Politecnico di Torino
IV Facoltà di Ingegneria
Corso di Laurea in Ing. Logistica e della Produzione
Corso di Laurea in Ing. dell'Organizzazione d'Impresa

06AZNEG/06AZNDI **- Fondamenti di Informatica**

Programmazione in C

Raccolta di lucidi

Anno accademico 2009/2010

Fulvio Corno

Programmazione in C

Unità
Primo programma in C

Primo programma in C

- Introduzione al linguaggio C
- Struttura minima di un file C
- Sottoinsieme minimale di istruzioni
- Compilare il primo programma
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitolo 1
  - Cabodi, Quer, Sonza Reorda: capitoli 1, 3
  - Dietel & Dietel: capitolo 1
- Dispense
  - Scheda: "Primo programma in C"

Primo programma in C

Introduzione al linguaggio C

Genesi del linguaggio C

- Sviluppato tra il 1969 ed il 1973 presso gli AT&T Bell Laboratories
  - B. Kernighan e D. Ritchie
  - Per uso interno
  - Legato allo sviluppo del sistema operativo Unix
- Nel 1978 viene pubblicato "The C Programming Language", prima specifica ufficiale del linguaggio
  - Detto "K&R"


Brian Kernighan

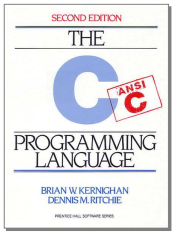

Dennis Ritchie

Obiettivi del linguaggio

- Insieme minimale di costrutti base
  - Semplicità del compilatore
- Sintassi estremamente sintetica
  - Talvolta criptica
- Pensato da programmatori per programmatori
  - Elevata efficienza
  - Per nulla user friendly
- Portabile
  - Indipendente dalla macchina
- Disponibilità di una libreria standard di funzioni

Evoluzione del linguaggio (1/2)

- 1978, K&R C
- 1989, ANSI C (o C89)
  - Frutto del lavoro di standardizzazione del comitato X3J11 dell'American National Standards Institute
  - Standard X3.159-1989 "Programming Language C"
  - Seconda edizione del K&R



Evoluzione del linguaggio (2/2)

- 1990, ISO C (o C90)
  - Ratifica da parte della International Organization for Standardization dello standard ANSI C
  - ISO/IEC 9899:1990
- 1999, ISO C99
  - Revisione compiuta negli anni '90
  - INCITS-ANSI/ISO/IEC 9899-1999
    - 550 pagine
    - http://www.open-std.org/jtc1/sc22/wg14/
  - Supportata da molti (non tutti) i compilatori

Diffusione attuale

- I linguaggi attualmente più diffusi al mondo sono:
  - C
  - C++, un'evoluzione del C
  - Java, la cui sintassi è tratta da C++
  - C#, estremamente simile a Java e C++
- Il linguaggio C è uno dei linguaggi più diffusi
- La sintassi del linguaggio C è ripresa da tutti gli altri linguaggi principali

Principali vantaggi del C

- Basato su relativamente pochi costrutti da apprendere
- Enorme disponibilità di documentazione ed esempi
- Buona disponibiltà di ambienti di sviluppo gratuiti
- Disponibile su qualsiasi configurazione hardware
- Elevata efficienza di elaborazione
- Adatto a vari tipi di applicazioni
  - Programmi di sistema
  - Elaborazione numerica
  - Programmi interattivi

Principali svantaggi del C

- Scarsa leggibilità di alcuni costrutti
- Facilità nel commettere errori di programmazione
  - Molti costrutti "pericolosi" sono permessi dal linguaggio e quindi non vengono segnalati dal compilatore
  - Alcuni errori di digitazione possono causare comportamenti errati
- Difficoltà nella realizzazione di interfacce grafiche
- Complessità nell'elaborazione dei testi

Un esempio

```
#include <stdio.h>

int main(void)
{
   printf("hello, world\n");

   return 0;
}
```

Primo programma in C

Struttura minima di un file C

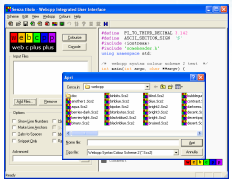
Struttura minima di un file C

- Applicazioni C in modo "console"
- Struttura del programma
- Commenti
- Direttive `#include`
- Definizione di variabili
- Corpo del `main`

Struttura minima di un file C

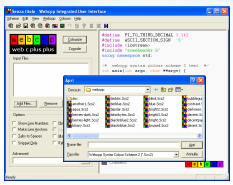
Applicazioni C in modo "console"

Tipi di applicazioni (1/4)

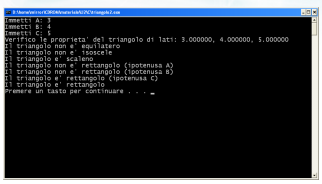


- Applicazioni grafiche
  - Interazione mediante mouse e finestre
  - Visualizzazione di testi e grafica
  - Elaborazione concorrente

Tipi di applicazioni (2/4)

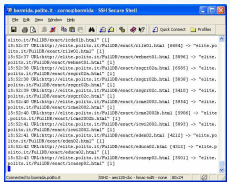


- Applicazioni grafiche
  - Interazione mediante mouse e finestre
  - Visualizzazione di testi e grafica
  - Elaborazione concorrente



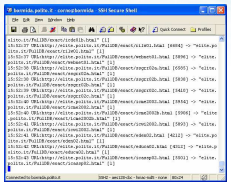
- Applicazioni "console"
  - Interazione mediante tastiera
  - Visualizzazione di soli caratteri
  - Elaborazione sequenziale

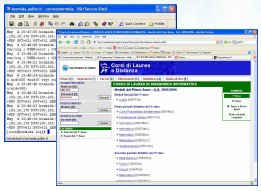
Tipi di applicazioni (3/4)



- Applicazioni batch
  - Nessuna interazione utente
  - Compiti lunghi e ripetitivi
  - Elaborazione numerica, trasferimenti in rete

Tipi di applicazioni (4/4)



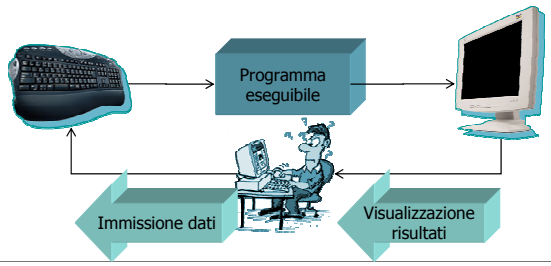


- Applicazioni batch
  - Nessuna interazione utente
  - Compiti lunghi e ripetitivi
  - Elaborazione numerica, trasferimenti in rete
- Applicazioni server
  - Nessuna interazione utente
  - Realizzano funzioni di sistema
  - Server locali o server Internet

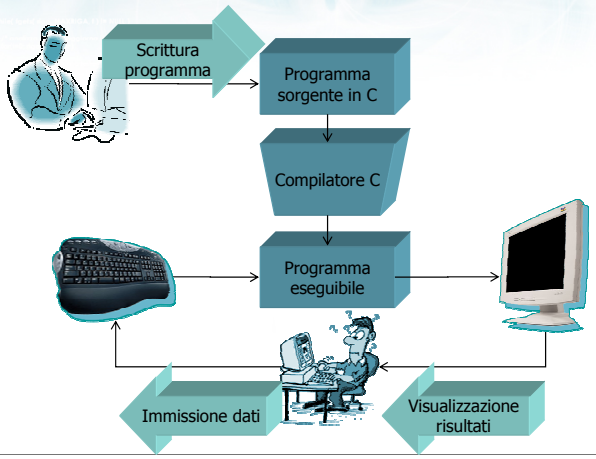
Applicazioni "console"

- Interazione utente limitata a due casi
  - Stampa di messaggi, informazioni e dati a video
  - Immissione di un dato via tastiera
- L'insieme tastiera+video viene detto terminale
- Nessuna caratteristica grafica
- Elaborazione
  - Sequenziale
  - Interattiva
  - Mono-utente

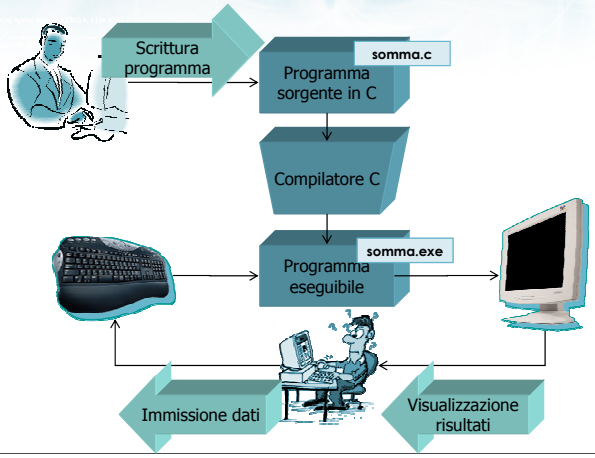
Modello di applicazioni "console"

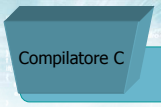


Modello di applicazioni "console"



Modello di applicazioni "console"



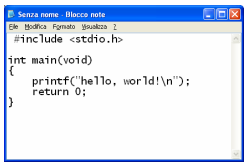


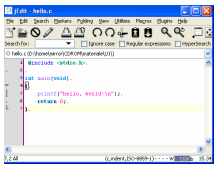
Compilatore C

- Traduce i programmi sorgenti scritti in linguaggio C in programmi eseguibili
- È a sua volta un programma eseguibile, a disposizione del programmatore
- Controlla l'assenza di errori di sintassi del linguaggio
- Non serve all'utente finale del programma
- Ne esistono diversi, sia gratuiti che commerciali

Scrittura del programma

- Un sorgente C è un normale file di testo
- Si utilizza un editor di testi
  - Blocco Note
  - Editor specializzati per programmatori



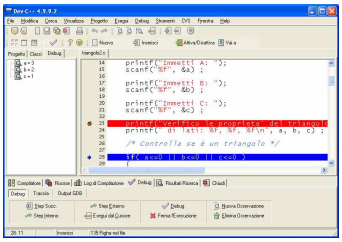


Editor per programmatori

- Colorazione ed evidenziazione della sintassi
- Indentazione automatica
- Attivazione automatica della compilazione
- Identificazione delle parentesi corrispondenti
- Molti disponibili, sia gratuiti che commerciali

Ambienti integrati

- Applicazioni software integrate che contengono al loro interno
  - Un editor di testi per programmatori
  - Un compilatore C
  - Un ambiente di verifica dei programmi (debugger)
- IDE: Integrated Development Environment



Struttura minima di un file C

Struttura del programma

Struttura di un sorgente in C

```
#include <stdio.h>

int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```


Struttura di un sorgente in C

```
#include <stdio.h>
```

Programma principale (funzione `main`)

```
int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Struttura di un sorgente in C

```
#include <stdio.h>

int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Parentesi graffe che delimitano il `main`

Struttura di un sorgente in C

```
#include <stdio.h>

int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Variabili utilizzate dal programma

Struttura di un sorgente in C

```
#include <stdio.h>

int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Istruzioni eseguibili

Struttura di un sorgente in C

```
#include <stdio.h>

int main(void)
{
   int a ;

   a = 3 ;

   printf("hello, world\n");
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Richiamo delle librerie utilizzate

In generale

```
#include delle librerie

int main(void)
{

   definizione variabili

   istruzioni eseguibili

}
```


Struttura minima di un file C

Commenti

Commenti

- Il testo presente in un sorgente C deve essere analizzato dal compilatore C, quindi deve sottostare a tutte le regole sintattiche del linguaggio
- Per aggiungere annotazioni, commenti, spiegazioni, note, ... si può usare un commento all'interno del sorgente

```
/* io sono un commento */
```


Sintassi

- Un commento è una qualsiasi sequenza di caratteri (anche su più righe) che:
  - Inizia con la coppia di caratteri /*
  - Termina con la coppia di caratteri */
- Non è permesso annidare commenti
  - All'interno di un commento non devono comparire i caratteri /*
- Tutto ciò che è compreso tra /* e */ viene ignorato dal compilatore C

Esempio

```
/* programma: hello.c
   autore: fulvio corno
 */

/* accedi alla libreria standard */
#include <stdio.h>

int main(void)
{
   int a ;  /* numero magico */

   a = 3 ;  /* assegno un valore */

   /* salutiamo l'utente */
   printf("hello, world\n") ;
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Spazi bianchi

- Oltre ai commenti, il compilatore ignora tutti gli spazi bianchi
  - Spazi tra un'istruzione e la successiva
  - Spazi ad inizio linea
  - Spazi intorno alla punteggiatura
  - Righe vuote
- La spaziatura viene utilizzata per rendere il sorgente C più ordinato e più facilmente comprensibile

Esempio

```
/* programma: hello.c  autore: fulvio corno */
/* accedi alla libreria standard */
#include <stdio.h>
int main(void)
{ int a ;  /* numero magico */ a = 3 ;
/* assegno un valore */
/* salutiamo l'utente */ printf("hello, world\n") ;
printf("the magic number is %d\n", a) ; return 0;
}
```


Esempio

```
/* programma: hello.c  autore: fulvio corno */
/* accedi alla libreria standard */
#include <stdio.h>
int main(void)
{ int a ;  /* numero magico */ a = 3 ;
/* assegno un valore */
/* salutiamo l'utente */ printf("hello, world\n") ;
printf("the magic number is %d\n", a) ; return 0;
}
```

```
#include <stdio.h>
int main(void)
{ int a ; a = 3 ; printf("hello, world\n") ;
printf("the magic number is %d\n", a) ; return 0; }
```

Struttura minima di un file C

Direttive `#include`

Librerie di funzioni

- Ogni compilatore C dispone di diverse librerie di funzioni già pronte per l'uso
- Il programmatore può utilizzare le funzioni di libreria
- È necessario dichiarare a quali librerie si vuole avere accesso
  - Direttive `#include` ad inizio programma
  - Aggiunge al programma le dichiarazioni di tutte le funzioni di tale libreria, permettendo al programmatore di richiamare tali funzioni

Sintassi

```
#include < NomeLibreria .h>
```

- Librerie principali:
  - `#include <stdio.h>`
    - Funzioni di lettura/scrittura su terminale e su file
  - `#include <stdlib.h>`
    - Funzioni base per interazione con sistema operativo
  - `#include <math.h>`
    - Funzioni matematiche
  - `#include <string.h>`
    - Elaborazione di testi

Avvertenze

- A differenza della regola generale, nelle direttive `#include` la spaziatura è importante
  - Il carattere # deve essere il primo della riga
  - Può esserci una sola `#include` per riga
  - La direttiva `#include` non va terminata con il ;
- Dimenticare una `#include` potrà portare ad errori nel corpo del `main`, quando si chiameranno le funzioni relative

Suggerimenti

- Iniziare sempre il sorgente C con le seguenti linee:

```
/* programma: NomeFile.c
   autore: NomeAutoreDelProgramma
   BreveDescrizioneDelProgramma
 */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main(void)
{

   . . . .

}
```


Struttura minima di un file C

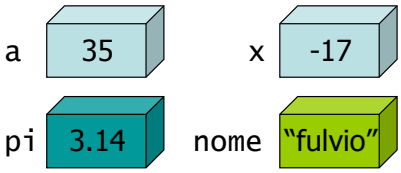
Definizione di variabili

Variabili

- Il programma memorizza le informazioni sulle quali lavora all'interno di variabili
- Ogni variabile è caratterizzata da:
  - Tipo di dato
  - Nome
  - Valore corrente

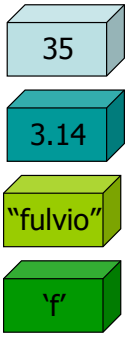
Variabili

- Il programma memorizza le informazioni sulle quali lavora all'interno di variabili
- Ogni variabile è caratterizzata da:
  - Tipo di dato
  - Nome
  - Valore corrente



Tipo di dato

- Definisce l'insieme dei valori ammissibili per la variabile



Numeri interi, positivi o negativi

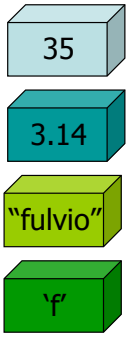
Numeri reali

Stringhe di testo

Singoli caratteri di testo

Tipo di dato

- Definisce l'insieme dei valori ammissibili per la variabile



Numeri interi, positivi o negativi

Numeri reali

Stringhe di testo

Singoli caratteri di testo

Nome

- Il programmatore assegna un nome a ciascuna variabile
- Dovrebbe rappresentare lo scopo dei valori contenuti nella variabile
- Sintetico, rappresentativo, mnemonico, facile da scrivere

Nomi ammissibili

- Il primo carattere deve essere una lettera
- I successivi possono essere lettere o numeri
- Lettere maiuscole e minuscole sono diverse
- Il simbolo _ viene considerato come una lettera
- Non devono essere nomi riservati dal linguaggio

Esempi di nomi

a b a1 a2

Esempi di nomi

a b a1 a2

num n N somma max

Esempi di nomi

a b a1 a2

num n N somma max

area perimetro perim

Esempi di nomi

a b a1 a2

num n N somma max

area perimetro perim

n_elementi Nelementi risultato

Esempi di nomi

a b a1 a2

num n N somma max

area perimetro perim

n_elementi Nelementi risultato

trovato nome risposta

Definizione di variabili

- Ogni variabile deve essere definita prima di poterla utilizzare
- Definizioni all'inizio della funzione `main`
- Sintassi della definizione
  - *TipoVariabile NomeVariabile* ;

```
int main(void)
{
   int a ;
   int b ;
   float x ;

   . . . . .
}
```

Definizione di variabili

- Ogni variabile deve essere definita prima di poterla utilizzare
- Definizioni all'inizio della funzione `main`
- Sintassi della definizione
  - *TipoVariabile NomeVariabile ;*
  - *TipoVariabile NomeVariabile, NomeVariabile ;*

```
int main(void)
{
   int a ;
   int b ;
   float x ;

   . . . . .
}
```

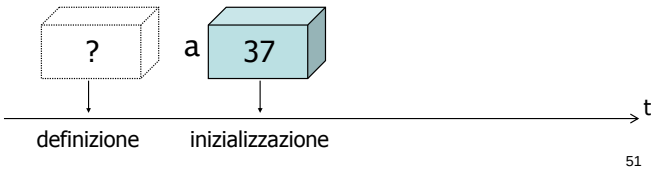
```
int main(void)
{
   int a, b ;
   float x ;

   . . . . .
}
```


Valore contenuto

- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
  - Variabili non inizializzate
- In momenti diversi il valore può cambiare

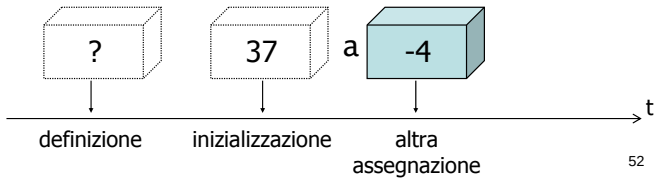
Valore contenuto

- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
  - Variabili non inizializzate
- In momenti diversi il valore può cambiare



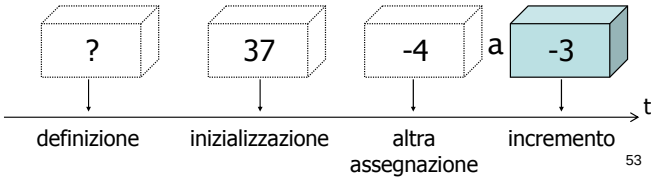
Valore contenuto

- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
  - Variabili non inizializzate
- In momenti diversi il valore può cambiare



Valore contenuto

- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
  - Variabili non inizializzate
- In momenti diversi il valore può cambiare



Struttura minima di un file C

Corpo del `main`

Istruzioni eseguibili

- La funzione `main`, dopo le definizioni di variabili, contiene le vere e proprie istruzioni eseguibili
- Ciascuna istruzione è terminata da `;`
- Tutte le istruzioni sono comprese nelle `{ ... }`
- Le istruzioni vengono eseguite in ordine
- Dopo aver eseguito l'ultima istruzione, il programma termina

Esempio

```
/* programma: hello.c
   autore: fulvio corno
 */

/* accedi alla libreria standard */
#include <stdio.h>

int main(void)
{
   int a ;  /* numero magico */

   a = 3 ;  /* assegno un valore */

   /* salutiamo l'utente */
   printf("hello, world\n") ;
   printf("the magic number is %d\n", a) ;

   return 0;
}
```

Tipologie di istruzioni

- Istruzioni operative
  - Lettura dati
    - `scanf("%d", &a) ;`
  - Stampa risultati
    - `printf("%d", a) ;`
  - Elaborazione numerica
    - `a = b + c ;`
    - `b = b + 1 ;`
    - `c = 42 ;`
    - `c = sqrt(a) ;`

Tipologie di istruzioni

- Istruzioni operative
  - Lettura dati
    - `scanf("%d", &a) ;`
  - Stampa risultati
    - `printf("%d", a) ;`
  - Elaborazione numerica
    - `a = b + c ;`
    - `b = b + 1 ;`
    - `c = 42 ;`
    - `c = sqrt(a) ;`
- Istruzioni di controllo
  - Modificano il controllo di flusso
    - Scelte
    - Iterazioni
    - Chiamate a funzioni
    - Interruzioni e salti
  - Predefinite dal linguaggio C
    - `if else while for return switch case break continue goto`

Primo programma in C

Sottoinsieme minimale di istruzioni

Sottoinsieme minimale di istruzioni

- I tipi `int` e `float`
- Istruzione `printf` – semplificata
- Istruzione `scanf` – semplificata
- Istruzione di assegnazione
- Semplici espressioni aritmetiche

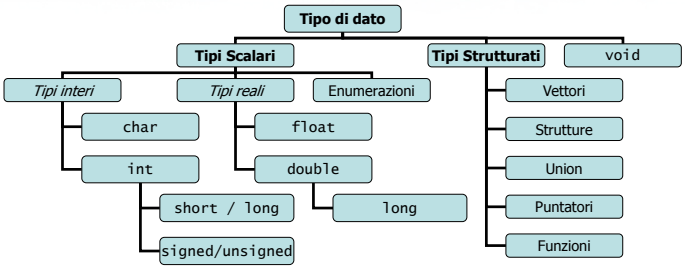
Sottoinsieme minimale di istruzioni

I tipi `int` e `float`

Tipi di dato

- Ogni costante, ogni variabile, ogni espressione appartiene ad un determinato tipo
- Il tipo determina
  - L'insieme dei valori che la costante, variabile o espressione può assumere
  - L'insieme delle operazioni lecite su tali valori
- I tipi possono essere
  - Semplici (o scalari): singoli valori
  - Strutturati: insiemi di più valori semplici

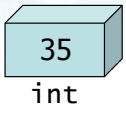
Il sistema dei tipi C



Caratteristiche generali

- I valori ammessi per ciascun tipo non sono fissati dallo standard
- Dipendono dal compilatore e dal sistema operativo
  - Ampiezza dei tipi di dato "naturale" per ogni calcolatore
- Maggior portabilità
- Maggior efficienza
- Nessuna garanzia di uniformità tra piattaforme diverse

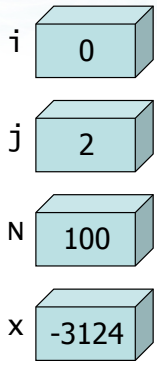
Il tipo int



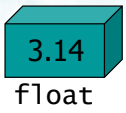
- Il tipo più importante del linguaggio C
- Valori interi, positivi o negativi
- Codificato in complemento a due
- Espresso solitamente su 16 bit oppure 32 bit
  - 16 bit: da –32 768 a +32 767
  - 32 bit: da –2 147 483 648 a +2 147 483 647
  - In generale: da `INT_MIN` a `INT_MAX`
    - `#include <limits.h>`

Esempi

```
int i, j ;
int N ;
int x ;

i = 0 ;
j = 2 ;
N = 100 ;
x = -3124 ;
```



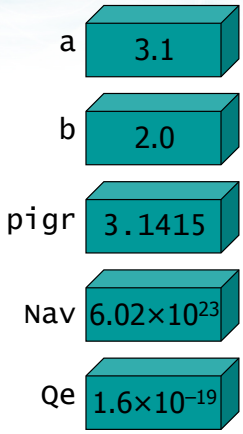
Il tipo float



- Valori reali
  - Frazionari
  - Esterni all'intervallo permesso per i numeri interi
- Codificato in virgola mobile, singola precisione
- Espresso solitamente su 32 bit
  - da $\pm 1.17 \times 10^{-38}$ a $\pm 3.40 \times 10^{+38}$
  - circa 6 cifre di precisione
  - In generale: da `FLT_MIN` a `FLT_MAX`
    - `#include <float.h>`

Esempi

```
float a, b ;
float pigr ;
float Nav, Qe ;

a = 3.1 ;
b = 2.0 ;
pigr = 3.1415926 ;
Nav = 6.022e23 ;
Qe = 1.6e-19 ;
```

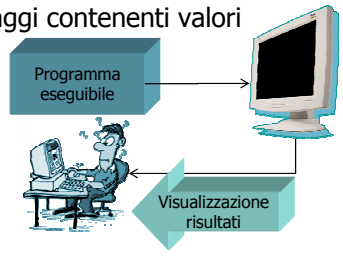


Sottoinsieme minimale di istruzioni

Istruzione `printf` – semplificata

Istruzioni di stampa

- Stampa di messaggi informativi
- Stampa di comando "a capo"
- Stampa di valori di variabili
- Stampa di valori di espressioni calcolate
- Stampa di messaggi contenenti valori



Stampa di messaggi

```
printf("Benvenuto\n") ;
printf("Immetti un numero: ") ;
printf("\n");
```

```
Prompt dei comandi

Benvenuto

Immetti un numero: _
```

Stampa di variabili

```
printf("%d ", j) ;
printf("%d\n", N) ;
printf("%f\n", pigr) ;
printf("%f\n", Nav) ;
```

```
Prompt dei comandi

2 100

3.141593

602200013124147500000000.000000
```

Stampa di espressioni

```
printf("%d\n", i-j) ;
printf("%d\n", N*2) ;
printf("%f\n", Nav * Qe) ;
```

```
Prompt dei comandi

-2

200

96352.000000
```

Stampa di messaggi e valori

```
printf("Risultato=%d\n", N*2) ;
printf("Angolo = %f radianti\n", pigr/4);
```

```
Risultato=200

Angolo = 0.785398 radianti
```

Sintassi istruzione `printf`

- `#include <stdio.h>`
- `printf("`*formato*`", `*valore/i*`) ;`
- Formato:
  - Testo libero (compresi spazi) → viene stampato letteralmente
  - Simboli `\n` → va a capo
  - Simboli `%d` → stampa un `int`
  - Simboli `%f` → stampa un `float`
- Valore/i:
  - Variabile o espressione
  - Di tipo `int` o `float`, corrispondente al simbolo %

Casi particolari (1/2)

- Per stampare il simbolo % occorre ripeterlo due volte
  - `printf("Sondaggio: %f%%\n", pSI ) ;`
    - `%f` → stampa `pSI`
    - `%%` → stampa un simbolo %
    - `\n` → va a capo
  - `Sondaggio: 43.12%`

Casi particolari (2/2)

- È possibile stampare più di un valore nella stessa istruzione
  - `printf("Voti: %d su %d\n", voti, tot ) ;`
    - primo %d → stampa `voti`
    - secondo %d → stampa `tot`
  - `Voti: 18 su 45`

Sottoinsieme minimale di istruzioni

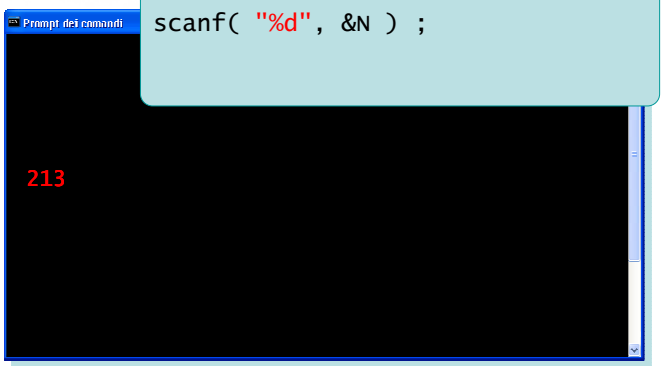
Istruzione `scanf` – semplificata

Istruzioni di lettura

- Lettura di un valore intero
- Lettura di un valore reale

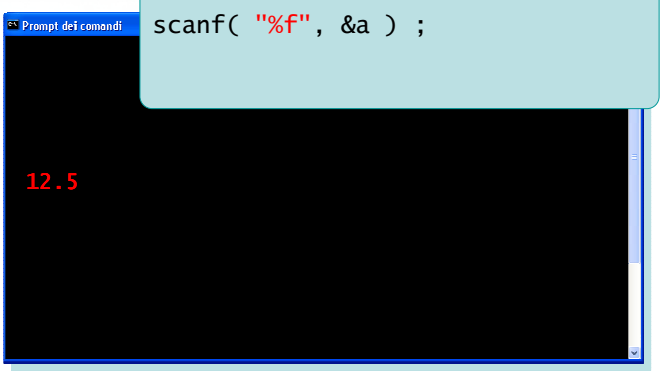
Lettura di un intero

```
scanf( "%d", &N ) ;
```



Lettura di un reale

```
scanf( "%f", &a ) ;
```



Sintassi istruzione `scanf`

- `#include <stdio.h>`
- `scanf("`*formato*`", &`*variabile*`) ;`
- Formato:
  - Simboli %d → legge un `int`
  - Simboli %f → legge un `float`
- Variabile:
  - Di tipo `int` o `float`, corrispondente al simbolo %
  - Sempre preceduta dal simbolo &

Suggerimento

- Combinare le istruzioni `printf` e `scanf` per guidare l'utente nell'immissione dei dati
  - Ogni `scanf` deve essere preceduta da una `printf` che indica quale dato il programma si aspetta

```
printf("Immetti il numero: ");
scanf("%d", &N) ;
printf("Numero immesso: %d\n", N);
```


Errore frequente

- Dimenticare il simbolo & nelle istruzioni `scanf`

```
printf("Immetti il numero: ");
scanf("%d", N) ;
```

forma corretta

```
printf("Immetti il numero: ");
scanf("%d", &N) ;
```


Errore frequente

- Dimenticare le variabili da stampare nelle istruzioni `printf`

```
printf("Numero immesso: %d\n");
```

forma corretta

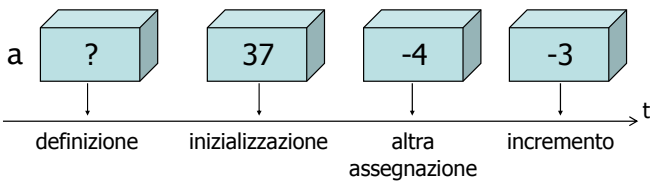
```
printf("Numero immesso: %d\n", N);
```


Sottoinsieme minimale di istruzioni

Istruzione di assegnazione

Assegnazione delle variabili

- Il valore di una variabile
  - Deve essere inizializzato, la prima volta
  - Può essere aggiornato, quante volte si vuole
- Per assegnare un nuovo valore ad una variabile si usa l'operatore =

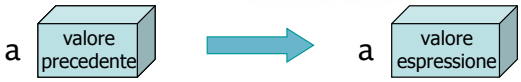


Assegnazione di variabili

- Assegnazione del valore di una costante
  - `i = 0 ;`
  - `a = 3.0 ;`
- Assegnazione del valore di un'altra variabile
  - `i = N ;`
  - `b = a ;`
- Assegnazione del valore di un'espressione
  - `j = N - i ;`
  - `b = a * 2 - 1 ;`

Sintassi dell'assegnazione

- *variabile* = *espressione* ;



- Passo 1: si valuta il valore corrente dell'espressione
  - Per tutte le variabili che compaiono nell'espressione, si usa il valore corrente
  - Può comparire anche la stessa variabile oggetto dell'assegnazione
- Passo 2: si memorizza tale valore nella variabile, cancellando il valore precedente

Esempi

- `N = 3 ;`

Esempi

- `N = 3 ;`
- `a = b ;`
  - Non confondere con `b = a ;`

Esempi

- `N = 3 ;`
- `a = b ;`
  - Non confondere con `b = a ;`
- `a = a + 1 ;`
  - Incrementa a di un'unità

Esempi

- `N = 3 ;`
- `a = b ;`
  - Non confondere con `b = a ;`
- `a = a + 1 ;`
  - Incrementa a di un'unità
- `a + 1 = a ;`
  - Errore

Quesito

- Che operazione svolge il seguente frammento di programma?

```
a = b ;
b = a ;
```

Risposta

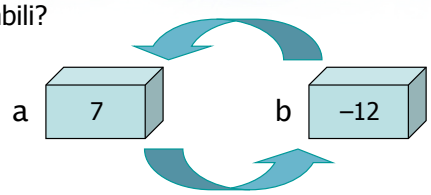
- Che operazione svolge il seguente frammento di programma?

```
a = b ;
b = a ;
```

- Il valore corrente di b viene copiato in a
  - Il valore vecchio di a viene perso
- Il (nuovo) valore corrente di a (uguale a b) viene ricopiato in b (operazione inutile)

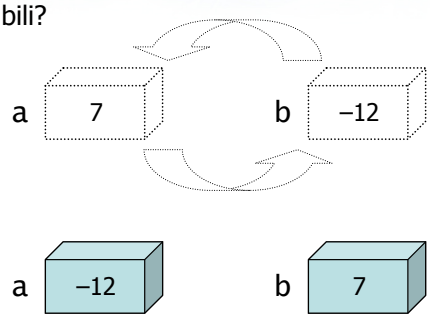
Quesito

- Come fare a scambiare tra di loro i valori di due variabili?



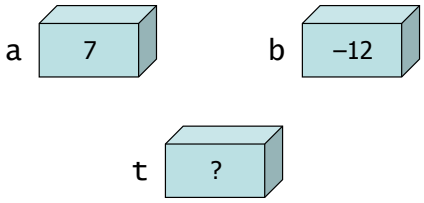
Quesito

- Come fare a scambiare tra di loro i valori di due variabili?



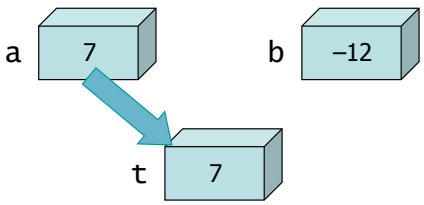
Risposta

```
t = a ;
a = b ;
b = t ;
```



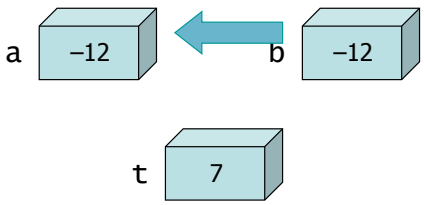
Risposta

```
t = a ;
a = b ;
b = t ;
```



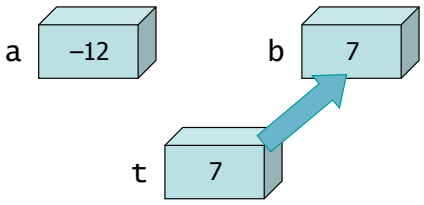
Risposta

```
t = a ;
a = b ;
b = t ;
```



Risposta

```
t = a ;
a = b ;
b = t ;
```



Sottoinsieme minimale di istruzioni

Semplici espressioni aritmetiche

Espressioni aritmetiche

- Ovunque sia richiesto il valore di una variabile, è possibile usare un'espressione aritmetica
  - Nei valori da stampare con la funzione `printf`
  - Nei valori da assegnare ad una variabile
- Le espressioni si possono costruire ricorrendo a:
  - Operatori: `+ - * /`
  - Parentesi: `( ... )`
  - Funzioni di libreria: `sqrt()`, `sin()`, `cos()`, ...

Operatori principali

- Somma: `a+b`
- Sottrazione: `a-b`
- Moltiplicazione: `a*b`
- Divisione: `a/b`
  - Divisione intera (risultato troncato) se entrambi gli operandi sono `int`
- Resto della divisione: `a%b`
  - Solo tra operandi `int`
- Cambio di segno: `-a`

Alcuni operatori avanzati

- Incremento: `i++`
- Decremento: `N--`
- Conversione ad intero: `(int)a`
- Conversione a reale: `(float)N`

Funzioni di libreria

- `#include <math.h>`
- Funzioni algebriche
  - `fabs`, `sqrt`, `cbrt`, `pow`, `hypot`, `ceil`, `floor`, `round`, `trunc`, `fmod`
- Funzioni esponenziali e logaritmiche
  - `exp`, `exp2`, `log`, `log10`, `log2`
- Funzioni trigonometriche e iperboliche
  - `cos`, `sin`, `tan`, `cosh`, `sinh`, `tanh`
- Funzioni trigonometriche e iperboliche inverse
  - `acos`, `asin`, `atan`, `atan2`, `acosh`, `asinh`, `atanh`

Parentesi

- Si possono costruire espressioni complicate a piacere utilizzando le parentesi
- Per maggiore leggibilità, abbondare con le parentesi ed usare la spaziatura e l'incolonnamento in modo ordinato
- Si utilizzano sempre le parentesi tonde

```
x1 = ( -b + sqrt( b*b - 4*a*c ) ) /
     ( 2*a ) ;
```

```
A = sqrt( p * (p-a) * (p-b) * (p-c)) ;
```

Primo programma in C

Compilare il primo programma

Compilare il primo programma

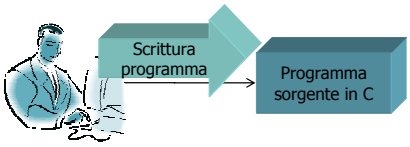
- Un semplice programma
- L'ambiente di sviluppo Dev-C++
- Codifica del programma
- Compilazione e correzione errori
- Esecuzione e verifica

Compilare il primo programma

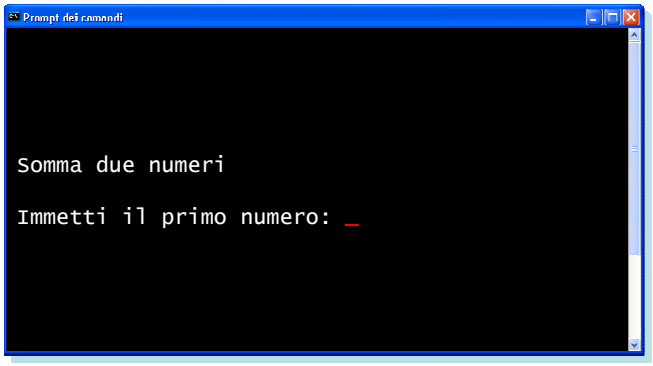
Un semplice programma

Esercizio "Somma due numeri"

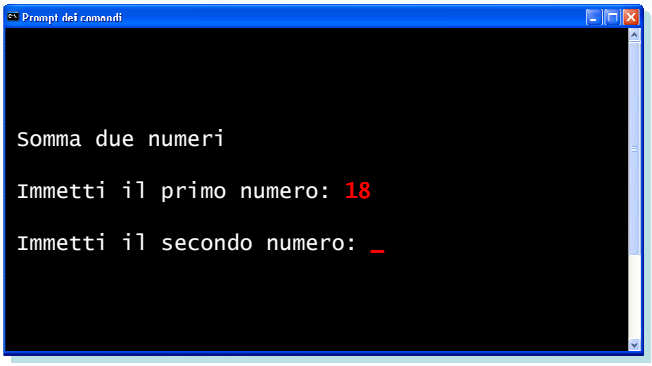
- Si realizzi un programma in linguaggio C che acquisisca da tastiera due numeri interi (detti A e B) e che stampi a video il valore della somma di tali numeri



Analisi



Analisi



Analisi

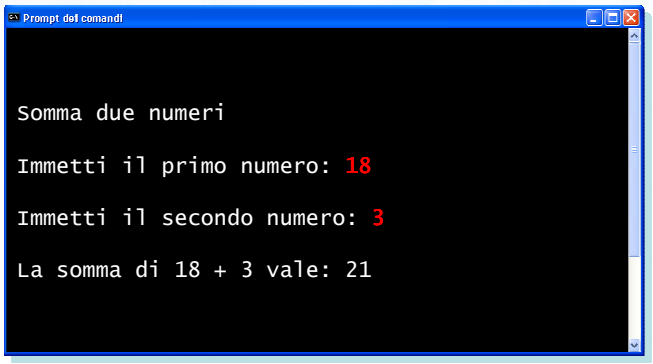
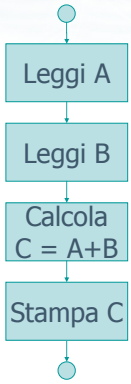
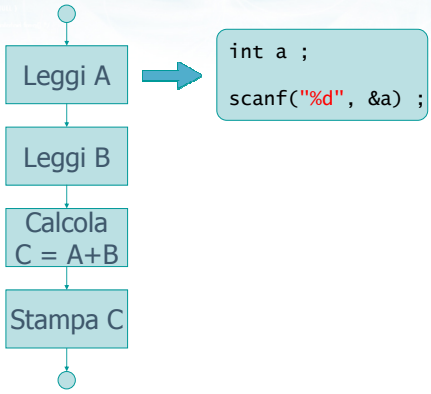


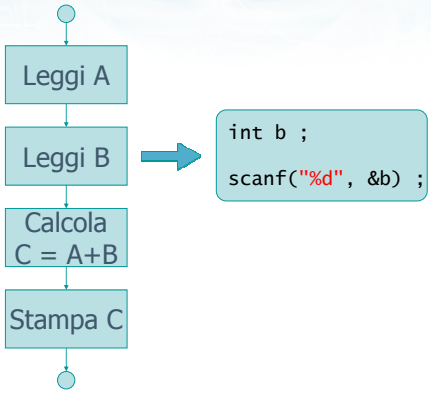
Diagramma di flusso



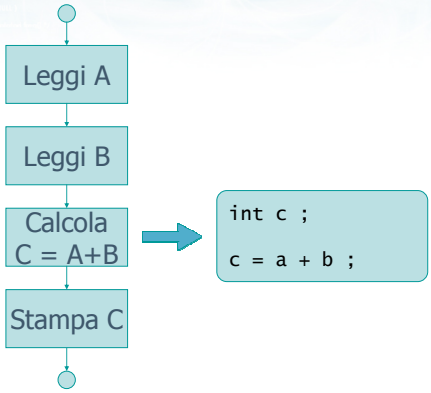
Traduzione in C (1/4)



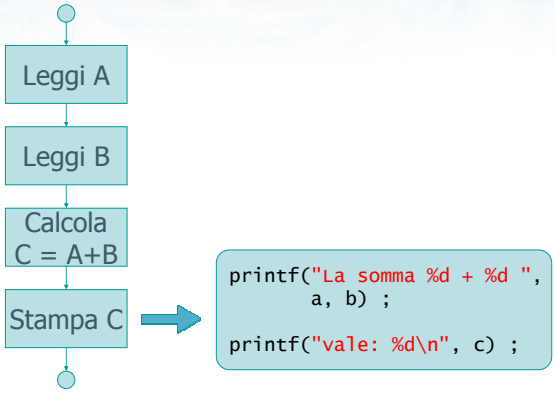
Traduzione in C (2/4)



Traduzione in C (3/4)



Traduzione in C (4/4)



Compilare il primo programma

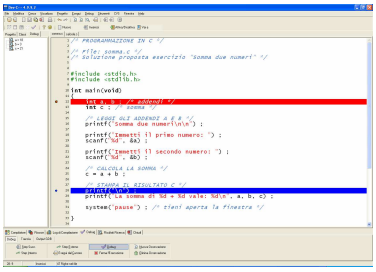
L'ambiente di sviluppo Dev-C++

Compilatori e IDE

- Occorre identificare ed installare
  - Un editor (possibilmente per programmatori)
  - Un compilatore
  - Un debugger
- Oppure trovare un Integrated Development Environment che integri tutte le funzionalità precedenti
- Esistono molte soluzioni gratuite

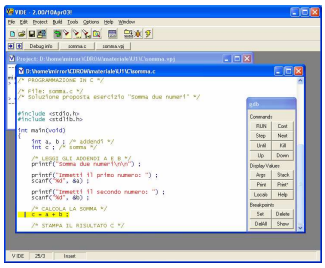
IDE per C in ambiente Windows

- Dev-C++
  - http://www.bloodshed.net



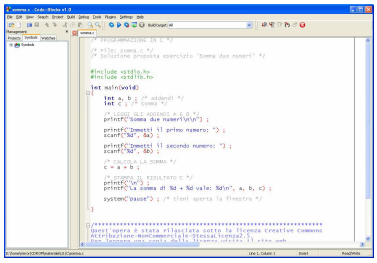
IDE per C in ambiente Windows

- V IDE
  - http://www.objectcentral.org



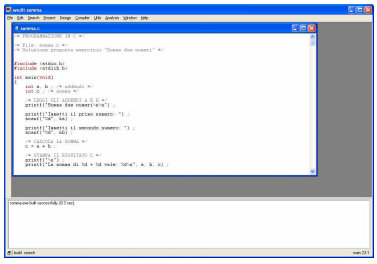
IDE per C in ambiente Windows

- Code::Blocks
  - http://www.codeblocks.org

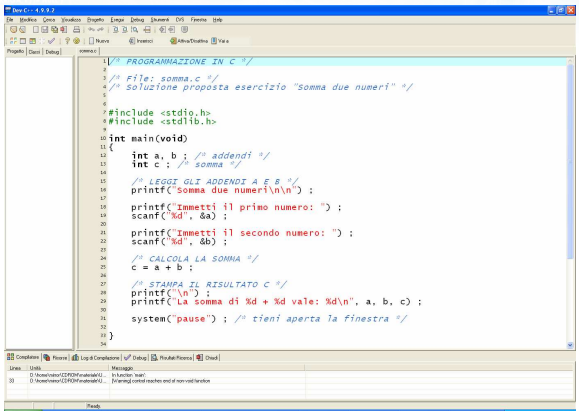


IDE per C in ambiente Windows

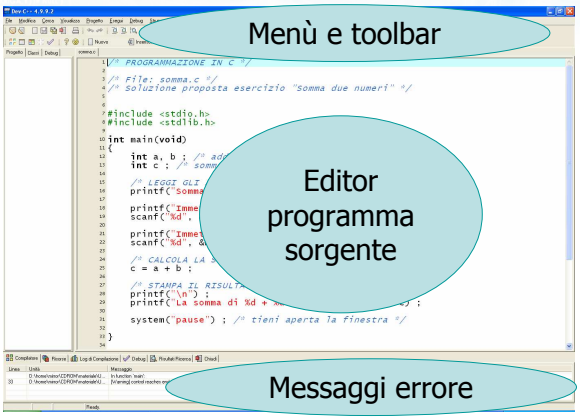
- lcc-win32
  - http://www.cs.virginia.edu/~lcc-win32/



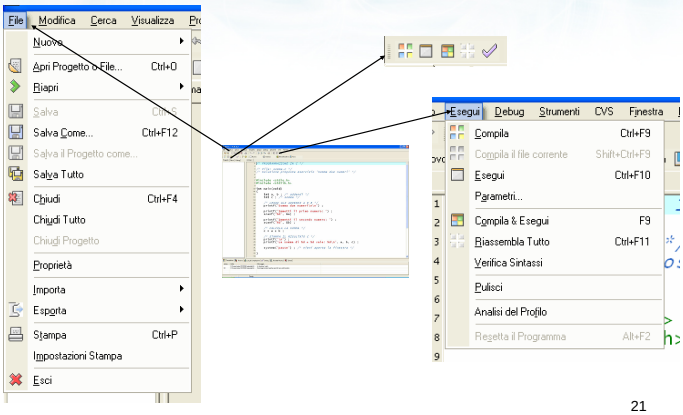
Interfaccia di Dev-C++



Interfaccia di Dev-C++



Menu principali

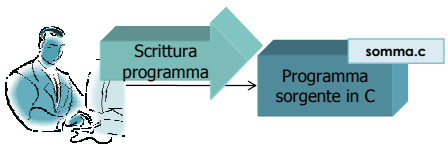


Compilare il primo programma

Codifica del programma

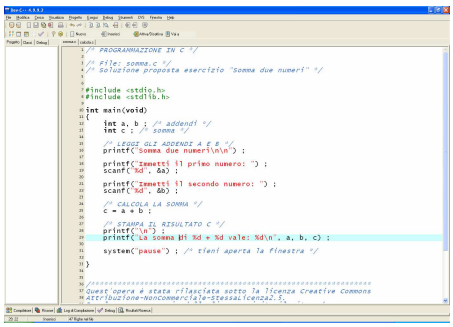
Codifica del programma

- A partire dal diagramma di flusso
- Utilizziamo un editor per immettere le istruzioni C
- Creiamo un file sorgente `somma.c`



Codifica "Somma due numeri"

- Codifichiamo il programma in Dev-C++



Soluzione proposta (1/2)

somma.c

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int a, b ; /* addendi */
    int c ; /* somma */

    /* LEGGI GLI ADDENDI A E B */
    printf("Somma due numeri\n\n") ;

    printf("Immetti il primo numero: ") ;
    scanf("%d", &a) ;
    printf("Immetti il secondo numero: ") ;
    scanf("%d", &b) ;
```


Soluzione proposta (2/2)

somma.c

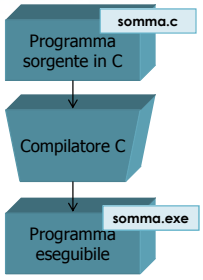
```
    /* CALCOLA LA SOMMA */
    c = a + b ;

    /* STAMPA IL RISULTATO C */
    printf("La somma di %d + %d vale: %d\n",
           a, b, c) ;
}
```


Compilare il primo programma

Compilazione e correzione errori

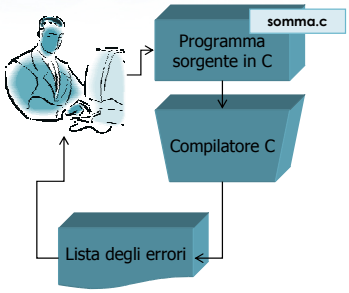
Compilazione del programma

- Attivare il compilatore sul programma sorgente `somma.c`
- Il compilatore verifica che non ci siano errori di sintassi
- In assenza di errori, viene generato il programma eseguibile `somma.exe`



Correzione errori di sintassi

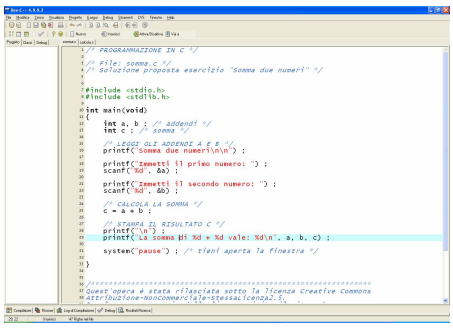
- Il compilatore genera una lista di messaggi di errore
- Capire il messaggio
- Identificare il punto errato nel programma
- Trovare la soluzione
- Correggere il programma
- Generare una nuova versione del file sorgente



Compilazione "Somma due numeri"

somma.c

- Compiliamo il programma

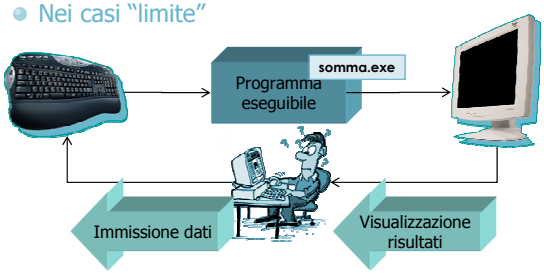


Compilare il primo programma

Esecuzione e verifica

Verifica del programma

- Ci mettiamo nei panni dell'utente finale
- Eseguiamo il programma
- Verifichiamo che funzioni correttamente
  - Nei casi "normali"
  - Nei casi "limite"



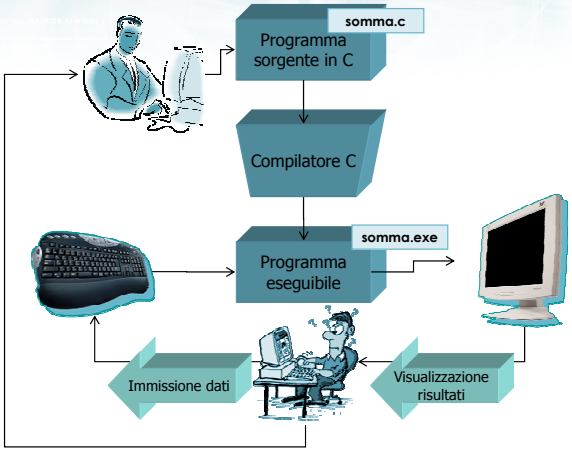
Errori in esecuzione

- Tipologie di errori possibili:
  - Crash del programma
    - Blocco imposto dal sistema operativo
  - Blocco del programma
    - Ciclo "infinito"
  - Risultati errati
    - (Quasi) sempre
    - Solo in alcuni casi (con alcuni dati ma non con altri)

Correzione errori di esecuzione

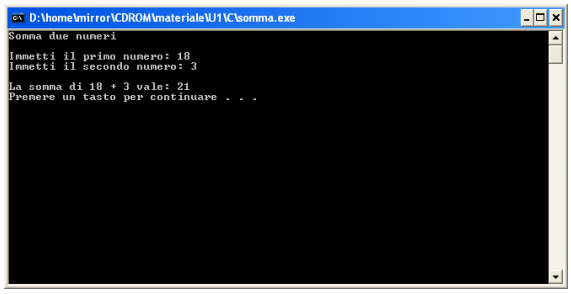
- Lavoro da "detective"
- Risalire dai sintomi alle cause del malfunzionamento
- Formulare delle ipotesi sulla causa dell'errore e verificarle
- Una volta trovato l'errore, cercare una soluzione
- A seconda della gravità, occorrerà modificare
  - Il sorgente C
  - L'algoritmo risolutivo
  - L'approccio generale

Correzione errori di esecuzione



Verifica "Somma due numeri"

- Eseguiamo il programma con alcuni dati di prova, verificandone il comportamento corretto



Primo programma in C

Esercizi proposti

Esercizi proposti

- Esercizio "Equazione di primo grado"
- Esercizio "Calcolo di aree"
- Esercizio "Somma minuti"

Esercizi proposti

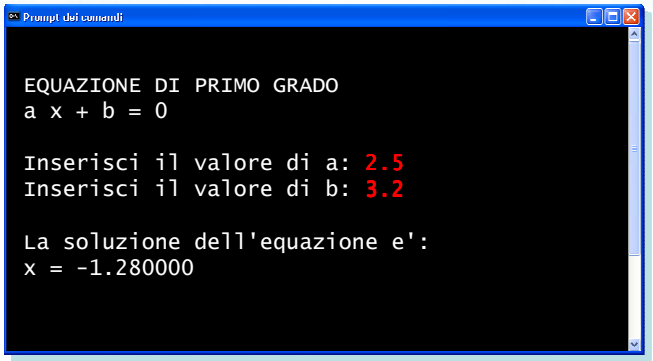
Esercizio "Equazione di primo grado"

Esercizio "Equazione di primo grado"

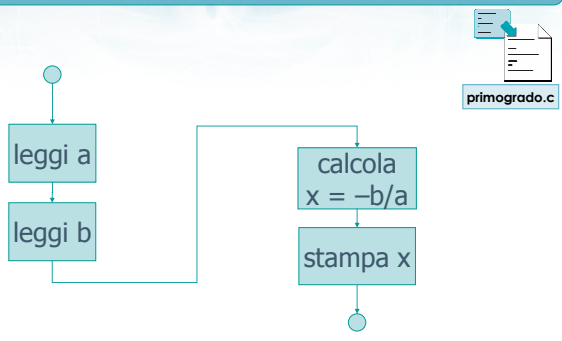
- Data l'equazione
  - a x + b = 0

  con a e b inseriti da tastiera, determinare il valore di x che risolve l'equazione

Analisi



Soluzione



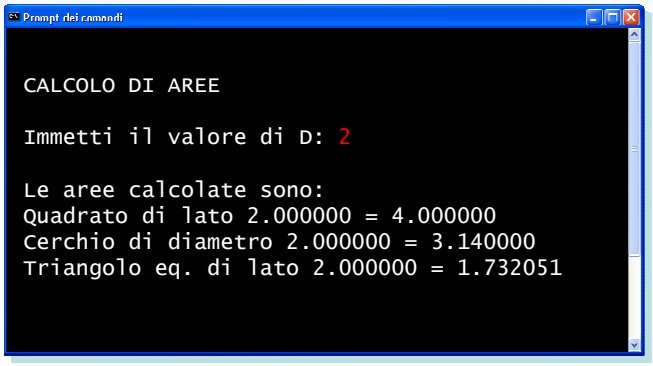
Esercizi proposti

Esercizio "Calcolo di aree"

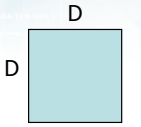
Esercizio "Calcolo di aree"

- Si scriva un programma in linguaggio C che, dato un numero reale immesso da tastiera, detto D, calcoli e stampi:
  - L'area del quadrato di lato D
  - L'area del cerchio di diametro D
  - L'area del triangolo equilatero di lato D

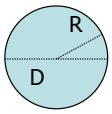
Analisi

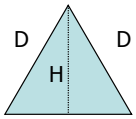


Aree

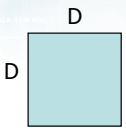


$$A = D^2$$

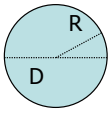




Aree

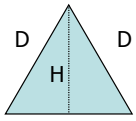


$$A = D^2$$

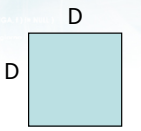


$$A = \pi \cdot R^2$$

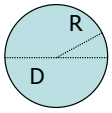
$$R = D/2$$



Aree

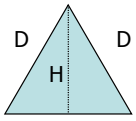


$$A = D^2$$



$$A = \pi \cdot R^2$$

$$R = D/2$$



$$A = \frac{D \cdot H}{2}$$

$$H = D \cdot \sin(60°) =$$

$$= D \cdot \sin(\pi/3) = D \cdot \sqrt{3}/2$$

Avvertenze

- Per le funzioni matematiche (`sin`, `sqrt`, …) occorre includere `math.h`
- Gli argomenti delle funzioni trigonometriche (`sin`, `cos`, …) devono essere espressi in radianti
- Il calcolo del quadrato si ottiene moltiplicando la variabile per se stessa: $D^2 = D \times D$
- Il valore di π deve essere definito dal programmatore in un'apposita variabile
  - La costante `M_PI`, definita in `math.h`, non è più supportata dallo standard ANSI C

Esercizi proposti

Esercizio "Somma minuti"

Esercizio "Somma minuti" (1/2)

- Un consulente deve calcolare il numero di ore e minuti per cui ha lavorato per un cliente
- Il consulente ha lavorato in due distinte sessioni di lavoro, per ciascuna delle quali ha annotato il numero di ore e il numero di minuti impiegati

Esercizio "Somma minuti" (2/2)

- Si scriva un programma in C che, a partire dalle ore e minuti della prima sessione di lavoro e dalle ore e minuti della seconda sessione di lavoro, calcoli il numero di ore e minuti complessivi

Analisi

```
Prompt dei comandi
SOMMA MINUTI

Sessione di lavoro 1:
Numero di ore: 2
Numero di minuti: 45

Sessione di lavoro 2:
Numero di ore: 1
Numero di minuti: 30

Tempo totale: 4 ore e 15 minuti
```


Aritmetica dell'orologio

- Diciamo:
  - ore1, min1 le ore/minuti della prima sessione
  - ore2, min2 le ore/minuti della seconda sessione
  - oretot, mintot le ore/minuti totali
- Non è possibile semplicemente sommare ore e minuti separatamente, in quanto min1+min2 potrebbe essere maggiore di 59
- Bisogna tener conto del "riporto" nella somma dei minuti

Soluzione

- mintot = (min1 + min2) modulo 60
- oretot = ore1 + ore2 + riporto
  - riporto = parte intera di (min1 + min2) / 60

Soluzione

- mintot = (min1 + min2) modulo 60
- oretot = ore1 + ore2 + riporto
  - riporto = parte intera di (min1 + min2) / 60

```
int ore1, ore2, oretot ;
int min1, min2, mintot, riporto ;

...

mintot = (min1 + min2) % 60 ;

riporto = (min1 + min2) / 60 ;

oretot = ore1 + ore2 + riporto ;
```

Primo programma in C

Sommario

Argomenti trattati

- Presentazione del linguaggio C
- Struttura base di un file sorgente in C
- Istruzioni minime per iniziare a programmare
  - Tipi fondamentali `int` e `float`
  - Istruzioni fondamentali di input/output
  - Istruzione di assegnazione
- Operazioni necessarie per compilare ed eseguire il programma

Suggerimenti

- Analizzare sempre il comportamento previsto del programma prima di iniziare a scrivere il sorgente
  - Interazione con l'utente
  - Risoluzione manuale con carta e penna
- Abbondare con i commenti
- Leggere con attenzione tutti i messaggi di errore e di warning del compilatore, e correggerli
- Verificare il programma con diversi dati di prova

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Scelte ed alternative

Scelte ed alternative

- Il controllo di flusso
- Istruzione `if-else`
- Condizioni complesse
- Istruzioni `if-else` annidate
- Istruzione `switch`
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitolo 3
  - Cabodi, Quer, Sonza Reorda: capitoli 2, 4
  - Dietel & Dietel: capitoli 2, 3
- Dispense
  - Scheda: "Istruzioni di scelta in C"

Scelte ed alternative

Il controllo di flusso

Il controllo di flusso

- Concetto di flusso e di scelta
- Rappresentazione grafica
- Condizioni booleane semplici
- Esempio

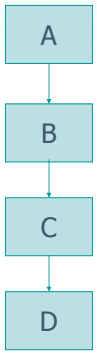
Il controllo di flusso

Concetto di flusso e di scelta

Flusso di esecuzione lineare

- Il programma C viene eseguito, di norma dalla prima istruzione all'ultima
- Il flusso di esecuzione è quindi normalmente di tipo lineare

```
istruzione_A  ;
istruzione_B  ;
istruzione_C  ;
istruzione_D  ;
```



Eccezioni

- In taluni casi il flusso di esecuzione deve variare:
  - in funzione del valore di qualche variabile di ingresso, occorre fare operazioni diverse
  - una certa operazione deve essere ripetuta più volte
  - alcune parti del programma vengono attivate solo se l'utente lo richiede
  - ...
- In tal caso, il flusso di esecuzione non segue più esattamente l'ordine di scrittura delle istruzioni nel codice sorgente

Scelte

- Il tipo più semplice di flusso non lineare è costituito dalle scelte (o alternative)

Se il voto è minore di 18,
allora prenota il prossimo appello,
altrimenti vai in vacanza.
Se il voto è maggiore di 28,
prenota il ristorante.

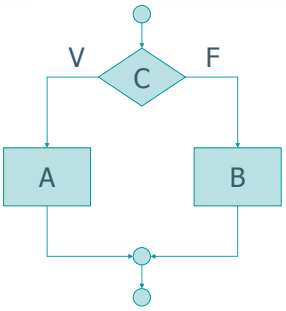
Anatomia di una scelta

- Una condizione di scelta è caratterizzata da tre informazioni:
  - la "condizione" che determina la scelta (il voto è minore di 18)
  - le "operazioni" da svolgere qualora la condizione sia "vera" (prenota il prossimo appello)
  - le "operazioni" da svolgere qualora la condizione sia "falsa" (vai in vacanza)
- Le "operazioni" potrebbero anche essere assenti

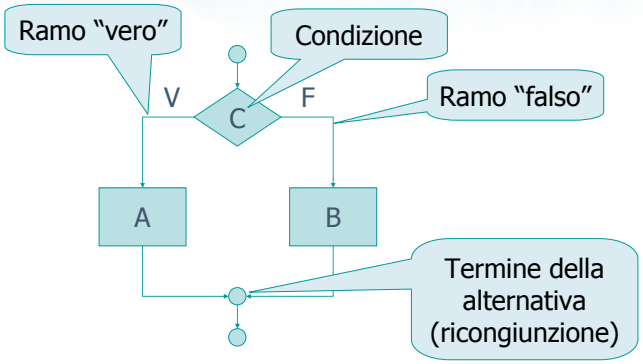
Il controllo di flusso

Rappresentazione grafica

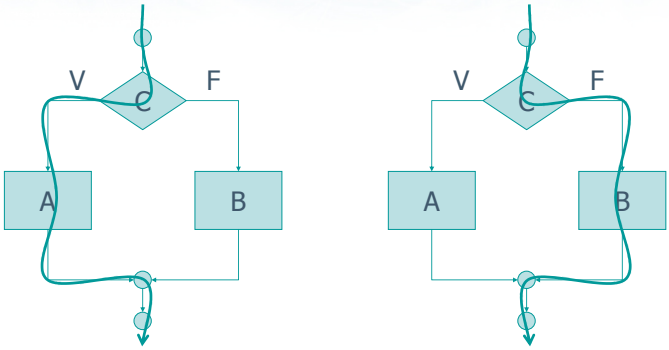
Notazione grafica



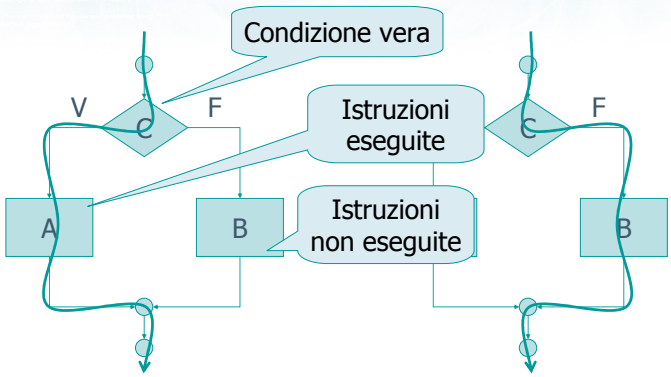
Notazione grafica



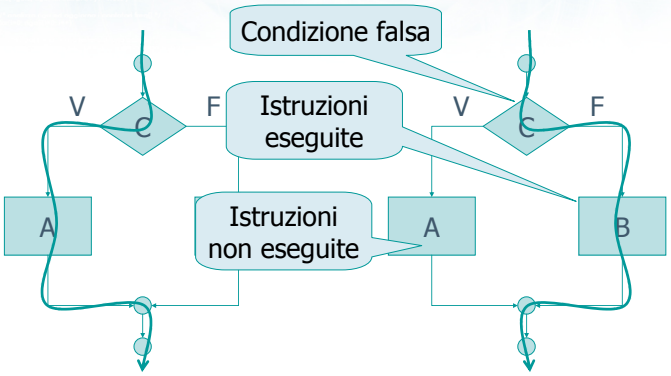
Flussi di esecuzione



Flussi di esecuzione



Flussi di esecuzione



Il controllo di flusso

Condizioni booleane semplici

Il concetto di condizione

- Che tipo di espressioni si utilizzano per esprimere la condizione C ?
  - devono dare una risposta univoca
    - Vero
    - Falso
  - sono basate sui valori delle variabili del programma
- Le espressioni di questo tipo sono dette booleane in quando generano dei valori che rispettano le leggi della logica di Boole (Vero o Falso)

Condizioni di confronto semplici

- Spesso le condizioni sono espresse sotto forma di confronti
- Confronto di uguaglianza
  - Uguale
  - Diverso
- Confronto di ordine
  - Maggiore
  - Minore
  - Maggiore o uguale
  - Minore o uguale

Il controllo di flusso

Esempio

Equazioni di primo grado

- Analizziamo il flusso di esecuzione di un programma in grado di risolvere le equazioni di primo grado:
- Data l'equazione
  - a x + b = 0

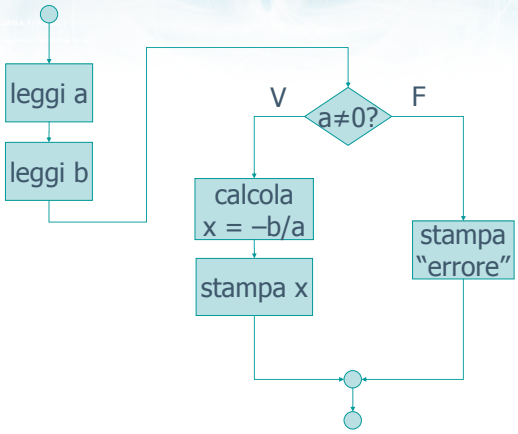
  con a e b inseriti da tastiera, determinare il valore di x che risolve l'equazione

Equazione risolutiva

- Dai corsi di matematica sappiamo che la soluzione dell'equazione:
  - $a\,x + b = 0$

  si può esprimere mediante la formula:
  - $x = -\,b / a$
- Tale formula è valida solo se $a \neq 0$
- Il programma dovrà comportarsi in modo diverso a seconda che a sia nullo o no

Soluzione



Scelte ed alternative

Istruzione `if-else`

Istruzione `if-else`

- Sintassi dell'istruzione
- Operatori di confronto
- Esercizio proposto di esempio
- Risoluzione esercizio (parte I)
- Esecuzione del programma
- Completamento esercizio
- Risoluzione esercizio (parte II)

Istruzione `if-else`

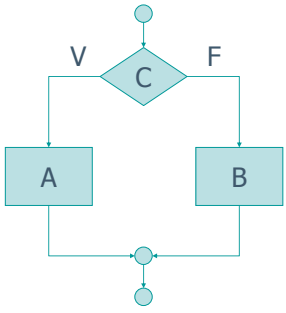
Sintassi dell'istruzione

Istruzioni di scelta in C

- L'operazione di scelta avviene mediante l'istruzione `if-else`
- Le condizioni di scelta possono essere semplici od elaborate
- Le scelte possono essere liberamente combinate tra loro, annidate
- In alcuni casi si può usare l'istruzione `switch` (trattata nella lezione "Istruzione `switch`")

Istruzione `if-else`

```
if ( C )
{
   A ;
}
else
{
   B ;
}
```



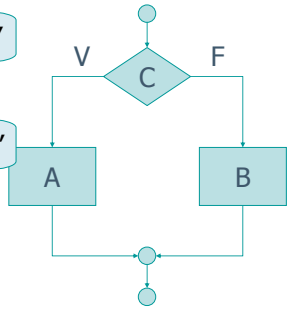
Istruzione `if-else`

```
if ( C )
{
   A ;
}
else
{
   B ;
}
```

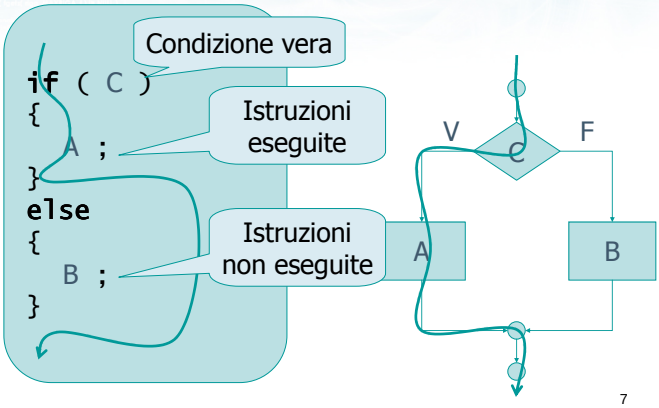
Condizione

Ramo "vero"

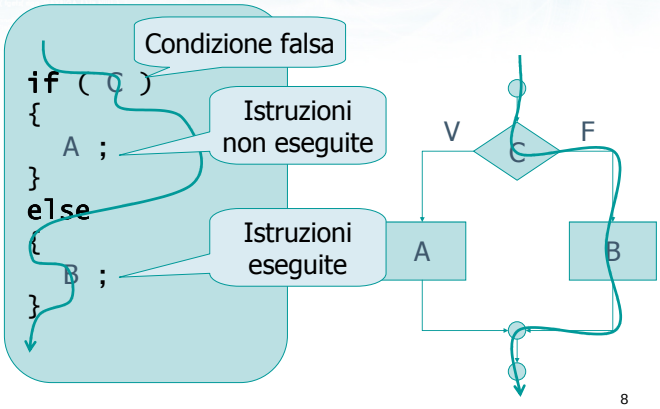
Ramo "falso"



Condizione vera



Condizione falsa



Esempio

es-posneg.c

```
int a, b ;

printf("Immetti un numero: ");
scanf("%d", &a) ;
if ( a > 0 )
{
   printf("positivo\n") ;
   b = a ;
}
else
{
   printf("negativo\n") ;
   b = -a ;
}
printf("Il valore assoluto di %d e' %d", a, b) ;
```


Suggerimento

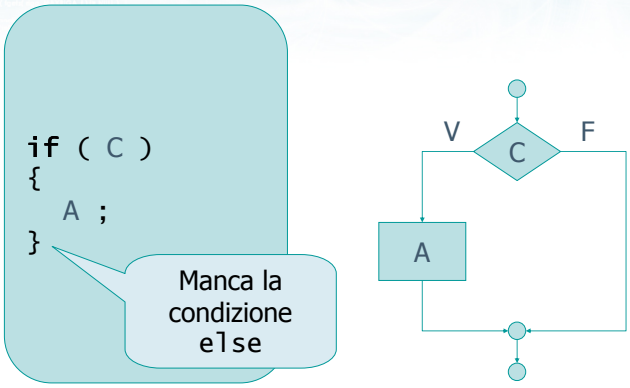
- Ad ogni nuova parentesi graffa aperta {, inserire degli spazi aggiuntivi ad inizio riga
- Tale tecnica, detta indentazione, permette una migliore leggibilità del codice
- È visivamente immediato riconoscere l'inizio e la fine dei blocchi di istruzioni
- Molti ambienti di sviluppo hanno funzioni di indentazione automatica o semi-automatica

Note

```
if ( C )
{
   A ;
}
else
{
   B ;
}
```

- La condizione C può essere semplice o complessa
- Il blocco A può essere composto da una sola istruzione, o da più istruzioni
- Il blocco B può essere composto da una sola istruzione, o da più istruzioni

Caso particolare: istruzione `if`



Esempio

es-valabs.c

```
int a ;

printf("Immetti un numero: ");
scanf("%d", &a) ;
if ( a < 0 )
{
   /* è negativo, gli cambio segno */
   a = -a ;
}
printf("Il valore assoluto e' %d", a) ;
```


Caso particolare: parentesi graffe

```
if ( C )
   A ;
else
{
   B ;
}
```

```
if ( C )
{
   A ;
}
else
   B ;
```

- Se il blocco A è composto da una sola istruzione, allora le parentesi graffe relative si possono omettere.
- Lo stesso vale per il blocco B.

```
if ( C )
   A ;
else
   B ;
```


Esempio

es-valabs2.c

```
int a ;

printf("Immetti un numero: ");
scanf("%d", &a) ;

if ( a < 0 ) /* è negativo, gli cambio segno */
   a = -a ;

printf("Il valore assoluto e' %d", a) ;
```


Errore frequente

- È errato mettere il simbolo di punto-e-virgola ; dopo l'istruzione `if`

```
if ( a > 0 ) ;
   a = -a ;
```

viene interpretato come

```
if ( a > 0 )
   /*nulla*/ ;
a = -a ;
```

forma corretta

```
if ( a > 0 )
   a = -a ;
```


Errore frequente

- È errato fidarsi della sola indentazione

```
if ( a > 0 )
   printf("neg");
   a = -a ;
```

viene interpretato come

```
if ( a > 0 )
   printf("neg");
a = -a ;
```

forma corretta

```
if ( a > 0 )
{
   printf("neg");
   a = -a ;
}
```


Suggerimento

- Anche se vi è una sola istruzione nel blocco "vero" o nel blocco "falso", utilizzare sempre le parentesi graffe
- In tal modo il programma sarà più leggibile, e sarà più facile aggiungere eventuali nuove istruzioni senza incappare in errori

Istruzione `if-else`

Operatori di confronto

Le condizioni

- La condizione C è solitamente basata su un'operazione di confronto
  - determinare se una variabile è uguale o meno a zero, o a un altro valore costante
  - determinare se una variabile è uguale ad un'altra variabile, o è diversa, o è maggiore, o minore, ...
  - determinare se una *espressione*, calcolata a partire da una o più variabili, è uguale, diversa, maggiore, minore, ... di una costante, o una variabile, o un'altra espressione

Operatori di confronto in C

- Uguaglianza
  - Uguale: `a == b`
  - Diverso: `a != b`
- Ordine
  - Maggiore: `a > b`
  - Minore: `a < b`
  - Maggiore o uguale: `a >= b`
  - Minore o uguale: `a <= b`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Esempi

- `if( a == 0 ) ...`
- `if( a == b ) ...`
- `if( a < 0 ) ...`
- `if( a+b > 3 ) ...`
- `if( x*y != y*x ) ...`
- `if( a/2 == (a+1)/2 ) ...`

Errore frequente

- Confondere l'operatore di assegnazione = con l'operatore di confronto ==
- Regola pratica: le parentesi tonde richiedono ==
- Regola pratica: il punto-e-virgola richiede =

```
/* assegnazione */
a = b + c ;

/* confronto */
if ( a == b+c )...
```

```
/* assegnazione */
a == b + c ;

/* confronto */
if ( a = b+c )...
```


Istruzione `if-else`

Esercizio proposto di esempio

Esercizio "Controlla A e B"

- Si scriva un programma in linguaggio C che legga due numeri da tastiera, detti A e B, e determini le seguenti informazioni, stampandole a video:
  - determini se B è un numero positivo o negativo
  - determini se A è un numero pari o dispari
  - calcoli il valore di A+B
  - determini quale scelta dei segni nell'espressione (±A) + (±B) porta al risultato massimo, e quale è questo valore massimo.

Struttura generale

- Leggi A e B
- Controlla il segno di B
  - Stampa il messaggio opportuno
- Controlla la parità di A
  - Stampa il messaggio opportuno
- Calcola A+B
  - Stampa il risultato
- ...l'ultimo punto è più difficile
  - ...ci pensiamo dopo!

Lettura dei dati

- Leggi A e B

```
int a, b ;

printf("Immetti A");
scanf("%d", &a) ;

printf("Immetti B");
scanf("%d", &b) ;
```


Controllo del segno

- Controlla il segno di B
  - Stampa il messaggio opportuno

```
if( b > 0 )
{
   printf("B e' positivo\n");
}
else
{
   printf("B e' negativo o nullo\n");
}
```


Controllo della parità

- Controlla la parità di A
  - Stampa il messaggio opportuno

```
if( "a è pari" )
{
   printf("A e' pari\n");
}
else
{
   printf("A e' dispari\n");
}
```


Numero pari

- Come determinare se un numero è pari?
- Calcoliamo la divisione per 2, e controlliamo il resto
  - Se il resto della divisione per 2 vale 0, allora il numero è pari
  - Se il resto della divisione per 2 vale 1, allora il numero è dispari
- Il calcolo del resto si ottiene con l'operatore %

```
if( "a è pari" )
```

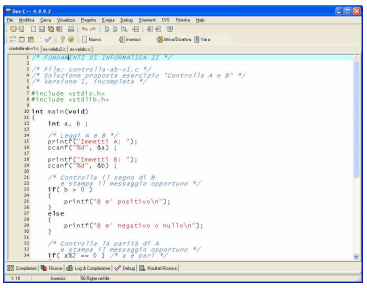
```
if( (a % 2) == 0 )
```


Istruzione if-else

Risoluzione esercizio (parte I)

Risoluzione esercizio "Controlla A e B"

- Risolviamo interattivamente l'esercizio

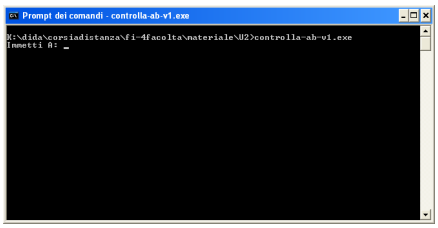


Istruzione if-else

Esecuzione del programma

Verifica esercizio "Controlla A e B"

- Compiliamo il programma
- Eseguiamolo con alcuni dati di prova, verificandone il comportamento corretto



Istruzione if-else

Completamento esercizio

Completamento esercizio "Controlla A e B"

- Abbiamo verificato il corretto funzionamento
- Rimane da implementare il punto:
  - determini quale scelta dei segni nell'espressione (±A) + (±B) porta al risultato massimo, e quale è questo valore massimo.
- Appaiono possibili diverse strategie:
  - Calcolare le 4 combinazioni e scegliere il massimo
  - Riscrivere algebricamente l'espressione

Strategia 1

- La prima strategia prevede di calcolare:
  - R1 = (+A) + (+B)
  - R2 = (+A) + (–B)
  - R3 = (–A) + (+B)
  - R4 = (–A) + (–B)
- Dopo avere calcolato queste 4 variabili, occorre confrontarle per determinare quale è maggiore di tutte le altre.
- In questo caso è inutilmente macchinoso.

Strategia 2

- Ragionando algebricamente, la massima somma che si può ottenere dall'espressione (±A) + (±B) sarà quando
  - ±A è positivo
  - ±B è positivo
- In altre parole, è sufficiente calcolare la somma dei valori assoluti
  - |A| + |B|

Valore assoluto

- Il valore assoluto di una variabile è pari a
  - il valore dell'opposto della variabile
    - se la variabile è negativa
  - il valore della variabile stessa
    - se la variabile è positiva

Valore assoluto

- Il valore assoluto di una variabile è pari a
  - il valore dell'opposto della variabile
    - se la variabile è negativa
  - il valore della variabile stessa
    - se la variabile è positiva

```
if( a<0 )
{
   va = -a ;
}
else
{
   va = a ;
}
```


Valore assoluto

- Il valore assoluto di una variabile è pari a
  - il valore dell'opposto della variabile
    - se la variabile è negativa
  - il valore della variabile stessa
    - se la variabile è positiva

```
if( a<0 )
{
   va = -a ;
}
else
{
   va = a ;
}
```

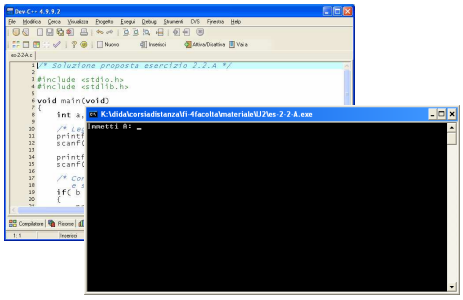
```
if( a<0 )
{
   a = -a ;
}
```


Istruzione `if-else`

Risoluzione esercizio (parte II)

Risoluzione esercizio "Controlla A e B"

- Completiamo l'esercizio, codificandolo e verificandolo



Scelte ed alternative

Condizioni complesse

Condizioni complesse

- Operatori booleani
- Operatori booleani in C
- Esercizio proposto
- Verifica della soluzione

Condizioni complesse

Operatori booleani

Logica Booleana

- Le condizioni "semplici" (es. confronti) forniscono un valore booleano (vero/falso)
- Spesso occorre prendere delle scelte in funzione del valore di più condizioni semplici
  - Es: x è compreso tra a e b?
    - x ≥ a, e contemporaneamente x ≤ b
  - Es: ci sono promossi?
    - voto1 ≥ 18, oppure voto2 ≥ 18
- A questo scopo si possono usare gli operatori booleani

Operatori booleani

- Date due qualsiasi condizioni booleane X ed Y (condizioni semplici, o a loro volta complesse):
- X **AND** Y
  - è vero se sia X che Y sono veri
- X **OR** Y
  - è vero se è vero X (indipendentemente da Y) oppure se è vero Y (indipendentemente da X) o se sono veri entrambi
- **NOT** X
  - è vero se X è falso, è falso se X è vero

Esempi

- x è compreso tra a e b?
  - (x ≥ a) **AND** (x ≤ b)
    - se so già che b ≥ a
  - ((b ≥ a) **AND** (x ≥ a) **AND** (x ≤ b)) **OR** ((b < a) **AND** (x ≤ a) **AND** (x ≥ b))
    - nel caso generale
- ci sono promossi?
  - (voto1 ≥ 18) **OR** (voto2 ≥ 18)

Condizioni complesse

Operatori booleani in C

Operatori booleani in C

Operatore booleano	Sintassi in C	Esempio
AND	&&	(x>=a)&&(x<=b)
OR	||	(v1>=18)||(v2>=18)
NOT	!	!(a>b)

Contesto di utilizzo

- Solitamente gli operatori booleani && || ! si utilizzano all'interno della condizione dell'istruzione `if`, per costruire condizioni complesse
  - Più avanti vedremo come si usano anche nella condizione del costrutto `while`
- Tali operatori lavorano su operandi che solitamente sono:
  - Condizioni semplici (es. `(a>b)||(a!=1)`)
  - Risultati di altri operatori booleani (es. `((a>b)&&(b>c))||(c==0)`)

Precedenza degli operatori

- Quando più operatori booleani e di confronto sono presenti nella stessa espressione, vengono valutati come segue:
  - Prima gli operatori di confronto
    - == != > < >= <=
  - Poi la negazione NOT
    - !
  - In seguito la congiunzione AND
    - &&
  - Infine la disgiunzione OR
    - ||

Esempi

```
if ( a>0 && b>0 )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( !(a>0 && b>0) )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```

```
if ( b>=a && x>=a && x<=b ||
b<a && x<=a && x>=b )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```

```
if ( b>=a && x>=a && x<=b ||
b<a && x<=a && x>=b )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```

```
if ( b>=a && x>=a && x<=b ||
b<a && x<=a && x>=b )
```


Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```

```
if ( b>=a && x>=a && x<=b ||
b<a && x<=a && x>=b )
```

Esempi

```
if ( a>0 && b>0 )
```

```
if ( a<=0 || b<=0 )
```

```
if ( a==0 || (a!=0 && b==0) )
```

```
if ( b>=a && x>=a && x<=b ||
b<a && x<=a && x>=b )
```

```
if ( ((b>=a) && (x>=a) && (x<=b)) ||
     ((b<a)  && (x<=a) && (x>=b))
   )
```


Suggerimento

- In presenza di espressioni complesse, è sempre conveniente abbondare con le parentesi
  - leggibilità
  - indipendenza dalle precedenze degli operatori

~~`if ( b>=a && x>=a && x<=b || b<a && x<=a && x>=b )`~~

```
if ( ((b>=a) && (x>=a) && (x<=b)) ||
     ((b<a)  && (x<=a) && (x>=b))
   )
```


Errore frequente

- È errato usare in successione più operatori di confronto senza collegarli mediante operatori booleani

`if ( a > b > 0 )` → forma corretta → `if ( (a > b) && (b > 0) )`

`if ( a==b==c )` → forma corretta → `if ( (a == b) && (b == c) )`

Errore frequente

- È errato "sottintendere" parte di un confronto
  - Esempio: "se a o b sono diversi da uno"

`if ( a || b != 1 )` → forma corretta

`if ( (a||b) != 1 )` → forma corretta

→ `if ( (a!=1) || (b!=1) )`

Condizioni complesse

Esercizio proposto

Esercizio "Classificazione triangolo 1"

- Si scriva un programma in linguaggio C che legga da tastiera i valori delle lunghezze dei tre lati di un triangolo (detti A, B e C), e determini:
  - se il triangolo è equilatero
  - se il triangolo è isoscele
  - se il triangolo è scaleno
  - se il triangolo è rettangolo
- Nota: si assuma, per il momento, che i valori A, B, C descrivano correttamente un triangolo

Analisi del problema

- Ricordiamo le condizioni matematiche relative alla classificazione dei triangoli:
  - Equilatero: le lunghezze dei tre lati A, B, C sono uguali tra loro
  - Isoscele: le lunghezze di [almeno] due dei tre lati A, B, C sono uguali tra loro
    - ogni triangolo equilatero è anche isoscele
  - Scaleno: le lunghezze dei tre lati A, B, C sono tutte diverse tra loro
  - Rettangolo: possiede un angolo retto
    - vale il teorema di Pitagora

Espressioni matematiche (I)

- Equilatero
  - A = B = C
  - (A = B) AND (B = C) AND (A = C)
  - (A = B) AND (B = C)
- Isoscele
  - (A = B) OR (B = C) OR (A = C)
- Scaleno
  - (A ≠ B) AND (A ≠ C) AND (B ≠ C)

Espressioni matematiche (II)

- Rettangolo
  - Teorema di Pitagora
    - Ipotenusa^2 = Cateto^2 + Cateto^2
  - L'ipotenusa può essere uno qualunque dei lati A, B oppure C
  - ($A^2 = B^2+C^2$) OR ($B^2 = A^2+C^2$) OR ($C^2 = A^2+B^2$)

Condizioni in C

- Equilatero
  - `a==b && b==c`
- Isoscele
  - `a==b || b==c || a==c`
- Scaleno
  - `a!=b && b!=c && a!=c`
- Rettangolo

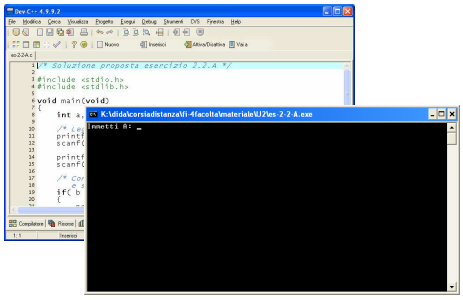
```
(a*a == b*b + c*c) ||
(b*b == a*a + c*c) ||
(c*c == a*a + b*b)
```


Condizioni complesse

Verifica della soluzione

Verifica "Classificazione triangolo 1"

- Analizziamo il codice dell'esercizio e verifichiamone il corretto funzionamento



Scelte ed alternative

Istruzioni `if-else` annidate

Istruzioni `if-else` annidate

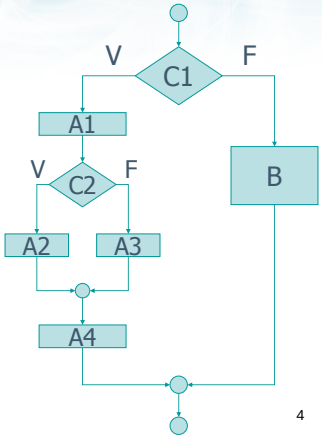
- Annidamento di istruzioni `if-else`
- Opzionalità del ramo `else`
- Catene `if-else if-...-else`
- Esercizio proposto
- Verifica della soluzione

Istruzioni `if-else` annidate

Annidamento di istruzioni `if-else`

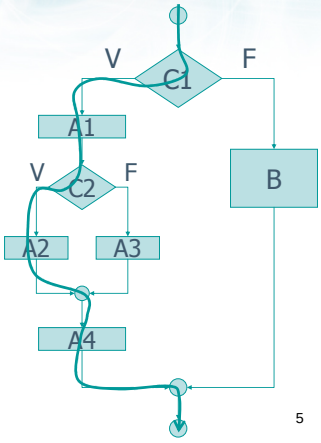
Scelte annidate

- Nelle istruzioni del blocco "vero" o del blocco "else", è possibile inserire altri blocchi di scelta
- In tal caso la seconda scelta risulta annidata all'interno della prima



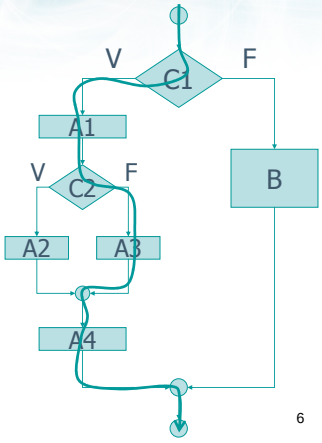
Caso 1

- C1 vero, C2 vero
  - Istruzioni eseguite: A1, A2, A4



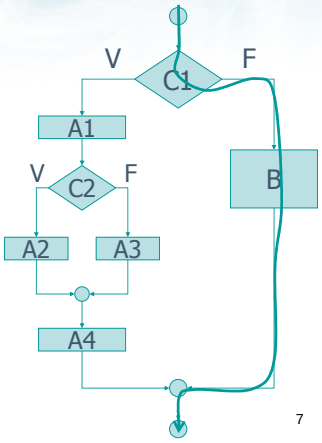
Caso 2

- C1 vero, C2 vero
  - Istruzioni eseguite: A1, A2, A4
- C1 vero, C2 falso
  - Istruzioni eseguite: A1, A3, A4



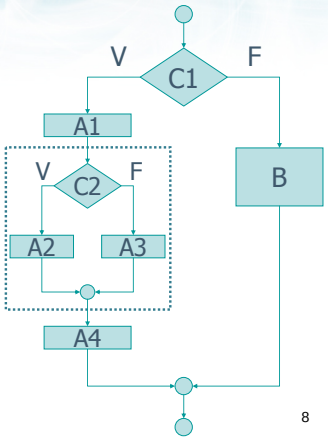
Caso 3

- C1 vero, C2 vero
  - Istruzioni eseguite: A1, A2, A4
- C1 vero, C2 falso
  - Istruzioni eseguite: A1, A3, A4
- C1 falso, C2 indifferente
  - Istruzioni eseguite: B



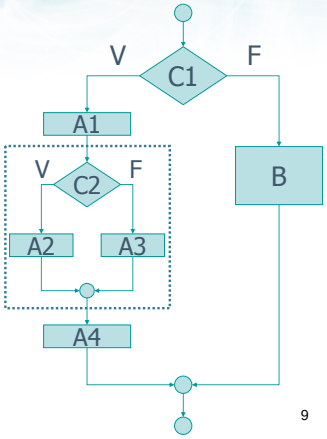
Corretto annidamento

- L'intero blocco di scelta più interno (dalla condizione fino al ricongiungimento) deve essere completamente contenuto all'interno di uno dei rami del blocco più esterno



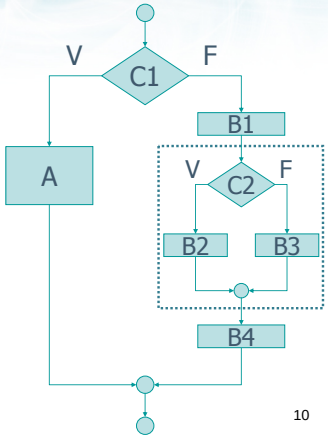
Sintassi C

```
if ( C1 )
{
   A1 ;
   if ( C2 )
   {
      A2 ;
   }
   else
   {
      A3 ;
   }
   A4 ;
}
else
{
   B ;
}
```

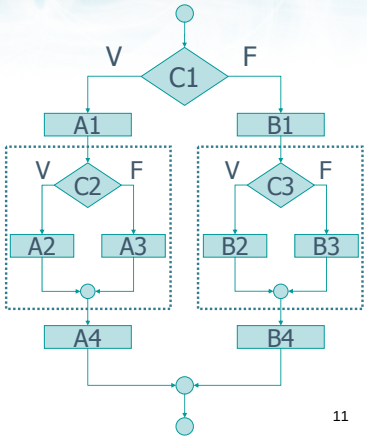


Sintassi C

```
if ( C1 )
{
   A ;
}
else
{
   B1 ;
   if ( C2 )
   {
      B2 ;
   }
   else
   {
      B3 ;
   }
   B4 ;
}
```



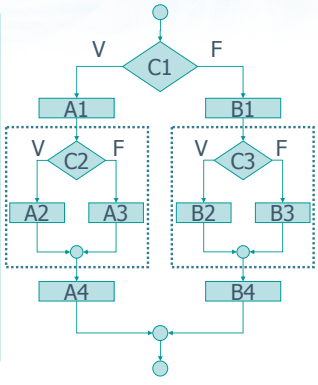
Caso generale



Sintassi C

```
if ( C1 )
{
   A1 ;
   if ( C2 )
   {
      A2 ;
   }
   else
   {
      A3 ;
   }
   A4 ;
}
```

```
else
{
   B1 ;
   if ( C3 )
   {
      B2 ;
   }
   else
   {
      B3 ;
   }
   B4 ;
}
```



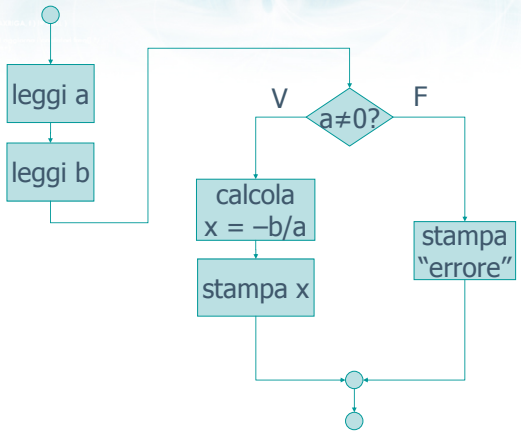
Sintassi C

- Un'istruzione `if` può comparire ovunque
  - anche all'interno del blocco "vero" o "falso" di un'altra istruzione `if`
- Occorre garantire il corretto annidamento delle istruzioni
  - le istruzioni annidate vanno completamente contenute tra le parentesi graffe {...}

Esempio

- Ricordiamo l'esercizio sull'algoritmo risolutivo delle equazioni di primo grado
  - a x + b = 0
- La soluzione è:
  - x = – b / a
    - solo se a≠0

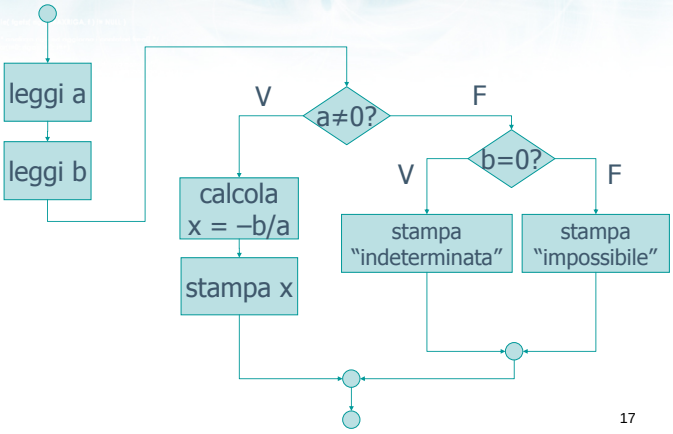
Soluzione (parziale)



Esempio

- Ricordiamo l'esercizio sull'algoritmo risolutivo delle equazioni di primo grado
  - a x + b = 0
- La soluzione è:
  - x = – b / a
    - solo se a≠0
  - x = indeterminato (infinite soluzioni)
    - se a=0 e b=0
  - x = impossibile (nessuna soluzione)
    - se a=0 e b≠0

Soluzione (completa)



Soluzione in C (1/2)

primogrado.c

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    float a, b ;
    float x ;

    printf("Risoluzione eq. di primo grado\n");
    printf("Equazione: a x + b = 0\n") ;

    /* Leggi A e B */
    printf("Immetti coefficiente a: ");
    scanf("%f", &a) ;

    printf("Immetti coefficiente b: ");
    scanf("%f", &b) ;
```

Soluzione in C (2/2)

```
    if( a != 0 )
    {
        x = - b / a ;
        printf("La soluzione e' x = %f\n", x) ;
    }
    else
    {
        if( b==0 )
        {
            printf("Equazione indeterminata\n");
        }
        else
        {
            printf("Equazione impossibile\n");
        }
    }
}
```

Istruzioni `if-else` annidate

Opzionalità del ramo `else`

Forme abbreviate

- Ricordiamo che:
  - Se il ramo "vero" oppure il ramo "falso" è composto da una sola istruzione, allora le parentesi graffe {...} sono opzionali
  - Se il ramo "falso" non contiene istruzioni, allora la clausola `else` si può omettere
- Nel contesto di `if` annidati, queste regole possono creare una potenziale ambiguità

Esempio

```
if( a>0 )
{
  c = a ;
}
else
{
  if( b>0 )
  {
    c = b ;
  }
  else
  {
    c = 0 ;
  }
}
```

=

```
if( a>0 )
  c = a ;
else
  if( b>0 )
    c = b ;
  else
    c = 0 ;
```


Esempio problematico

```
if( a>0 )
if( b>0 )
c = a + b ;
else
c = 0 ;
```

?

```
if( a>0 )
  if( b>0 )
    c = a + b ;
  else
    c = 0 ;
```

```
if( a>0 )
  if( b>0 )
    c = a + b ;
else
  c = 0 ;
```


Regola

- Ogni clausola `else`, in assenza di parentesi graffe che ne esplicitino l'attribuzione, è da intendersi riferita all'istruzione `if` più vicina (per la quale non sia ancora stata attribuita una clausola `else`)

```
if( a>0 )
if( b>0 )
c = a + b ;
else
c = 0 ;
```

```
if( a>0 )
{
  if( b>0 )
    c = a + b ;
  else
    c = 0 ;
}
```

Suggerimento

- In presenza di `if` annidate, è conveniente abbondare con le parentesi graffe

```
if( a>0 )
if( b>0 )
c = a + b ;
else
c = 0 ;
```

```
if( a>0 )
{
  if( b>0 )
    c = a + b ;
}
else
  c = 0 ;
```

```
if( a>0 )
{
  if( b>0 )
    c = a + b ;
  else
    c = 0 ;
}
```

Istruzioni `if-else` annidate

Catene `if-else` `if-...-else`

Catene di istruzioni condizionali

- Talvolta occorre verificare, in sequenza, una serie di condizioni particolari, trovando la prima condizione vera tra quelle possibili
- Esempio:
  - Dato un numero intero tra 1 e 12, che rappresenta il mese corrente, stampare il nome del mese per esteso ("Gennaio" ... "Dicembre")

Soluzione

```
if( mese == 1 )
    printf("Gennaio\n") ;
else
{
    if( mese == 2 )
        printf("Febbraio\n") ;
    else
    {
        if( mese == 3 )
            printf("Marzo\n") ;
        else
        {
            if( mese == 4 )
                printf("Aprile\n") ;
            else
            {
                ..... /* continua fino a 12 */
            }
        }
    }
}
```

mesi.c

Analisi della soluzione

```
if( mese == 1 )
    printf("Gennaio\n") ;
else
{
    if( mese == 2 )
        printf("Febbraio\n") ;
    else
    {
        if( mese == 3 )
            printf("Marzo\n") ;
        else
        {
            if( mese == 4 )
                printf("Aprile\n") ;
            else
            {
                if( mese == 5 )
                    printf("Maggio\n") ;
                else
                {
                    if( mese == 6 )
                        printf("Giugno\n") ;
                    else
                    {
                        if( mese == 7 )
                            printf("Luglio\n") ;
                        else
                        {
                            if( mese == 8 )
                                printf("Agosto\n") ;
                            else
                            {
                                if( mese == 9 )
                                    printf("Settembre\n") ;
                                else
                                {
                                    if( mese == 10 )
                                        printf("Ottobre\n") ;
                                    else
                                    {
                                        if( mese == 11 )
                                            printf("Novembre\n") ;
                                        else
                                        {
                                            if( mese == 12 )
                                                printf("Dicembre\n") ;
                                            else
                                                printf("MESE ERRATO!\n") ;
                                        }
                                    }
                                }
                            }
                        }
                    }
                }
            }
        }
    }
}
```

- Annidamento eccessivo
- Scarsa leggibilità
- Difficile identificazione delle {...} corrispondenti
- Esistono formattazioni migliori?

Soluzione più leggibile

```
if( mese == 1 )
    printf("Gennaio\n") ;
else if( mese == 2 )
    printf("Febbraio\n") ;
else if( mese == 3 )
    printf("Marzo\n") ;
else if( mese == 4 )
    printf("Aprile\n") ;
else if( mese == 5 )
    printf("Maggio\n") ;
.............................
else if( mese == 9 )
    printf("Settembre\n") ;
else if( mese == 10 )
    printf("Ottobre\n") ;
else if( mese == 11 )
    printf("Novembre\n") ;
else if( mese == 12 )
    printf("Dicembre\n") ;
else
    printf("MESE ERRATO!\n") ;
```

mesi2.c

In generale...

- In ogni ramo "falso" c'è una sola istruzione: la `if-else` annidata
  - anche se a sua volta contiene altre istruzioni, è comunque considerata una sola istruzione
- È possibile omettere le parentesi nel ramo `else`
- È conveniente rimuovere l'indentazione
- Ricordarsi dell'`else` finale

```
if ( C1 )
{
    A1 ;
}
else if ( C2 )
{
    A2 ;
}
else if ( C3 )
{
    A3 ;
}
.........
else
{
    An ;
}
```


Istruzioni `if-else` annidate

Esercizio proposto

Esercizio "Classificazione triangolo 2"

- Si scriva un programma in linguaggio C che legga da tastiera i valori delle lunghezze dei tre lati di un triangolo (detti A, B e C), e determini:
  - se il triangolo è equilatero
  - se il triangolo è isoscele
  - se il triangolo è scaleno
  - se il triangolo è rettangolo
- Il programma, prima di classificare il triangolo, controlli se i numeri A, B, C rappresentano correttamente un triangolo

Analisi

- Data una terna di numeri A, B, C, non è detto che si possa costruire un triangolo con tali lunghezze dei lati
  - La lunghezza di ciascun lato deve essere un numero positivo
  - Ogni lato deve essere minore della somma degli altri due
  - Ogni lato deve essere maggiore della differenza degli altri due

Struttura proposta

```
if (i lati non sono positivi)
{
    printf("errore") ;
}
else if (ogni lato non è minore della somma degli altri)
{
    printf("errore") ;
}
else if (ogni lato non è maggiore della differenza degli altri)
{
    printf("errore") ;
}
else
{
    /* caso normale */
    ...vedi triangolo.c
}
```

Condizioni booleane

- "i lati non sono positivi"
  - `a<=0 || b<=0 || c<=0`
- "ogni lato non è minore della somma degli altri"
  - `a>=b+c || b>=a+c || c>=a+b`
- "ogni lato non è maggiore della differenza degli altri"
  - attenzione: la differenza va presa in valore assoluto!
  - prima di calcolare A-B, occorre verificare che A>B, altrimenti bisogna calcolare B-A

Condizioni booleane (2)

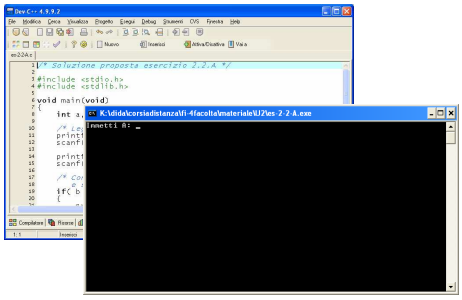
- "ogni lato non è maggiore della differenza degli altri"
  - `( (b>c)  && a <= b-c ) ||`
  - `( (b<=c) && a <= c-b ) ||`
  - `( (a>c)  && b <= a-c ) ||`
  - `( (a<=c) && b <= c-a ) ||`
  - `( (a>b)  && c <= b-a ) ||`
  - `( (a<=b) && c <= a-b )`

Istruzioni `if-else` annidate

Verifica della soluzione

Verifica "Classificazione triangolo 2"

- Analizziamo il codice dell'esercizio e verifichiamone il corretto funzionamento



Scelte ed alternative

Istruzione switch

Istruzione switch

- Sintassi dell'istruzione
- Particolarità dell'istruzione
- Esercizio proposto
- Verifica della soluzione

Istruzione switch

Sintassi dell'istruzione

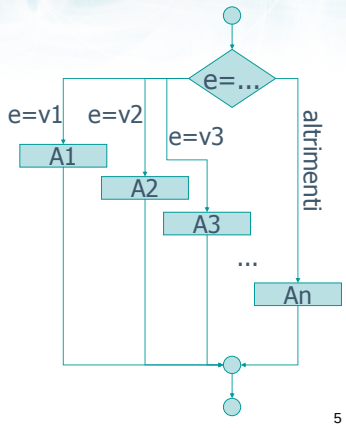
Scelte multiple

- Quando occorre compiere una sequenza di scelte, in funzione del valore di una variabile, occorre una catena di `if-else`
- Lo stesso risultato si può ottenere in forma più compatta mediante l'istruzione `switch`

```
if( mese == 1 )
    printf("Gennaio\n") ;
else if( mese == 2 )
    printf("Febbraio\n") ;
else if( mese == 3 )
    printf("Marzo\n") ;
else if( mese == 4 )
    printf("Aprile\n") ;
else if( mese == 5 )
    printf("Maggio\n") ;
.........................
else if( mese == 9 )
    printf("Settembre\n") ;
else if( mese == 10 )
    printf("Ottobre\n") ;
else if( mese == 11 )
    printf("Novembre\n") ;
else if( mese == 12 )
    printf("Dicembre\n") ;
else
    printf("MESE ERRATO!\n") ;
```

Sintassi istruzione switch

```
switch ( e )
{
    case v1:
        A1 ;
    break ;

    case v2:
        A2 ;
    break ;

    case v3:
        A3 ;
    break ;
    ..........
    default:
        An ;
}
```



Precisazioni (1/2)

- L'espressione e può essere una variabile oppure un'espressione aritmetica
  - Il tipo di dato deve essere `int`, `char` o `enum`
- Ciascun caso è identificato da una costante
  - L'espressione e viene confrontata con il valore delle costanti v1...vn
  - Il tipo di dato deve essere compatibile
- Ciascun caso è delimitato da `case`...`break`
  - Non vi sono parentesi graffe {...}

Precisazioni (2/2)

- I casi possono apparire in qualsiasi ordine
  - Devono essere tutti diversi
- Verrà selezionato al più un caso
- Il caso `default` viene valutato se e solo se nessuno degli altri casi è stato considerato
  - Opzionale, ma sempre consigliato

L'istruzione `break`

- Il significato di `break` è di portare l'esecuzione del programma fino al termine del costrutto `switch`
  - "Salta alla chiusa graffa": }
- In assenza di `break`, l'esecuzione proseguirebbe attraverso il caso successivo
  - Né il prossimo `case`, né eventuali parentesi graffe, possono fermare l'esecuzione lineare

Casi multipli

- Potrebbe essere necessario eseguire lo stesso codice in corrispondenza di diversi valori dell'espressione
- È possibile accomunare più casi, indicandoli consecutivamente

```
switch( ora )
{
  case 12:
    pranzo = 1 ;
    break;
  case 13:
    pranzo = 1 ;
    break;
}
```

→

```
switch( ora )
{
  case 12:
  case 13:
    pranzo = 1 ;
    break;
}
```


Esempio

```
switch( mese )
{
    case 1:
        printf("Gennaio\n") ;
        break ;
    case 2:
        printf("Febbraio\n") ;
        break ;
    case 3:
        printf("Marzo\n") ;
        break ;
    case 4:
        printf("Aprile\n") ;
        break ;
    ..........................
    case 12:
        printf("Dicembre\n") ;
        break ;
    default:
        printf("MESE ERRATO!\n") ;
}
```

mesi3.c

Istruzione `switch`

Particolarità dell'istruzione

Istruzione atipica

- L'istruzione `switch` è anomala sotto diversi punti di vista:
  - Non utilizza le parentesi graffe per l'annidamento delle istruzioni interne
  - Prevede solamente il controllo di uguaglianza `e==v`
  - Richiede che i valori da confrontare siano costanti
  - `break` e `default` sono opzionali
- Occorre una forte disciplina nell'utilizzarla correttamente!

Errore frequente

- È errato dimenticare l'istruzione `break` al termine di ogni caso

```
switch( ora )
{
  case 12:
    pranzo = 1 ;
  case 20:
    cena = 1 ;
    break;
}
```

viene interpretato come

```
switch( ora )
{
  case 12:
    pranzo = 1 ;
    cena = 1 ;
    break;
  case 20:
    cena = 1 ;
    break;
}
```

forma corretta

```
switch( ora )
{
  case 12:
    pranzo = 1 ;
    break;
  case 20:
    cena = 1 ;
    break;
}
```

Errore frequente

- È errato utilizzare variabili come valori dei singoli `case`
- Se non è possibile usare costanti, allora utilizzare delle catene di `if-else`

forma corretta

```
switch( ora )
{
  case orapranzo:
    pranzo = 1 ;
    break;
  case oracena:
    cena = 1 ;
    break;
}
```

```
if( ora==orapranzo )
{
  pranzo = 1 ;
}
else if( ora==oracena )
{
  cena = 1 ;
}
```

Errore frequente

- È errato pensare di poter fare confronti di ordine, o confronti multipli
- Utilizzare sequenze di `if-else`
- Non usare `else if` se i casi non sono mutuamente esclusivi

forma corretta

```
switch( ora )
{
  case <12:
    mattino = 1 ;
    break;
  case 12 || 20:
    pasti = 1 ;
    break;
}
```

```
if( ora<12 )
{
  mattino = 1 ;
}
if( ora==12 || 20 )
{
  pasti = 1 ;
}
```

Suggerimento

- Utilizzare sempre l'istruzione `default`
- Anche se non vi è ragione apparente, è opportuno che il programma intercetti valori errati della variabile

```
switch( ora )
{
  ......

  default:
    printf("Errore: valore inatteso
            %d per la variabile ora\n", ora) ;
}
```


Suggerimento

- Posizionare l'istruzione `default` come ultimo caso dello `switch`
  - Potrebbe stare ovunque
  - Maggiore leggibilità
- L'istruzione `break` finale si può omettere

Istruzione `switch`

Esercizio proposto

Esercizio "Semplice calcolatrice"

- Si scriva un programma in linguaggio C che implementi una semplice calcolatrice in grado di compiere le 4 operazioni (+ − × ÷) tra numeri interi
- Il programma presenti un semplice menù da cui l'utente indichi (con un numero tra 1 e 4) l'operazione da svolgere
- In seguito il programma acquisirà da tastiera i due operandi e stamperà il risultato dell'operazione

Soluzione (1/4)

calcola.c

```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int op ;
    int a, b, c ;
    int err ;

    printf("Semplice calcolatrice\n\n") ;

    printf("Inserisci 1 per la somma\n");
    printf("Inserisci 2 per la sottrazione\n");
    printf("Inserisci 3 per la moltiplicazione\n");
    printf("Inserisci 4 per la divisione\n");

    printf("La tua scelta:") ;
    scanf("%d", &op) ;
```

Soluzione (2/4)

```
printf("Inserisci il primo operando: ") ;
scanf("%d", &a) ;

printf("Inserisci il secondo operando: ") ;
scanf("%d", &b) ;
```


Soluzione (3/4)

```
    err = 0 ;
    switch( op )
    {
        case 1:
            c = a + b ;
            break ;
        case 2:
            c = a - b ;
            break ;
        case 3:
            c = a * b ;
            break ;
        case 4:
            c = a / b ;
            break ;
        default:
            printf("Operazione errata\n") ;
            err = 1 ;
    }
```

Soluzione (3/4)

```
err = 0 ;
switch( op
{
    case 1
        c
        br
    case 2
        c
        br
    case 3
        c
        br
    case 4:
        c = a / b ;
        break ;
    default:
        printf("Operazione errata\n") ;
        err = 1 ;
}
```

```
case 4:
    if( b == 0 )
    {
        printf("Divisione per zero!\n");
        err = 1 ;
    }
    else
    {
        c = a / b ;
    }
    break ;
```

Soluzione (4/4)

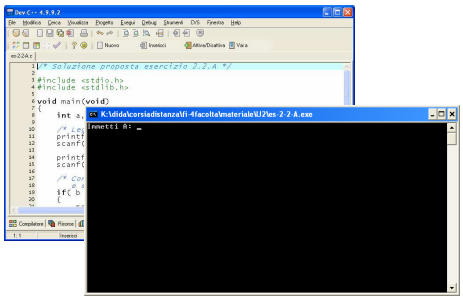
```
    if( err == 0 )
    {
        printf("Il risultato vale: %d\n", c) ;
    }

}
```

Istruzione switch

Verifica della soluzione

Verifica "Semplice calcolatrice"

- Analizziamo il codice dell'esercizio e verifichiamone il corretto funzionamento



Scelte ed alternative

Esercizi proposti

Esercizi proposti

- Esercizi sul calcolo del massimo
- Esercizio "Equazione di secondo grado"
- Esercizio "Re e Regina"

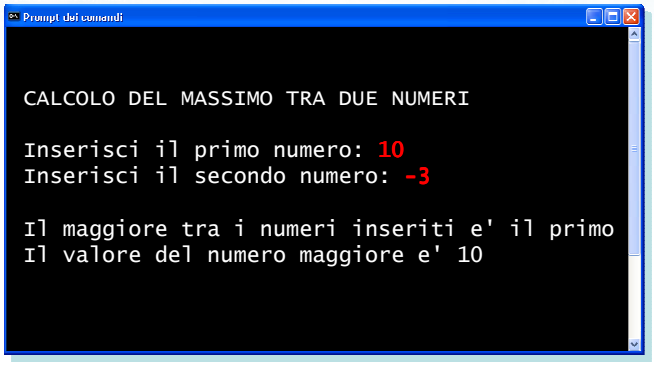
Esercizi proposti

Esercizi sul calcolo del massimo

Esercizio "Calcolo del massimo"

- Si scriva un programma in linguaggio C che acquisisca due numeri interi da tastiera e:
  - determini, stampando un messaggio opportuno quale dei due numeri (il primo o il secondo) sia maggiore
  - stampi il valore di tale numero
- Si trascuri il caso in cui i due numeri siano uguali

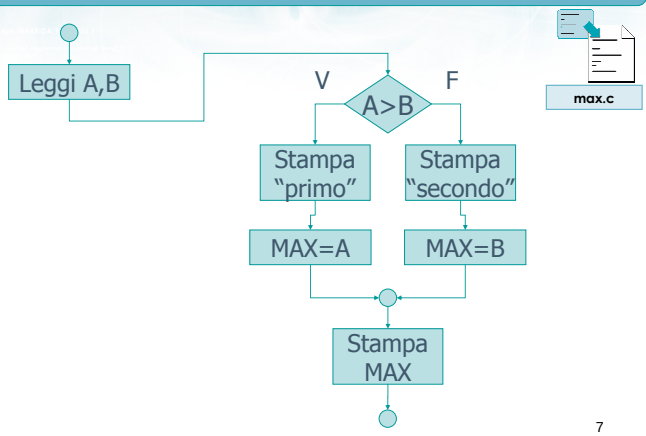
Esempio



Analisi

- Chiamiamo A e B i due numeri introdotti dall'utente
- Chiamiamo MAX il valore del maggiore tra i due
- Occorre verificare la condizione A>B
  - Se A>B, allora MAX sarà pari ad A
  - Altrimenti, MAX sarà pari a B

Diagramma di flusso



max.c

Esercizio "Calcolo del massimo a 3"

- Si scriva un programma in linguaggio C che acquisisca **tre** numeri interi da tastiera e:
  - determini, stampando un messaggio opportuno quale dei **tre** numeri (il primo, il secondo o il terzo) sia maggiore
  - stampi il valore di tale numero
- Si trascuri il caso in cui i numeri siano uguali

Esempio

```
Prompt dei comandi

CALCOLO DEL MASSIMO TRA TRE NUMERI

Inserisci il primo numero: 10
Inserisci il secondo numero: -3
Inserisci il terzo numero: 15

Il maggiore tra i numeri inseriti e' il terzo
Il valore del numero maggiore e' 15
```


Analisi

- Chiamiamo A, B, C i tre numeri inseriti
- Si può procedere in due modi
  - Usando istruzioni `if` annidate
    - se A>B, allora controlla se A>C ...

max3-if.c

Analisi

- Chiamiamo A, B, C i tre numeri inseriti
- Si può procedere in due modi
  - Usando istruzioni `if` annidate
    - se A>B, allora controlla se A>C ...

max3-if.c

  - Usando espressioni condizionali complesse
    - se A>B e A>C ...

max3-and.c

Esercizi proposti

Esercizio "Equazione di secondo grado"

Esercizio "Equazione di secondo grado"

- Data l'equazione
  - $a\,x^2 + b\,x + c = 0$

  con a, b e c inseriti da tastiera, determinare il valore (o i valori) di x che risolvono l'equazione

Analisi

- Ricordiamo la formula risolutiva per le equazioni di secondo grado
- La quantità sotto radice è il discriminante Δ
- Vi sono vari casi possibili
  - se a≠0 o a=0
  - se Δ>0, Δ=0 o Δ<0

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$\Delta = b^2 - 4ac$$

Casi possibili

Caso	Situazione	Soluzione/i
$a = 0$	Equazione di primo grado	$x = -c/b$ Impossibile se $b = 0, c \neq 0$ Indeterminata se $b = 0, c = 0$
$a \neq 0$ $\Delta > 0$	Due soluzioni reali distinte	$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$
$a \neq 0$ $\Delta = 0$	Due soluzioni reali coincidenti	$x_1 = x_2 = \frac{-b}{2a}$
$a \neq 0$ $\Delta < 0$	Due soluzioni complesse coniugate	$x_{1,2} = R \pm iC$ $R = -b/2a, \quad C = \sqrt{\lvert b^2 - 4ac \rvert}/2a$

Soluzione

- Acquisire a, b, c
- Se a=0, risolvere l'equazione di primo grado
  - Ri-usiare il codice già scritto, facendo attenzione al nome delle variabili
- Calcolare il discriminante Δ
  - Il nome della variabile sarà delta
- In funzione del segno di delta, usare la formula opportuna

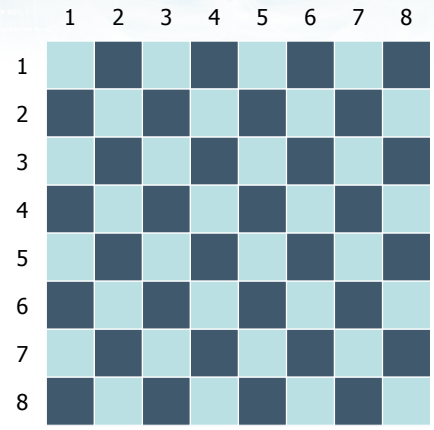
Esercizi proposti

Esercizio "Re e Regina"

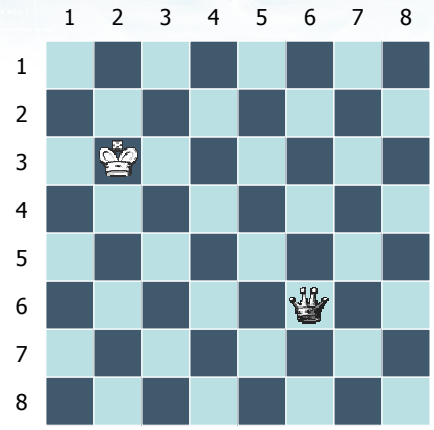
Esercizio "Re e Regina"

- Su una scacchiera 8x8 sono posizionati due pezzi: il Re bianco e la Regina nera
- Si scriva un programma in linguaggio C che, acquisite le posizioni del Re e della Regina, determini se la Regina è in posizione tale da poter mangiare il Re
  - Le posizioni dei due pezzi sono identificate mediante la riga e la colonna su cui si trovano, espresse come numeri interi tra 1 e 8

Analisi



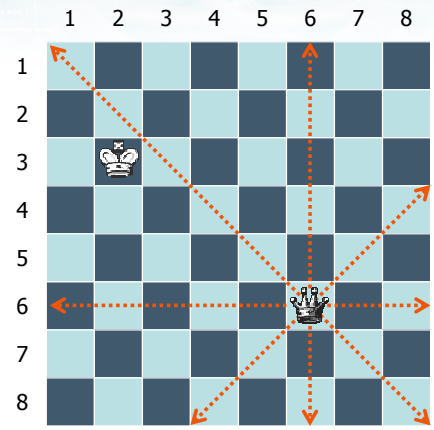
Analisi



Re bianco:
riga 3
colonna 2

Regina nera:
riga 6
colonna 6

Analisi

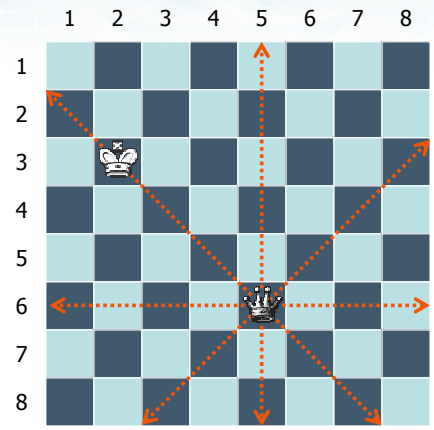


Re bianco:
riga 3
colonna 2

Regina nera:
riga 6
colonna 6

Il Re è salvo

Analisi



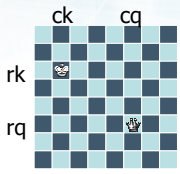
Re bianco:
riga 3
colonna 2

Regina nera:
riga 6
colonna 5

Il Re è
sotto scacco

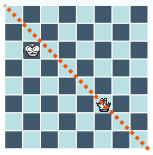
Soluzione (1/2)

- Acquisire le coordinate dei pezzi
  - Re: `rk`, `ck`
  - Regina: `rq`, `cq`
- Controllare se la Regina
  - è sulla stessa riga del Re
    - `rq==rk`
  - è sulla stessa colonna del Re
    - `cq==ck`
  - è sulla stessa diagonale discendente del Re
  - è sulla stessa diagonale ascendente del Re

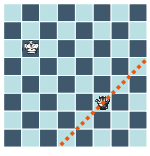


Soluzione (1/2) - Diagonali

- Diagonale discendente
  - r–c costante
  - `rq-cq == rk-ck`



- Diagonale ascendente
  - r+c costante
  - `rq+cq == rk+ck`



Soluzione (2/2)

- Conviene utilizzare una variabile logica `scacco`
  - inizializzare `scacco=0`
  - fare i vari tipi di controlli
  - se si verifica una condizione di scacco, porre `scacco=1`
- Al termine dei controlli, in funzione del valore di `scacco`, stampare il messaggio opportuno
  - se `scacco==0`, il Re è salvo
  - se `scacco==1`, il Re è sotto scacco

Scelte ed alternative

Sommario

Argomenti trattati

- Ramificazione del flusso di esecuzione
- Istruzione `if-else`
- Condizioni Booleane semplici e complesse
- Annidamento di istruzioni `if-else`
- Istruzione `switch`

Tecniche di programmazione

- Catene di istruzioni `if-else if-...-else`
- Annidamento delle istruzioni o condizioni Booleane complesse
- Uso di variabili logiche per tenere traccia delle condizioni incontrate
- Istruzione `switch` per sostituire alcuni tipi di catene `if-else if`
- Uso di `else` e `default` per catturare condizioni anomale

Suggerimenti

- Analizzare sempre tutti i casi possibili prima di iniziare a scrivere il programma
- Abbondare con le parentesi graffe
- Curare l'indentazione
- Aggiungere commenti in corrispondenza della clausola `else` e della graffa di chiusura

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Cicli ed iterazioni

Cicli ed iterazioni

- La ripetizione
- Istruzione `while`
- Schemi ricorrenti nei cicli
- Istruzione `for`
- Approfondimenti
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitolo 3
  - Cabodi, Quer, Sonza Reorda: capitolo 4
  - Dietel & Dietel: capitolo 4
- Dispense
  - Scheda: "Cicli ed iterazioni in C"

Cicli ed iterazioni

La ripetizione

La ripetizione

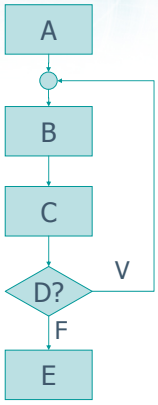
- Concetto di ciclo
- Struttura di un ciclo
- Numero di iterazioni note
- Numero di iterazioni ignote

La ripetizione

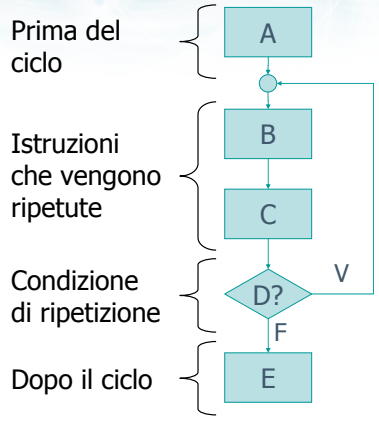
Concetto di ciclo

Flusso di esecuzione ciclico

- È spesso utile poter **ripetere** alcune parti del programma più volte
- Nel diagramma di flusso, corrisponde a "tornare indietro" ad un blocco precedente
- Solitamente la ripetizione è controllata da una condizione booleana



Flusso di esecuzione ciclico



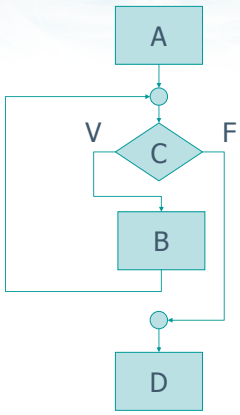
Errore frequente

- Ogni ciclo porta in sé il rischio di un grave errore di programmazione: il fatto che il ciclo venga ripetuto indefinitamente, senza mai uscire
- Il programmatore deve garantire che ogni ciclo, dopo un certo numero di iterazioni, venga terminato
  - La condizione booleana di controllo dell'iterazione **deve** divenire falsa

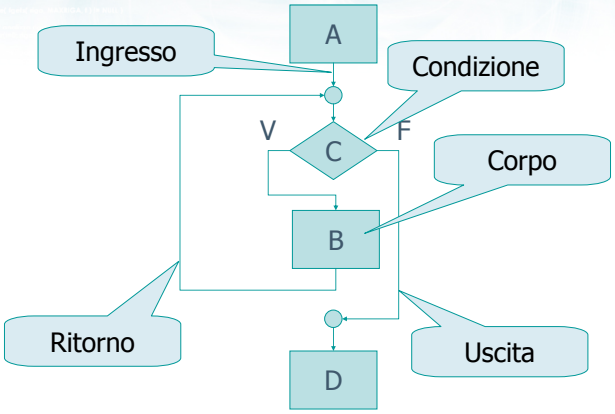
Istruzioni eseguibili ed eseguite

- Istruzioni eseguibili
  - Le istruzioni che fanno parte del programma
  - Corrispondono alle istruzioni del sorgente C
- Istruzioni eseguite
  - Le istruzioni effettivamente eseguite durante una specifica esecuzione del programma
  - Dipendono dai dati inseriti
- Nel caso di scelte, alcune istruzioni eseguibili non verranno eseguite
- Nel caso di cicli, alcune istruzioni eseguibili verranno eseguite varie volte

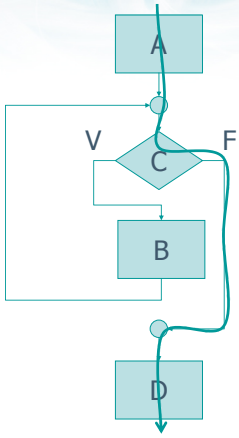
Notazione grafica (while)



Notazione grafica (while)



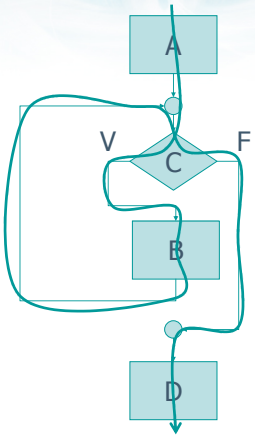
Flussi di esecuzione



Zero iterazioni

A
C → Falso
D

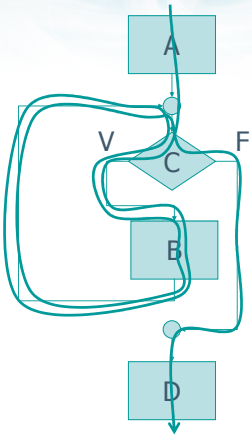
Flussi di esecuzione



Una iterazione

A
C → Vero
B
C → Falso
D

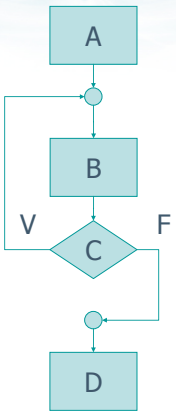
Flussi di esecuzione



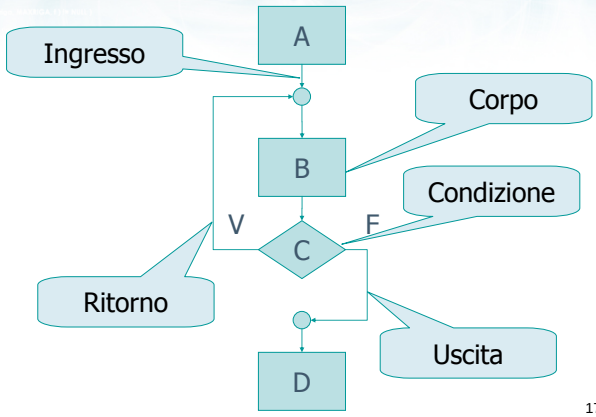
Due iterazioni

A
C → Vero
B
C → Vero
B
C → Falso
D

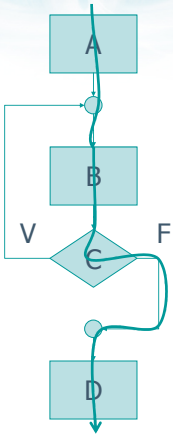
Notazione grafica (do-while)



Notazione grafica (do-while)



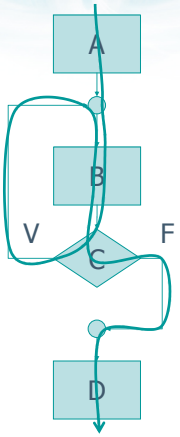
Flussi di esecuzione



Una iterazione

A
B
C → Falso
D

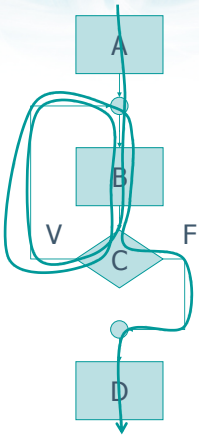
Flussi di esecuzione



Due iterazioni

A
B
C → Vero
B
C → Falso
D

Flussi di esecuzione



Tre iterazioni

A
B
C → Vero
B
C → Vero
B
C → Falso
D

La ripetizione

Struttura di un ciclo

Problemi

- Nello strutturare un ciclo occorre garantire:
  - Che il ciclo possa terminare
  - Che il numero di iterazioni sia quello desiderato
- Il corpo centrale del ciclo può venire eseguito più volte:
  - La prima volta lavorerà con variabili che sono state inizializzate al di fuori del ciclo
  - Le volte successive lavorerà con variabili che possono essere state modificare nell'iterazione precedente
  - Garantire la correttezza sia della prima, che delle altre iterazioni

Anatomia di un ciclo (1/5)

- Conviene concepire il ciclo come 4 fasi
  - Inizializzazione
  - Condizione di ripetizione
  - Corpo
  - Aggiornamento

Anatomia di un ciclo (2/5)

- Conviene concepire il ciclo come 4 fasi
  - Inizializzazione
    - Assegnazione del valore iniziale a tutte le variabili che vengono lette durante il ciclo (nel corpo o nella condizione)
  - Condizione di ripetizione
  - Corpo
  - Aggiornamento

Anatomia di un ciclo (3/5)

- Conviene concepire il ciclo come 4 fasi
  - Inizializzazione
  - Condizione di ripetizione
    - Condizione, di solito inizialmente vera, che al termine del ciclo diventerà falsa
    - Deve dipendere da variabili che saranno modificate all'interno del ciclo (nel corpo o nell'aggiornamento)
  - Corpo
  - Aggiornamento

Anatomia di un ciclo (4/5)

- Conviene concepire il ciclo come 4 fasi
  - Inizializzazione
  - Condizione di ripetizione
  - Corpo
    - Le istruzioni che effettivamente occorre ripetere
    - Sono lo scopo per cui il ciclo viene realizzato
    - Posso usare le variabili inizializzate
    - Posso modificare le variabili
  - Aggiornamento

Anatomia di un ciclo (5/5)

- Conviene concepire il ciclo come 4 fasi
  - Inizializzazione
  - Condizione di ripetizione
  - Corpo
  - Aggiornamento
    - Modifica di una o più variabili in grado di aggiornare il valore della condizione di ripetizione
    - Tengono "traccia" del progresso dell'iterazione

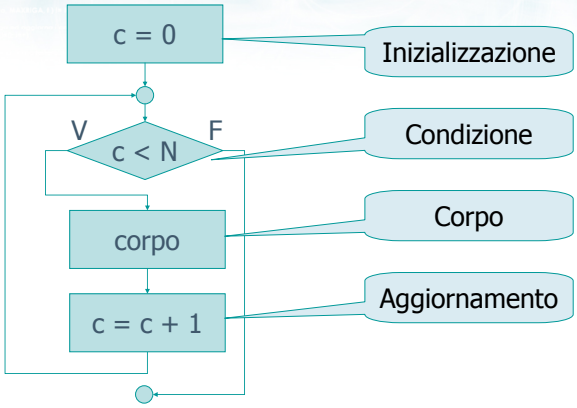
La ripetizione

Numero di iterazioni note

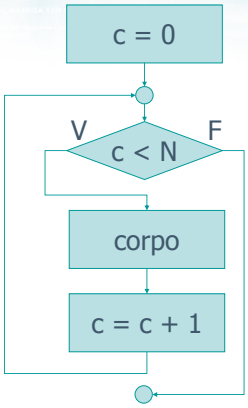
Tipologie di cicli

- Cicli in cui il numero di iterazioni sia noto a priori, ossia prima di entrare nel ciclo stesso
  - Solitamente si usa una variabile "contatore"
  - L'aggiornamento consiste in un incrememto o decremento della variabile
- Cicli in cui il numero di iterazioni non sia noto a priori, ma dipenda dai dati elaborati nel ciclo
  - Solitamente si una una condizione dipendente da una variabile letta da tastiera oppure calcolata nel corpo del ciclo
  - Difficile distinguere il corpo dall'aggiornamento
  - Problema di inizializzazione

Cicli con iterazioni note

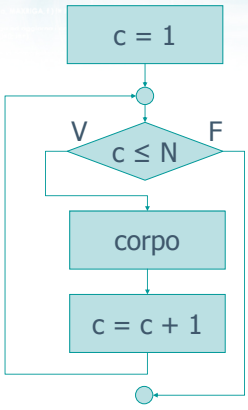


Cicli con iterazioni note



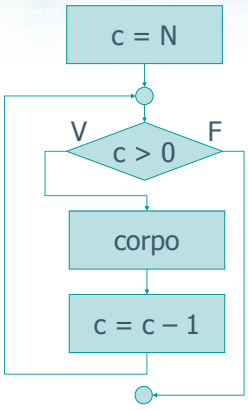
- Forma 0...N-1
- Prima iterazione:
  - c=0
- Ultima iterazione:
  - c=N-1
- Corpo ripetuto:
  - N volte
- Al termine del ciclo
  - c=N
  - condizione falsa

Cicli con iterazioni note



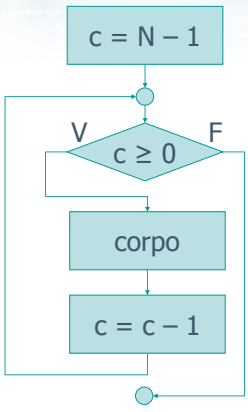
- Forma 1...N
- Prima iterazione:
  - c=1
- Ultima iterazione:
  - c=N
- Corpo ripetuto:
  - N volte
- Al termine del ciclo
  - c=N+1
  - condizione falsa

Cicli con iterazioni note



- Forma N...1
- Prima iterazione:
  - c=N
- Ultima iterazione:
  - c=1
- Corpo ripetuto:
  - N volte
- Al termine del ciclo
  - c=0
  - condizione falsa

Cicli con iterazioni note

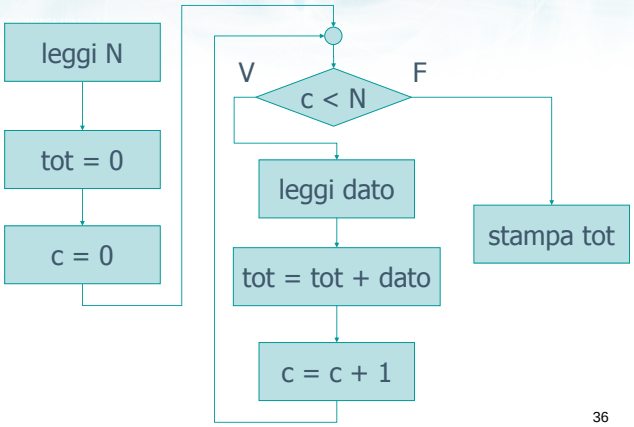


- Forma N-1...0
- Prima iterazione:
  - c=N-1
- Ultima iterazione:
  - c=0
- Corpo ripetuto:
  - N volte
- Al termine del ciclo
  - c=-1
  - condizione falsa

Esempio

- Acquisire da tastiera una sequenza di numeri interi e stamparne la somma.
- Il programma
  - inizialmente chiede all'utente quanti numeri intende inserire
  - in seguito richiede uno ad uno i dati
  - infine stampa la somma

Soluzione



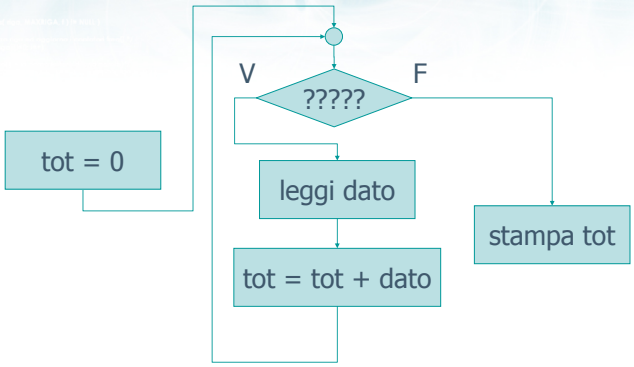
La ripetizione

Numero di iterazioni ignote

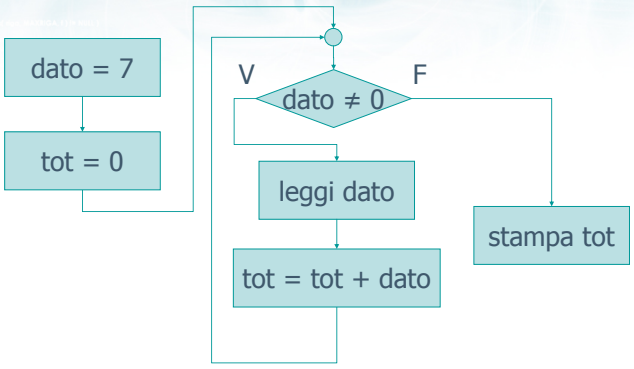
Cicli con iterazioni ignote

- Non esiste uno schema generale
- Esempio:
  - Acquisire da tastiera una sequenza di numeri interi e stamparne la somma.
  - Il termine della sequenza viene indicato inserendo un dato pari a zero.

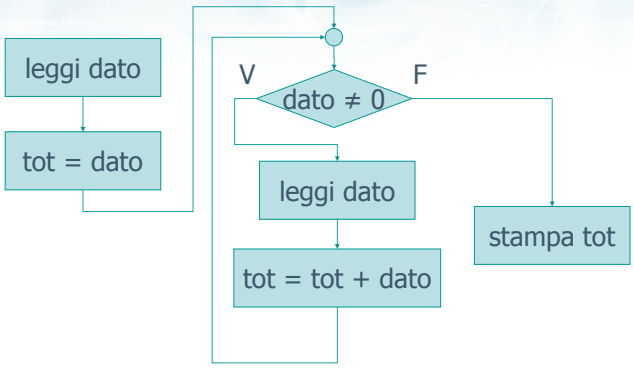
Soluzione parziale



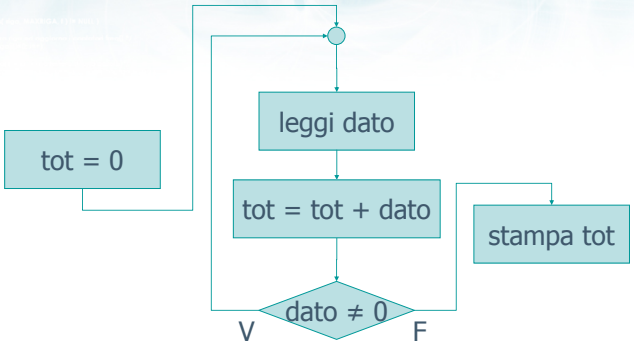
Soluzione 1



Soluzione 2

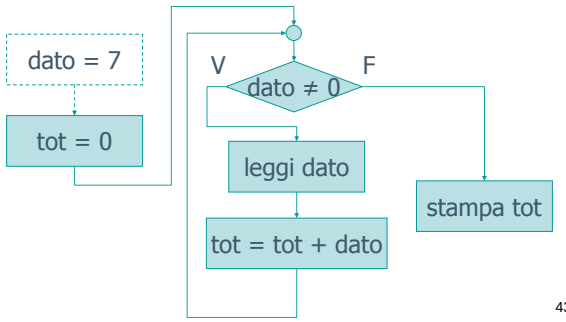


Soluzione 3



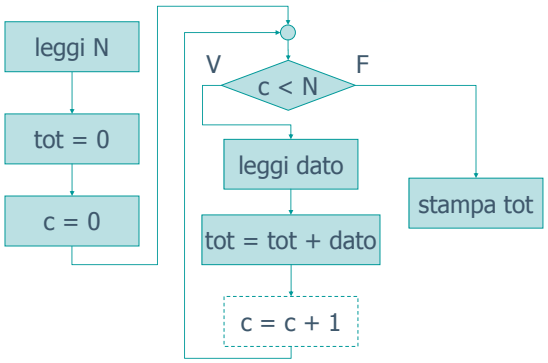
Errore frequente

- Dimenticare l'inizializzazione di una variabile utilizzata all'interno del ciclo



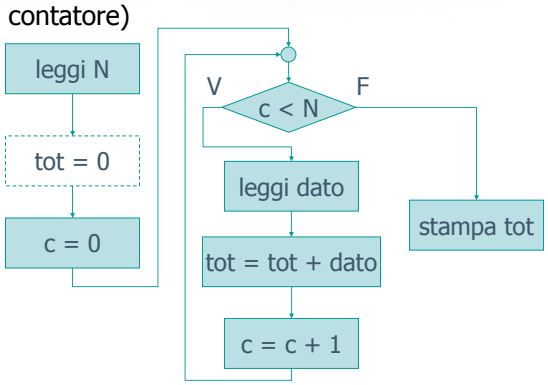
Errore frequente

- Dimenticare l'incremento della variabile contatore



Errore frequente

- Dimenticare di inizializzare le altre variabili (oltre al contatore)



Cicli ed iterazioni

Istruzione `while`

Istruzione `while`

- Sintassi dell'istruzione
- Esercizio "Media aritmetica"
- Esecuzione del programma
- Cicli `while` annidati
- Esercizio "Quadrato"

Istruzione `while`

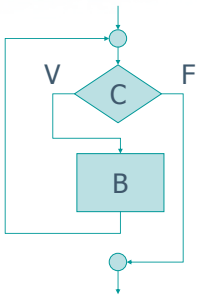
Sintassi dell'istruzione

Istruzioni di ripetizione in C

- Nel linguaggio C esistono tre distinte istruzioni di iterazione
  - `while`
  - `do-while`
  - `for`
- La forma più generale è l'istruzione di tipo `while`
- L'istruzione `do-while` si usa in taluni contesti (es. controllo errori di input)
- L'istruzione `for` è usatissima, soprattutto per numero di iterazioni noto

Istruzione `while`

```
while ( C )
{
   B ;
}
```



Comportamento del `while`

```
while ( C )
{
   B ;
}
```

1. Valuta la condizione C
2. Se C è falsa, salta completamente l'iterazione e vai all'istruzione che segue la }
3. Se C è vera, esegui una volta il blocco di istruzioni B
4. Al termine del blocco B, ritorna al punto 1. per rivalutare la condizione C

Numero di iterazioni note

```
int i, N ;

i = 0 ;
while ( i < N )
{
   /* Corpo dell'iterazione */
   ...

   i = i + 1 ;
}
```


Esempio

num1-10.c

```
int i ;

i = 1 ;
while ( i <= 10 )
{
   printf("Numero = %d\n", i) ;
   i = i + 1 ;
}
```


Esempio

fatt.c

```
int i, n ;
float f ;
....  /* leggi n */  ....
i = 2 ;
f = 1.0 ;
while ( i <= n )
{
   f = f * i ;
   i = i + 1 ;
}
printf("Fattoriale di %d = %f\n",
   n, f);
```


Particolarità

- Nel caso in cui il corpo del `while` sia composto di una sola istruzione, si possono omettere le parentesi graffe
  - Non succede quasi mai

```
while ( C )
{
   B ;
}
```

```
while ( C )
   B ;
```

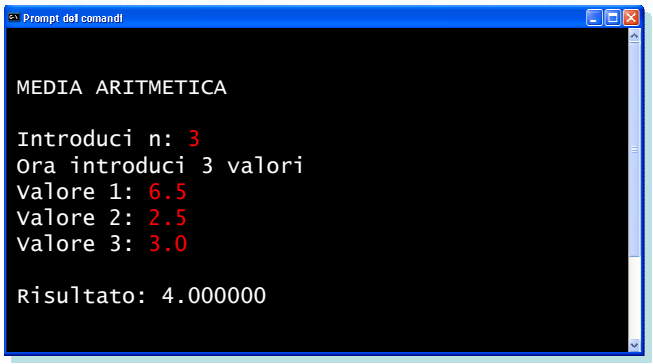
Istruzione `while`

Esercizio "Media aritmetica"

Esercizio "Media aritmetica"

- Si realizzi un programma C in grado di
  - Leggere un numero naturale n
  - Leggere n numeri reali
  - Calcolare e visualizzare la media aritmetica di tali numeri
- Osservazione
  - Attenzione al caso in cui n≤0

Analisi



Algoritmo

- Acquisisci n
- Inizializza totale = 0
- Ripeti n volte
  - Acquisisci un dato
  - Somma il dato al totale dei dati acquisiti
- Calcola e stampa la media = totale / n

Algoritmo

- Acquisisci n
- Se n>0
  - Inizializza totale = 0
  - Ripeti n volte
    - Acquisisci un dato
    - Somma il dato al totale dei dati acquisiti
  - Calcola e stampa la media = totale / n
- Altrimenti stampa messaggio di errore

Traduzione in C (1/3)

media.c

```c
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    int i, n ;
    float dato ;
    float somma ;

    printf("MEDIA ARITMETICA\n");

    /* Leggi n */
    printf("Introduci n: ");
    scanf("%d", &n) ;
```


Traduzione in C (2/3)

media.c

```c
    /* Controlla la correttezza del valore n */

    if( n>0 )
    {
        /* n corretto... procedi! */
        ....vedi lucido seguente....
    }
    else
    {
        /* n errato in quanto e' n <= 0 */
        printf("Non ci sono dati da inserire\n");
        printf("Impossibile calcolare la media\n");
    }

} /* main */
```


Traduzione in C (3/3)

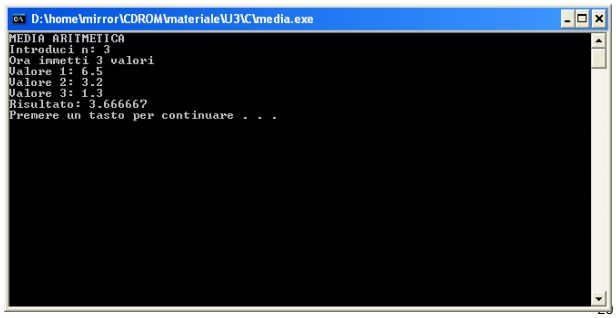
media.c

```c
        /* Leggi i valori e calcola la media */
        printf("Ora immetti %d valori\n", n) ;

        somma = 0.0 ;
        i = 0 ;
        while( i < n )
        {
            printf("Valore %d: ", i+1) ;
            scanf("%f", &dato) ;

            somma = somma + dato ;

            i = i + 1 ;
        }

        printf("Risultato: %f\n", somma/n) ;
```

Istruzione while

Esecuzione del programma

Verifica "Media aritmetica"



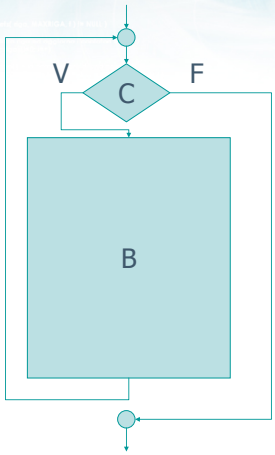
Istruzione while

Cicli while annidati

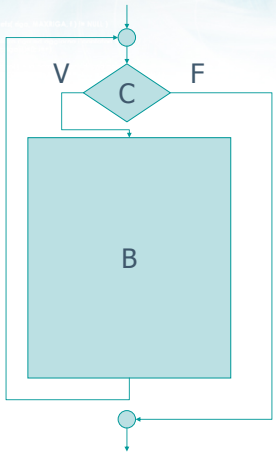
Annidamento di cicli

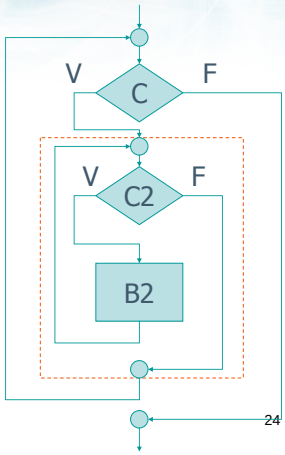
- All'interno del corpo del ciclo `while` è possibile racchiudere qualsiasi altra istruzione C
- In particolare, è possibile racchiudere un'istruzione `while` all'interno di un'altra istruzione `while`
- In tal caso, per ogni singola iterazione del ciclo `while` più esterno, vi saranno tutte le iterazioni previste per il ciclo più interno

Cicli while annidati



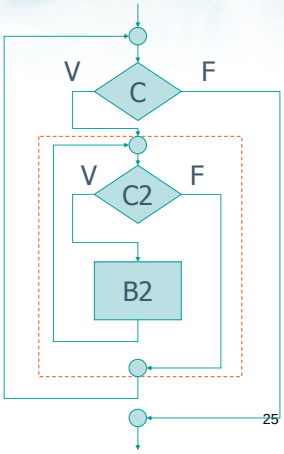
Cicli while annidati





Cicli while annidati

```
while( C )
{
    while( C2 )
    {
        B2 ;
    }
}
```



Esempio

conta99.c

```
i = 0 ;
while( i<N )
{
    j = 0 ;
    while( j<N )
    {
        printf("i=%d - j=%d\n", i, j);

        j = j + 1 ;
    }

    i = i + 1 ;
}
```


Esempio

conta99.c

```
i = 0 ;
while( i<N )
{
  j = 0 ;
  while( j<N )
  {
    printf("i=%d - j=%d\n", i, j);
    j = j + 1 ;
  }
  i = i + 1 ;
}
```

```
i=0 - j=0
i=0 - j=1
i=0 - j=2
i=1 - j=0
i=1 - j=1
i=1 - j=2
i=2 - j=0
i=2 - j=1
i=2 - j=2
```

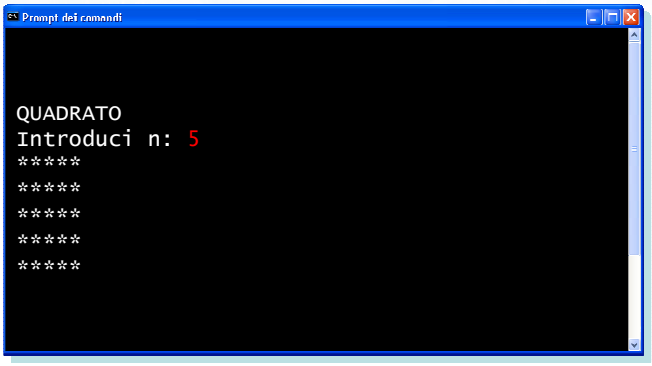

Istruzione while

Esercizio "Quadrato"

Esercizio "Quadrato"

- Si realizzi un programma C in grado di
  - Leggere un numero naturale n
  - Visualizzare un quadrato di lato n costituito da asterischi

Analisi



Algoritmo

- Acquisisci n
- Ripeti n volte
  - Stampa una riga di n asterischi

Algoritmo

- Acquisisci n
- Ripeti n volte
  - Stampa una riga di n asterischi
    - Ripeti n volte
      - Stampa un singolo asterisco
    - Vai a capo

Traduzione in C


```
i = 0 ;
while( i<n )
{
    j = 0 ;
    while( j<n )
    {
        printf("*") ;
        j = j + 1 ;
    }

    printf("\n");

    i = i + 1 ;
}
```


Traduzione in C


```
i = 0 ;
while( i<n )
{
    j = 0 ;
    while( j<n )
    {
        printf("*") ;
        j = j + 1 ;
    }

    printf("\n");

    i = i + 1 ;
}
```

Ripeti n volte

Stampa una riga di n asterischi

Traduzione in C


```
i = 0 ;
while( i<n )
{
    j = 0 ;
    while( j<n )
    {
        printf("*") ;
        j = j + 1 ;
    }

    printf("\n");

    i = i + 1 ;
}
```

Stampa n asterischi

Vai a capo

Traduzione in C


```
i = 0 ;
while( i<n )
{
    j = 0 ;
    while( j<n )
    {
        printf("*") ;
        j = j + 1 ;
    }

    printf("\n");

    i = i + 1 ;
}
```

Ripeti n volte

Stampa un asterisco

Cicli ed iterazioni

Schemi ricorrenti nei cicli

Schemi ricorrenti nei cicli

- Contatori
- Accumulatori
- Flag
- Esistenza e universalità

Schemi ricorrenti nei cicli

Contatori

Contatori

- Spesso in un ciclo è utile sapere
  - Quante iterazioni sono state fatte
  - Quante iterazioni rimangono da fare
  - Quale numero di iterazione sia quella corrente
- Per questi scopi si usano delle "normali" variabili intere, dette **contatori**
  - Inizializzate prima del ciclo
  - Incrementate/decrementate ad ogni iterazione
  - Oppure incrementate/decrementate ogni volta che si riscontra una certa condizione

Contatori

```
int i, N ;
...
i = 0 ;
while ( i < N )
{
   printf("Iterazione %d\n", i+1) ;
   i = i + 1 ;
}
```


Esempio

- Scrivere un programma in C che
  - legga dall'utente 10 numeri interi
  - al termine dell'inserimento, stampi
    - quanti tra i numeri inseriti sono positivi
    - quanti tra i numeri inseriti sono negativi
    - quanti tra i numeri inseriti sono nulli

Soluzione (1/3)

contaposneg.c

```
int npos, nneg, nzero ;

....

npos = 0 ;
nneg = 0 ;
nzero = 0 ;
```


Soluzione (2/3)

contaposneg.c

```
i = 0 ;
while( i<n )
{
    printf("Inserisci dato %d: ", i+1);
    scanf("%d", &dato);

    if( dato>0 )
        npos = npos + 1 ;
    else if( dato<0 )
        nneg = nneg + 1 ;
    else
        nzero = nzero + 1 ;

    i = i + 1 ;
}
```


Soluzione (3/3)

contaposneg.c

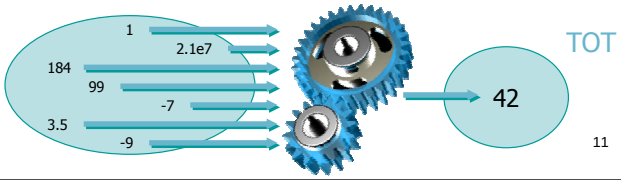
```
printf("Numeri positivi: %d\n", npos);
printf("Numeri negativi: %d\n", nneg);
printf("Numeri nulli: %d\n", nzero);
```


Schemi ricorrenti nei cicli

Accumulatori

Accumulatori (1/2)

- Spesso in un ciclo occorre calcolare un valore TOT che dipende dall'insieme dei valori analizzati nelle singole iterazioni
- Esempi:
  - TOT = sommatoria dei dati analizzati
  - TOT = produttoria dei dati analizzati
  - TOT = massimo, minimo dei dati analizzati



Accumulatori (2/2)

- In questo caso si usano delle variabili (intere o reali) dette **accumulatori**
  - Inizializzare TOT al valore che dovrebbe avere in assenza di dati (come se fosse n=0)
  - Ad ogni iterazione, aggiornare TOT tenendo conto del dato appena analizzato
  - Al termine del ciclo, TOT avrà il valore desiderato

Esempio: somma primi 10 interi

- Si scriva un programma in C che stampi il valore della somma dei primi 10 numeri interi

Analisi

- Inizializzazione di TOT
  - Qual è la somma dei primi 0 numeri interi?
    - TOT = 0
- Aggiornamento di TOT
  - Sapendo che TOT è la somma dei primi (i-1) numeri interi, e sapendo che il prossimo numero intero da sommare vale i, quanto dovrà valere TOT?
    - TOT = TOT + i

Soluzione: somma primi 10 interi

```
int tot ;

i = 1 ;
tot = 0 ;
while( i<=10 )
{
    tot = tot + i ;

    i = i + 1 ;
}

printf("La somma dei numeri da 1 a 10 ");
printf("vale %d\n", tot) ;
```


Esempio: fattoriale di K

- Si scriva un programma in C che, dato un numero intero K, calcoli e stampi il fattoriale di K
- TOT = K!

$$TOT = K! = \prod_{i=1}^{i=K} i$$

Analisi

- Inizializzazione di TOT
  - Qual è il valore del fattoriale per K=0?
    - TOT = 1.0
- Aggiornamento di TOT
  - Sapendo che TOT è pari al fattoriale di i-1, e sapendo che il prossimo numero da considerare è i, quanto dovrà valere TOT?
    - TOT = TOT * i

Soluzione: fattoriale di K

```
float tot ;

i = 1 ;
tot = 1.0 ;
while( i<=K )
{
    tot = tot * i ;

    i = i + 1 ;
}

printf("Il fattoriale di %d ", K);
printf("vale %f\n", tot) ;
```

Esempio: massimo

- Si scriva un programma in C che
  - acquisisca da tastiera N numeri reali
  - stampi il valore massimo tra i numeri acquisiti

Analisi

- Inizializzazione di TOT
  - Qual è il valore del massimo in un insieme di 0 numeri?
    - Non esiste, non è definito!
    - TOT = numero molto piccolo, che non possa certamente essere scambiato con il massimo
- Aggiornamento di TOT
  - Sapendo che TOT è pari al massimo dei primi i-1 dati, e sapendo che il prossimo dato da considerare è d, quanto dovrà valere TOT?
    - Se d<=TOT, allora TOT rimane il massimo
    - Se d>TOT, allora il nuovo massimo sarà TOT=d

Esempio: massimo

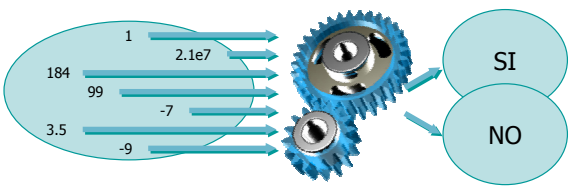
```
int max ;

i = 0 ;
max = INT_MIN ;
while( i<N )
{
    scanf("%d", &dato) ;
    if(dato>max)
        max = dato ;

    i = i + 1 ;
}

printf("Massimo = %d\n", max);
```


Schemi ricorrenti nei cicli

Flag

Flag, indicatori, variabili logiche

- Spesso occorre analizzare una serie di dati per determinare se si verifica una certa condizione
- Esempi:
  - Tra i dati inseriti esiste il numero 100?
  - Esistono due numeri consecutivi uguali?



Problemi

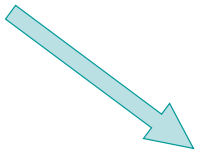
- Nel momento in cui si "scopre" il fatto, non si può interrompere l'elaborazione ma occorre comunque terminare il ciclo
- Al termine del ciclo, come fare a "ricordarsi" se si era "scoperto" il fatto o no?

Una possibile soluzione

- Per sapere
  - Se una certa condizione si verifica
    è possibile contare
  - Quante volte quella condizione si verifica
    ed in seguito verificare
  - Verificare se il conteggio è diverso da zero
- Ci riconduciamo ad un problema risolubile per mezzo di una variabile contatore

Esempio 1

Tra i dati inseriti esiste il numero 100?



Conta quante volte tra i dati inseriti compare il numero 100

Il conteggio è > 0 ?

Soluzione (1/3)

trova100v1.c

```
int i, n ;
int dato ;
int conta ;
/* conta il numero di "100" letti */

printf("TROVA 100\n");

n = 10 ;

printf("Inserisci %d numeri\n", n);
```


Soluzione (2/3)

trova100v1.c

```
conta = 0 ;

i = 0 ;
while( i<n )
{
    printf("Inserisci dato %d: ", i+1);
    scanf("%d", &dato);

    if( dato == 100 )
        conta = conta + 1 ;

    i = i + 1 ;
}
```

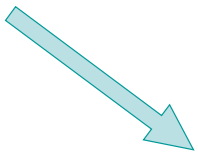

Soluzione (3/3)

trova100v1.c

```
if( conta != 0 )
    printf("Ho trovato il 100\n");
else
    printf("NON ho trovato il 100\n");
```


Esempio 2

Esistono due numeri consecutivi uguali?



Conta quante volte due numeri consecutivi sono uguali

Il conteggio è > 0 ?

Soluzione (1/3)

ugualiv1.c

```
    int i, n ;
    int dato ;
    int precedente ;
    int conta ;
/* conta il numero di "doppioni" trovati */

    printf("TROVA UGUALI\n");

    n = 10 ;

    printf("Inserisci %d interi\n", n);
```


Soluzione (2/3)

ugualiv1.c

```
conta = 0 ;

precedente = INT_MAX ;
/* ipotesi: l'utente non lo inserira' mai */

i = 0 ;
while( i<n )
{
    printf("Inserisci dato %d: ", i+1);
    scanf("%d", &dato);

    if( dato == precedente )
        conta = conta + 1 ;

    precedente = dato ;

    i = i + 1 ;
}
```


Soluzione (3/3)

ugualiv1.c

```
if( conta != 0 )
    printf("Ho trovato dei numeri
            consecutivi uguali\n");
else
    printf("NON ho trovato dei numeri
            consecutivi uguali\n");
```

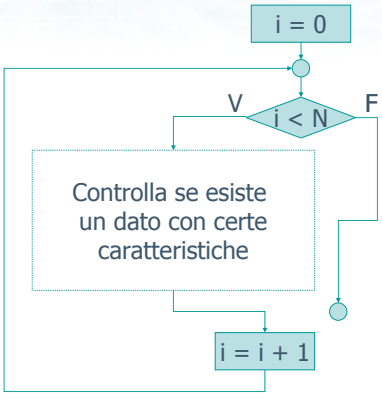

Svantaggi

- Il contatore determina quante volte si verifica la condizione ricercata
- In realtà non mi serve sapere quante volte, ma solo se si è verificata almeno una volta
- Usiamo un contatore "degenere", che una volta arrivato ad 1 non si incrementa più

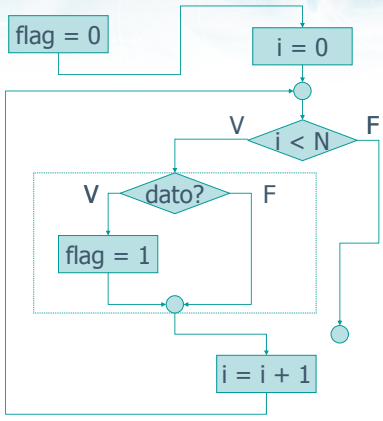
Variabili "flag"

- Variabili intere che possono assumere solo due valori
  - Variabile = 0 ⇒ la condizione non si è verificata
  - Variabile = 1 ⇒ la condizione si è verificata
- Viene inizializzata a 0 prima del ciclo
- Se la condizione si verifica all'interno del ciclo, viene posta a 1
- Al termine del ciclo si verifica il valore
- Sinonimi: Flag, Variabile logica, Variabile booleana, Indicatore

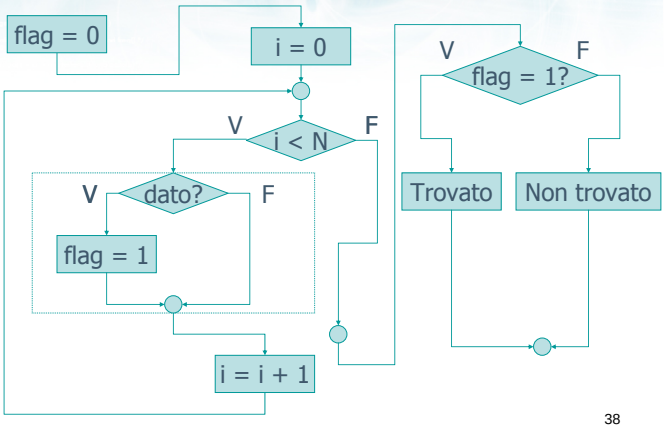
Analisi



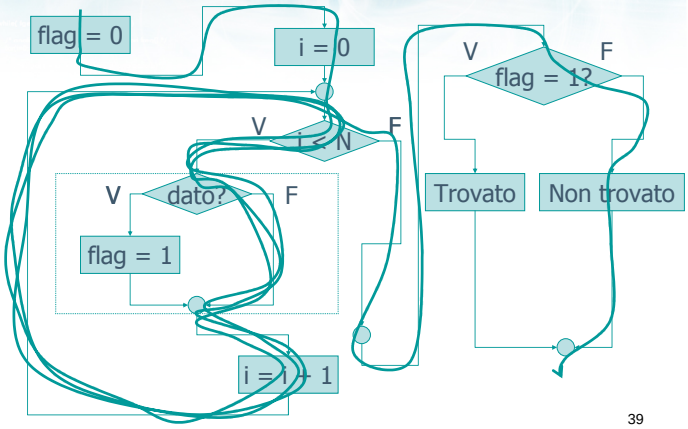
Analisi



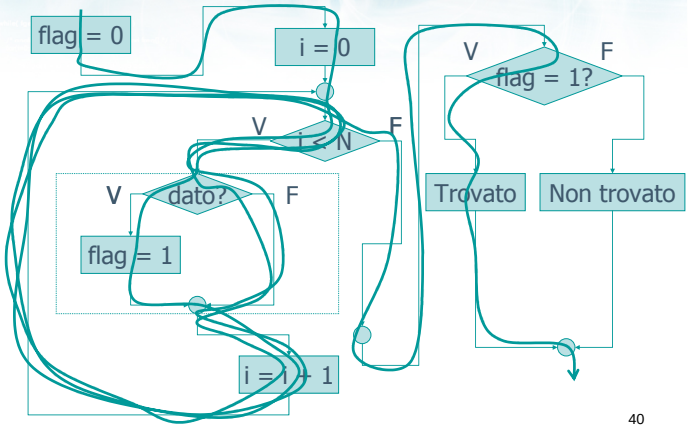
Analisi



Analisi



Analisi



Soluzione con flag – esempio 1

```
int trovato ; /* ho visto il numero "100"? */
....
trovato = 0 ;

i = 0 ;
while( i<n )
{
    scanf("%d", &dato);

    if( dato == 100 )
        trovato = 1 ;

    i = i + 1 ;
}

if( trovato != 0 )
    printf("Trovato il numero 100\n");
else
    printf("NON trovato il numero 100\n");
```

trova100v2.c

Soluzione con flag – esempio 2

```
int doppi ; /* trovati "doppioni" ? */
...
doppi = 0 ;

precedente = INT_MAX ;
i = 0 ;
while( i<n )
{
    scanf("%d", &dato);

    if( dato == precedente )
        doppi = 1 ;

    precedente = dato ;
    i = i + 1 ;
}

if( doppi != 0 )
    printf("Trovati consecutivi uguali\n");
```

ugualiv2.c

Schemi ricorrenti nei cicli

Esistenza e universalità

Ricerca di esistenza o universalità

- L'utilizzo dei flag è può essere utile quando si desiderino verificare delle proprietà su un certo insieme di dati
  - È vero che tutti i dati verificano la proprietà?
  - È vero che almeno un dato verifica la proprietà?
  - È vero che nessun dato verifica la proprietà?
  - È vero che almeno un dato non verifica la proprietà?

Esempi

- Verificare che tutti i dati inseriti dall'utente siano positivi
- Determinare se una sequenza di dati inseriti dall'utente è crescente
- Due numeri non sono primi tra loro se hanno almeno un divisore comune
  - esiste almeno un numero che sia divisore dei due numeri dati
- Un poligono regolare ha tutti i lati di lunghezza uguale
  - ogni coppia di lati consecutivi ha uguale lunghezza

Formalizzazione

- È vero che tutti i dati verificano la proprietà?
  - $\forall x : P(x)$
- È vero che almeno un dato verifica la proprietà?
  - $\exists x : P(x)$
- È vero che nessun dato verifica la proprietà?
  - $\forall x : \text{not } P(x)$
- È vero che almeno un dato non verifica la proprietà?
  - $\exists x : \text{not } P(x)$

Realizzazione (1/2)

- Esistenza: $\exists x : P(x)$
  - Inizializzo flag F = 0
  - Ciclo su tutte le x
    - Se P(x) è vera
      - Pongo F = 1
  - Se F = 1, l'esistenza è dimostrata
  - Se F = 0, l'esistenza è negata

Realizzazione (1/2)

- Esistenza: $\exists x : P(x)$
  - Inizializzo flag F = 0
  - Ciclo su tutte le x
    - Se P(x) è vera
      - Pongo F = 1
  - Se F = 1, l'esistenza è dimostrata
  - Se F = 0, l'esistenza è negata
- Universalità: $\forall x : P(x)$
  - Inizializzo flag F = 1
  - Ciclo su tutte le x
    - Se P(x) è falsa
      - Pongo F = 0
  - Se F = 1, l'universalità è dimostrata
  - Se F = 0, l'universalità è negata

Realizzazione (2/2)

- Esistenza: ∃x : not P(x)
  - Inizializzo flag F = 0
  - Ciclo su tutte le x
    - Se P(x) è falsa
      - Pongo F = 1
  - Se F = 1, l'esistenza è dimostrata
  - Se F = 0, l'esistenza è negata
- Universalità: ∀x : not P(x)
  - Inizializzo flag F = 1
  - Ciclo su tutte le x
    - Se P(x) è vera
      - Pongo F = 0
  - Se F = 1, l'universalità è dimostrata
  - Se F = 0, l'universalità è negata

Esempio 1

- Verificare che tutti i dati inseriti dall'utente siano positivi

```
int positivi ;
...
positivi = 1 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato <= 0 )
        positivi = 0 ;
    ....
    i = i + 1 ;
}
if( positivi == 1 )
    printf("Tutti positivi\n");
```


Esempio 2

- Determinare se una sequenza di dati inseriti dall'utente è crescente

```
int crescente ;
...
crescente = 1 ;
precedente = INT_MIN ;
i = 0 ;
while( i<n )
{
    ...
    if( dato < precedente )
        crescente = 0 ;
    precedente = dato ;
    ....
    i = i + 1 ;
}
```


Esempio 3

- Due numeri non sono primi tra loro se hanno almeno un divisore comune

```
int A, B ;
int noprimi ;
...
noprimi = 0 ;
i = 2 ;
while( i<=A )
{
    ...
    if( (A%i==0) && (B%i==0) )
        noprimi = 1 ;
    ....
    i = i + 1 ;
}
```


Esempio 4

- Un poligono regolare ha tutti i lati di lunghezza uguale

```
int rego ;
...
rego = 1 ;
precedente = INT_MIN ;
i = 0 ;
while( i<n )
{
    ...
    if( lato != precedente )
        rego = 0 ;
    precedente = lato ;
    ....
    i = i + 1 ;
}
```


Errore frequente

- Resettare il flag al valore di inizializzazione, dimenticando di fatto eventuali condizioni incontrate in precedenza

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
        trovato = 1 ;
    else
        trovato = 0 ;
    ...
    i = i + 1 ;
}
```

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
        trovato = 1 ;

    ...
    i = i + 1 ;
}
```

Errore frequente

- Passare ai fatti non appena trovato il primo elemento che soddisfa la proprietà

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
    {
        trovato = 1 ;
        printf("W!\n");
    }
    ...
    i = i + 1 ;
}
```

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
        trovato = 1 ;
    ....
    i = i + 1 ;
}
if(trovato==1)
    printf("W!\n");
```

Errore frequente

- Pensare che al primo fallimento si possa determinare che la proprietà è falsa

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
        trovato = 1 ;
    else
        printf("NO!\n");
    ...
    i = i + 1 ;
}
```

```
trovato = 0 ;
i = 0 ;
while( i<n )
{
    ...
    if( dato == 100 )
        trovato = 1 ;
    ....
    i = i + 1 ;
}
if(trovato==0)
    printf("NO!\n");
```

Cicli ed iterazioni

Istruzione `for`

Istruzione `for`

- Sintassi dell'istruzione
- Operatori di autoincremento
- Cicli `for` annidati

Istruzione `for`

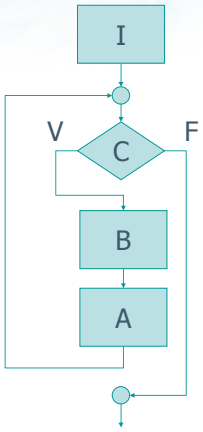
Sintassi dell'istruzione

Istruzione `for`

- L'istruzione fondamentale è `while`
  - La condizione solitamente valuta una variabile di controllo
  - Occorre ricordarsi l'inizializzazione della variabile di controllo
  - Occorre ricordarsi di aggiornare (incrementare, ...) la variabile di controllo
- L'istruzione `for` rende più semplice ricordare queste cose

Istruzione `for`

```
for ( I; C; A )
{
   B ;
}
```



Istruzione `for`

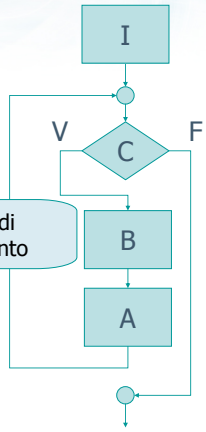
Istruzione di inizializzazione

```
for ( I; C; A )
{
   B ;
}
```

Istruzione di aggiornamento

Corpo

Condizione



Esempio

num1-10v2.c

```
int i ;

for ( i=1; i<=10; i=i+1 )
{
   printf("Numero = %d\n", i) ;
}
```


Equivalenza for←→while

```
for ( I; C; A )
{
   B ;
}
```

```
I ;
while ( C )
{
   B ;
   A ;
}
```


Esempio

```
int i ;

for ( i=1; i<=10; i=i+1 )
{
   printf("%d\n", i) ;
}
```

```
int i ;

i = 1 ;
while ( i <= 10 )
{
   printf("%d\n", i) ;
   i = i + 1 ;
}
```


Utilizzo prevalente (1/2)

- Le istruzioni di inizializzazione `I` e di aggiornamento `A` possono essere qualsiasi
- Solitamente `I` viene utilizzata per inizializzare il contatore di controllo del ciclo, e quindi è del tipo
  - `i = 0`
- Solitamente `A` viene utilizzata per incrementare (o decrementare) il contatore, e quindi è del tipo
  - `i = i + 1`

```
for ( I; C; A )
{
   B ;
}
```


Utilizzo prevalente (2/2)

- L'istruzione `for` può sostituire un qualsiasi ciclo `while`
- Solitamente viene utilizzata, per maggior chiarezza, nei cicli con numero di iterazioni noto a priori

Cicli for con iterazioni note

```
int i ;

for ( i=0; i<N; i=i+1 )
{
   .......
}
```

```
int i ;

for ( i=1; i<=N; i=i+1 )
{
   .......
}
```

```
int i ;

for ( i=N; i>0; i=i-1 )
{
   .......
}
```

```
int i ;

for ( i=N-1; i>=0; i=i-1)
{
   .......
}
```

Casi particolari (1/6)

- Se non è necessario inizializzare nulla, si può omettere l'istruzione I
  - `for( ; i != 0 ; i = i - 1 )`
  - La condizione C viene comunque valutata prima della prima iterazione, pertanto le variabili coinvolte dovranno essere inizializzate prima dell'inizio del ciclo
  - Il simbolo `;` è sempre necessario

```
for ( I; C; A )
{
   B ;
}
```


Casi particolari (2/6)

- Se l'aggiornamento viene fatto nel ciclo, si può omettere l'istruzione A
  - `for( dato = INT_MIN; dato != 0 ; )`
  - La responsabilità di aggiornare la variabile di controllo (`dato`) è quindi del corpo B del ciclo
  - Il simbolo `;` è sempre necessario

```
for ( I; C; A )
{
   B ;
}
```


Casi particolari (3/6)

- Se occorre inizializzare più di una variabile, è possibile farlo separando le varie inizializzazioni con un simbolo `,`
  - `for( i=0, j=0; i<N; i=i+1 )`
  - Solitamente uno solo è il contatore del ciclo, gli altri saranno altri contatori, accumulatori o flag

```
for ( I; C; A )
{
   B ;
}
```


Casi particolari (4/6)

- Se occorre aggiornare più di una variabile, è possibile farlo separando i vari aggiornamenti con un simbolo `,`
  - `for( i=0; i<N; i=i+1, k=k-1 )`

```
for ( I; C; A )
{
   B ;
}
```


Casi particolari (5/6)

- Nel caso in cui si ometta sia I che A, il ciclo `for` degenera nel ciclo `while` equivalente
  - `for( ; i<N; )`
  - `while( i<N )`

```
for ( I; C; A )
{
   B ;
}
```


Casi particolari (6/6)

- È possibile omettere la condizione C, in tal caso viene considerata come sempre vera
  - `for( i=0; ; i=i+1)`
  - Questo costrutto genera un ciclo infinito. È necessario che il ciclo venga interrotto con un altro meccanismo (`break`, `return`, `exit`)
  - Talvolta si incontra anche un ciclo infinito "puro"
    - `for( ; ; )`

```
for ( I; C; A )
{
   B ;
}
```

Istruzione for

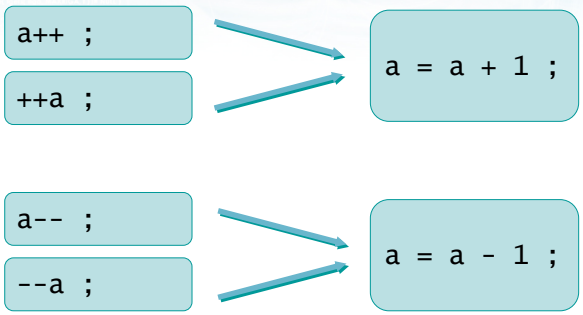
Operatori di autoincremento

Istruzione di aggiornamento

- Nella maggioranza dei casi, l'istruzione di aggiornamento A consiste in un incremento
  - `i = i + 1`
- oppure in un decremento
  - `i = i - 1`
- Il linguaggio C dispone di operatori specifici per semplificare la sintassi di queste operazioni frequenti

```
for ( I; C; A )
{
   B ;
}
```


Operatore di auto-incremento



Cicli for con iterazioni note

```
int i ;

for ( i=0; i<N; i++ )
{
   .......
}
```

```
int i ;

for ( i=1; i<=N; i++ )
{
   .......
}
```

```
int i ;

for ( i=N; i>0; i-- )
{
   .......
}
```

```
int i ;

for ( i=N-1; i>=0; i-- )
{
   .......
}
```

Istruzione for

Cicli for annidati

Annidamento di cicli

- Come sempre, all'interno del corpo B di un ciclo (`for` o `while`) è possibile annidare altri cicli (`for` o `while`)
- Non vi è limite al livello di annidamento
- I cicli più interni sono sempre eseguiti "più velocemente" dei cicli più esterni

Esempio

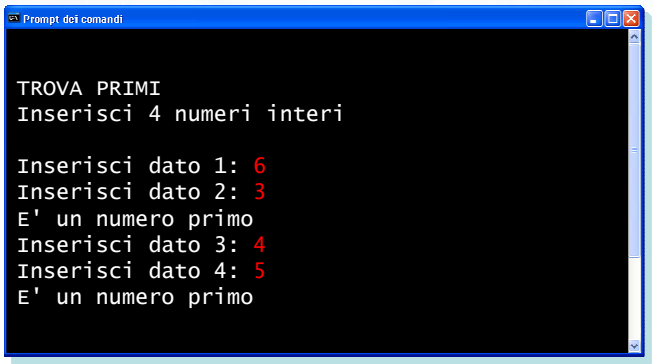
conta99v2.c

```
for( i=0; i<N; i++ )
{
    for( j=0; j<N; j++ )
    {
        printf("i=%d - j=%d\n", i, j);
    }
}
```

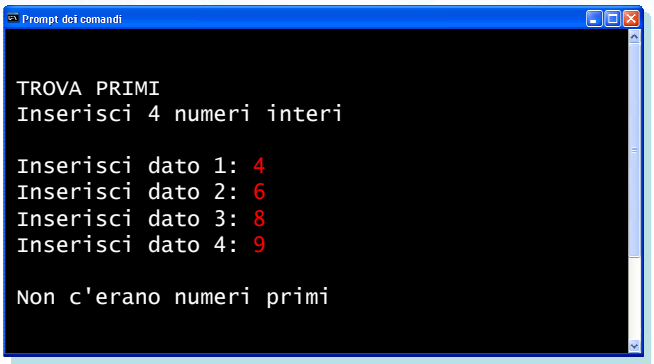

Esercizio

- Si scriva un programma in linguaggio C che
  - acquisisca da tastiera 10 numeri
  - per ciascuno di tali numeri determini se è un numero primo, stampando immediatamente un messaggio opportuno
  - al termine, se nessuno tra i numeri inseriti era un numero primo, stampi un messaggio opportuno

Analisi (1/2)



Analisi (2/2)



Numero primo

10primi.c

```
primo = 1 ;
for( j=2; j<dato; j++)
{
    if( dato%j == 0 )
        primo = 0 ;
}
```


Stampa se non ci sono primi

10primi.c

```
for( i=0; i<n; i++ )
{
    scanf("%d", &dato);

    ....determina se è un numero primo....

    if( primo == 1 )
    {
        printf("E' un numero primo\n");
    }
}
```

Stampa se è un primo


```
trovato = 0 ;
for( i=0; i<n; i++ )
{
    ....acquisisci dato e determina se è un numero primo....
    if( primo == 1 )
    {
        printf("E' un numero primo\n");
        trovato = 1 ;
    }
}
if( trovato == 0 )
    printf("Non ci sono primi\n") ;
```


Vista d'insieme


```
trovato = 0 ;
for( i=0; i<n; i++ )
{
    scanf("%d", &dato);

    primo = 1 ;
    for( j=2; j<dato; j++ )
    {
        if( dato%j == 0 )
            primo = 0 ;
    }

    if( primo == 1 )
    {
        printf("E' un numero primo\n");
        trovato = 1 ;
    }
}

if( trovato == 0 )
    printf("Non c'erano numeri primi\n");
```


Vista d'insieme

```
trovato = 0 ;
for( i=0; i<n; i++ )
{
    scanf("%d", &dato);

    primo = 1 ;
    for( j=2; j<dato; j++ )
    {
        if( dato%j == 0 )
            primo = 0 ;
    }

    if( primo == 1 )
    {
        printf("E' un numero primo\n");
        trovato = 1 ;
    }
}

if( trovato == 0 )
    printf("Non c'erano numeri primi\n");
```

Ciclo esterno

Ciclo interno

Vista d'insieme

```
trovato = 0 ;
for( i=0; i<n; i++ )
{
    scanf("%d", &dato);

    primo = 1 ;
    for( j=2; j<dato; j++ )
    {
        if( dato%j == 0 )
            primo = 0 ;
    }

    if( primo == 1 )
    {
        printf("E' un numero primo\n");
        trovato = 1 ;
    }
}

if( trovato == 0 )
    printf("Non c'erano numeri primi\n");
```

Flag interno: numero primo

Inizializzazione

Aggiornamento

Verifica

Vista d'insieme

```
trovato = 0 ;
for( i=0; i<n; i++ )
{
    scanf("%d", &dato);

    primo = 1 ;
    for( j=2; j<dato; j++ )
    {
        if( dato%j == 0 )
            primo = 0 ;
    }

    if( primo == 1 )
    {
        printf("E' un numero primo\n");
        trovato = 1 ;
    }
}

if( trovato == 0 )
    printf("Non c'erano numeri primi\n");
```

Flag esterno: nessun primo

Inizializzazione

Aggiornamento

Verifica

Cicli ed iterazioni

Approfondimenti

Approfondimenti

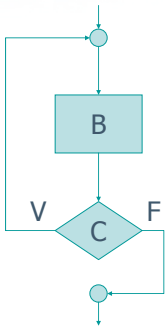
- Istruzione do-while
- Istruzione break
- Istruzione continue

Approfondimenti

Istruzione do-while

Istruzione do-while

```
do {
   B ;
} while ( C ) ;
```



Confronto

- Istruzione while
  - Condizione valutata prima di ogni iterazione
  - Numero minimo di iterazioni: 0
  - Per uscire: condizione falsa
- Istruzione do-while
  - Condizione valutata al termine di ogni iterazione
  - Numero minimo di iterazioni: 1
  - Per uscire: condizione falsa

Esempio

- Acquisire un numero compreso tra 1 e 10 da tastiera
- Nel caso in cui l'utente non inserisca il numero correttamente, chiederlo nuovamente

Soluzione

```
printf("Numero tra 1 e 10\n");
do {

    scanf("%d", &n) ;

} while ( n<1 || n>10 ) ;
```


Soluzione migliore

```
printf("Numero tra 1 e 10\n");
do {

    scanf("%d", &n) ;

    if( n<1 || n>10 )
        printf("Errore: ripeti\n");

} while ( n<1 || n>10 ) ;
```


Esempio

- Si scriva un programma in C che calcoli la somma di una sequenza di numeri interi
- La sequenza termina quando l'utente inserisce il dato 9999

Soluzione

```
somma = 0 ;
do {

    scanf("%d", &dato) ;

    if( n != 9999 )
        somma = somma + dato ;

} while ( dato != 9999 ) ;
```


Approfondimenti

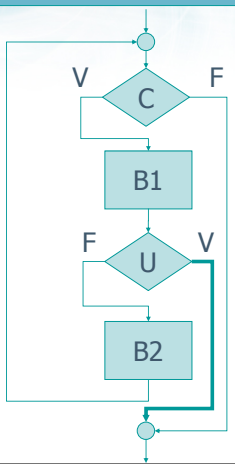
Istruzione break

Interruzione dei cicli

- Di norma, un ciclo termina quando la condizione di controllo diventa falsa
  - Necessario arrivare al termine del corpo per poter valutare la condizione
- Talvolta potrebbe essere comodo interrompere prematuramente l'esecuzione di un ciclo
  - A seguito di condizioni di errore
  - Quando è stato trovato ciò che si cercava

Istruzione break

```
while ( C )
{
   B1 ;
   if ( U )
      break ;
   B2 ;
}
```



Funzionamento

- Quando viene eseguita l'istruzione `break`
  - Viene interrotta l'esecuzione del corpo del ciclo
  - Il flusso di esecuzione passa all'esterno del ciclo che contiene il `break`
  - Si esce dal ciclo anche se la condizione di controllo è ancora vera
  - In caso di cicli annidati, si esce solo dal ciclo più interno
- Funziona con cicli `while`, `for`, do-`while`

Esempio

- Si scriva un programma in C che calcoli la somma di una sequenza di numeri interi
- La sequenza termina quando l'utente inserisce il dato 9999

Soluzione

```
somma = 0 ;
do {

    scanf("%d", &dato) ;

    if( dato == 9999 )
        break;

    somma = somma + dato ;

} while ( 1 ) ;
```


Esempio

- Si scriva un programma in C che determini se un numero inserito da tastiera è primo

Soluzione

```
scanf("%d", &dato) ;
primo = 1 ;
for ( i=2; i<dato; i++ )
{
    if( dato%i == 0 )
    {
        primo = 0 ;
        break ; /* inutile continuare */
    }
}
```

Note

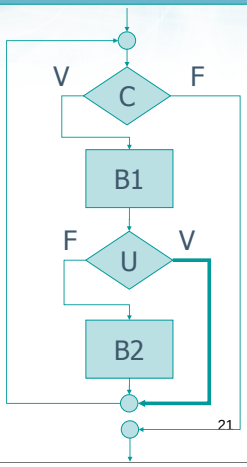
- L'istruzione `break` crea programmi non strutturati: usare con cautela
- Non è possibile uscire da più cicli annidati contemporaneamente

Approfondimenti

Istruzione `continue`

Istruzione `continue`

```
while ( C )
{
   B1 ;
   if ( U )
      continue ;
   B2 ;
}
```



Funzionamento

- Quando viene eseguita l'istruzione `continue`
  - Viene interrotta l'esecuzione del corpo del ciclo
  - Il flusso di esecuzione passa al termine del corpo
  - Nel caso di cicli `for`, viene eseguita l'istruzione di aggiornamento
  - Viene nuovamente valutata la condizione
  - Il ciclo continua normalmente
  - In caso di cicli annidati, si esce solo dal ciclo più interno
- Funziona con cicli `while`, `for`, do-`while`

Esempio

```
somma = 0 ;
do {

    scanf("%d", &dato) ;

    if( dato == 9999 )
        continue ; /* non considerarlo */

    somma = somma + dato ;

} while ( dato != 9999 ) ;
```


Avvertenze

- L'istruzione `continue` è oggettivamente poco utilizzata
- Crea un "salto" poco visibile: accompagnarla sempre con commenti evidenti

Cicli ed iterazioni

Esercizi proposti

Esercizi proposti

- Esercizio "Decimale-binario"
- Esercizio "Massimo Comun Divisore"
- Esercizio "Triangolo di Floyd"

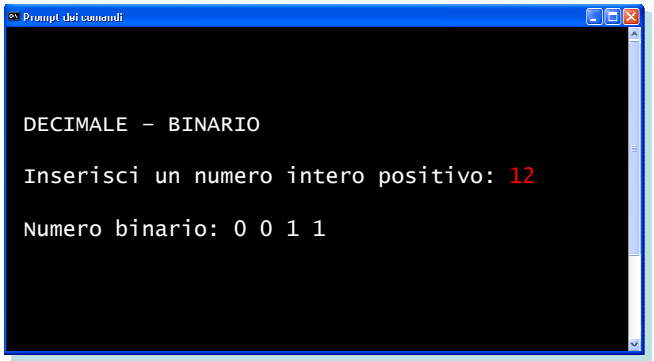
Esercizi proposti

Esercizio "Decimale-binario"

Esercizio "Decimale-binario"

- Si realizzi un programma in C in grado di
  - Leggere un numero naturale n
  - Convertire tale numero dalla base 10 alla base 2
  - Visualizzare il risultato, a partire dalla cifra meno significativa

Analisi



Divisioni successive

- n = 12
- n%2 = 0 → cifra 0
- n = n / 2 = 6
- n%2 = 0 → cifra 0
- n = n / 2 = 3
- n%2 = 1 → cifra 1
- n = n / 2 = 1
- n%2 = 1 → cifra 1
- n = n / 2 = 0 ➔ STOP

N	N % 2
12	0
6	0
3	1
1	1
0	

Soluzione

bin-dec.c

```
while( n!=0 )
{
    if( n%2 == 1 )
        printf("1 ") ;
    else
        printf("0 ") ;

    n = n / 2 ;
}
```


Esercizi proposti

Esercizio "Massimo Comun Divisore"

Esercizio "Massimo Comun Divisore"

- Si scriva un programma in C in grado di calcolare il massimo comun divisore (MCD) di due numeri interi.
- Il MCD è definito come il massimo tra i divisori comuni ai due numeri.

Analisi

- Diciamo N1 e N2 i numeri inseriti dall'utente
- Il MCD di N1 e N2 è il **massimo** tra i numeri che sono **divisori** sia di N2, sia di N1.
  - Troviamo i divisori di N1 ...
  - ... tra quelli che sono anche divisori di N2 ...
  - ... calcoliamo il massimo

Algoritmo

bin-dec.c

- k_max = 0
- for k = da 1 a N1
  - se k è un divisore di N1
    - se k è un divisore di N2
      - aggiorna k_max = k
- MCD = k_max

Esercizi proposti

Esercizio "Triangolo di Floyd"

Esercizio "Triangolo di Floyd"

Esercizio "Triangolo di Floyd"

- Scrivere un programma C per la rappresentazione del triangolo di Floyd.
- Il programma riceve da tastiera un numero interno N.
- Il programma visualizza le prima N righe del triangolo di Floyd.

```
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
```

N=5

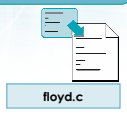
Analisi

- Occorre stampare i primi numeri interi in forma di triangolo
- La riga k-esima ha k elementi

```
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
```

N=5

Algoritmo



- cont = 1
- for riga = da 1 a N
  - for colonna = da 1 a riga
    - stampa cont
    - cont++
  - vai a capo

```
1
2  3
4  5  6
7  8  9  10
11  12  13  14  15
```

N=5

Cicli ed iterazioni

Sommario

Argomenti trattati

- Ripetizione del del flusso di esecuzione
- Inizializzazione, Condizione, Aggiornamento, Corpo
- Istruzione `while`
- Istruzione `for`
- Cicli annidati

Tecniche di programmazione

- Cicli con numero di iterazioni note o ignote
- Contatori
- Accumulatori
- Flag

Schemi ricorrenti

- Calcolo di somme, medie, ...
- Calcolo di max, min
- Ricerca di esistenza
- Ricerca di universalità
- Controllo dei dati in input

Suggerimenti

- Ricordare di verificare sempre le 4 parti del ciclo
  - Inizializzazione, Condizione, Corpo, Aggiornamento
- Le complicazioni nascono da
  - Cicli annidati
  - Condizioni if annidate in cicli
  - Annidamento di flag o ricerche
- Procedere sempre per gradi
  - Pseudo-codice o flow chart
  - Identificare chiaramente il ruolo dei diversi cicli

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Vettori

Vettori

- Strutture dati complesse
- I vettori in C
- Operazioni elementari sui vettori
- Esercizi guidati sui vettori
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 1 e 5
  - Cabodi, Quer, Sonza Reorda: capitolo 5
  - Dietel & Dietel: capitolo 6
- Dispense
  - Scheda: "Vettori in C"

Vettori

Strutture dati complesse

Strutture dati complesse

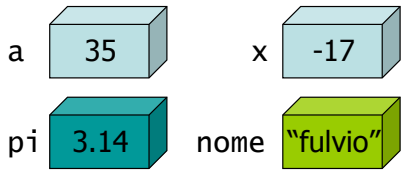
- Tipi di dato strutturati
- Introduzione ai vettori
- Caratteristiche dei vettori

Strutture dati complesse

Tipi di dato strutturati

Tipi di dato strutturati

- Finora abbiamo utilizzato dei tipi di dato semplici
  - `int`, `float`
  - Ogni variabile può contenere **un solo** valore
- Il linguaggio C permette di definire tipi di dato complessi, aggregando più variabili semplici

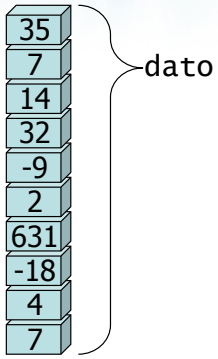


Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro

Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



Esigenze

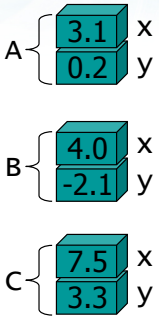
- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro

pitagora

1	2	3	4	5
2	4	6	8	10
3	6	9	12	15
4	8	12	16	20
5	10	15	20	25
6	12	18	24	30

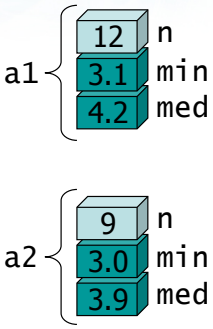
Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



Esigenze

- Leggere da tastiera 10 numeri, e stamparli in ordine inverso
- Calcolare e stampare la tavola pitagorica
- Calcolare l'area del triangolo date le coordinate dei vertici
- Per ogni auto calcolare il tempo medio e minimo sul giro



Dati strutturati (1/2)

- Raggruppare più variabili semplici in un'unica struttura complessa
- Diversi meccanismi di aggregazione
  - Vettore
  - Matrice
  - Struttura

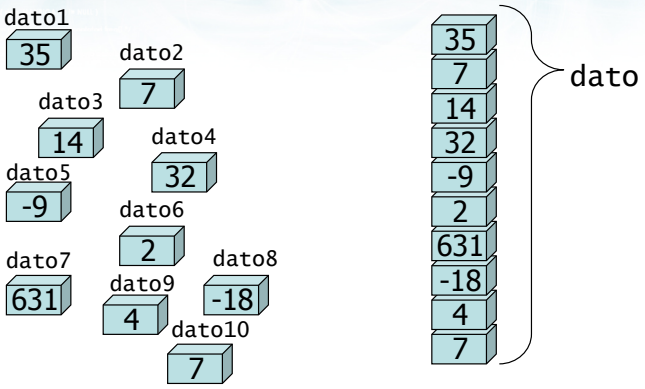
Dati strutturati (2/2)

- Una sola variabile, di tipo complesso, memorizza **tutti** i dati della struttura complessa
  - `Vettore_di_int dato`
  - `Matrice_di_int pitagora`
  - `Struttura_xy A, B, C`
  - `Struttura_auto a1, a2`

Strutture dati complesse

Introduzione ai vettori

Variabili e vettori



Da evitare...

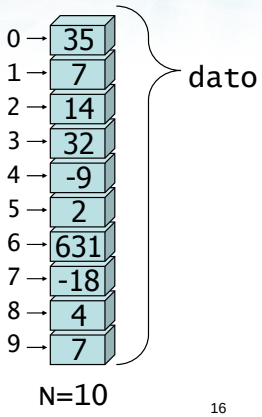
```
int main(void)
{
   int dato1, dato2, dato3, dato4, dato5 ;
   int dato6, dato7, dato8, dato9, dato10 ;
   . . . . .
   scanf("%d", &dato1) ;
   scanf("%d", &dato2) ;
   scanf("%d", &dato3) ;
   . . .
   scanf("%d", &dato10) ;

   printf("%d\n", dato10) ;
   printf("%d\n", dato9) ;
   printf("%d\n", dato8) ;
   . . .
   printf("%d\n", dato1) ;
}
```

Vettori

- Meccanismo di strutturazione più semplice
- Utilizzato per aggregare serie di dati
- Permette di memorizzare tutti i dati acquisiti o calcolati, e potervi accedere
  - In qualsiasi momento
  - In qualsiasi ordine
- I singoli dati sono distinti dal loro numero d'ordine
  - Primo, secondo, ..., ultimo dato
  - Ciascun dato può avere un valore diverso

Vettori

- Sequenza lineare di dati elementari
- Elementi tutti dello stesso tipo
- Numero di elementi fisso (N)
- Elementi identificati dal proprio indice
  - da 0 a N-1



...così è meglio!

```
int main(void)
{
   int dato[10] ;
   . . . . .
   for( i=0; i<10; i++)
      scanf("%d", &dato[i]) ;

   for( i=9; i>=0; i--)
      printf("%d\n", dato[i]) ;
}
```

Insieme o separati?

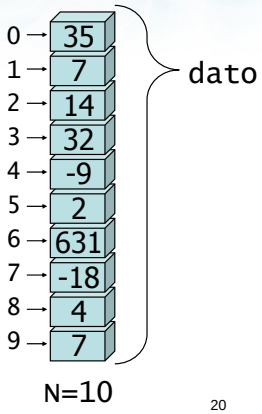
- I singoli dati vengono tenuti insieme in un'**unica variabile** di tipo vettore
  - `int dato[10]`
- Le operazioni (lettura, scrittura, elaborazione) avvengono però sempre **un dato alla volta**
  - `scanf("%d", &dato[i])`
  - `printf("%d\n", dato[i])`
- Ogni singolo dato è identificato da
  - Nome del vettore
  - Posizione all'interno del vettore

Strutture dati complesse

Caratteristiche dei vettori

Caratteristiche dei vettori

- Caratteristiche statiche
  - Nome
    - dato
  - Tipo di dato base
    - int
  - Dimensione totale
    - 10
- Caratteristiche dinamiche
  - Valori assunti dalle singole celle
    - 35, 7, 14, 32, ...



Esempio

- Leggere da tastiera 10 numeri e stamparli in ordine inverso
  - Tipo base: int
  - Dimensione: 10
  - Lettura delle celle da 0 a 9
  - Stampa delle celle da 9 a 0

Vincoli

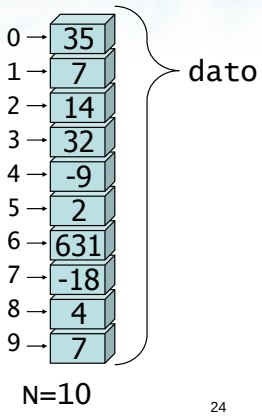
- Tutte le celle di un vettore avranno lo **stesso nome**
- Tutte le celle di un vettore devono avere lo **stesso tipo base**
- La dimensione del vettore è **fissa** e deve essere determinata al momento della sua definizione
  - La dimensione è sempre un numero intero
- Ogni cella ha **sempre** un valore
  - Impossibile avere celle "vuote"
  - Le celle non inizializzate contengono valori ignoti

Accesso alle celle

- Ciascuna cella è identificata dal proprio **indice**
- Gli indici sono sempre numeri **interi**
  - In C, gli indici partono da **0**
- Ogni cella è a tutti gli effetti una **variabile** il cui tipo è pari al tipo base del vettore
- Ogni cella, indipendentemente dalle altre
  - deve essere inizializzata
  - può essere letta/stampata
  - può essere aggiornata da istruzioni di assegnazione
  - può essere usata in espressioni aritmetiche

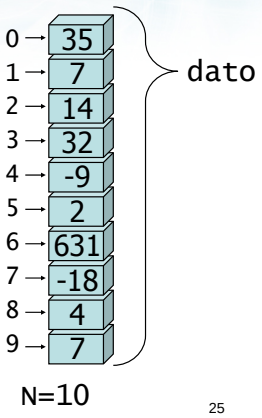
Errore frequente

- Non confondere mai l'indice con il contenuto
  - `dato[5] = 2`
  - `dato[9] == dato[1]`
  - `dato[i]>dato[j]`
  - `i>j`



Errore frequente

- **Non si può** effettuare alcuna operazione sul vettore
  - `dato = 0`
  - `printf("%d", dato)`
- Occorre operare sui singoli elementi
  - Solitamente all'interno di un ciclo `for`



N=10

Vettori

I vettori in C

I vettori in C

- Sintassi della definizione
- Definizione di costanti
- Operazioni di accesso

I vettori in C

Sintassi della definizione

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

- Stesse regole che valgono per i nomi delle variabili.
- I nomi dei vettori devono essere diversi dai nomi delle variabili.

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

- `int`
- `float`
- In futuro vedremo: `char`, `struct`

Definizione di vettori in C

```
int dato[10] ;
```

Tipo di dato base

Nome del vettore

Numero di elementi

- Intero positivo
- Costante nota a tempo di compilazione
- Impossibile cambiarla in seguito

Esempi

```
int dato[10] ;
float lati[8] ;
int numeriprimi[100] ;
int is_primo[1000] ;
```


Esempi

```
int dato[10] ;
float lati[8] ;
int numeriprimi[100] ;
int is_primo[1000] ;
```

0	1
1	2
2	3
3	5
4	7
5	11
6	13
7	17
8	19
9	23
10	29
11	31
12	37
...	...

Esempi

```
int dato[10] ;
float lati[8] ;
int numeriprimi[100] ;
int is_primo[1000] ;
```

0	0
1	1
2	1
3	1
4	0
5	0
6	0
7	1
8	0
9	0
10	0
11	1
12	0
...	...

Errore frequente

- Dichiarare un vettore usando una variabile anziché una costante

```
int N = 10 ;
int dato[N] ;
```

```
int dato[10] ;
```


Errore frequente

- Dichiarare un vettore usando una variabile non ancora inizializzata

```
int N ;
int dato[N] ;

. . .

scanf("%d",&N) ;
```

```
int dato[10] ;
```

Errore frequente

- Dichiarare un vettore usando il nome dell'indice

```
int i ;
int dato[i] ;

. . .
for(i=0; i<10; i++)
  scanf("%d",&dato[i]) ;
```

```
int dato[10] ;
```


I vettori in C

Definizione di costanti

Costanti

- La dimensione di un vettore deve essere specificata utilizzando una costante intera positiva
  - **Costante** = valore numerico già noto al momento della compilazione del programma

```
int dato[10] ;
```


Problemi

```
int i ;
int dato[10] ;

. . .

for(i=0; i<10; i++)
  scanf("%d", &dato[i]) ;

for(i=0; i<10; i++)
  printf("%d\n", dato[i]) ;
```

Devono essere tutte uguali. Chi lo garantisce?

Problemi

```
int i ;
int dato[10] ;

. . .

for(i=0; i<10; i++)
  scanf("%d", &dato[i]) ;

for(i=0; i<10; i++)
  printf("%d\n", dato[i]) ;
```

Se volessi lavorare con 20 dati dovrei modificare in tutti questi punti.

Soluzione

- Per risolvere i problemi visti si può ricorrere alle **costanti simboliche**
  - Associamo un nome simbolico ad una costante
  - Nel programma usiamo sempre il nome simbolico
  - Il compilatore si occuperà di sostituire, ad ogni occorrenza del nome, il valore numerico della costante

Costanti simboliche

- Costrutto `#define`
  - Metodo originario, in tutte le versioni del C
  - Usa una sintassi particolare, diversa da quella del C
  - Definisce costanti valide su tutto il file
  - Non specifica il tipo della costante
- Modificatore `const`
  - Metodo più moderno, nelle versioni recenti del C
  - Usa la stessa sintassi di definizione delle variabili
  - Specifica il tipo della costante

Costrutto `#define`

```
#define N 10

int main(void)
{
   int dato[N] ;
   . . .
}
```

Definizione della costante

Uso della costante

Particolarità (1/2)

- La definizione non è terminata dal segno ;
- Tra il nome della costante ed il suo valore vi è solo uno spazio (non il segno =)
- Le istruzioni `#define` devono essere una per riga
- Solitamente le `#define` vengono poste subito dopo le `#include`

```
#define N 10
```


Particolarità (2/2)

- Non è possibile avere una `#define` ed una variabile con lo stesso nome
- Per convenzione, le costanti sono indicate da nomi TUTTI_MAIUSCOLI

```
#define N 10
```


Esempio

```
#define MAX 10

int main(void)
{
  int i ;
  int dato[MAX] ;

  . . .

  for(i=0; i<MAX; i++)
    scanf("%d", &dato[i]) ;

  for(i=0; i<MAX; i++)
    printf("%d\n", dato[i]) ;
}
```


Modificatore `const`

```
int main(void)
{
   const int N = 10 ;

   int dato[N] ;

   . . .
}
```

Definizione della costante

Uso della costante

Sintassi

- Stessa sintassi per dichiarare una variabile
- Parola chiave `const`
- Valore della costante specificato dal segno =
- Definizione terminata da segno ;
- Necessario specificare il tipo (es. `int`)
- Il valore di `N` non si può più cambiare

```
const int N = 10 ;
```


Esempio

```
int main(void)
{
  const int MAX = 10 ;
  int i ;
  int dato[MAX] ;

  . . .

  for(i=0; i<MAX; i++)
    scanf("%d", &dato[i]) ;

  for(i=0; i<MAX; i++)
    printf("%d\n", dato[i]) ;
}
```


Suggerimenti

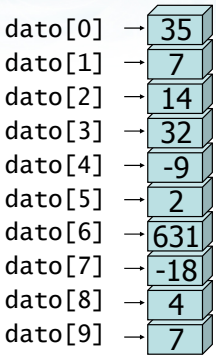
- Utilizzare sempre il modificatore `const`
- Permette maggiori controlli da parte del compilatore
- Gli eventuali messaggi d'errore sono più chiari
- Aggiungere sempre un commento per indicare lo scopo della variabile
- Utilizzare la convenzione di assegnare nomi TUTTI_MAIUSCOLI alle costanti

I vettori in C

Operazioni di accesso

Accesso ai valori di un vettore

- Ciascun elemento di un vettore è equivalente ad una singola variabile avente lo stesso tipo base del vettore
- È possibile accedere al contenuto di tale variabile utilizzando l'operatore di **indicizzazione**: `[ ]`




```
int dato[10] ;
```


Sintassi

```
nomevettore[ valoreindice ]
```

Come nella dichiarazione

Valore intero compreso tra 0 e (dimensione del vettore – 1)

Costante, variabile o espressione aritmetica con valore intero

Vincoli

- Il valore dell'indice deve essere compreso tra 0 e N-1. La responsabilità è del programmatore
- Se l'indice non è un numero intero, viene automaticamente troncato
- Il nome di un vettore può essere utilizzato solamente con l'operatore di indicizzazione

Uso di una cella di un vettore

- L'elemento di un vettore è utilizzabile come una qualsiasi variabile:
  - utilizzabile all'interno di un'espressione
    - `tot = tot + dato[i] ;`
  - utilizzabile in istruzioni di assegnazione
    - `dato[0] = 0 ;`
  - utilizzabile per stampare il valore
    - `printf("%d\n", dato[k]) ;`
  - utilizzabile per leggere un valore
    - `scanf("%d\n", &dato[k]) ;`

Esempi

- `if (dato[i]==0)`
  - se l'elemento contiene zero
- `if (dato[i]==dato[i+1])`
  - due elementi consecutivi uguali
- `dato[i] = dato[i] + 1 ;`
  - incrementa l'elemento `i`-esimo
- `dato[i] = dato[i+1] ;`
  - copia un dato dalla cella successiva

Vettori

Operazioni elementari sui vettori

Operazioni elementari sui vettori

- Definizioni
- Stampa di un vettore
- Lettura di un vettore
- Copia di un vettore
- Ricerca di un elemento
- Ricerca del massimo o minimo
- Vettori ad occupazione variabile

Operazioni elementari sui vettori

Definizioni

Definizioni (1/2)

vettori.c

```
const int N = 10 ;
   /* dimensioni dei vettori */

int v[N] ; /* vettore di N interi */

float r[N] ;
   /* vettore di N reali */

int i, j ;
   /* indici dei cicli */
```


Definizioni (2/2)

vettori.c

```
const int M = 100 ;
   /* dimensioni dei vettori */

int w[N] ; /* vettore di N interi */
int h[M] ; /* vettore di M interi */

int dato ;
   /* elemento da ricercare */
```


Operazioni elementari sui vettori

Stampa di un vettore

Stampa di un vettore

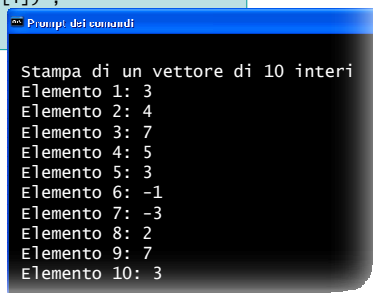
- Occorre stampare un elemento per volta, all'interno di un ciclo `for`
- Ricordare che
  - gli indici del vettore variano tra 0 e `N-1`
  - gli utenti solitamente contano tra 1 e `N`
  - `v[i]` è l'elemento `(i+1)`-esimo

Stampa vettore di interi


```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    printf("%d\n", v[i]) ;
}
```


Stampa vettore di interi

```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    printf("%d\n", v[i]) ;
}
```

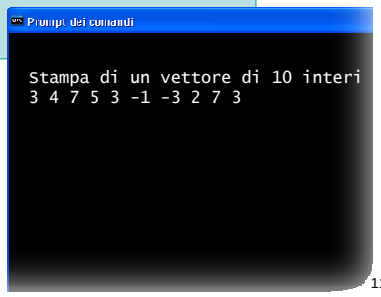


Stampa in linea


```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("%d ", v[i]) ;
}
printf("\n") ;
```


Stampa in linea

```
printf("Vettore di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("%d ", v[i]) ;
}
printf("\n") ;
```



Stampa vettore di reali


```
printf("Vettore di %d reali\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    printf("%f\n", r[i]) ;
}
```

Avvertenze

- Anche se il vettore è di reali, l'indice è sempre intero
- Separare sempre i vari elementi almeno con uno spazio (o un a capo)

Operazioni elementari sui vettori

Lettura di un vettore

Lettura di un vettore

- Occorre leggere un elemento per volta, all'interno di un ciclo `for`
- Ricordare l'operatore & nella `scanf`

Lettura vettore di interi

vettori.c

```
printf("Lettura di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    scanf("%d", &v[i]) ;
}
```


Lettura vettore di interi

```
printf("Lettura di %d interi\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    scanf("%d", &v[i]) ;
}
```

```
Prompt dei comandi

Lettura di un vettore di 10 interi
Elemento 1: 3
Elemento 2: 4
Elemento 3: 7
Elemento 4: 5
Elemento 5: 3
Elemento 6: -1
Elemento 7: -3
Elemento 8: 2
Elemento 9: 7
Elemento 10: 3
```


Lettura vettore di reali

vettori.c

```
printf("Lettura di %d reali\n", N) ;

for( i=0; i<N; i++ )
{
    printf("Elemento %d: ", i+1) ;
    scanf("%f", &r[i]) ;
}
```

Avvertenze

- Anche se il vettore è di reali, l'indice è sempre intero
- Fare precedere sempre ogni lettura da una `printf` esplicativa

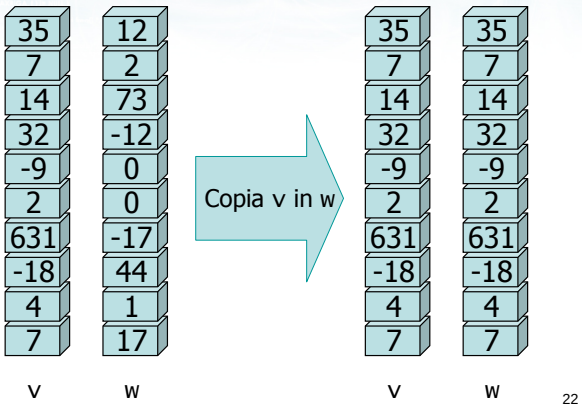
Operazioni elementari sui vettori

Copia di un vettore

Copia di un vettore

- Più correttamente, si tratta si copiare **il contenuto** di un vettore in un altro vettore
- Occorre copiare un elemento per volta dal vettore "sorgente" al vettore "destinazione", all'interno di un ciclo `for`
- I due vettori devono avere lo stesso numero di elementi, ed essere dello stesso tipo base

Copia di un vettore



Copia di un vettore

vettori.c

```
/* copia il contenuto di v[] in w[] */

for( i=0; i<N; i++ )
{
    w[i] = v[i] ;
}
```


Avvertenze

- Nonostante siano coinvolti **due** vettori, occore **un solo ciclo** `for`, e **un solo indice** per accedere agli elementi di entrambi i vettori
- Assolutamente non tentare di fare la copia in una sola istruzione!

```
w = v ;
w[] = v[] ;
w[N] = v[N] ;
w[1,N] = v[1,N] ;
```

Operazioni elementari sui vettori

Ricerca di un elemento

Ricerca di un elemento

- Dato un valore numerico, verificare
  - se **almeno uno** degli elementi del vettore è uguale al valore numerico
  - in caso affermativo, dire dove si trova
  - in caso negativo, dire che non esiste
- Si tratta di una classica istanza del problema di "ricerca di esistenza"

Ricerca di un elemento (1/3)

vettori.c

```
int dato ; /* dato da ricercare */
int trovato ; /* flag per ricerca */
int pos ;     /* posizione elemento */

...

printf("Elemento da ricercare? ");
scanf("%d", &dato) ;
```


Ricerca di un elemento (2/3)

vettori.c

```
trovato = 0 ;
pos = -1 ;

for( i=0 ; i<N ; i++ )
{
    if( v[i] == dato )
    {
        trovato = 1 ;
        pos = i ;
    }
}
```


Ricerca di un elemento (3/3)

vettori.c

```
if( trovato==1 )
{
    printf("Elemento trovato "
       "alla posizione %d\n", pos+1) ;
}
else
{
    printf("Elemento non trovato\n");
}
```


Varianti

- Altri tipi di ricerche
  - Contare quante volte è presente l'elemento cercato
  - Cercare se esiste almeno un elemento maggiore (o minore) del valore specificato
  - Cercare se esiste un elemento approssimativamente uguale a quello specificato
  - ...

Operazioni elementari sui vettori

Ricerca del massimo o minimo

Ricerca del massimo

- Dato un vettore (di interi o reali), determinare
  - quale sia l'elemento di valore massimo
  - quale sia la posizione in cui si trova tale elemento
- Conviene applicare la stessa tecnica per l'identificazione del massimo già vista in precedenza
  - Conviene inizializzare il max al valore del primo elemento

Ricerca del massimo (1/2)

vettori.c

```
float max ;  /* valore del massimo */
int posmax ; /* posizione del max */

...
max = r[0] ;
posmax = 0 ;

for( i=1 ; i<N ; i++ )
{
    if( r[i]>max )
    {
        max = r[i] ;
        posmax = i ;
    }
}
```


Ricerca del massimo (2/2)

vettori.c

```
printf("Il max vale %f e si ", max) ;
printf("trova in posiz. %d\n", posmax) ;
```


Operazioni elementari sui vettori

Vettori ad occupazione variabile

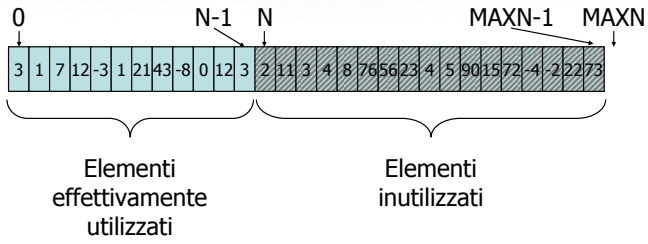
Occupazione variabile

- La principale limitazione dei vettori è la loro dimensione fissa, definita come costante al tempo di compilazione del programma
- Molto spesso non si conosce l'effettivo numero di elementi necessari fino a quando il programma non andrà in esecuzione
- Occorre identificare delle tecniche che ci permettano di lavorare con vettori di dimensione fissa, ma occupazione variabile

Tecnica adottata

- Dichiarare un vettore di dimensione sufficientemente ampia da contenere il massimo numero di elementi nel caso peggiore
  - Esempio: `MAXN`
- La parte iniziale del vettore sarà occupata dagli elementi, la parte finale rimarrà inutilizzata
- Dichiarare una variabile che tenga traccia dell'effettiva occupazione del vettore
  - Esempio: `N`

Tecnica adottata



Esempio

```
/* dimensione massima */
const int MAXN = 100 ;

int v[MAXN] ; /* vettore di dim. max. */

int N ; /* occupazione effettiva
           del vettore */

...

N = 0 ; /* inizialmente "vuoto" */
```


Regole di utilizzo

- All'inizio del programma si inizializza N al numero effettivo di elementi
  - Esempio: `scanf("%d", &N);`
- Verificare sempre che `N<=MAXN`
- Durante l'esecuzione, utilizzare sempre `N`, e mai `MAXN`
  - Esempio: `for(i=0; i<N; i++)`
- Gli elementi da `v[N]` a `v[MAXN-1]` vengono ignorati (costituiscono memoria "sprecata")

Crescita del vettore

- Un vettore ad occupazione variabile può facilmente crescere, aggiungendo elementi "in coda"

```
v[N] = nuovo_elemento ;
N++ ;
```


Esempio

- Acquisire da tastiera una serie di numeri reali, e memorizzarli in un vettore.
- La serie di numeri è terminata da un valore uguale a 0

- Il valore di `N` non è noto all'inizio del programma, ma viene aggiornato via via che si leggono gli elementi

Soluzione (1/3)

leggi0.c

```
const int MAXN = 100 ;

float v[MAXN] ;
float dato ;

int N ;
int i ;

printf("Inserisci gli elementi\n") ;
printf("(per terminare 0)\n") ;
```


Soluzione (2/3)

leggi0.c

```
N = 0 ; /* vettore inizialm. vuoto */

/* leggi il primo dato */
i = 0 ;
printf("Elemento %d: ", i+1) ;
scanf("%f", &dato) ;
i++ ;

/* aggiungi al vettore */
if( dato != 0.0 )
{
    v[N] = dato ;
    N++ ;
}
```


Soluzione (3/3)

leggi0.c

```
while(dato!= 0.0)
{
    /* leggi il dato successivo */
    printf("Elemento %d: ", i+1) ;
    scanf("%f", &dato) ;
    i++ ;

    /* aggiungi al vettore */
    if( dato != 0.0 )
    {
        v[N] = dato ;
        N++ ;
    }
}
```


Esercizio "Positivi e Negativi"

- Si realizzi un programma che legga da tastiera una sequenza di numeri interi (terminata da 0), e che successivamente stampi
  - tutti i numeri positivi presenti nella sequenza, nello stesso ordine
  - tutti i numeri negativi presenti nella sequenza, nello stesso ordine

Analisi

```
Prompt dei comandi
INSERISCI UNA SEQUENZA (0 per terminare)
Elemento: 3
Elemento: -4
Elemento: 1
Elemento: 2
Elemento: -3
Elemento: 0

Numeri positivi:
3 1 2
Numeri negativi:
-4 -3
```


Approccio risolutivo

- Definiamo 3 vettori ad occupazione variabile:
  - `seq`, di occupazione `N`, che memorizza la sequenza iniziale
  - `pos`, di occupazione `Np`, che memorizza i soli elementi positivi
  - `neg`, di occupazione `Nn`, che memorizza i soli elementi negativi
- Il programma inizialmente acquisirà da tastiera il vettore `seq`, in seguito trascriverà ciascun elemento nel vettore più opportuno

Soluzione (1/4)

posneg.c

```
const int MAXN = 100 ;

int seq[MAXN] ;
int pos[MAXN] ;
int neg[MAXN] ;

int N, Np, Nn ;

int i ;
int dato ;

/* vettori inizialmente vuoti */
N = 0 ;
Np = 0 ;
Nn = 0 ;
```


Soluzione (2/4)

posneg.c

```
/* LETTURA SEQUENZA INIZIALE */

/* vedi esempio precedente ... */
```


Soluzione (3/4)

posneg.c

```
for( i=0 ; i<N ; i++ )
{
    if(seq[i] > 0)
    {
        /* positivo => in pos[] */
        pos[Np] = seq[i] ;
        Np++ ;
    }
    else
    {
        /* negativo => in neg[] */
        neg[Nn] = seq[i] ;
        Nn++ ;
    }
}
```


Soluzione (4/4)

posneg.c

```
printf("Numeri positivi:\n") ;
for(i=0; i<Np; i++)
    printf("%d ", pos[i]) ;
printf("\n");

printf("Numeri negativi:\n") ;
for(i=0; i<Nn; i++)
    printf("%d ", neg[i]) ;
printf("\n");
```

Vettori

Esercizi guidati sui vettori

Esercizi guidati sui vettori

- Esercizio "Elementi comuni"
- Esercizio "Ricerca duplicati"
- Esercizio "Sottosequenza"
- Esercizio "Poligono"

Esercizi guidati sui vettori

Esercizio "Elementi comuni"

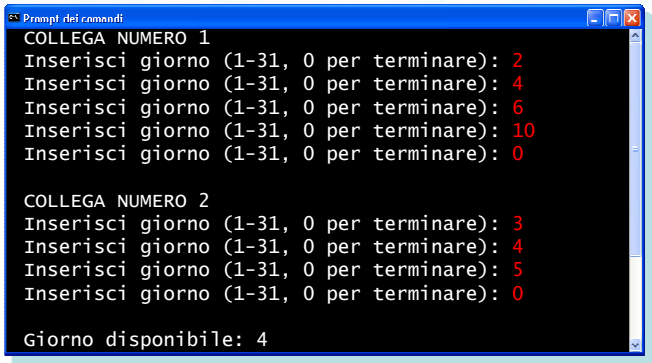
Esercizio "Elementi comuni" (1/2)

- Due colleghi intendono fissare una riunione, pertanto devono identificare dei giorni nei quali sono entrambi liberi da impegni. A tale scopo, essi realizzano un programma C che permetta a ciascuno di immettere le proprie disponibilità, e che identifichi i giorni nei quali entrambi sono liberi

Esercizio "Elementi comuni" (2/2)

- In particolare, in una prima fase il programma acquisisce le disponibilità dei due colleghi
  - Per ciascun collega il programma acquisisce un elenco di numeri interi (supponiamo compresi tra 1 e 31), che indicano i giorni del mese in cui essi sono disponibili. L'immissione dei dati termina inserendo 0.
- Nella seconda fase, il programma identificherà i giorni in cui entrambi i colleghi sono disponibili, e li stamperà a video

Analisi (1/3)



Analisi (2/3)

```
Prompt del comandi
COLLEGA NUMERO 1
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 6
Inserisci giorno (1-31, 0 per terminare): 10
Inserisci giorno (1-31, 0 per terminare): 0

COLLEGA NUMERO 2
Inserisci giorno (1-31, 0 per terminare): 3
Inserisci giorno (1-31, 0 per terminare): 5
Inserisci giorno (1-31, 0 per terminare): 7
Inserisci giorno (1-31, 0 per terminare): 0

Purtroppo non vi e' NESSUN giorno disponibile
```

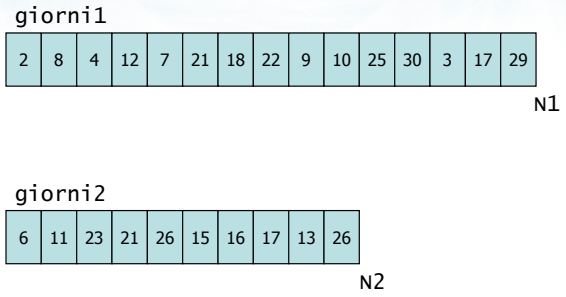

Analisi (3/3)

```
Prompt del comandi
COLLEGA NUMERO 1
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 6
Inserisci giorno (1-31, 0 per terminare): 0

COLLEGA NUMERO 2
Inserisci giorno (1-31, 0 per terminare): 2
Inserisci giorno (1-31, 0 per terminare): 3
Inserisci giorno (1-31, 0 per terminare): 4
Inserisci giorno (1-31, 0 per terminare): 0

Giorno disponibile: 2
Giorno disponibile: 4
```

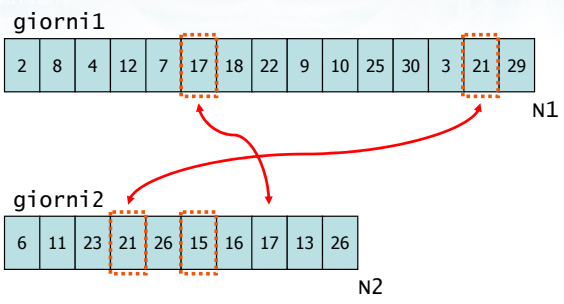

Algoritmo

- Acquisisci le disponibilità del collega 1
  - Vettore `giorni1[]` di N1 elementi
- Acquisisci le disponibilità del collega 2
  - Vettore `giorni2[]` di N2 elementi
- Verifica se vi sono elementi di `giorni1[]` che siano anche elementi di `giorni2[]`
  - Se sì, stampa tali elementi
  - Se no, stampa un messaggio

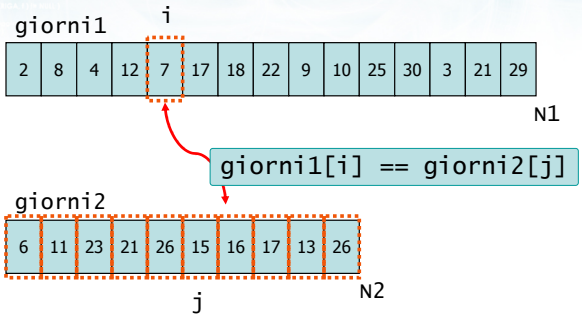
Ricerca elementi comuni



Ricerca elementi comuni



Ricerca elementi comuni



Soluzione (1/5)

riunione.c

```
const int MAXN = 100 ;

int N1, N2 ;
int giorni1[MAXN] ; /* giorni collega 1 */
int giorni2[MAXN] ; /* giorni collega 2 */

int giorno ;
int i, j ;
int trovato ;
  /* flag: giorni1[i] in giorni2[]? */
int fallito ;
  /* flag: trovato almeno un giorno? */
```


Soluzione (2/5)

riunione.c

```
/* DISPONIBILITA' COLLEGA 1 */
printf("COLLEGA NUMERO 1\n");
N1 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
    giorni1[N1] = giorno ;
    N1++ ;

    printf("Inserisci giorno (1-31): ");
    scanf("%d", &giorno) ;
}
printf("Collega 1 ha inserito %d giorni\n",
    N1);
```


Soluzione (3/5)

riunione.c

```
/* DISPONIBILITA' COLLEGA 2 */
printf("COLLEGA NUMERO 2\n");
N2 = 0 ;
printf("Inserisci giorno (1-31): ");
scanf("%d", &giorno) ;

while( giorno != 0 )
{
    giorni2[N2] = giorno ;
    N2++ ;

    printf("Inserisci giorno (1-31): ");
    scanf("%d", &giorno) ;
}
printf("Collega 2 ha inserito %d giorni\n",
    N2);
```


Soluzione (4/5)

riunione.c

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
  /* ...verifica se è disponibile
       il collega 2... */

  /* ...in caso affermativo stampalo */
  if( trovato == 1 )
  {
    printf("Giorno disponibile: %d\n",
      giorni1[i]) ;
    fallito = 0 ;
  }
}
```


Soluzione (4/5)

riunione.c

```
/* RICERCA DEGLI ELEMENTI COMUNI */
fallito = 1 ;
/* Per ogni giorno del collega 1... */
for( i=0 ; i<N1; i++ )
{
  /* ...verifica se è disponibile
       il collega 2... */

  /* ...in caso affermativo stampalo */
  if( tr
  {
    prin
      gi
    fall
  }
}
```

```
trovato = 0 ;
for( j=0; j<N2; j++ )
{
  if( giorni2[j] == giorni1[i] )
    trovato = 1 ;
}
```


Soluzione (5/5)

riunione.c

```
/* Se non ne ho trovato nessuno,
   stampo un messaggio */
if(fallito==1)
  printf("NESSUN giorno disponibile\n");
```

Esercizi guidati sui vettori

Esercizio "Ricerca duplicati"

Esercizio "Ricerca duplicati" (1/2)

- La società organizzatrice di un concerto vuole verificare che non ci siano biglietti falsi. A tale scopo, realizza un programma in linguaggio C che acquisisce i numeri di serie dei biglietti e verifica che non vi siano numeri duplicati

Esercizio "Ricerca duplicati" (2/2)

- In particolare, il programma acquisisce innanzitutto il numero di biglietti venduti, N, ed in seguito acquisisce i numeri di serie degli N biglietti
- Al termine dell'acquisizione, il programma stamperà "Tutto regolare" se non si sono riscontrati duplicati, altrimenti stamperà il numero di serie dei biglietti duplicati

Analisi (1/2)

```
Prompt dei comandi

RICERCA BIGLIETTI DUPLICATI

Numero totale di biglietti: 5
Numero di serie del biglietto 1: 1234
Numero di serie del biglietto 2: 4321
Numero di serie del biglietto 3: 1423
Numero di serie del biglietto 4: 1242
Numero di serie del biglietto 5: 3321
Tutto regolare
```


Analisi (2/2)

```
Prompt dei comandi

RICERCA BIGLIETTI DUPLICATI

Numero totale di biglietti: 5
Numero di serie del biglietto 1: 1234
Numero di serie del biglietto 2: 4321
Numero di serie del biglietto 3: 1234
Numero di serie del biglietto 4: 1242
Numero di serie del biglietto 5: 3321
ATTENZIONE: biglietto 1234 duplicato!
```


Algoritmo

- Acquisizione del valore di N
- Lettura dei numeri di serie
  - Utilizziamo un vettore: `int serie[MAXN]`
- **Ricerca dei duplicati**
- Stampa dei messaggi finali

Ricerca dei duplicati

- Prendi un elemento per volta
  - `elem = serie[i] ;`
- Cerca se **altri** elementi del vettore sono **uguali** a tale elemento
  - uguali ⇒ `(elem == serie[j])`
  - altri ⇒ `(i != j)`
- Si tratta di una ricerca di esistenza per ogni elemento considerato

Soluzione (1/5)

biglietti.c

```
const int MAXN = 100 ;

int N ; /* num tot biglietti */
int serie[MAXN] ; /* numeri serie */

int elem ;
int i, j ;

/* flag: trovato almeno un duplic.? */
int dupl ;

/* flag per ricerca di esistenza */
int trovato ;
```


Soluzione (2/5)

biglietti.c

```
printf("RICERCA DUPLICATI\n") ;
printf("\n");

/* ACQUISIZIONE VALORE DI N */
do{

  printf("Num tot di biglietti: ") ;
  scanf("%d", &N) ;
  if (N<2 || N>MAXN)
    printf("N=%d non valido\n", N) ;

} while(N<2 || N>MAXN) ;
```


Soluzione (3/5)

biglietti.c

```
/* LETTURA DEI NUMERI DI SERIE */
for( i=0 ; i<N ; i++ )
{
  printf("Numero serie biglietto %d: ",
         i+1) ;
  scanf("%d", &serie[i]) ;
}
```


Soluzione (4/5)

biglietti.c

```
/* RICERCA DEI DUPLICATI */
dupl = 0 ;
for( i=0 ; i<N ; i++ )
{
  /* verifica se serie[i] e' duplicato */
  elem = serie[i] ;

  /* => ricerca esistenza di elem
        all'interno di serie[] */

  if(trovato == 1)
  {
    printf("ATTENZIONE: %d duplicato\n",
           elem) ;
    dupl = 1 ;
  }
}
```


Soluzione (4/5)

biglietti.c

```
/* RICERCA DEI DUPLICATI */
dupl = 0 ;
for( i=0 ; i<N ; i++ )
{
  /* verifica se serie[i] e' duplicato */
  elem = serie[i] ;

  /* => ricerca esistenza di elem
        all'interno di serie[] */

  if(trovato
  {
    printf("A
           el
    dupl = 1
  }
}
```

```
trovato = 0 ;
for( j=0 ; j<N ; j++ )
{
  if( (i!=j) &&
      (elem == serie[j]) )
    trovato = 1 ;
}
```

Soluzione (5/5)

biglietti.c

```
/* STAMPA DEI MESSAGGI FINALI */
if(dupl==0)
    printf("Tutto regolare\n");
```


Esercizi guidati sui vettori

Esercizio "Sottosequenza"

Esercizio "Sottosequenza" (1/2)

- In un esercizio di telepatia, un sensitivo scommette di essere in grado di indovinare almeno 3 numeri consecutivi, in una sequenza di 100 numeri pensati da uno spettatore
- Per garantire l'oggettività dell'esperimento, viene realizzato un programma in C per la verifica dell'avvenuta telepatia
  - Per maggior generalità, il programma viene realizzato in modo da controllare sequenze di almeno K numeri consecutivi, all'interno di sequenze di N numeri.

Esercizio "Sottosequenza" (2/2)

- In particolare, il programma acquisisce innanzitutto la sequenza di N numeri pensati dallo spettatore. Si ipotizza che tali numeri siano interi positivi, compresi tra 1 e 10000
- In seguito, il programma acquisisce dal sensitivo una sequenza di K numeri
- Il programma verifica se esiste, nella sequenza di N numeri, una sottosequenza di K numeri esattamente uguale a quella inserita dal sensitivo
- I valori di N e K sono introdotti dall'utente all'inizio del programma

Analisi

```
Prompt dei comandi
Lunghezza della sequenza complessiva: 6
Lunghezza della sequenza da indovinare: 3
Inserire la sequenza complessiva
Elemento 1: 3
Elemento 2: 4
Elemento 3: 5
Elemento 4: 6
Elemento 5: 7
Elemento 6: 8
Inserire la sequenza da indovinare telepaticamente
Elemento 1: 5
Elemento 2: 6
Elemento 3: 7
Complimenti! Hai ottime capacita' telepatiche
```

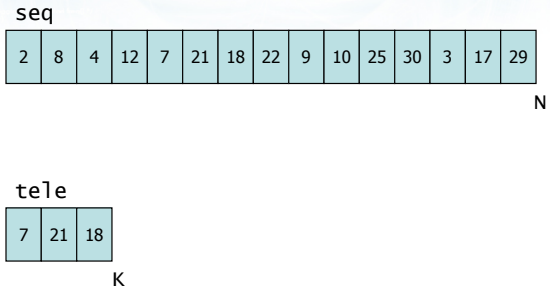

Algoritmo (1/2)

- Chiamiamo
  - `seq[]` la sequenza di N elementi
  - `tele[]` la sottosequenza di K elementi (K<N)

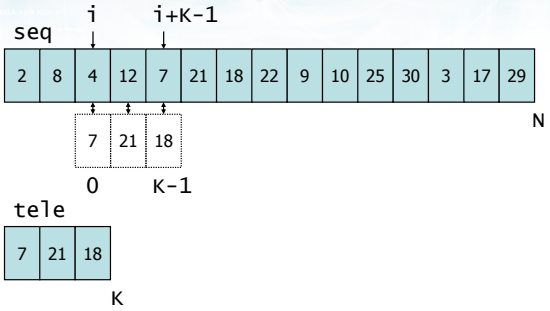
Algoritmo (2/2)

- Verifichiamo se
  - i primi K elementi di `seq[]` sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 1 a K sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 2 a K+1 sono uguali ai K elementi di `tele[]`
  - i K elementi di `seq[]` con indice da 3 a K+2 sono uguali ai K elementi di `tele[]`
  - ...

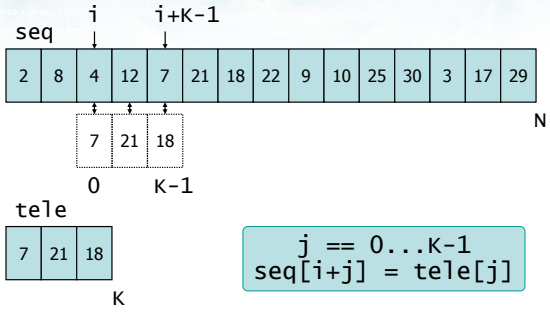
Sottosequenze



Sottosequenze



Sottosequenze




```
j == 0...K-1
seq[i+j] = tele[j]
```


Soluzione (1/4)

sensitivo.c

```
const int MAXN = 100 ;
const int MAXK = 10 ;

int N ; /* lunghezza seq. completa */
int K ; /* lunghezza sottosequenza */
int seq[MAXN] ; /* seq. completa */
int tele[MAXK] ; /* seq. telepatica */

int i, j ;
int trovato ; /* flag: ricorda se ha
   trovato una sottosequenza uguale */
int errore ; /* flag: verifica che
   TUTTI gli elementi della
   sottosequenza siano uguali */
```


Soluzione (2/4)

sensitivo.c

```
printf("ESPERIMENTO DI TELEPATIA\n") ;
printf("\n");

/* ACQUISIZIONE LUNGHEZZA SEQUENZE */
...leggi da tastiera i valori di N e K...

/* ACQUISIZIONE SEQUENZA COMPLESSIVA */
...leggi da tastiera il vettore seq[] di N elementi...

/* ACQUISIZIONE SEQ. DA INDOVINARE */
...leggi da tastiera il vettore tele[] di K elementi...
```

Soluzione (3/4)

sensitivo.c

```
trovato = 0 ;
/* considera tutti i possibili
   punti di partenza (i) */
for( i=0; i<N-K; i++ )
{
    /* verifica se seq[] nelle
       posizioni da (i) a (i+K-1) e'
       uguale a tele[] nelle posizioni
       da (0) a (K-1) */

    /* la sottosequenza era corretta? */
    if(errore==0)
        trovato=1 ;
}
```


Soluzione (3/4)

sensitivo.c

```
trovato = 0 ;
/* considera tutti i possibili
   punti di partenza (i) */
for( i=0; i<N-K; i++ )
{
    /* verifica se seq[] nelle
       posizioni da (i) a (i+K-1) e'
       uguale a tele[] nelle posizioni
       da (0) a (K-1) */

    /* la
    if(err
        tr
}
```

```
errore = 0 ;
for( j=0; j<K; j++ )
{
    if( seq[i+j] != tele[j] )
        errore = 1 ;
}
```


Soluzione (4/4)

sensitivo.c

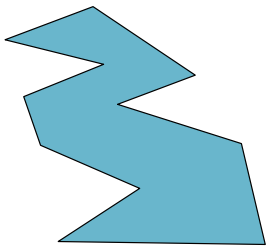
```
/* STAMPA RISULTATO DELLA VERIFICA */
if( trovato==1 )
  printf("Complimenti!\n");
else
  printf("Esperimento non riuscito\n");
```


Esercizi guidati sui vettori

Esercizio "Poligono"

Esercizio "Poligono" (1/2)

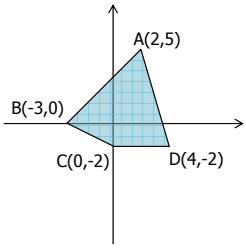
- Uno studente di geometria deve misurare il perimetro di una serie di poligoni irregolari, di cui conosce le coordinate cartesiane (x,y) dei vertici. Per far ciò realizza un programma in C



Esercizio "Poligono" (2/2)

- In particolare, il programma innanzitutto acquisisce il numero di vertici di cui è composto il poligono. Chiamiamo N tale numero
- In seguito, il programma acquisisce le N coppie (x,y) corrispondenti agli N vertici
- Il programma infine stampa la lunghezza complessiva del perimetro del poligono irregolare:
  - sommando delle lunghezze di ciascun lato
  - lato determinato dalle coordinate dei vertici

Analisi (1/2)



- AB = $\sqrt{(2-(-3))^2+(5-0)^2}$
- BC = $\sqrt{(-3-0)^2+(0-(-2))^2}$
- CD = $\sqrt{(0-4)^2+(-2-(-2))^2}$
- DA = $\sqrt{(4-2)^2+(-2-5)^2}$

- Perimetro =
  - = AB + BC + CD + DA

Analisi (2/2)

```
Prompt dei comandi
CALCOLO DEL PERIMETRO

Numero di vertici: 4
Inserire le coordinate dei vertici
Vertice 1:  x = 2
            y = 5
Vertice 2:  x = -3
            y = 0
Vertice 3:  x = 0
            y = -2
Vertice 4:  x = 4
            y = -2

Lunghezza del perimetro: 21.956729
```


Struttura dati

- In questo problema i dati da memorizzare non sono semplici numeri, ma coppie di numeri
- Possiamo utilizzare due vettori
  - Vettore x[], contenente le ascisse dei punti
  - Vettore y[], contenente le ordinate dei punti
- Esempio:
  - Punto A(2,5) ⇒ x[0]=2 ; y[0]=5 ;
- In tutte le elaborazioni, i due vettori verranno usati con **uguale valore dell'indice**
  - Vettori "paralleli"

Soluzione (1/3)

poligoni.c

```
const int MAXN = 10 ;

int N ; /* numero di vertici */

/* vettori "paralleli" */
float x[MAXN] ;
float y[MAXN] ;

int i ;

float lato ;
float perimetro ;
```


Soluzione (2/3)

poligoni.c

```
/* ACQUISIZIONE NUMERO VERTICI */
...leggi da tastiera il valore di N...

/* ACQUISIZIONE COORDINATE VERTICI */
printf("Inserire coordinate\n");
for( i=0; i<N; i++ )
{
    printf("Vertice %d:  x = ", i+1) ;
    scanf("%f", &x[i]) ;
    printf("           y = ") ;
    scanf("%f", &y[i]) ;
}
```


Soluzione (3/3)

poligoni.c

```
perimetro = 0 ;

for( i=0; i<N-1; i++ )
{
  /* (x[i],y[i])-(x[i+1],y[i+1]) */
  lato = sqrt(   (x[i]-x[i+1])*(x[i]-x[i+1])
               + (y[i]-y[i+1])*(y[i]-y[i+1]) ) ;
  perimetro = perimetro + lato ;
}

/* ultimo lato:
  (x[N-1],y[N-1])-(x[0],y[0]) */
  lato = sqrt(   (x[N-1]-x[0])*(x[N-1]-x[0])
               + (y[N-1]-y[0])*(y[N-1]-y[0]) ) ;
  perimetro = perimetro + lato ;
```

Vettori

Sommario

Argomenti trattati

- La struttura dati vettoriale
- Dichiarazione di vettori in C
- Accesso agli elementi del vettore
- Vettori con occupazione variabile

Tecniche di programmazione

- Lettura e scrittura di vettori
- Ricerca di elementi
- Ricerche di duplicati, di sottosequenze, di elementi comuni, ...
- Vettori paralleli

Vettori e cicli

- Per elaborare il contenuto di un vettore sono spesso necessari dei cicli
  - Operazioni semplici: scansione del vettore per stampa, lettura, ricerca, ...
  - Operazioni complesse: possono richiedere più cicli annidati

Errore frequente

- Non è detto che ad **ogni vettore** corrisponda **un ciclo**
- Controesempio: nella ricerca di elementi duplicati, vi è un solo vettore, ma due cicli annidati
- Controesempio: nel calcolo del perimetro, vi sono due vettori, ma un solo ciclo

Errore frequente

- Al vettore **non è associato alcun indice** particolare, per scandirne gli elementi
- Lo stesso vettore può essere usato con indici diversi
- Lo stesso indice può essere applicato a vettori diversi

Suggerimenti

- Tenere separate le operazioni di lettura/scrittura dalle elaborazioni vere e proprie
- Procedere per gradi, in modalità top-down, cercando di riconoscere ove possibile le strutture note
  - ricerca di un elemento
  - verifica di esistenza
  - verifica di universalità
- Non confondere i vari "flag" utilizzati in caso di cicli annidati

Avvertenze

- La combinazione di vettori e cicli crea un **fortissimo incremento nella complessità** dei programmi realizzabili
  - Questo è il punto più ripido nella curva di apprendimento
- Prima di procedere oltre, allenarsi con molti esercizi di programmazione

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Caratteri e stringhe

Caratteri e stringhe

- Dati testuali
- Il tipo char
- Vettori di caratteri
- Operazioni elementari sulle stringhe
- Funzioni di libreria
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 1 e 5
  - Cabodi, Quer, Sonza Reorda: capitolo 5
  - Dietel & Dietel: capitolo 8
- Dispense
  - Scheda: "Caratteri e stringhe in C"

Caratteri e stringhe

Dati testuali

Dati testuali

- Tipi di dato testuali
- Caratteri
- Stringhe

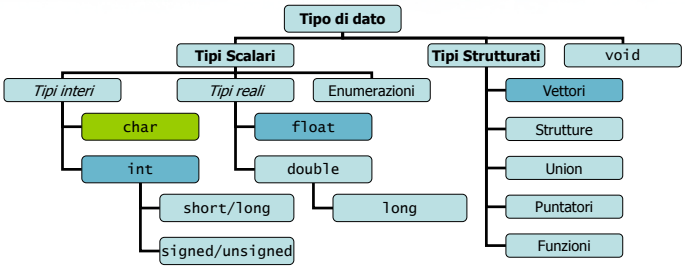
Dati testuali

Tipi di dato testuali

Tipi di dato testuali

- I programmi visti finora erano in grado di elaborare esclusivamente informazioni numeriche
  - Numeri interi (`int`), numeri reali (`float`)
  - Variabili singole o vettori
- In molti casi è necessario elaborare informazioni di tipo testuale
  - `Vuoi continuare (s/n)?`
  - Conta le parole di un testo scritto
  - Gestisci una rubrica di nomi e numeri di telefono
  - …

Il sistema dei tipi C



Rappresentazione dei testi

- Il calcolatore è in grado di rappresentare i caratteri alfabetici, numerici ed i simboli speciali di punteggiatura
- Ad ogni diverso carattere viene assegnato, **convenzionalmente**, un codice numerico corrispondente
- Il programma in C lavora sempre con i codici numerici
- Le funzioni di input/output sono in grado di accettare e mostrare i caratteri corrispondenti

Codice ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.lookuptables.com

Caratteri e stringhe

- Il codice ASCII permette di rappresentare un singolo **carattere**

y	7	w	!	%

- Nelle applicazioni pratiche spesso serve rappresentare sequenze di caratteri: **stringhe**

F	u	l	v	i	o

0	6	A	Z	N

0	1	1	-	5	6	4	6	3	3	2

Dualità caratteri - numeri

- Ogni carattere è rappresentato dal suo codice ASCII

y	7	w	!	%
121	55	87	33	37

- Ogni stringa è rappresentata dai codici ASCII dei caratteri di cui è composta

F	u	l	v	i	o
70	117	108	118	105	111

0	6	A	Z	N
48	54	65	90	78

0	1	1	-	5	6	4	6	3	3	2
48	49	49	45	53	54	52	54	51	51	50

Dati testuali

Caratteri

Caratteri in C

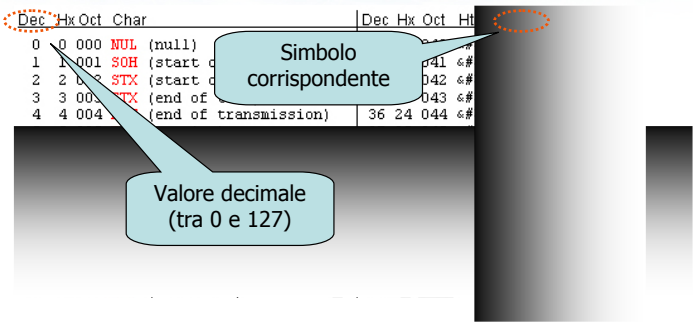
- Ogni carattere viene rappresentato dal proprio codice ASCII
- Sono sufficienti 7 bit per rappresentare ciascun carattere
  - Il C usa variabili di 8 bit (1 byte)
- Non sono previste le lettere accentate né altri simboli diacritici
  - Richiedono estensioni speciali e librerie specifiche

Codice ASCII

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

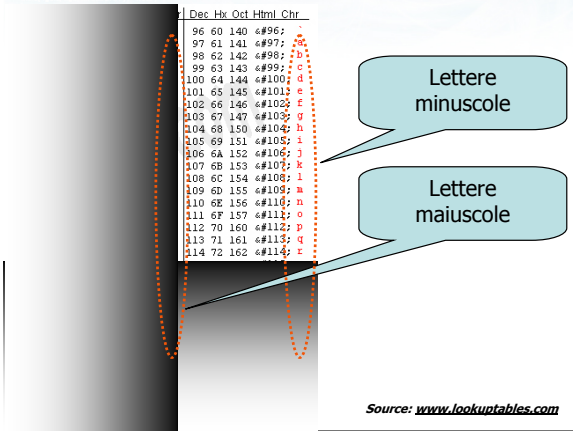
Source: www.lookuptables.com

Codice ASCII



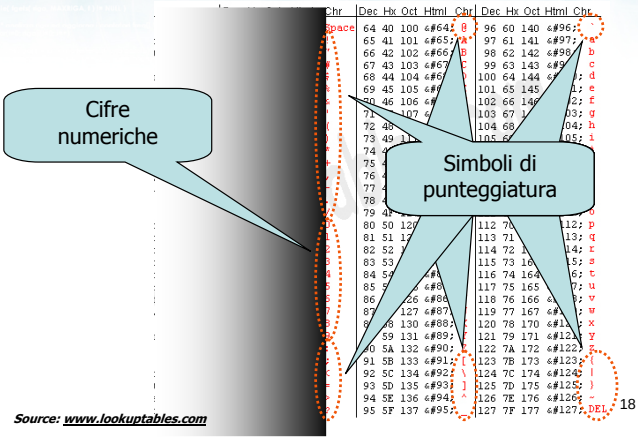
Source: www.lookuptables.com

Codice ASCII



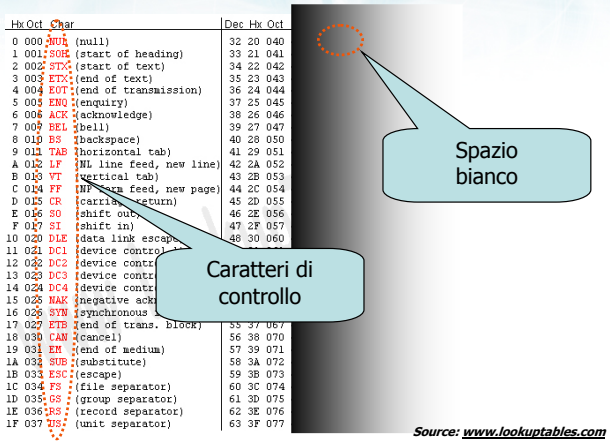
Source: www.lookuptables.com

Codice ASCII



Source: www.lookuptables.com

Codice ASCII



Source: www.lookuptables.com

Caratteristiche del codice ASCII

- Le lettere maiuscole sono tutte consecutive, ed in ordine alfabetico
- Le lettere minuscole sono tutte consecutive, ed in ordine alfabetico
- Le lettere maiuscole vengono "prima" delle minuscole
- Le cifre numeriche sono tutte consecutive, in ordine dallo 0 al 9
- I simboli di punteggiatura sono sparsi

Caratteri di controllo

- Caratteri speciali, non visualizzabili
- Rappresentano comandi di stampa, e non simboli da stampare
- Esempi:
  - 7 – BEL: emetti un "bip"
  - 8 – BS: cancella l'ultimo carattere
  - 10 – LF: avanza di una riga
  - 13 – CR: torna alla prima colonna
  - 27 – ESC: tasto "Esc"
- Per alcuni esiste una sequenza di escape in C: \n

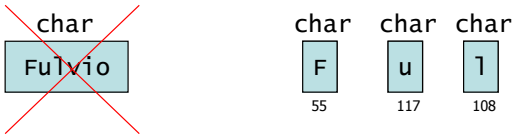
Errore frequente

- Non confondere il carattere ASCII che rappresenta una cifra numerica con il valore decimale associato a tale cifra

- Per chiarezza useremo gli apici per indicare i caratteri

Errore frequente

- Pensare che un singolo carattere possa memorizzare più simboli



Dati testuali

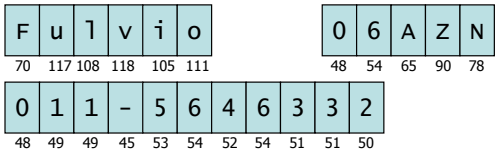
Stringhe

Stringhe

- Una **stringa** è una struttura dati capace di memorizzare **sequenze di caratteri**
- In C non esiste un tipo di dato specifico
- Si usano **vettori di caratteri**
- La lunghezza di una stringa è tipicamente variabile durante l'esecuzione del programma
  - Occorrerà gestire l'occupazione variabile dei vettori di caratteri

Caratteristiche delle stringhe

- Memorizzate come singoli caratteri, ma il loro significato è dato dall'intera sequenza di caratteri
- Lunghezza variabile
- Mix di lettere/cifre/punteggiatura/spazi
- Solitamente non contengono caratteri di controllo



Manipolazione delle stringhe

- Occorre trattare l'insieme di caratteri memorizzato nel vettore come un'unica "variabile"
- Ogni operazione elementare sulle stringhe coinvolgerà tipicamente dei cicli che scandiscono il vettore
- Molte funzioni di libreria sono già disponibili per compiere le operazioni più frequenti ed utili

Errore frequente

- Non confondere una stringa composta da cifre numeriche con il valore decimale associato a tale sequenza

Caratteri e stringhe

Il tipo char

Il tipo char

- Variabili char
- Input/output di char
- Operazioni sui char
- Esercizio "Quadrati di lettere"

Il tipo char

Variabili char

Variabili char

- I caratteri in C si memorizzano in variabili di tipo char

```
char lettera ;
```

- Le costanti di tipo char si indicano ponendo il simbolo corrispondente tra singoli apici

```
lettera = 'Q' ;
```


Apici

- Non confondere i 3 tipi di apici presenti sulla tastiera:

Apice singolo (apostrofo)	'	In C, delimita singoli caratteri
Apice doppio (virgolette)	"	In C, delimita stringhe di caratteri
Apice rovesciato (accento grave)	`	Non utilizzato in C

Dualità dei char

- Sintatticamente, i char non sono altro che degli int di piccola dimensione
  - Ogni operazione possibile su un int, è anche possibile su un char
  - Ovviamente solo alcune di tali operazioni avranno senso sull'interpretazione testuale (ASCII) del valore numerico

Esempi

```
int i ;
char c ;

c = 'A' ;
```


Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
```


Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
```


Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
```


Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
```


Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
if (c == 'Z') ...
```

Esempi

```
int i ;
char c ;

c = 'A' ;
c = 65 ; /* equivalente! */
i = c ; /* i sarà 65 */
c = c + 1 ; /* c sarà 66 = 'B' */
c = c * 2 ; /* non ha senso... */
if (c == 'Z') ...
for( c='A'; c<='Z'; c++) ...
```


Caratteri speciali

- Per alcuni caratteri di controllo il linguaggio C definisce una particolare **sequenza di escape** per poterli rappresentare

C	ASCII	Significato
'\n'	LF – 10	A capo
'\t'	TAB – 9	Tabulazione
'\b'	BS – 8	Backspace – cancella ultimo car.
'\a'	BEL – 7	Emette un "bip"
'\r'	CR – 13	Torna alla prima colonna

Punteggiatura speciale in C

- Alcuni caratteri hanno un significato particolare dentro gli apici. Per poterli inserire come carattere esistono apposite sequenze di escape

C	ASCII	Significato
'\\'	\	Immette un backslash
'\''	'	Immette un apice singolo
'\"'	"	Immette un apice doppio
'*ooo*'	*ooo*	Immette in carattere ASCII con codice (ottale) *ooo*
'\x*hh*'	*hh*	Immette in carattere ASCII con codice (esadecimale) *hh*

Il tipo char

Input/output di char

Input/output di char

- Esistono due insiemi di funzioni che permettono di leggere e stampare variabili di tipo char:
  - Le funzioni `printf/scanf`, usando lo specificatore di formato "%c"
  - Le funzioni `putchar` e `getchar`
- In entrambi i casi è sufficiente includere la libreria `<stdio.h>`
- È possibile mescolare liberamente le due famiglie di funzioni

Stampa di caratteri

```
char ch ;

printf("%c", ch) ;
```

```
char ch ;

putchar(ch) ;
```

Lettura di caratteri

```
char ch ;

scanf("%c", &ch) ;
```

```
char ch ;

ch = getchar() ;
```


Suggerimenti (1/2)

- La funzione `printf` è più comoda quando occorre stampare altri caratteri insieme a quello desiderato
  - `printf("La risposta e': %c\n", ch) ;`
  - `printf("Codice: %c%d\n", ch, num ) ;`
- La funzione `putchar` è più comoda quando occorre stampare semplicemente il carattere
  - `for(ch='a'; ch<='z'; ch++)`
    `putchar(ch) ;`

Suggerimenti (2/2)

- La funzione `getchar` è generalmente più comoda in tutti i casi
  - `printf("Vuoi continuare (s/n)? ");`
    `ch = getchar() ;`

Bufferizzazione dell'input-output

- Tutte le funzioni della libreria `<stdio.h>` gestiscono l'input-output in modo **bufferizzato**
  - Per maggior efficienza, i caratteri non vengono trasferiti immediatamente dal programma al terminale (o viceversa), ma solo a gruppi
  - È quindi possibile che dopo una `putchar`, il carattere **non** compaia **immediatamente** sullo schermo
  - Analogamente, la `getchar` **non** restituisce il carattere finché l'utente non preme **invio**

Conseguenza pratica

```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: _
```

Il programma stampa l'invito ad inserire un dato

Conseguenza pratica

```
char ch,ch2 ;

printf("Dato: ");

ch = getchar() ;

ch2 = getchar() ;
```

```
Dato: _
```

`getchar` blocca il programma in attesa del dato

Conseguenza pratica

```
char ch,ch2 ;
printf("Dato: ");
ch = getchar() ;
ch2 = getchar() ;
```

```
Dato: a_
```

L'utente immette 'a', il programma non lo riceve

Conseguenza pratica

```
char ch,ch2 ;
printf("Dato: ");
ch = getchar() ;
ch2 = getchar() ;
```

```
Dato: a
_
```

L'utente immette Invio, il programma prosegue

Conseguenza pratica

```
char ch,ch2 ;
printf("Dato: ");
ch = getchar() ;
ch2 = getchar() ;
```

```
Dato: a
_
```

Ora ch='a', il programma fa un'altra getchar()

Conseguenza pratica

```
char ch,ch2 ;
printf("Dato: ");
ch = getchar() ;
ch2 = getchar() ;
```

```
Dato: a
_
```

Il programma **non** si blocca in attesa dell'utente

Conseguenza pratica

```
char ch,ch2 ;
printf("Dato: ");
ch = getchar() ;
ch2 = getchar() ;
```

```
Dato: a
_
```

C'era già un carattere pronto: Invio! ch2='\n'

Consigli pratici

- Ricordare che l'utente deve sempre premere Invio, anche se il programma richiede un singolo carattere
- Ricordare che, se l'utente inserisce più di un carattere, questi verranno restituiti uno ad uno nelle `getchar` successive
- Ricordare che l'Invio viene letto come tutti gli altri caratteri

Soluzione proposta

```
char ch, temp ;

printf("Dato: ");

ch = getchar() ; /* leggi il dato */

/* elimina eventuali caratteri successivi
ed il \n che sicuramente ci sarà */
do {
    temp = getchar() ;
} while (temp != '\n') ;
```


Soluzione proposta

```
char ch, temp ;

printf("Dato: ");

ch = getchar() ; /

/* elimina eventua
ed il \n che sicur
do {
    temp = getchar() ;
} while (temp != '\n') ;
```

```
/* forma più compatta */
while ( getchar()!='\n' )
  /*niente*/ ;
```


Il tipo char

Operazioni sui char

Operazioni sui char

- Le operazioni lecite sui `char` derivano direttamente dalla combinazione tra
  - Le operazioni permesse sugli `int`
  - La disposizione dei caratteri nella tabella ASCII
  - Le convenzioni lessicali della nostra lingua scritta

Conversione ASCII-Carattere

- Una variabile di tipo `char` è allo stesso tempo
  - Il valore numerico del codice ASCII del carattere
    - `printf("%d", ch) ;`
    - `i = ch ;`
    - `ch = j ;`
    - `ch = 48 ;`
  - Il simbolo corrispondente al carattere ASCII
    - `printf("%c", ch) ;`
    - `putchar(ch) ;`
    - `ch = 'Z' ;`
    - `ch = '4' ;`

Esempio (1/3)

char-int.c

```
int i ;
char ch ;

printf("Immetti codice ASCII (32-126): ");

scanf("%d", &i) ;

ch = i ;

printf("Il carattere %c ha codice %d\n",
       ch, i) ;
```

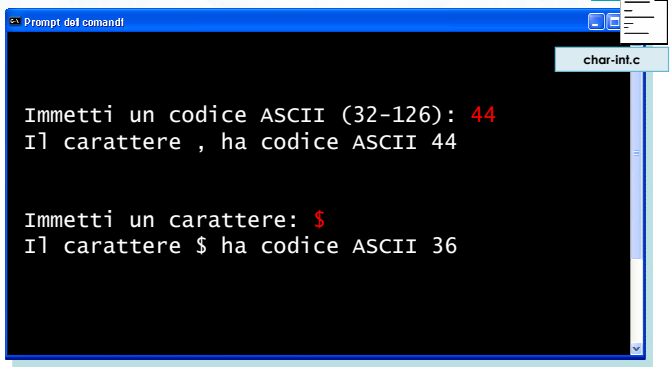
Esempio (2/3)

char-int.c

```
printf("Immetti un carattere: ") ;
ch = getchar() ;

while( getchar() != '\n' )
    /**/ ;

i = ch ;

printf("Il carattere %c ha codice %d\n",
       ch, i) ;
```


Esempio (3/3)

char-int.c



Scansione dell'alfabeto

- È possibile generare tutte le lettere dell'alfabeto, in ordine, grazie al fatto che nella tabella ASCII esse compaiono consecutive e ordinate

```
char ch ;

for( ch = 'A' ; ch <= 'Z' ; ch++ )
    putchar(ch) ;

putchar('\n') ;
```


Verifica se è una lettera

- Per sapere se un carattere è alfabetico, è sufficiente verificare se cade nell'intervallo delle lettere (maiuscole o minuscole)

```
if( ch>='A' && ch<='Z' )
    printf("%c lettera maiuscola\n", ch) ;

if( ch>='a' && ch<='z' )
    printf("%c lettera minuscola\n", ch) ;

if( (ch>='A' && ch<='Z') ||
    (ch>='a' && ch<='z') )
    printf("%c lettera\n", ch) ;
```

Verifica se è una cifra

- Per sapere se un carattere è numerico ('0'-'9'), è sufficiente verificare se cade nell'intervallo delle cifre

```
if( ch>='0' && ch<='9' )
    printf("%c cifra numerica\n", ch) ;
```


Valore di una cifra

- Conoscere il valore decimale di un carattere numerico ('0'-'9'), è sufficiente calcolare la "distanza" dalla cifra '0'

```
if( ch>='0' && ch<='9' )
{
    printf("%c cifra numerica\n", ch) ;
    val = ch - '0' ;
    printf("Il suo valore e': %d", val ) ;
}
```

Da minuscolo a maiuscolo (1/2)

- I codici ASCII delle lettere maiuscole e delle minuscole differiscono solamente per una costante:
  - `'A'` = 65 ... `'Z'` = 90
  - `'a'` = 97 ... `'z'` = 122
- Se `ch` è una lettera minuscola
  - `ch - 'a'` è la sua posizione nell'alfabeto
  - `( ch - 'a' ) + 'A'` è la corrispondente lettera maiuscola

Da minuscolo a maiuscolo (2/2)

- Possiamo interpretare la conversione come una traslazione della quantità (`'A'-'a'`)

```
if( ch>='a' && ch<='z' )
{
    printf("%c lettera minuscola\n", ch) ;
    ch2 = ch + ('A'-'a') ;
    printf(La maiuscola e': %c\n", ch2) ;
}
```


Confronto alfabetico

- Se due caratteri sono **entrambi maiuscoli** (o entrambi minuscoli) è sufficiente confrontare i rispettivi codici ASCII

```
if( ch < ch2 )
  printf("%c viene prima di %c", ch, ch2) ;
else
  printf("%c viene prima di %c", ch2, ch) ;
```


Il tipo char

Esercizio "Quadrati di lettere"

Esercizio "Quadrati di lettere"

- Si scriva un programma in linguaggio C che stampi su video una serie di quadrati, composti dalle successive lettere dell'alfabeto, di dimensioni sempre crescenti:
  - Un quadrato 1x1 di lettere A
  - Un quadrato 2x2 di lettere B
  - Un quadrato 3x3 di lettere C
  - ...eccetera

Analisi

```
Prompt dei comandi
Quanti quadrati vuoi stampare? 4
A

BB
BB

CCC
CCC
CCC

DDDD
DDDD
DDDD
DDDD
```

Soluzione (1/2)

quadrati.c

```
int i, N ;
int riga, col ;
char ch ;

printf("Quanti quadrati? ") ;
scanf("%d", &N) ;

while(N<1 || N>26)
{
  printf("Deve essere tra 1 e 26\n");
  printf("Quanti quadrati? ") ;
  scanf("%d", &N) ;
}
```


Soluzione (2/2)

quadrati.c

```
for( i=0; i<N; i++ )
{
    /* stampa un quadrato
       di dimensione (i+1) */

    ch = i + 'A' ;

    for(riga=0; riga<i+1; riga++)
    {
        for(col=0; col<i+1; col++)
            putchar(ch);
        putchar('\n') ;
    }
    putchar('\n') ;
}
```

Caratteri e stringhe

Vettori di caratteri

Vettori di caratteri

- Il tipo stringa
- Terminatore nullo
- Input/output di stringhe

Vettori di caratteri

Il tipo stringa

Stringhe in C

- Nel linguaggio C non è supportato esplicitamente alcun tipo di dato "stringa"
- Le informazioni di tipo stringa vengono memorizzate ed elaborate ricorrendo a semplici **vettori di caratteri**

```
char saluto[10] ;
```

B	u	o	n	g	i	o	r	n	o

Esempio

- Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max 20 caratteri), e stampi a video un saluto per l'utente stesso

```
Prompt dei comandi
Come ti chiami? Fulvio

Buongiorno, Fulvio!
```


Soluzione (1/3)

saluti.c

```
const int MAX = 20 ;
char nome[MAX] ;
int N ;
char ch ;
int i ;

printf("Come ti chiami? ") ;

N = 0 ;
```

Soluzione (2/3)

saluti.c

```
ch = getchar() ;

while( ch != '\n' && N<MAX )
{
    nome[N] = ch ;
    N++ ;
    ch = getchar() ;
}
```


Soluzione (3/3)

saluti.c

```
printf("Buongiorno, ") ;

for(i=0; i<N; i++)
    putchar( nome[i] ) ;

printf("!\n") ;
```


Commenti (1/2)

- Qualsiasi operazione sulle stringhe si può realizzare agendo opportunamente su vettori di caratteri, gestiti con occupazione variabile
- Così facendo, però vi sono alcuni svantaggi
  - Per ogni vettore di caratteri, occorre definire un'opportuna variabile che ne indichi la lunghezza
  - Ogni operazione, anche elementare, richiede l'uso di cicli `for`/`while`

Commenti (2/2)

- Alcune convenzioni ci possono aiutare
  - Gestire in modo standard i vettori di caratteri usati per memorizzare stringhe
  - Apprendere le tecniche solitamente utilizzate per compiere le operazioni più frequenti
- Molte funzioni di libreria seguono queste convenzioni
  - Conoscere le funzioni di libreria ed utilizzarle per accelerare la scrittura del programma

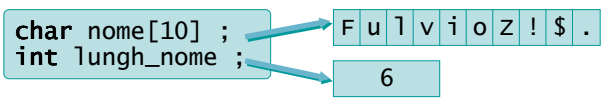
Vettori di caratteri

Terminatore nullo

Lunghezza di una stringa

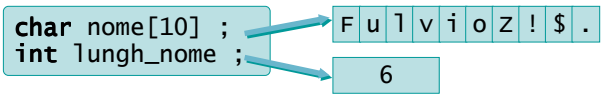
- Vi sono due tecniche per determinare la lunghezza di una stringa
  1. utilizzare una variabile intera che memorizzi il numero di caratteri validi

```
char nome[10] ;
int lungh_nome ;
```

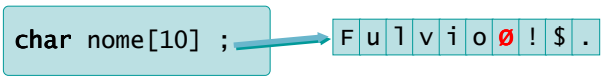


Lunghezza di una stringa

- Vi sono due tecniche per determinare la lunghezza di una stringa
  1. utilizzare una variabile intera che memorizzi il numero di caratteri validi



  2. utilizzare un carattere "speciale", con funzione di **terminatore**, dopo l'ultimo carattere valido

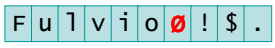


Carattere terminatore

- Il carattere "terminatore" deve avere le seguenti caratteristiche
  - Fare parte della tabella dei codici ASCII
    - Deve essere rappresentabile in un `char`
  - Non comparire mai nelle stringhe utilizzate dal programma
    - Non deve confondersi con i caratteri "normali"
- Inoltre il vettore di caratteri deve avere una posizione libera in più, per memorizzare il terminatore stesso

Terminatore standard in C

- Per convenzione, in C si sceglie che tutte le stringhe siano rappresentate mediante un carattere terminatore
- Il terminatore corrisponde al carattere di codice ASCII pari a zero
  - `nome[6] = 0 ;`
  - `nome[6] = '\0' ;`



Vantaggi

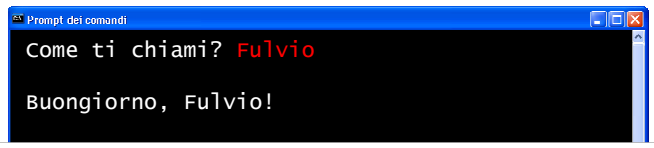
- Non è necessaria un'ulteriore variabile intera per ciascuna stringa
- L'informazione sulla lunghezza della stringa è interna al vettore stesso
- Tutte le funzioni della libreria standard C rispettano questa convenzione
  - Si aspettano che la stringa sia terminata
  - Restituiscono sempre stringhe terminate

Svantaggi

- Necessario 1 byte in più
  - Per una stringa di N caratteri, serve un vettore di N+1 elementi
- Necessario ricordare di aggiungere sempre il terminatore
- Impossibile rappresentare stringhe contenenti il carattere ASCII 0

Esempio

- Si realizzi un programma in linguaggio C che acquisisca da tastiera il nome dell'utente (una stringa di max 20 caratteri), e stampi a video un saluto per l'utente stesso



Soluzione (1/3)

saluti0.c

```
const int MAX = 20 ;
char nome[MAX+1] ;
char ch ;
int i ;

printf("Come ti chiami? ") ;

i = 0 ;
```


Soluzione (2/3)

saluti0.c

```
i = 0 ;

ch = getchar() ;

while( ch != '\n' && i<MAX )
{
    nome[i] = ch ;
    i++ ;
    ch = getchar() ;
}
/* aggiunge terminatore nullo */
nome[i] = '\0' ;
```


Soluzione (3/3)

saluti0.c

```
printf("Buongiorno, ") ;

for(i=0; nome[i]!='\0'; i++)
    putchar( nome[i] ) ;

printf("!\n") ;
```


Vettori di caratteri

Input/output di stringhe

I/O di stringhe

- Diamo per scontato di utilizzare la convenzione del terminatore nullo
- Si possono utilizzare
  - Funzioni di lettura e scrittura carattere per carattere
    - Come nell'esercizio precedente
  - Funzioni di lettura e scrittura di stringhe intere
    - `scanf` e `printf`
    - `gets` e `puts`

Lettura di stringhe con `scanf`

- Utilizzare la funzione `scanf` con lo specificatore di formato `"%s"`
- La variabile da leggere deve essere il nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
  - Non utilizzare la &
- Legge ciò che viene immesso da tastiera, fino al primo spazio o fine linea (esclusi)
  - Non adatta a leggere nomi composti (es. `"Pier Paolo"`)

Esempio

```
const int MAX = 20 ;
char nome[MAX+1] ;

printf("Come ti chiami? ") ;

scanf("%s", nome) ;
```


Lettura di stringhe con gets

- La funzione `gets` è pensata appositamente per acquisire una stringa
- Accetta un parametro, che corrisponde al nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
- Legge ciò che viene immesso da tastiera, fino al fine linea (escluso), e compresi eventuali spazi
  - Possibile leggere nomi composti (es. `"Pier Paolo"`)

Esempio

```
const int MAX = 20 ;
char nome[MAX+1] ;

printf("Come ti chiami? ") ;

gets(nome) ;
```


Scrittura di stringhe con printf

- Utilizzare la funzione `printf` con lo specificatore di formato `"%s"`
- La variabile da stampare deve essere il nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
- È possibile combinare la stringa con altre variabili nella stessa istruzione

Esempio

```
printf("Buongiorno, ") ;
printf("%s", nome) ;
printf("!\n") ;


printf("Buongiorno, %s!\n", nome) ;
```


Scrittura di stringhe con puts

- La funzione `puts` è pensata appositamente per stampare una stringa
- La variabile da stampare deve essere il nome di un vettore di caratteri
  - Non utilizzare le parentesi quadre
- Va a capo automaticamente
  - Non è possibile stampare altre informazioni sulla stessa riga

Esempio

```
printf("Buongiorno, ") ;
puts(nome) ;

/* No!! printf("!\n") ; */
```


Conclusione

- Utilizzare sempre la convenzione del terminatore nullo
- Ricordare di allocare un elemento in più nei vettori di caratteri
- Utilizzare quando possibile le funzioni di libreria predefinite
  - In lettura, prediligere `gets`
  - In scrittura
    - `printf` è indicata per messaggi composti
    - `puts` è più semplice se si ha un dato per riga

Caratteri e stringhe

Operazioni elementari sulle stringhe

Operazioni elementari sulle stringhe

- Lunghezza
- Copia di stringhe
- Concatenazione di stringhe
- Confronto di stringhe
- Ricerca di sotto-stringhe
- Ricerca di parole

Operazioni elementari sulle stringhe

Lunghezza

Lunghezza di una stringa

- La lunghezza di una stringa si può determinare ricercando la posizione del terminatore nullo

```
char s[MAX+1] ;
int lun ;
```

s	S	a	l	v	e	ø	3	r	W	t
	0	1	2	3	4	5				

Calcolo della lunghezza

```
const int MAX = 20 ;
char s[MAX+1] ;
int lun ;
int i ;

... /* lettura stringa */

for( i=0 ; s[i] != 0 ; i++ )
    /* Niente */ ;

lun = i ;
```


La funzione strlen

- Nella libreria standard C è disponibile la funzione `strlen`, che calcola la lunghezza della stringa passata come parametro
- Necessario includere `<string.h>`

```
const int MAX = 20 ;
char s[MAX+1] ;
int lun ;

... /* lettura stringa */

lun = strlen(s) ;
```

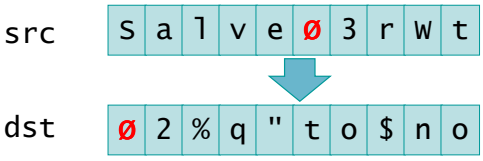
Operazioni elementari sulle stringhe

Copia di stringhe

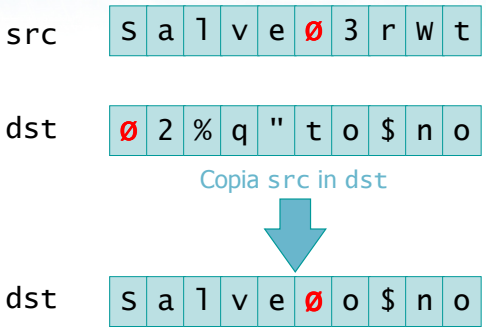
Copia di stringhe

- L'operazione di copia prevede di ricopiare il contenuto di una prima stringa "sorgente", in una seconda stringa "destinazione"

```
char src[MAXS+1] ;
char dst[MAXD+1] ;
```



Risultato della copia



Copia

```
const int MAXS = 20, MAXD = 30 ;
char src[MAXS+1] ;
char dst[MAXD+1] ;
int i ;

... /* lettura stringa src */

for( i=0 ; src[i] != 0 ; i++ )
     dst[i] = src[i] ; /* copia */

dst[i] = 0 ; /* aggiunge terminatore */
```


La funzione strcpy

- Nella libreria standard C, includendo <string.h>, è disponibile la funzione strcpy, che effettua la copia di stringhe
  - Primo parametro: stringa destinazione
  - Secondo parametro: stringa sorgente

```
const int MAXS = 20, MAXD = 30 ;
char src[MAXS+1] ;
char dst[MAXD+1] ;

... /* lettura stringa src */

strcpy(dst, src) ;
```


Avvertenze

- Nella stringa destinazione vi deve essere un numero sufficiente di locazioni libere
  - MAXD+1 >= strlen(src)+1
- Il contenuto precedente della stringa destinazione viene perso
- La stringa sorgente non viene modificata
- Il terminatore nullo
  - Deve essere aggiunto in coda a dst
  - La strcpy pensa già autonomamente a farlo

Errore frequente

- Per effettuare una copia di stringhe **non** si può assolutamente utilizzare l'operatore =
- Necessario usare `strcpy`

```
dst = src ;
```

```
dst[] = src[] ;
```

```
dst[MAXD] = src[MAXC] ;
```

```
strcpy(dst, src);
```


Operazioni elementari sulle stringhe

Concatenazione di stringhe

Concatenazione di stringhe

- L'operazione di concatenazione corrisponde a creare una nuova stringa composta dai caratteri di una prima stringa, **seguiti** dai caratteri di una seconda stringa

Semplificazione

- Per maggior semplicità, in C l'operazione di concatenazione scrive il risultato **nello stesso vettore** della prima stringa
- Il valore precedente della prima stringa viene così perso
- Per memorizzare altrove il risultato, o per non perdere la prima stringa, è possibile ricorrere a stringhe temporanee ed alla funzione `strcpy`

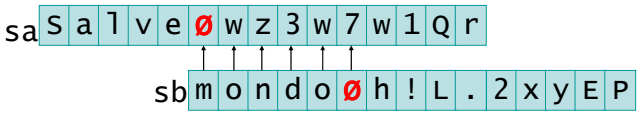
Esempio

Algoritmo di concatenazione

- Trova la fine della prima stringa

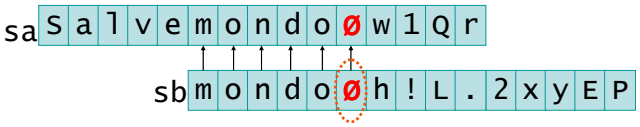
Algoritmo di concatenazione

- Trova la fine della prima stringa
- Copia la seconda stringa nel vettore della prima, a partire della posizione del terminatore nullo (sovrascrivendolo)



Algoritmo di concatenazione

- Trova la fine della prima stringa
- Copia la seconda stringa nel vettore della prima, a partire della posizione del terminatore nullo (sovrascrivendolo)
- Termina la copia non appena trovato il terminatore della seconda stringa



Concatenazione

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int la ;
int i ;

... /* lettura stringhe */

la = strlen(sa) ;

for( i=0 ; sb[i] != 0 ; i++ )
    sa[la+i] = sb[i] ; /* copia */

sa[la+i] = 0 ; /* terminatore */
```


La funzione `strcat`

- Nella libreria standard C, includendo `<string.h>`, è disponibile la funzione `strcat`, che effettua la concatenazione di stringhe
  - Primo parametro: prima stringa (destinazione)
  - Secondo parametro: seconda stringa

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;

... /* lettura stringhe */

strcat(sa, sb) ;
```


Avvertenze (1/2)

- Nella prima stringa vi deve essere un numero sufficiente di locazioni libere
  - `MAX+1 >= strlen(sa)+strlen(sb)+1`
- Il contenuto precedente della prima stringa viene perso
- La seconda stringa non viene modificata
- Il terminatore nullo
  - Deve essere aggiunto in coda alla prima stringa
  - La `strcat` pensa già autonomamente a farlo

Avvertenze (2/2)

- Per concatenare 3 o più stringhe, occorre farlo due a due:
  - `strcat(sa, sb);`
  - `strcat(sa, sc);`
- È possibile concatenare anche stringhe costanti
  - `strcat(sa, "!");`

Operazioni elementari sulle stringhe

Confronto di stringhe

Confronto di stringhe

- Il confronto di due stringhe (es.: `sa` e `sb`), mira a determinare se:
  - Le due stringhe sono uguali
    - hanno uguale lunghezza e sono composte dagli stessi caratteri nello stesso ordine
  - Le due stringhe sono diverse
  - La stringa `sa` precede la stringa `sb`
    - secondo l'ordine lessicografico imposto dal codice ASCII
    - parzialmente compatibile con l'ordine alfabetico
  - La stringa `sa` segue la stringa `sb`

Confronto di uguaglianza

- Ogni carattere di `sa` deve essere uguale al carattere corrispondente di `sb`
- Il terminatore nullo deve essere nella stessa posizione
- I caratteri successivi al terminatore vanno ignorati

sa	S	a	l	v	e	Ø	o	4	d	1	Ø	w	1	Q	r
sb	S	a	l	v	e	Ø	h	!	L	.	2	x	y	E	P

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```


Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[M
int ugual
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Flag: ricerca di universalità della condizione `sa[i]==sb[i]`

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0 ; i++ )
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Cicla fino al terminatore di `sa` o di `sb` (il primo che si incontra)

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

Verifica che tutti i caratteri incontrati siano uguali. Se no, poni a 0 il flag `uguali`.

Confronto di uguaglianza

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int uguali ;
int i ;
...

uguali = 1 ;
for( i=0 ; sa[i]!=0
{
    if(sa[i]!=sb[i])
        uguali = 0 ;
}
if(sa[i]!=0 || sb[i]!=0)
    uguali = 0 ;
```

In questo punto sicuramente una delle due stringhe è arrivata al terminatore. Se non lo è anche l'altra, allora non sono uguali!

Confronto di ordine

- Verifichiamo se `sa` "è minore di" `sb`. Partiamo con `i=0`
- Se `sa[i]<sb[i]`, allora `sa` è minore
- Se `sa[i]>sb[i]`, allora `sa` non è minore
- Se `sa[i]=sb[i]`, allora bisogna controllare i caratteri successivi (`i++`)
- Il terminatore nullo conta come "minore" di tutti

sa	S	a	l	v	e	Ø	o	4	d	1	Ø	w	1	Q	r

sb	S	a	l	u	t	e	Ø	!	L	.	2	x	y	E	P

Confronto di ordine (1/2)

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int minore ;
int i ;
...
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
          && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}
```


Confronto di ordine (2/2)

```
if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    printf("%s e' minore di %s\n",
           sa, sb ) ;
```


Commenti

```
minore = 0 ;
for( i=0 ; sa[i]
          && minore
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}

if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    ...
```

Ricerca di esistenza della condizione `sa[i]<sb[i]`.

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore
    if(sa[i]>sb
        minore
}

if(minore==0 &&
    minore=1 ;

if(minore==1)
    ...
```

Cicla fino al primo terminatore nullo, oppure fino a che non si "scopre" chi è minore.
In altre parole, continua a ciclare solo finché le stringhe "sembrano" uguali.

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=
           && mino
{
    if(sa[i]<sb[i])
        minore = 1 ;
    if(sa[i]>sb[i])
        minore = -1 ;
}
                  sa
if(minore==1)
    ...
```

Sicuramente sa è minore di sb
Flag: `minore = 1`

Se flag `minore==0` continua a ciclare

Sicuramente sa non è minore di sb
Flag: `minore = -1`

Commenti

```
minore = 0 ;
for( i=0 ; sa[i]!=0 && sb[i]!=0
           && minore==0;  i++ )
{
    if(sa[i]<sb[i])
        minore = 1
    if(sa[i]>sb[i])
        minore = -1
}

if(minore==0 && sa[i]==0 && sb[i]!=0)
    minore=1 ;

if(minore==1)
    ...
```

Se finora erano uguali, ma sa è più corta di sb, allora sa è minore

La funzione `strcmp`

- Nella libreria standard C, includendo `<string.h>`, è disponibile la funzione `strcmp`, che effettua il confronto di stringhe
  - Primo parametro: prima stringa
  - Secondo parametro: seconda stringa
  - Valore restituito:
    - `<0` se la prima stringa è minore della seconda
    - `==0` se le stringhe sono uguali
    - `>0` se la prima stringa è maggiore della seconda

Confronti vari

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int ris ;

...

ris = strcmp(sa, sb) ;

if(ris<0)
    printf("%s minore di %s\n", sa, sb);
if(ris==0)
    printf("%s uguale a %s\n", sa, sb);
if(ris>0)
    printf("%s maggiore di %s\n", sa, sb);
```

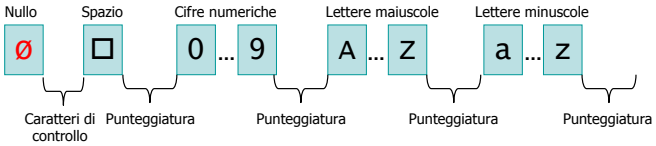
Suggerimento

- Per ricordare il significato del valore calcolato da `strcmp`, immaginare che la funzione faccia una "sottrazione" tra le due stringhe
  - `sa - sb`
    - Negativo: sa minore
    - Positivo: sa maggiore
    - Nullo: uguali

```
const int MAX = 20 ;
char sa[MAX] ;
char sb[MAX] ;
int ris ;
...
ris = strcmp(sa, sb) ;
```

Ordinamento delle stringhe

- La funzione `strcmp` lavora confrontando tra loro i codici ASCII dei caratteri
- Il criterio di ordinamento è quindi dato dalla posizione dei caratteri nella tabella ASCII



Conseguenze

- Ogni lettera maiuscola precede ogni lettera minuscola
  - `Ciao` precede `ciao`
  - `Zulu` precede `apache`
- Gli spazi contano, e precedono le lettere
  - `Qui Quo Qua` precede `QuiQuoQua`
- I simboli di punteggiatura contano, ma non vi è una regola intuitiva
- L'ordinamento che si ottiene è lievemente diverso da quello "standard" alfabetico

Operazioni elementari sulle stringhe

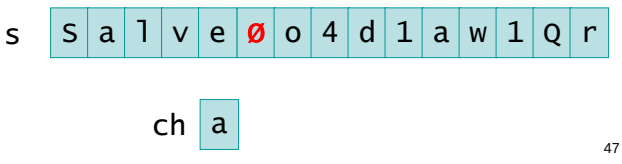
Ricerca di sotto-stringhe

Ricerca in una stringa

- È possibile concepire diversi tipi di ricerche da compiersi all'interno di una stringa:
  - Determinare se un determinato carattere compare all'interno di una stringa data
  - Determinare se una determinata stringa compare integralmente all'interno di un'altra stringa data, in una posizione arbitraria

Ricerca di un carattere (1/2)

- Detti:
  - `s` una stringa arbitraria
  - `ch` un carattere qualsiasi
- Determinare se la stringa `s` contiene (una o più volte) il carattere `ch` al suo interno, in qualsiasi posizione



Ricerca di un carattere (2/2)

```
const int MAX = 20 ;
char s[MAX] ;
char ch ;
int trovato ;
int i ;

...

trovato = 0 ;
for( i=0 ; s[i]!=0 && trovato==0; i++ )
{
    if( s[i]==ch )
        trovato = 1 ;
}
```

La funzione strchr (1/2)

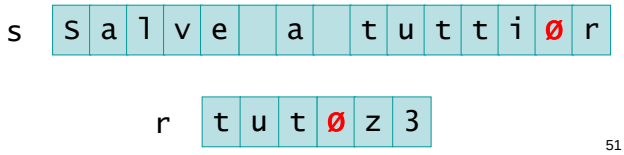
- Nella libreria standard C, includendo <string.h>, è disponibile la funzione strchr, che effettua la ricerca di un carattere
  - Primo parametro: stringa in cui cercare
  - Secondo parametro: carattere da cercare
  - Valore restituito:
    - !=NULL se il carattere c'è
    - ==NULL se il carattere non c'è

La funzione strchr (2/2)

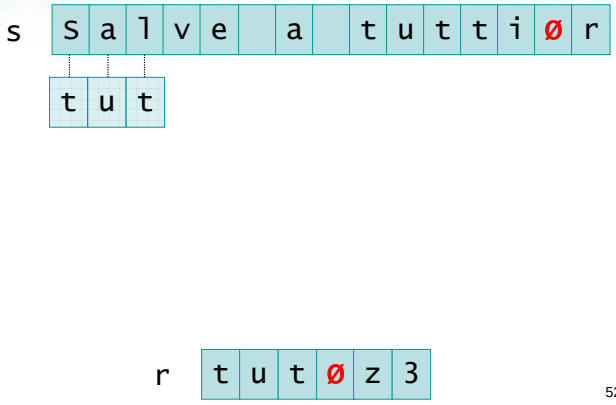
```
const int MAX = 20 ;
char s[MAX] ;
char ch ;

...

if(strchr(s, ch)!=NULL)
   printf("%s contiene %c\n", s, ch) ;
```


Ricerca di una sotto-stringa

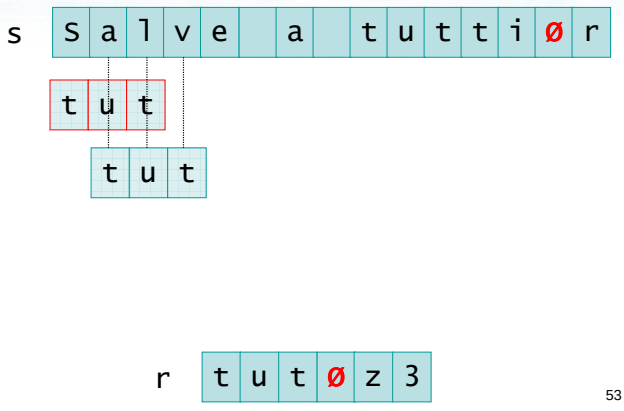
- Detti:
  - s una stringa arbitraria
  - r una stringa da ricercare
- Determinare se la stringa s contiene (una o più volte) la stringa r al suo interno, in qualsiasi posizione



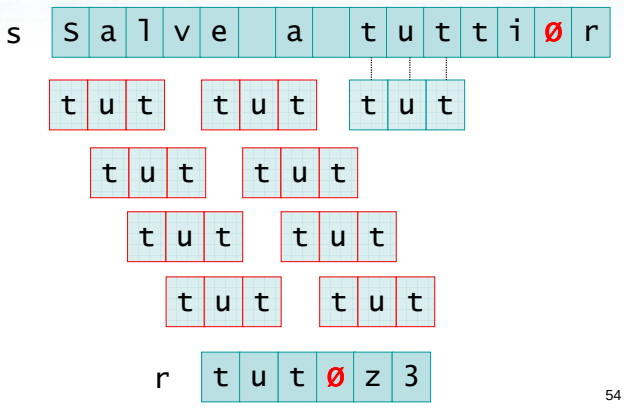
Esempio



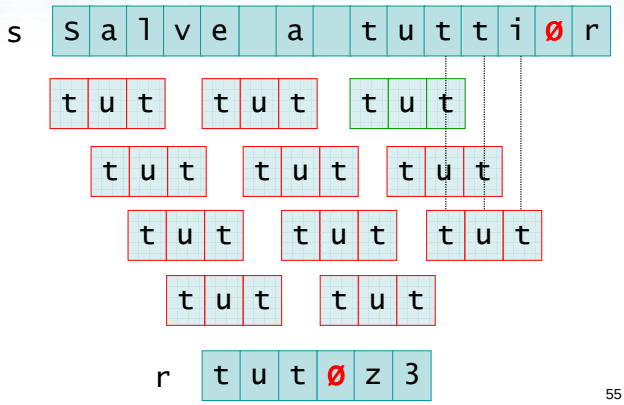
Esempio



Esempio

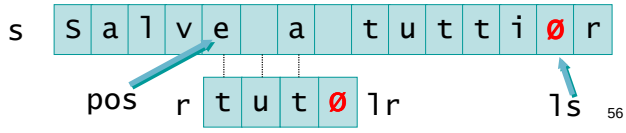


Esempio



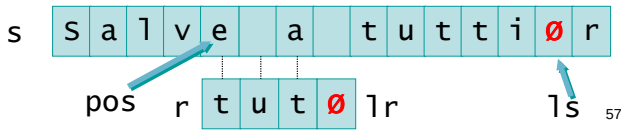
Algoritmo di ricerca

- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione possibile di `r` all'interno di `s`: `pos = 0...ls-lr` (compresi)



Algoritmo di ricerca

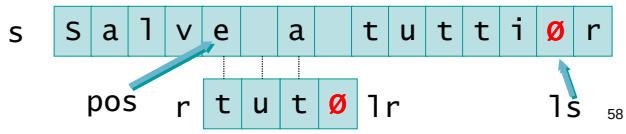
- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione possibile di `r` all'interno di `s`: `pos = 0...ls-lr` (compresi)
  - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`, tra `pos` e `pos+lr-1`
  - Se sì, `trovato = 1`



Algoritmo di ricerca

- `lr = strlen(r);`
- `trovato = 0`
- Per ogni posizione po
  `pos = 0...ls-lr` (compresi)
  - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
  - Se sì, `trovato = 1`

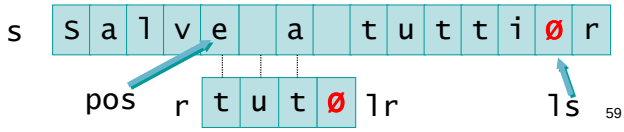
```
diversi = 0 ;
for(i=0; i<lr; i++)
    if(r[i]!=s[pos+i])
        diversi = 1 ;
```



Algoritmo di ricerca

- `lr = strlen(r); ls = strlen(s)`
- `trovato = 0`
- Per ogni posizione po
  `pos = 0...ls-lr`
  - Controlla se i caratt
    coincidono con i ca
    `pos+lr-1`
  - Se sì, `trovato = 1`

```
if(diversi==0)
    trovato=1 ;
```



Ricerca di una sotto-stringa (1/2)

substr.c

```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;
int lr, ls, pos ;
int i ;
int trovato, diversi ;

...

ls = strlen(s);
lr = strlen(r);
```

Ricerca di una sotto-stringa (2/2)


```
trovato = 0 ;
for(pos=0; pos<=ls-lr; pos++)
{
  /* confronta r[0...lr-1] con  s[pos...pos+lr-1] */
    diversi = 0 ;
    for(i=0; i<lr; i++)
        if(r[i]!=s[pos+i])
            diversi = 1 ;

    if(diversi==0)
        trovato=1 ;
}

if(trovato==1)
    printf("Trovato!\n");
```


La funzione strstr (1/2)

- Nella libreria standard C, includendo <string.h>, è disponibile la funzione strstr, che effettua la ricerca di una sottostringa
  - Primo parametro: stringa in cui cercare
  - Secondo parametro: sotto-stringa da cercare
  - Valore restituito:
    - !=NULL se la sotto-stringa c'è
    - ==NULL se la sotto-stringa non c'è

La funzione strstr (2/2)

```
const int MAX = 20 ;
char s[MAX] ;
char r[MAX] ;

...

if(strstr(s, r)!=NULL)
    printf("Trovato!\n");
```


Operazioni elementari sulle stringhe

Ricerca di parole

Ricerca di parole

- Talvolta non interessa trovare una qualsiasi sotto-stringa, ma solamente verificare se una **parola completa** è presente in una stringa

s1	C	i	a	o		n	o	n	n	o	ø	t	2	"	r
s2	o	g	g	i		n	o	n		c	'	e	'	ø	4

r	n	o	n	ø	z	3

Definizioni (1/2)

- **Lettera**: carattere ASCII facente parte dell'alfabeto maiuscolo ('A'...'Z') o minuscolo ('a'...'z')
- **Parola**: insieme consecutivo di lettere, separato da altre parole mediante spazi, numeri o simboli di punteggiatura

Definizioni (2/2)

- **Inizio di parola**: lettera, prima della quale non vi è un'altra lettera
  - Non vi è un altro carattere (inizio stringa)
  - Vi è un altro carattere, ma non è una lettera
- **Fine di parola**: lettera, dopo la quale non vi è un'altra lettera
  - Non vi è un altro carattere (fine stringa)
  - Vi è un altro carattere, ma non è una lettera

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in quella posizione, `s[pos]`, è un inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == 0 ||
    s[pos-1] non è una lettera )
```

```
if( pos == 0 ||
    !( (s[pos-1]>='a' && s[pos-1]<='z') ||
       (s[pos-1]>='A' && s[pos-1]<='Z') )
  )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in inizio di parola
    - Controlla se i caratteri di `r`, tra 0 e `lr-1`, coincidono con i caratteri di `s`,tra `pos` e `pos+lr-1`
    - Controlla se `s[pos+lr-1]` è una fine di parola
    - Se entrambi i controlli sono ok, allora `trovato=1`

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```


Algoritmo di ricerca

- Per ogni possibile posizione `pos` di `r` all'interno di `s`
  - Se il carattere in

```
if( pos == ls-lr ||
    s[pos+lr] non è una lettera )
```

```
if( pos == ls-lr ||
    !( (s[pos+lr]>='a' && s[pos+lr]<='z') ||
       (s[pos+lr]>='A' && s[pos+lr]<='Z') )
  )
```

Ricerca di una parola (1/2)

parola.c

```
trovato = 0 ;
for(pos=0; pos<=ls-lr; pos++)
{
    if( pos==0 ||
        !( (s[pos-1]>='a' &&
            s[pos-1]<='z') ||
           (s[pos-1]>='A' &&
            s[pos-1]<='Z') ) )
    {
        diversi = 0 ;
        for(i=0; i<lr; i++)
            if(r[i]!=s[pos+i])
                diversi = 1 ;
```


Ricerca di una parola (2/2)

parola.c

```
        if( diversi==0 &&
            ( pos == ls-lr ||
              !( (s[pos+lr]>='a' &&
                  s[pos+lr]<='z') ||
                 (s[pos+lr]>='A' &&
                  s[pos+lr]<='Z') )
            ) )
        {
            trovato=1 ;
        }
    }
}
```


La funzione strparola

- Nella libreria standard C non esiste alcuna funzione che svolga automaticamente la ricerca di una parola intera!!!
- Occorre identificare, ogni volta, se il compito da svolgere è riconducibile ad una o più funzioni di libreria
- Eventualmente si combinano tra loro più funzioni di libreria diverse
- In alcuni casi occorre però ricorrere all'analisi carattere per carattere

Caratteri e stringhe

Funzioni di libreria

Funzioni di libreria

- Introduzione
- Lunghezza di stringhe
- Classificazione di caratteri
- Trasformazione di caratteri
- Copia e concatenazione
- Confronto di stringhe
- Ricerca in stringhe
- Conversione numero-stringa

Funzioni di libreria

Introduzione

Librerie sulle stringhe

- La libreria standard C dispone di molte funzioni predisposte per lavorare su caratteri e stringhe
- Tali funzioni si trovano prevalentemente in due librerie:
  - `<ctype.h>` funzioni operanti su caratteri
  - `<string.h>` funzioni operanti su stringhe
- Tutte le funzioni di libreria accettano e generano stringhe correttamente terminate

Suggerimenti

- Quando possibile, utilizzare sempre le funzioni di libreria
  - Sono più veloci
  - Sono maggiormente collaudate
- In ogni caso, ricordare che è sempre possibile effettuare le operazioni direttamente:
  - Sui caratteri, ricorrendo alla codifica ASCII
  - Sulle stringhe, ricorrendo alla rappresentazione come vettori di caratteri

Rappresentazione

Nome funzione	**`strlen`**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`int`** : la lunghezza della stringa
Descrizione	Calcola la lunghezza della stringa `s`
Esempio	`lun = strlen(s) ;`

Convenzioni

- Assumiamo che nel seguito di questa lezione siano valide le seguenti definizioni

```
const int MAX = 20 ;
char s[MAX] ;
char s1[MAX] ;
char s2[MAX] ;
char r[MAX] ;
int lun ;
int n ;
char ch ;
float x ;
```


Funzioni di libreria

Lunghezza di stringhe

Lunghezza di stringhe

- Definite in `<string.h>`
- Determina la lunghezza di una stringa data
  - strlen

strlen

Nome funzione	**strlen**
Libreria	`#include <string.h>`
Parametri in ingresso	s : stringa
Valore restituito	**int** : la lunghezza della stringa
Descrizione	Calcola la lunghezza della stringa s
Esempio	`lun = strlen(s) ;`

Funzioni di libreria

Classificazione di caratteri

Classificazione di caratteri

- Definite in `<ctype.h>`
- Analizzano un singolo carattere, identificandone la tipologia
  - Lettera
    - Maiuscola
    - Minuscola
  - Cifra
  - Punteggiatura
  - isalpha
  - isupper
  - islower
  - isdigit
  - isalnum
  - isxdigit
  - ispunct
  - isgraph
  - isprint
  - isspace
  - iscntrl

isalpha

Nome funzione	**isalpha**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera maiuscola o minuscola (A...Z, a...z), "falso" altrimenti
Esempio	`if(isalpha(ch))` `{ ... }`

isupper

Nome funzione	**isupper**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera maiuscola (A...Z), "falso" altrimenti
Esempio	`if(isupper(ch))` `{ ... }`

islower

Nome funzione	**islower**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera minuscola (a...z), "falso" altrimenti
Esempio	`if(islower(ch))` `{ ... }`

isdigit

Nome funzione	**isdigit**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una cifra numerica (0...9), "falso" altrimenti
Esempio	`if(isdigit(ch))` `{ ... }`

isalnum

Nome funzione	**isalnum**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una lettera oppure una cifra numerica, "falso" altrimenti. Equivalente a `isalpha(ch)||isdigit(ch)`
Esempio	`if(isalnum(ch))` `{ ... }`

isxdigit

Nome funzione	**isxdigit**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è una una cifra numerica oppure una lettera valida in base 16 (a...f, A...F), "falso" altrimenti.
Esempio	`if(isxdigit(ch))` `{ ... }`

ispunct

Nome funzione	`ispunct`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un simbolo di punteggiatura (!"#$%&'()*+,-./:;<=>?@ [\]^_`{|}~), "falso" altrimenti.
Esempio	`if(ispunct(ch))` `{ ... }`

isgraph

Nome funzione	`isgraph`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un qualsiasi simbolo visibile (lettera, cifra, punteggiatura), "falso" altrimenti.
Esempio	`if(isgraph(ch))` `{ ... }`

isprint

Nome funzione	`isprint`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è un qualsiasi simbolo visibile oppure lo spazio, "falso" altrimenti.
Esempio	`if(isprint(ch))` `{ ... }`

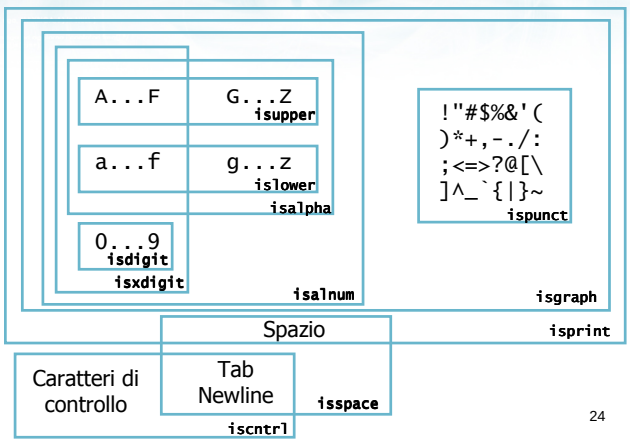
isspace

Nome funzione	`isspace`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se il carattere `ch` è invisibile (spazio, tab, a capo), "falso" altrimenti.
Esempio	`if(isspace(ch))` `{ ... }`

iscntrl

Nome funzione	`iscntrl`
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	Vero/falso
Descrizione	Ritorna "vero" se `ch` è un carattere di controllo (ASCII 0...31, 127), "falso" altrimenti.
Esempio	`if(iscntrl(ch))` `{ ... }`

Vista d'insieme



Funzioni di libreria

Trasformazione di caratteri

Trasformazione di caratteri

- Definite in `<ctype.h>`
- Convertono tra lettere maiuscole e lettere minuscole
  - toupper
  - tolower

toupper

Nome funzione	**toupper**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	**char** : carattere maiuscolo
Descrizione	Se `ch` è una lettera minuscola, ritorna l'equivalente carattere maiuscolo, se no ritorna `ch` stesso
Esempio	`for(i=0; s[i]!=0; i++)` `  s[i] = toupper(s[i]) ;`

tolower

Nome funzione	**tolower**
Libreria	`#include <ctype.h>`
Parametri in ingresso	`ch` : carattere
Valore restituito	**char** : carattere maiuscolo
Descrizione	Se `ch` è una lettera minuscola, ritorna l'equivalente carattere maiuscolo, se no ritorna `ch` stesso
Esempio	`for(i=0; s[i]!=0; i++)` `  s[i] = tolower(s[i]) ;`

Funzioni di libreria

Copia e concatenazione

Copia e concatenazione

- Definite in `<string.h>`
- Trasferiscono il contenuto di una stringa in un'altra
  - Sostituendolo
  - Accodandolo
  - strcpy
  - strncpy
  - strcat
  - strncat

strcpy

Nome funzione	**strcpy**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa
Valore restituito	nessuno utile
Descrizione	Copia il contenuto della stringa `src` all'interno della stringa `dst` (che deve avere lunghezza sufficiente).

Esempio	`strcpy(s1, s2) ;`
	`strcpy(s, "") ;`
	`strcpy(s1, "ciao") ;`

strncpy

Nome funzione	**strncpy**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa `n` : numero max caratteri
Valore restituito	nessuno utile
Descrizione	Copia il contenuto della stringa `src` (massimo n caratteri) all'interno della stringa `dst`.
Esempio	`strncpy(s1, s2, 20) ;` `strncpy(s1, s2, MAX) ;`

strcat

Nome funzione	**strcat**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa
Valore restituito	nessuno utile
Descrizione	Accoda il contenuto della stringa `src` alla fine della stringa `dst` (che deve avere lunghezza sufficiente).
Esempio	`strcat(s1, s2) ;` `strcat(s1, " ") ;`

strncat

Nome funzione	**strncat**
Libreria	`#include <string.h>`
Parametri in ingresso	`dst` : stringa `src` : stringa `n` : numero max caratteri
Valore restituito	nessuno utile
Descrizione	Accoda il contenuto della stringa `src` (massimo n caratteri) alla fine della stringa `dst`.
Esempio	`strncat(s1, s2) ;`

Funzioni di libreria

Confronto di stringhe

Confronto di stringhe

- Definite in `<string.h>`
- Confrontano due stringhe sulla base dell'ordine lessicografico imposto dalla tabella dei codici ASCII
  - strcmp
  - strncmp

strcmp

Nome funzione	**strcmp**
Libreria	`#include <string.h>`
Parametri in ingresso	`s1` : stringa `s2` : stringa
Valore restituito	**`int`** : risultato confronto
Descrizione	Risultato <0 se `s1` precede `s2` Risultato ==0 se `s1` è uguale a `s2` Risultato >0 se `s1` segue `s2`
Esempio	`if(strcmp(s, r)==0) {...}` `while(strcmp(r,"fine")!=0) {...}`

strncmp

Nome funzione	**strncmp**
Libreria	`#include <string.h>`
Parametri in ingresso	`s1` : stringa `s2` : stringa `n` : numero max caratteri
Valore restituito	**`int`** : risultato confronto
Descrizione	Simile a `strcmp`, ma confronta solo i primi n caratteri, ignorando i successivi.
Esempio	`if(strncmp(r, "buon", 4)==0)` (buongiorno, buonasera, buonanotte)

Funzioni di libreria

Ricerca in stringhe

Ricerca

- Definite in `<string.h>`
- Ricercano all'interno di una stringa data
  - Se compare un carattere
  - Se compare una sotto-stringa
  - Se compare una sequenza qualsiasi composta di caratteri dati

- strchr
- strstr
- strspn
- strcspn

strchr

Nome funzione	**strchr**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `ch` : carattere
Valore restituito	`==NULL` oppure `!=NULL`
Descrizione	Risultato `!=NULL` se il carattere `ch` compare nella stringa. Risultato `==NULL` se non compare.
Esempio	`if(strchr(s, '.')!=NULL)...` `if(strchr(s, ch)==NULL)...`

strstr

Nome funzione	**strstr**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	`==NULL` oppure `!=NULL`
Descrizione	Risultato `!=NULL` se la sotto-stringa `r` compare nella stringa `s`. Risultato `==NULL` se non compare.
Esempio	`if(strstr(s, "xy")!=NULL)...` `if(strstr(s, s1)==NULL)...`

strspn

Nome funzione	**strspn**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	**`int`** : lunghezza sequenza iniziale
Descrizione	Calcola la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri presenti in `r` (in qualsiasi ordine).
Esempio	`lun = strspn(s, " ") ;` `lun = strspn(s, " :,;.") ;`

strcspn

Nome funzione	**strcspn**
Libreria	`#include <string.h>`
Parametri in ingresso	`s` : stringa `r` : stringa
Valore restituito	**`int`** : lunghezza sequenza iniziale
Descrizione	Calcola la lunghezza della parte iniziale di `s` che è composta esclusivamente da caratteri **non presenti** in `r` (in qualsiasi ordine).
Esempio	`lun = strcspn(s, " ") ;` `lun = strcspn(s, " :,;.") ;`

Funzioni di libreria

Conversione numero-stringa

Conversioni numero-stringa

- Definite in `<stdlib.h>`
- Mettono in relazione un valore numerico (intero o reale) con la sua rappresentazione come caratteri all'interno di una stringa
  - "372" ↔ 372 (int)
  - "3.0" ↔ 3.0 (float)
  - atoi
  - atof
- In futuro:
  - sscanf
  - sprintf

atoi

Nome funzione	**atoi**
Libreria	`#include <stdlib.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`int`** : valore estratto
Descrizione	Analizza la stringa `s` ed estrae il valore intero in essa contenuto (a partire dai primi caratteri).
Esempio	`n = atoi(s) ;` `n = atoi("232abc") ;`

atof

Nome funzione	**atof**
Libreria	`#include <stdlib.h>`
Parametri in ingresso	`s` : stringa
Valore restituito	**`double/float`** : valore estratto
Descrizione	Analizza la stringa `s` ed estrae il valore reale in essa contenuto (a partire dai primi caratteri).
Esempio	`x = atof(s) ;` `x = atof("2.32abc") ;`

Caratteri e stringhe

Esercizi proposti

Esercizi proposti

- Esercizio "Parola palindroma"
- Esercizio "Iniziali maiuscole"
- Esercizio "Alfabeto farfallino"

Esercizi proposti

Esercizio "Parola palindroma"

Esercizio "Parola palindroma"

- Sia data una parola inserita da tastiera.
  - Si consideri che la parola può contenere sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al massimo 30 caratteri
- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la parola inserita
  - Aggiornare la parola in modo che tutti i caratteri siano minuscoli, e visualizzarla
  - Verificare se la parola è palindroma

Palindromia

- Una parola è detta **palindroma** se può essere letta indifferentemente da sinistra verso destra e da destra verso sinistra
  - Esempi:

o	t	t	o

m	a	d	a	m

Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa minuscolo
- Verifica se è palindroma
- Stampa se è palindroma

Analisi

- Acquisisci parola
- Stampa parola
- Converti in
- Stampa mi
- Verifica se
- Stampa se

```
const int MAX = 30 ;
char parola[MAX+1] ;
printf("Inserisci parola: ") ;
scanf("%s", parola) ;
```


Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa mi
- Verifica se
- Stampa se

```
printf("Parola inserita: %s\n",
   parola) ;
```


Analisi

- Acquisisci parola
- Stampa parola
- Converti in minuscolo
- Stampa minuscolo
- Veri
- Star

```
char minusc[MAX+1] ;
int i ;

strcpy(minusc, parola) ;

for(i=0; minusc[i]!=0; i++)
{
    minusc[i] = tolower( minusc[i] ) ;
}
```

Analisi

- Acquisisci p
- Stampa par
- Converti in
- Stampa minuscolo
- Verifica se è palindroma
- Stampa se è palindroma

```
printf("Parola minuscola: %s\n",
   minusc) ;
```


Analisi

- Acquisisci p
- Stampa par
- Converti in
- Stampa mi
- Verifica se
- Stampa se

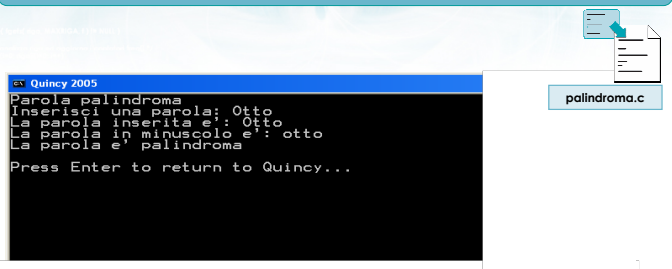
```
int i, j, palin, lun ;

i = 0 ;
j = strlen( minusc ) - 1 ;

palin = 1 ;

while( i<j && palin==1 )
{
    if(minusc[i] != minusc[j])
        palin = 0 ;
    i++ ;  j-- ;
}
```


Analisi

- Acquisisci parola
- Stampa
- Conver
- Stampa
- Verifica se è palindroma
- Stampa se è palindroma

```
if(palin==1)
    printf("E' palindroma\n") ;
else
    printf("Non e' palindroma\n") ;
```

Soluzione




```
Quincy 2005
Parola palindroma
Inserisci una parola: Otto
La parola inserita e': Otto
La parola in minuscolo e': otto
La parola e' palindroma

Press Enter to return to Quincy...
```


Esercizi proposti

Esercizio "Iniziali maiuscole"

Esercizio "Iniziali maiuscole" (1/2)

- Scrivere un programma che legga una frase introdotta da tastiera
  - La frase è terminata dall'introduzione del carattere di invio
  - La frase contiene sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al più 100 caratteri

Esercizio "Iniziali maiuscole" (2/2)

- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la frase inserita
  - Costruire una nuova frase in cui il primo carattere di ciascuna parola nella frase di partenza è stato reso maiuscolo. Tutti gli altri caratteri devono essere resi minuscoli
  - Visualizzare la nuova frase

Esempio

```
cHe bElLA gIOrnaTa
```

```
Che Bella Giornata
```


Analisi

- La frase inserita può contenere degli spazi: occorrerà usare `gets` e non `scanf`
- Ogni lettera inziale di parola va convertita in maiuscolo
- Ogni lettera non iniziale di parola va convertita in minuscolo
- Ogni altro carattere va lasciato immutato

Conversione delle lettere

iniziali.c

```
for(i=0; frase[i]!=0; i++)
{
    if( isalpha(frase[i]) &&
        ( i==0 || !isalpha(frase[i-1]) ) )
    {
        frase[i] = toupper( frase[i] ) ;
    }
    else
    {
        frase[i] = tolower( frase[i] ) ;
    }
}
```


Esercizi proposti

Esercizio "Alfabeto farfallino"

Esercizio "Alfabeto farfallino" (1/2)

- Scrivere un programma che legga una frase introdotta da tastiera
  - La frase è terminata dall'introduzione del carattere di invio
  - La frase contiene sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al più 100 caratteri

Esercizio "Alfabeto farfallino" (2/2)

- Il programma deve svolgere le seguenti operazioni:
  - Visualizzare la frase inserita
  - Costruire una nuova frase nel cosiddetto "alfabeto farfallino"
  - Visualizzare la nuova frase

L'alfabeto farfallino

- La traduzione nell'alfabeto farfallino di una parola segue le seguenti regole:
  - Tutte le consonanti sono tradotte in modo identico
  - Ogni vocale è tradotta uguale a se stessa, seguita da **altri due** caratteri:
    - la lettera 'f' (se la vocale è minuscola) o la lettera 'F' (se la vocale è maiuscola)
    - una copia della vocale stessa
- Esempi:
  - a ⟶ afa
  - con ⟶ cofon

Esempio

Vacanze di NATALE

↓

Vafacafanzefe difi NAFATAFALEFE

Approccio risolutivo

- Copiamo la stringa `frase` in una nuova stringa `farfa`
- `lun=0`
- Per ogni carattere `frase[i]`
  - Se non è una vocale, va accodato a `farfa[lun]`
  - Se è una vocale, occorre accodare a `farfa[lun]` i 3 caratteri: `frase[i]`, poi `'f'` o `'F'`, poi ancora `frase[i]`
  - Incrementare `lun` (di 1 oppure 3)
- Infine aggiungere a `farfa[lun]` il terminatore nullo

Conversione alfabeto (1/2)

farfallino.c

```
lun = 0 ;
for(i=0; frase[i]!=0; i++)
{   if( isalpha(frase[i]) )
    {   /* lettera alfabetica */
        ch = tolower(frase[i]) ;
        if( ch=='a' || ch=='e' || ch=='i'
            || ch=='o' || ch=='u' )
        {   /* vocale: trasforma */
          1 farfa[lun] = frase[i] ;
          2 if(isupper(frase[i]))
                farfa[lun+1] = 'F' ;
            else  farfa[lun+1] = 'f' ;
          3 farfa[lun+2] = frase[i] ;

            lun = lun + 3 ;
        }
```


Conversione alfabeto (2/2)

farfallino.c

```
        else
        {
            /* consonante: copia */
            farfa[lun] = frase[i] ;
            lun++ ;
        }
    }
    else
    {
        /* altro carattere: copia */
        farfa[lun] = frase[i] ;
        lun++ ;
    }
}
farfa[lun] = 0 ; /* terminatore */
```

Caratteri e stringhe

Sommario

Argomenti trattati

- Caratteri e stringhe
- Il tipo `char`
- Vettori di `char`
- Stringhe: parole, frasi
- Operazioni fondamentali sulle stringhe
  - Classificazione
  - Ricerca
  - Copia e concatenazione
  - Conversione

Tecniche di programmazione

- Terminatore nullo
- Librerie `<string.h>` e `<ctype.h>`
- Manipolazione di stringhe come vettori di caratteri
- Manipolazione di stringhe attraverso le funzioni di libreria
- Identificazione di parole all'interno di frasi

Stringhe e vettori

- Molte operazioni sulle stringhe sono ricondotte ad analoghe operazioni sui vettori
- Molti problemi "astratti" su vettori numerici assumono forma più "concreta" nel caso delle stringhe
- I cicli sono solitamente governati dal controllo del terminatore di fine stringa

Errore frequente

- L'input/output nelle stringhe è spesso problematico
- Utilizzando la funzione `scanf`, in presenza di spazi interni alla stringa rimarranno dei caratteri "non letti", che daranno fastidio alle successive `scanf`
- Quando possibile, ricorrere alla funzione `gets` per la lettura di una stringa

Errore frequente

- Le stringhe hanno lunghezza variabile; i vettori che le contengono hanno lunghezza fissa
- È possibile, con una chiamata a `strcpy` o `strcat`, scrivere oltre la dimensione del vettore
  - Grave errore di programmazione, che può portare alla corruzione di dati in altre variabili
- Abituarsi a verificare la lunghezza prima di copiare le stringhe

```
if(strlen(a) + strlen(b) + 1 <= MAX)
    strcat(a,b) ;
else
    ERRORE!!!
```

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Matrici - Vettori di stringhe

Matrici – Vettori di stringhe

- Matrici
- Definizione di matrici in C
- Operazioni elementari sulle matrici
- Vettori di stringhe
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 1 e 5
  - Cabodi, Quer, Sonza Reorda: capitolo 5
  - Dietel & Dietel: capitolo 6
- Dispense
  - Scheda: "Matrici e Vettori di stringhe in C"

Matrici – Vettori di stringhe

Matrici

Matrici

- Matrici bidimensionali
- Matrici come vettori di vettori
- Matrici pluridimensionali

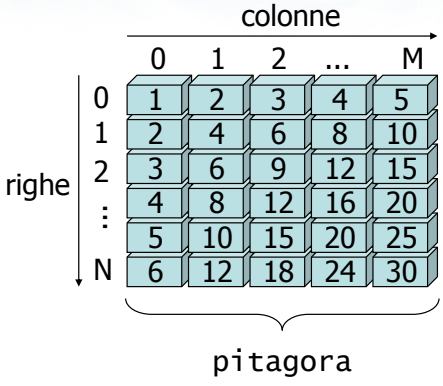
Matrici

Matrici bidimensionali

Il concetto di matrice

- La **matrice** (**array**) è un'estensione logica del concetto di vettore
  - Vettore = Sequenza uni-dimensionale di valori
    - Tutti dello stesso tipo
    - Identificati da un indice intero
    - Dimensione fissa
  - Matrice = Schiera bi- (o n-) dimensionale di valori
    - Tutti dello stesso tipo
    - Identificati da 2 (o n) indici interi
    - Dimensioni fisse

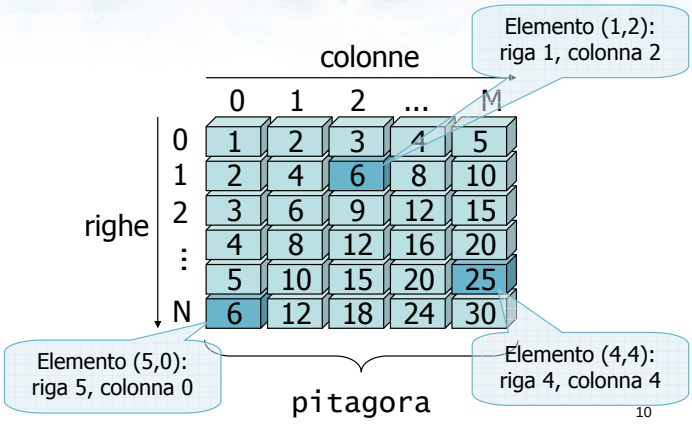
Matrice bidimensionale



Caratteristiche

- Una matrice bi-dimensionale è caratterizzata da
  - Nome : `pitagora`
  - Numero di righe : N
  - Numero di colonne : M
  - Tipo degli elementi : `int`
- Le righe sono numerate da 0 ad N-1
- Le colonne sono numerate da 0 ad M-1
- In totale ci sono N×M elementi
- In generale M≠N; per matrici quadrate, M=N

Identificazione degli elementi



Lavorare con le matrici

- Ogni operazione su una matrice deve essere svolta lavorando singolarmente su ciascuno degli elementi
- Ciò solitamente significa dover ricorrere a due cicli annidati

```
for(i=0; i<N; i++)  /* righe */
{
    for(j=0; j<M; j++)  /* colonne */
    {
        somma = somma + matrice[i][j] ;
    }
}
```

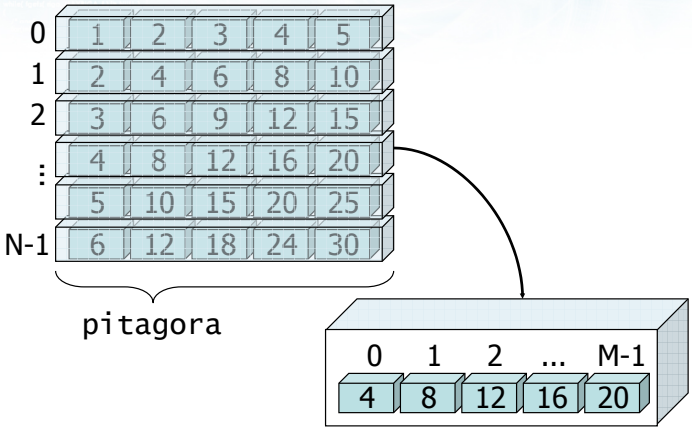

Matrici

Matrici come vettori di vettori

Vettori di vettori

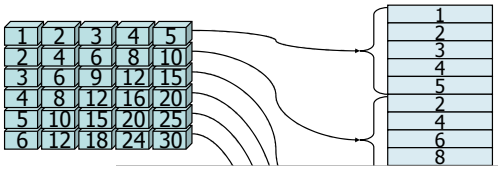
- Un altro modo di vedere le matrici è concepirle come vettori di vettori:
  - Un vettore di N oggetti (righe)
    - Ciascun oggetto (riga) è composto da M elementi (colonne)
- Questa prende il nome di **rappresentazione "per righe"** di una matrice

Esempio



Codifica

- Nella realtà, poiché la memoria di un calcolatore è uni-dimensionale, le matrici vengono effettivamente memorizzate "per righe"

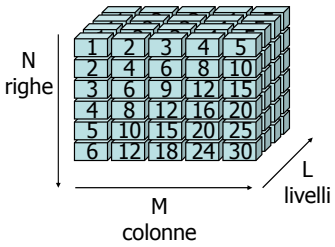


Matrici

Matrici pluridimensionali

Matrici con più dimensioni

- Il concetto di matrice può essere generalizzato anche a più di 2 dimensioni
  - Non vi sono, a priori, limiti sul numero di dimensioni



N×M×L elementi:

elemento(i, j, k)
$0 \le i \le N-1$
$0 \le j \le M-1$
$0 \le k \le L-1$

Caratteristiche

- Anche le matrici a più dimensioni condividono i vincoli di base dell'intera famiglia degli array:
  - Tipo di elementi uniforme
  - Dimensioni fissate a priori
  - Indici interi a partire da 0
- In pratica è molto raro utilizzare più di 3 dimensioni

Matrici – Vettori di stringhe

Definizione di matrici in C

Definizione di matrici in C

- Sintassi della definizione
- Operazioni di accesso

Definizione di matrici in C

Sintassi della definizione

Definizione di matrici in C

```
int mat[10][5] ;
```

Tipo di dato base | Nome della matrice | Numero di righe | Numero di colonne

Definizione di matrici in C

```
int mat[10][5] ;
```

Tipo di dato base | Nome della matrice | Numero di righe | Numero di colonne

- `int`
- `float`
- `char`
- In futuro vedremo: `struct`

Definizione di matrici in C

```
int mat[10][5] ;
```

Tipo di dato base | Nome della matrice | Numero di righe | Numero di colonne

- Stesse regole che valgono per i nomi di variabili e vettori
- I nomi delle matrici devono essere diversi dai nomi di altre variabili o vettori

Definizione di matrici in C

```
int mat[10][5] ;
```

- Tipo di dato base
- Nome della matrice
- Numero di righe
- Numero di colonne

- Interi positivi
- Costanti note a tempo di compilazione
  - `const` oppure `#define`
- Uguali o diverse

Esempi

- `int pitagora[10][10] ;`

	0	1	2	3	4	5	6	7	8	9
0	1	2	3	4	5	6	7	8	9	10
1	2	4	6	8	10	12	14	16	18	20
2	3	6	9	12	15	18	21	24	27	30
3	4	8	12	16	20	24	28	32	36	40
4	5	10	15	20	25	30	35	40	45	50
5	6	12	18	24	30	36	42	48	54	60
6	7	14	21	28	35	42	49	56	63	70
7	8	16	24	32	40	48	56	64	72	80
8	9	18	27	36	45	54	63	72	81	90
9	10	20	30	40	50	60	70	80	90	100

Esempi

- `int pitagora[10][10] ;`
- `char tris[3][3] ;`

	0	1	2
0	.	.	.
1	.	.	.
2	.	.	.

	0	1	2
0	.	.	.
1	.	X	.
2	.	.	.

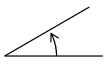
	0	1	2
0	O	.	.
1	.	X	.
2	.	.	.

	0	1	2
0	O	X	.
1	.	X	.
2	.	.	.

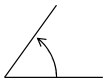
Esempi

- `int pitagora[10][10] ;`
- `char tris[3][3] ;`
- `float rot[2][2] ;`

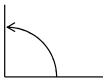
	0	1
0	0.707	0.707
1	-0.707	0.707



	0	1
0	0.500	0.866
1	-0.866	0.500



	0	1
0	0.000	1.000
1	-1.000	0.000



	0	1
0	cos β	sin β
1	-sin β	cos β

Matrici a più dimensioni

```
int mat[10][5] ;
```

- Più dimensioni
- Numero di elementi per ciascuna dimensione

```
float dati[10][5][6] ;
```


Errore frequente

- Dichiarare una matrice usando variabili anziché costanti per le dimensioni

```
int N = 10 ;
int mat[N][N] ;
```

```
int mat[10][10] ;
```

```
const int N = 10 ;
int mat[N][N] ;
```

Errore frequente

- Dichiarare una matrice usando il nome degli indici

```
int i, j ;
int mat[i][j] ;

. . .
for(i=0; i<10; i++)
  for(j=0; j<10; j++)
    scanf("%d",&mat[i][j]) ;
```

```
const int N = 10 ;
int mat[N][N] ;
```

Errore frequente

- Dichiarare una matrice usando il simbolo di "virgola"

```
const int N = 10 ;
const int M = 20 ;

int mat[N,M] ;
```

```
int mat[N][M] ;
```


Definizione di matrici in C

Operazioni di accesso

Accesso ai valori di una matrice

- Ciascun elemento di una matrice è una variabile del tipo base
- Per accedere a tale elemento si usa l'operatore di **indicizzazione**: [ ]
- Vi sono tanti indici quante sono le dimensioni della matrice
  - Ogni indice è racchiuso da una coppia di parentesi quadre

Sintassi

```
nomematrice[ valoreindice1 ][ valoreindice2 ]
```

Costante, variabile o espressione aritmetica con valore intero

Sintassi

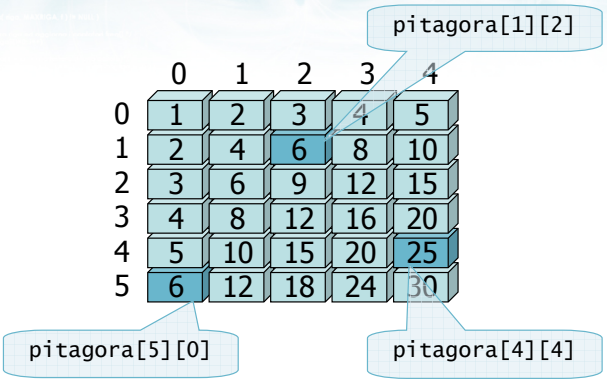
```
nomematrice[ valoreindice1 ][ valoreindice2 ]
```

Valore intero compreso tra 0 e numero di righe -1

Costante, variabile o espressione aritmetica con valore intero

Valore intero compreso tra 0 e numero di colonne -1

Esempi




```
int pitagora[6][5] ;
```


Vincoli (1/2)

- In una matrice NxMxKx..., il valore dell'indice deve essere compreso tra 0 e N-1/M-1/K-1/....
  - La responsabilità è del programmatore
- Se qualche indice non è un numero intero, viene automaticamente troncato

```
pitagora[i][j] = (i+1)*(j+1) ;

x = pitagora[1][2] ;
```


Vincoli (2/2)

- Una variabile di tipo matrice può essere utilizzata solamente mediante l'operatore di indicizzazione
  - Occorre agire individualmente sui singoli elementi
  - Non è possibile agire sull'intera matrice in una sola istruzione

```
pitagora[i][j] = (i+1)*(j+1) ;

x = pitagora[1][2] ;
```


Uso di una cella di un vettore

- L'elemento di una matrice è utilizzabile come una qualsiasi variabile:
  - Utilizzabile all'interno di un'espressione
    - `tot = tot + mat[i][j] ;`
  - Utilizzabile in istruzioni di assegnazione
    - `mat[0][0] = 0 ;`
  - Utilizzabile per stampare il valore
    - `printf("%d\n", mat[z][k]) ;`
  - Utilizzabile per leggere un valore
    - `scanf("%d\n", &mat[k][z]) ;`

Matrici – Vettori di stringhe

Operazioni elementari sulle matrici

- Definizioni
- Stampa di una matrice
- Lettura di una matrice
- Copia di una matrice
- Somme di riga o di colonna
- Ricerca di un elemento
- Ricerca del massimo o del minimo

Operazioni elementari sulle matrici

Definizioni

matrici.c

```
const int N = 10 ;
const int M = 5 ; /* dimensioni massime */

float mat[N][M] ; /* matrice 10x5 di reali */
float mat2[N][M] ; /* uguali dimensioni */

float sr[N] ; /* somma per righe */
float sc[M] ; /* somma per colonne */

int i, j ; /* indici dei cicli */
```


matrici.c

```
int trovato ; /* flag */
int riga, col ; /* risultati ricerca */

float dato ; /* elemento da ricercare */

float somma, sommar, sommac ;
    /* per calcolo di somme */

float maxr, maxc ; /* massimi */
```


Operazioni elementari sulle matrici

Stampa di una matrice

Stampa di matrici

- Occorre stampare un elemento per volta, all'interno di cicli `for`
- Sono necessari due cicli annidati
  - Il ciclo esterno per scandire le righe (da 0 a N-1)
  - Il ciclo interno per scandire ciascuna colonna (da 0 a M-1) della riga data
- Si può stampare "per righe" (caso normale) o "per colonne" (trasposta)

Stampa per righe matrice di reali

matrici.c

```
printf("Matrice: %d x %d\n", N, M);

for(i=0; i<N; i++)
{
        Stampa la riga i-esima
}
```


Stampa per righe matrice di reali

matrici.c

```
printf("Matrice: %d x %d\n", N, M);

for(i=0; i<N; i++)
{
    /* Stampa la riga i-esima */
    for(j=0; j<M; j++)
    {
        printf("%f ", mat[i][j]) ;
    }
    printf("\n");
}
```


Esempio

Prompt dei comandi

```
Matrice: 10 righe, 5 colonne
1.000000 0.500000 0.333333 0.250000 0.200000
2.000000 1.000000 0.666667 0.500000 0.400000
3.000000 1.500000 1.000000 0.750000 0.600000
4.000000 2.000000 1.333333 1.000000 0.800000
5.000000 2.500000 1.666667 1.250000 1.000000
6.000000 3.000000 2.000000 1.500000 1.200000
7.000000 3.500000 2.333333 1.750000 1.400000
8.000000 4.000000 2.666667 2.000000 1.600000
9.000000 4.500000 3.000000 2.250000 1.800000
10.000000 5.000000 3.333333 2.500000 2.000000
```


Stampa per colonne matrice di reali

matrici.c

```
printf("Matrice: %d x %d\n", N, M);

for(j=0; j<M; j++)
{
    for(i=0; i<N; i++)
    {
        printf("%f ", mat[i][j]) ;
    }
    printf("\n");
}
```


Esempio

Prompt dei comandi

```
Matrice: 10 righe, 5 colonne
1.00 2.00 3.00 4.00 5.00 6.00 7.00 8.00 9.00 10.00
0.50 1.00 1.50 2.00 2.50 3.00 3.50 4.00 4.50 5.00
0.33 0.67 1.00 1.33 1.67 2.00 2.33 2.67 3.00 3.33
0.25 0.50 0.75 1.00 1.25 1.50 1.75 2.00 2.25 2.50
0.20 0.40 0.60 0.80 1.00 1.20 1.40 1.60 1.80 2.00
```

Operazioni elementari sulle matrici

Lettura di una matrice

Lettura di matrici

- Occorre leggere un elemento per volta
- Si procede per righe (o per colonne)
- Si utilizzano solitamente due cicli annidati

Lettura per righe matrice di reali

matrici.c

```
printf("Immetti matrice %d x %d\n",
        N, M) ;

for(i=0; i<N; i++)
{
    printf("Riga %d:\n", i+1) ;
    for(j=0; j<M; j++)
    {
        printf("Elemento (%d,%d): ",
               i+1, j+1) ;
        scanf("%f", &mat[i][j]) ;
    }
}
```


Esempio

```
Prompt dei comandi
Immetti una matrice 10 x 5
Riga 1:
Elemento (1,1): 3.2
Elemento (1,2): 1
Elemento (1,3): -12.4
Elemento (1,4): 2.112
Elemento (1,5): 23
Riga 2:
Elemento (2,1): 23.1
Elemento (2,2): 2.11
Elemento (2,3): .22
Elemento (2,4): 3.14
Elemento (2,5): 2.71
```


Operazioni elementari sulle matrici

Copia di una matrice

Copia di matrici (1/2)

- L'operazione di copia di una matrice "sorgente" in una "destinazione" richiede che ciascun elemento venga copiato individualmente
- La matrice destinazione deve avere dimensioni uguali o superiori a quelle della sorgente
- L'operazione di copia avviene ovviamente a livello del singolo elemento

Copia di matrici (2/2)

matrici.c

```
for(i=0; i<N; i++)
    for(j=0; j<M; j++)
        mat2[i][j] = mat[i][j] ;
```


Operazioni elementari sulle matrici

Somme di riga o di colonna

Sommatorie in matrici

- Il calcolo di totali sui dati contenuti in una matrice può corrispondere a tre diverse operazioni:
  - Somma degli elementi di ciascuna riga (totali di riga)
  - Somma degli elementi di ciascuna colonna (totali di colonna)
  - Somma di tutti gli elementi della matrice

Esempio

`float mat[N][M] ;`

1.00	0.50	0.33	0.25	0.20
2.00	1.00	0.67	0.50	0.40
3.00	1.50	1.00	0.75	0.60
4.00	2.00	1.33	1.00	0.80
5.00	2.50	1.67	1.25	1.00
6.00	3.00	2.00	1.50	1.20
7.00	3.50	2.33	1.75	1.40
8.00	4.00	2.67	2.00	1.60
9.00	4.50	3.00	2.25	1.80
10.00	5.00	3.33	2.50	2.00

Esempio

`float mat[N][M] ;`

`float sr[N] ;`

1.00	0.50	0.33	0.25	0.20	2.28
2.00	1.00	0.67	0.50	0.40	4.56
3.00	1.50	1.00	0.75	0.60	6.85
4.00	2.00	1.33	1.00	0.80	9.13
5.00	2.50	1.67	1.25	1.00	11.41
6.00	3.00	2.00	1.50	1.20	13.70
7.00	3.50	2.33	1.75	1.40	15.98
8.00	4.00	2.67	2.00	1.60	18.26
9.00	4.50	3.00	2.25	1.80	20.55
10.00	5.00	3.33	2.50	2.00	22.83
55.00	27.50	18.33	13.75	11.00	

`float sc[M] ;`

Somma per righe

matrici.c

```
for(i=0 ; i<N ; i++)
{
    somma = 0.0 ;
    for(j=0; j<M; j++)
        somma = somma + mat[i][j] ;
    sr[i] = somma ;
}

for(i=0; i<N; i++)
    printf("Somma riga %d = %f\n",
           i+1, sr[i]) ;
```

Somma per colonne

matrici.c

```
for(j=0 ; j<M ; j++)
{
    somma = 0.0 ;
    for(i=0; i<N; i++)
        somma = somma + mat[i][j] ;
    sc[j] = somma ;
}

for(j=0; j<M; j++)
    printf("Somma colonna %d = %f\n",
           j+1, sc[j]) ;
```


Somma complessiva

matrici.c

```
somma = 0.0 ;
for(i=0 ; i<N ; i++)
{
    for(j=0; j<M; j++)
        somma = somma + mat[i][j] ;
}

printf("Somma complessiva = %f\n",
       somma) ;
```


Operazioni elementari sulle matrici

Ricerca di un elemento

Ricerca di elementi

- Dato un valore dato, ricercare se esso esiste (almeno una volta) nella matrice
- In caso affermativo, specificare la riga e la colonna
- Si utilizzano i soliti due cicli annidati

Ricerca elemento (1/2)

matrici.c

```
printf("Dato da ricercare: ");
scanf("%f", &dato) ;

trovato = 0 ;
riga = -1 ;
col  = -1 ;

for(i=0; i<N && trovato==0; i++)
    for(j=0; j<M && trovato==0; j++)
        if( mat[i][j]==dato )
        {
            trovato=1 ;
            riga = i ;
            col = j ;
        }
```


Ricerca elemento (2/2)

matrici.c

```
if(trovato==1)
  printf("Dato %f presente: (%d,%d)\n",
    dato, riga, col) ;
else
  printf("Dato %f non presente\n",
      dato) ;
```

Operazioni elementari sulle matrici

Ricerca del massimo o del minimo

Massimi e minimi

- Per quanto riguarda il calcolo di massimi e minimi si possono distinguere i 3 scenari:
  - Massimo/minimo degli elementi di ciascuna riga
  - Massimo/minimo degli elementi di ciascuna colonna
  - Massimo/minimo tra tutti gli elementi della matrice

Massimo per righe


```
for(i=0; i<N; i++)
{
    col = 0 ;
    maxr = mat[i][0] ;
    for(j=1; j<M; j++)
        if( mat[i][j] > maxr )
        {
            maxr = mat[i][j] ;
            col = j ;
        }

    printf("Max di riga %d vale %f
       e si trova nella colonna %d\n",
        i+1, maxr, col+1) ;
}
```


Massimo per colonne


```
for(j=0; j<M; j++)
{
    riga = 0 ;
    maxc = mat[0][j] ;
    for(i=1; i<N; i++)
        if( mat[i][j] > maxc )
        {
            maxc = mat[i][j] ;
            riga = i ;
        }

    printf("Max di colonna %d vale %f
       e si trova nella riga %d\n",
        j+1, maxc, riga+1) ;
}
```


Osservazioni

- Le operazioni qui citate sono gli elementi fondamentali dell'elaborazione di matrici
- Nei problemi concreti si osserveranno delle combinazioni di tali operazioni
- Occorre analizzare il problema, scomporlo nei suoi sottoproblemi e combinare opportunamente le varie tecniche

Esercizio "Max Sum Abs"

- Data una matrice NxN, determinare la riga in cui la somma dei valori assoluti degli elementi sia massima

$$r = \max_i \left(\sum_j \left| M_{ij} \right| \right)$$

Soluzione 1

- Inizializza `max`
- Per ogni riga `i`:
  - Calcola la somma `sommar` dei valori assoluti di tale riga
  - Confronta `sommar` con il `max` corrente, ed eventualmente aggiorna il `max`
- Stampa `max`

Soluzione 1

maxsumabs1.c

```
max = -1.0 ;

for(i=0; i<N; i++)
{
    sommar = 0.0 ;
    for(j=0; j<M; j++)
    {
        sommar = sommar +
                 fabs(mat[i][j]) ;
    }

    if(sommar>max)
        max = sommar ;
}
printf("R = %f\n", max) ;
```


Soluzione 2

- Calcola un vettore di appoggio, di N elementi, contenente le sommatorie per ogni riga
- Trova il max all'interno di questo vettore di appoggio
- Soluzione più lunga dal punto di vista del codice, ma più semplice da concepire e realizzare

Soluzione 2 (1/2)

maxsumabs2.c

```
for(i=0; i<N; i++)
{
   sommar = 0.0 ;
   for(j=0; j<M; j++)
   {
       sommar = sommar +
                fabs(mat[i][j]) ;
   }
   sr[i] = sommar ;
    }
```


Soluzione 2 (2/2)

maxsumabs2.c

```
max = -1.0 ;

for(i=0; i<N; i++)
    if(sr[i]>max)
        max = sr[i] ;

printf("R = %f\n", max) ;
```

Matrici – Vettori di stringhe

Vettori di stringhe

Vettori di stringhe

- Matrici di caratteri
- Vettori di stringhe
- I/O di vettori di stringhe

Vettori di stringhe

Matrici di caratteri

Matrici di caratteri

- Nel definire una matrice, è ovviamente possibile usare il tipo base **`char`**
- Permette di memorizzare una tabella NxM di caratteri ASCII
- Ogni posizione `[i][j]` deve contenere **un carattere**
  - Non può essere vuota
  - Non può contenere più di un carattere

```
char tris[3][3] ;
```

	0	1	2
0	O	X	.
1	.	X	.
2	.	.	.

Esercizio "Verifica Sudoku"

- Si realizzi un programma in C che verifichi la corretta soluzione della griglia di un "Sudoku"
- Il programma acquisisce da tastiera la griglia 9x9, in cui ciascun elemento è un carattere tra 1 e 9
- Il programma deve verificare se il Sudoku è stato correttamente risolto

Esempio

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Analisi

- Tutti i valori devono essere singoli caratteri tra '1' e '9'
- Su ciascuna delle 9 righe, non devono esserci valori ripetuti
- Su ciascuna delle 9 colonne, non devono esserci valori ripetuti
- In ciascuno dei 9 blocchi 3x3, non devono esserci valori ripetuti

Soluzione (1/10)

sudoku.c

```
const int N = 9 ;

char sudoku[N][N] ;

int i, j, k ; /* indici dei cicli */
int r1, c1, r2, c2 ;
char ch ;
int err, good ; /* flag */

printf("Verifica Sudoku\n") ;
```


Soluzione (2/10)

sudoku.c

```
for(i=0; i<N; i++)
{
    printf("Riga %d:\n", i+1) ;
    for(j=0; j<N; j++)
    {
        Acquisisci un carattere
        ch tra '1' e '9'
        sudoku[i][j] = ch ;
    }
}
```


Soluzione (2/10)

sudoku.c

```
do {
    printf("  Colonna %d: ", j+1) ;
    ch = getchar() ;
    if( ch<'1' || ch>'9' )
        printf("Errata - ripeti\n") ;

    /* elimina fino fine linea */
    while( getchar()!='\n')
        /*nop*/ ;

} while( ch<'1' || ch>'9') ;
```


Soluzione (3/10)

sudoku.c

```
/* Stampa il tabellone */
for(i=0; i<9; i++)
{
    for(j=0; j<9; j++)
    {
        printf("%c ", sudoku[i][j]) ;

        if(j==2 || j==5)
            printf("  ") ;
    }
    printf("\n");

    if(i==2 || i==5)
        printf("\n") ;
}
```


Soluzione (4/10)

sudoku.c

```
good = 1 ; /* flag generale */

/* Verifica le righe */
for(i=0; i<N; i++)
{
    printf("Riga %d: ", i+1) ;
    err = 0 ;

    /* ricerca duplicati su col. j,k */
    for(j=0; j<N; j++)
        for(k=j+1; k<N; k++)
            if(sudoku[i][j]==
                        sudoku[i][k])
                err = 1 ;
```

Soluzione (5/10)

sudoku.c

```
    if(err==0)
        printf("OK\n");
    else
    {
        printf("Errore!\n");
        good = 0 ;
    }
}
```


Soluzione (6/10)

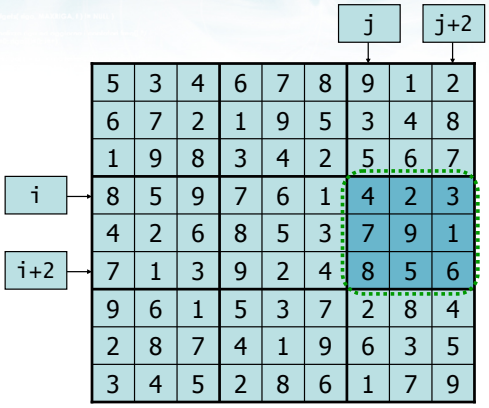
sudoku.c

```
for(i=0; i<N; i++) /* Colonne */
{   printf("Colonna %d: ", i+1) ;
    err = 0 ;
    /* ricerca dupl. su righe j,k */
    for(j=0; j<N; j++)
      for(k=j+1; k<N; k++)
        if(sudoku[j][i]==sudoku[k][i])
          err = 1 ;

    if(err==0) printf("OK\n");
    else
    {   printf("Errore!\n");
        good = 0 ;
    }
}
```

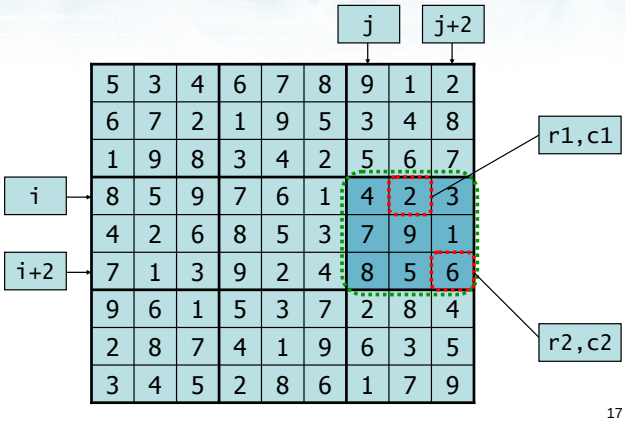

Ricerca per blocchi

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Ricerca per blocchi



Ricerca per blocchi



Soluzione (7/10)

sudoku.c

```
for(i=0; i<N; i=i+3)
{
  for(j=0; j<N; j=j+3)
  {
    printf("Blocco (%d,%d)-(%d,%d): ",
         i+1, j+1, i+3, j+3) ;

    /* ricerca duplicati nel blocco */
    /* Confronta [r1][c1]
                 con [r2][c2] */

    err = 0 ;
```

Soluzione (8/10)

sudoku.c

```
for(r1=i; r1<i+3; r1++)
for(c1=j; c1<j+3; c1++)
{   /* elemento [r1][c1]... */
    for(r2=i; r2<i+3; r2++)
    for(c2=j; c2<j+3; c2++)
    {
        /* ..rispetto a [r2][c2] */
        if( ((r1!=r2)||(c1!=c2)) &&
            sudoku[r1][c1]==
                    sudoku[r2][c2] )
        {
            err = 1 ;
        }
    } /*r2,c2*/
} /*r1,c1*/
```


Soluzione (9/10)

sudoku.c

```
        if(err==0)
            printf("OK\n");
        else
        {
            printf("Errore!\n");
            good = 0 ;
        }
    } /*j*/
} /*i*/
```


Soluzione (10/10)

sudoku.c

```
if(good==1)
    printf("Sudoku valido!\n");
else
    printf("Sudoku errato...\n");
```


Vettori di stringhe

Vettori di stringhe

Vettori di stringhe

- Una matrice di caratteri può anche essere vista come:
  - Un vettore di vettori di caratteri, cioè
  - Un vettore di stringhe
- Si tratta di un metodo diverso di interpretare la stessa struttura dati

Esempio

```
char tris[3][3] ;
```

	0	1	2
0	O	X	.
1	.	X	.
2	.	.	.

```
char nomi[5][10] ;
```

	0	1	2	3	4	5	6	7	8	9
0	F	u	l	v	i	o	\0	x	!	w
1	A	n	t	o	n	i	o	\0	.	z
2	C	r	i	s	t	i	n	a	\0	u
3	E	l	e	n	a	\0	5	g	r	d
4	D	a	v	i	d	e	\0	$	2	e

Caratteristiche

5 stringhe di lunghezza variabile

Lunghezza max 9 caratteri

Terminatore nullo in ogni stringa (riga)

```
char nomi[5][10] ;
```

	0	1	2	3	4	5	6	7	8	9
0	F	u	l	v	i	o	\0	x	!	w
1	A	n	t	o	n	i	o	\0	.	z
2	C	r	i	s	t	i	n	a	\0	u
3	E	l	e	n	a	\0	5	g	r	d
4	D	a	v	i	d	e	\0	$	2	e

Sintassi

- Definizione

```
char vett[MAX][LUN] ;
```

Nome del vettore di stringhe

Numero di stringhe

Lunghezzza massima di ciascuna stringa (compreso terminatore nullo)

Sintassi

- Definizione

```
char vett[MAX][LUN] ;
```

- Accesso al singolo carattere

```
vett[i][j]
```

Carattere (j+1)-esimo della stringa (i+1)-esima, da usarsi per l'elaborazione carattere-per-carattere

Sintassi

- Definizione

```
char vett[MAX][LUN] ;
```

- Accesso

Stringa (i+1)-esima, da usarsi con le funzioni di libreria

- Accesso all'intera stringa

```
vett[i]
```


Esempio 1

- Dato un vettore di stringhe, determinare quante volte è presente la lettera 'A' (maiuscola o minuscola)

```
cont = 0 ;
for(i=0; i<MAX; i++)
{
   for(j=0; vett[i][j]!=0; j++)
   {
      if(toupper(vett[i][j])=='A')
         cont++ ;
   }
}
```


Esempio 2

- Dato un vettore di stringhe, determinare se esistono due stringhe identiche

```
uguali = 0 ;
for(i=0; i<MAX; i++)
{
   for(k=i+1; k<MAX; k++)
   {
      if(strcmp(vett[i], vett[k])==0)
         uguali=1 ;
   }
}
```

Occupazione variabile

- In un vettore di stringhe, ogni riga (ogni stringa) è intrinsecamente un vettore di caratteri ad occupazione variabile
  - Terminatore nullo per indicare la lunghezza effettiva
- Il numero di stringhe effettivamente memorizzato potrebbe non riempire l'intero vettore
  - Variabile intera che indica l'effettiva occupazione del vettore

Esempio

```
const int MAX = 5 ;
const int LUN = 9 ;

char nomi[MAX][LUN+1] ;
int N ;
```

N=3

	0	1	2	3	4	5	6	7	8	9
0	F	u	l	v	i	o	\0	x	!	w
1	A	n	t	o	n	i	o	\0	.	z
2	C	r	i	s	t	i	n	a	\0	u
3	e	4	1	)	a	\0	5	g	r	d
4	1	%	<	d	d	e	g	$	2	e

Errore frequente

- Confondere una stringa (vettore di caratteri) con un vettore di stringhe (matrice di caratteri)

```
char s[LUN+1] ;
```

- s[i] è un singolo carattere
- s è l'intera stringa

```
char v[MAX][LUN+1] ;
```

- v[i][j] è il singolo carattere
- v[i] è un'intera stringa
- v è l'intera matrice

Vettori di stringhe

I/O di vettori di stringhe

Stampa (1/2)

- La stampa del contenuto di un vettore di stringhe si ottiene semplicemente stampando ciascuno degli elementi
  - Si può utilizzare `puts` oppure `printf`

Stampa (2/2)

vettstr.c

```
for(i=0; i<N; i++)
{
    puts(vett[i]) ;
}
```

Lettura

- Acquisire da tastiera un vettore di stringhe
- Un ciclo per ciascuna delle stringhe da leggere
  - Lunghezza nota a priori
  - Lunghezza determinata dalla lettura di un certo dato (es.: "FINE")
- Acquisizione, nel ciclo, di ciascuna delle stringhe
  - Utilizzo della funzione `gets`
  - Eventualmente, lettura in una stringa d'appoggio per la verifica di correttezza, prima di ricopiare nel vettore destinazione

Dimensione nota a priori (1/2)

vettstr.c

```
char vett[MAX][LUN+1] ;
char s[250] ;
. . .

do {
    printf("Quante stringhe? ") ;
    gets(s) ;
    N = atoi(s) ;
    if(N<1 || N>MAX)
        printf("Valore errato: deve
          essere tra 1 e %d\n", MAX) ;
} while(N<1 || N>MAX) ;
```


Dimensione nota a priori (2/2)

vettstr.c

```
for(i=0; i<N; i++)
{   printf("Stringa %d: ", i+1) ;
    gets(s) ;
    if (strlen(s)==0)
    {   printf("Vuota - ripeti\n");
        i-- ;
    }
    else if(strlen(s)>LUN)
    {   printf("Troppo lunga -
                max %d chr\n", LUN) ;
        i-- ;
    }
    else
        strcpy(vett[i], s) ;
}
```


Lettura terminata da "FINE"

vettstr.c

```
N = 0 ;
end = 0 ;
do {
    printf("Stringa %d: ", N+1) ;
    gets(s) ;
    if (strlen(s)==0)
        printf("Vuota - ripeti\n");
    else if(strlen(s)>LUN)
        printf("Troppo lunga\n") ;
    else if(strcmp(s, "FINE")==0)
        end = 1 ;
    else
    {   strcpy(vett[N], s) ;
        N++ ;
    }
} while(end==0) ;
```

Matrici – Vettori di stringhe

Esercizi proposti

Esercizi proposti

- Esercizio "Statistiche testo"
- Esercizio "Magazzino"

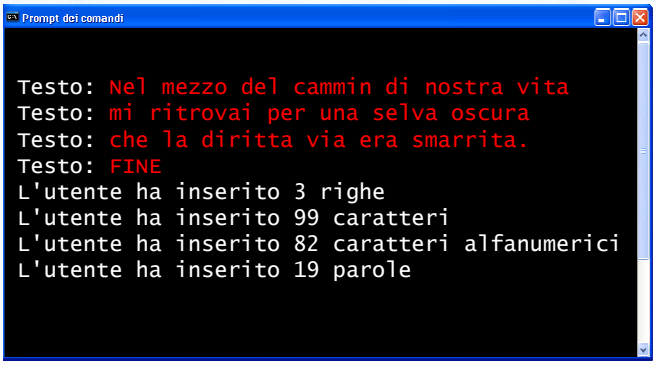
Esercizi proposti

Esercizio "Statistiche testo"

Esercizio "Statistiche testo"

- Un utente inserisce una serie di frasi da tastiera, su più righe
- L'inserimento termina quando l'utente inserisce la parola FINE su una riga da sola
- Il programma deve determinare:
  - 1) Quante righe sono state inserite dall'utente
  - 2) Quanti caratteri sono stati inseriti
  - 3) Quanti caratteri alfanumerici sono stati inseriti
  - 4) Quante parole sono state inserite

Analisi



Soluzione (1/5)


```
const int MAXRIGHE = 2000 ;
const int LUN = 80 ;

char testo[MAXRIGHE][LUN] ;
int Nrighe ; /* righe inserite */
char riga[LUN*10] ;
int i, j ;
int caratteri, caralfa, parole ;
```

Soluzione (2/5)

statistichetesto.c

```
Nrighe = 0 ;
do {
   printf("Testo: ");
   gets(riga) ;

   if( strcmp(riga, "FINE")!=0 )
   {
      /*copia riga in testo[Nrighe] ;*/
      strcpy( testo[Nrighe] , riga ) ;

      Nrighe++ ;
   }
} while( strcmp(riga, "FINE")!=0 ) ;
```


Soluzione (3/5)

statistichetesto.c

```
printf("L'utente ha inserito"
       " %d righe\n", Nrighe);

caratteri = 0 ;
for(i=0; i<Nrighe; i++)
    caratteri = caratteri +
                strlen(testo[i]) ;

printf("L'utente ha inserito"
       " %d caratteri\n", caratteri) ;
```


Soluzione (4/5)

statistichetesto.c

```
caralfa = 0 ;
for(i=0; i<Nrighe; i++)
{
    for(j=0; testo[i][j]!=0; j++)
    {
        if( isalnum(testo[i][j] ) )
            caralfa++ ;
    }
}

printf("L'utente ha inserito "
       "%d caratteri alfanumerici\n",
       caralfa) ;
```


Soluzione (5/5)

statistichetesto.c

```
parole = 0 ;
for(j=0; j<Nrighe; j++)
{
   for(i=0; testo[j][i]!=0; i++)
   {
      if( isalpha(testo[j][i]) &&
      (i==0 || !isalpha(testo[j][i-1])))
      {
         parole ++ ;
      }
   }
}

printf("L'utente ha inserito "
       "%d parole\n", parole) ;
```


Esercizi proposti

Esercizio "Magazzino"

Esercizio "Magazzino" (1/2)

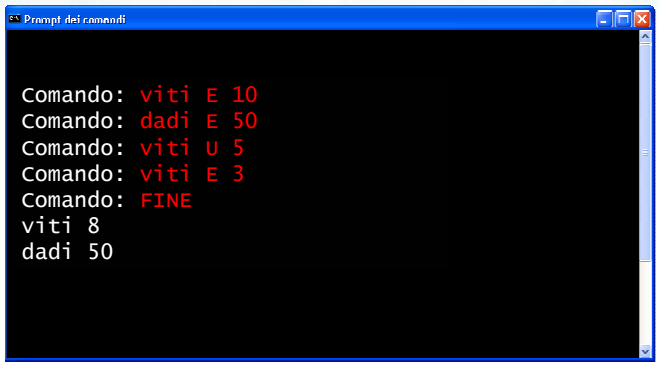
- Un'azienda deve tenere traccia dei beni presenti in un magazzino
- L'utente inserisce da tastiera dei "comandi" nel seguente formato:
  - `bene EU quantità`

  dove:
  - `bene` è il nome di un bene
  - `EU` è la lettera `'E'` per entrata, `'U'` per uscita
  - `quantità` è la quantità di bene entrata o uscita

Esercizio "Magazzino" (2/2)

- L'utente termina il caricamento inserendo un comando pari a "FINE". In tal caso il programma deve stampare le quantità di beni presenti a magazzino

Analisi



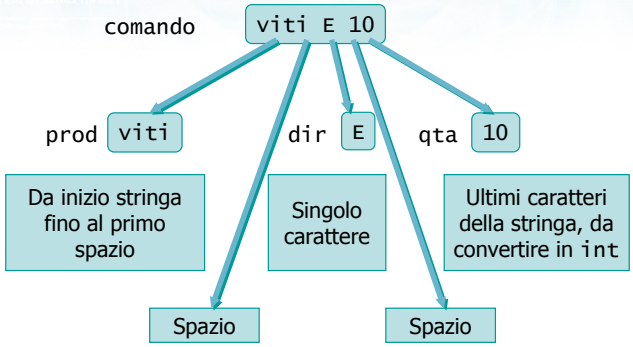
Problemi

- Estrazione dei 3 parametri dal comando immesso dall'utente
  - Nome prodotto: stringa
  - Direzione: carattere
  - Quantità: intero

`viti E 10`

- Memorizzazione di nomi di prodotti e quantità relative
  - Vettori paralleli
- Aggiornamento delle giacenze

Stringa di comando



Analisi del comando (1/2)

magazzino.c

```
i = 0 ;
while(comando[i]!=' ')
{
    prod[i] = comando[i] ;
    i++ ;
}
prod[i] = 0 ;

/* salta lo spazio */
i++ ;

dir = comando[i] ; /* 'E' o 'U' */
```


Analisi del comando (2/2)

magazzino.c

```
/* salta lo spazio */
i++ ;

/* da qui a fine: la quantità */
j = 0 ;
while(comando[i]!=0)
{
    temp[j] = comando[i] ;
    i++ ;
    j++ ;
}
temp[j] = 0 ;
qta = atoi(temp) ;
```

Osservazione

- L'analisi di una stringa di comando composta da più campi è sempre un'operazione complessa
- Nel caso in cui si dovessero gestire delle condizioni anomale (es. più spazi consecutivi) o di errore (es. manca la quantità), il codice diverrebbe estremamente articolato
- Vedremo più avanti la funzione `sscanf` che può aiutare in questi casi

Rappresentazione del magazzino

- Occorre memorizzare, per ciascun prodotto
  - Il nome
  - La quantità corrente
- Si possono usare due vettori "paralleli"

```
char prodotti[MAX][LUN+1] ;
int quantita[MAX] ;

int N ; /* occupazione effettiva
           vettori prodotti e quantita */
```


Inserimento di un prodotto

- Determinare se il prodotto è già in magazzino
  - Ricerca del nome del prodotto nel vettore `prodotto`
- Se c'è già, incrementa la quantità
- Se non c'è ancora, aggiungi una riga ai vettori

Inserimento di un prodotto


```
/* trova la posizione del prodotto */
trovato = -1 ;
for(i=0; i<N; i++)
    if(strcmp(prodotti[i], prod)==0)
        trovato = i ;

if( trovato != -1 ) /* esiste già */
    quantita[trovato] =
        quantita[trovato] + qta ;

else /* prodotto nuovo */
{   strcpy(prodotti[N], prod) ;
    quantita[N] = qta ;
    N++ ;
}
```


Eliminazione di un prodotto

- Determinare se il prodotto è già in magazzino
  - Ricerca del nome del prodotto nel vettore `prodotto`
- Se c'è già, decrementa la quantità
- Se non c'è ancora, errore

Eliminazione di un prodotto


```
/* trova la posizione del prodotto */
trovato = -1 ;
for(i=0; i<N; i++)
    if(strcmp(prodotti[i], prod)==0)
        trovato = i ;

if( trovato == -1 )
    printf("Prodotto %s non "
     "trovato in magazzino\n", prod);
else
    quantita[trovato] =
        quantita[trovato] - qta ;
```

Matrici – Vettori di stringhe

Sommario

Argomenti trattati

- Matrici bi-dimensionali e pluri-dimensionali
- Matrici di numeri interi e reali
  - Definizione
  - Operazioni frequenti
- Matrici di caratteri
- Vettori di stringhe
  - Caso particolare di matrici di caratteri
  - Operazioni frequenti

Tecniche di programmazione

- Usare matrici per memorizzare schiere di dati numerici
- Usare vettori di stringhe per memorizzare stringhe di testo di lunghezza variabile
- Compiere operazioni di ricerca nei vettori di stringhe

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
Funzioni

Funzioni

- Tipi di dato
- Funzioni in C
- Modifica dei parametri
- Parametri "by reference"
- La funzione `main()`
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitoli 2 e 4
  - Cabodi, Quer, Sonza Reorda: capitoli 3 e 7
  - Dietel & Dietel: capitolo 5
- Dispense
  - Scheda: "Tipi di dato in C"
  - Scheda: "Funzioni in C"

Funzioni

Tipi di dato

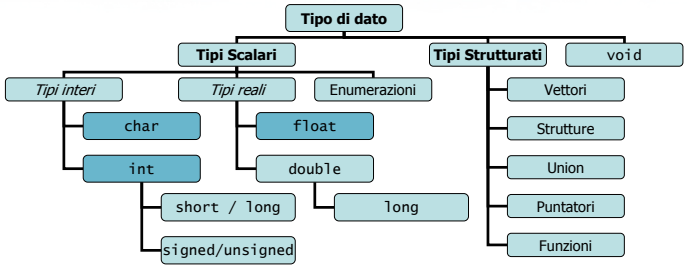
Tipi di dato

- I tipi scalari in C
- Input/output dei tipi scalari
- Conversioni di tipo

Tipi di dato

I tipi scalari in C

Il sistema dei tipi C



I tipi interi in C

Tipo	Descrizione	Esempi
`char`	Caratteri ASCII	`'a' '7' '!'`
`int`	Interi…	`+2 -18 0 +24221`
`short int`	… con meno bit	
`long int`	… con più bit	
`unsigned int`	Interi senza segno…	`0 1 423 23234`
`unsigned short int`	… con meno bit	
`unsigned long int`	… con più bit	

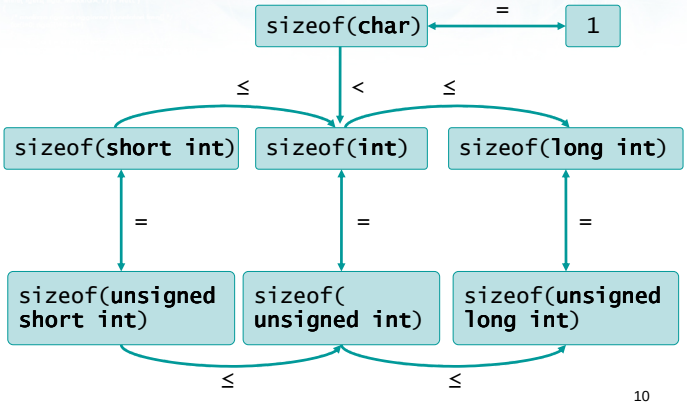
Quanti bit?

- Lo standard C non specifica l'ampiezza, in bit, dei tipi di dato fondamentali
- L'operatore `sizeof` può essere usato per ricavare una costante pari al numero di byte occupato da ciascun tipo

```
sizeof(char)
```

```
sizeof(int)
```

Specifiche del C



Intervallo di rappresentazione

Tipo	Min	Max
char	CHAR_MIN	CHAR_MAX
int	INT_MIN	INT_MAX
short int	SHRT_MIN	SHRT_MAX
long int	LONG_MIN	LONG_MAX
unsigned int	0	UINT_MAX
unsigned short int	0	USHRT_MAX
unsigned long int	0	ULONG_MAX

```
#include <limits.h>
```


Compilatori a 32 bit

Tipo	N. Bit	Min	Max
char	8	-128	127
int	32	-2147483648	2147483647
short int	16	-32768	32767
long int	32	-2147483648	2147483647
unsigned int	32	0	4294967295
unsigned short int	16	0	65536
unsigned long int	32	0	4294967295

I tipi reali in C

Tipo	Descrizione
float	Numeri reali in singola precisione
double	Numeri reali in doppia precisione
long double	Numeri reali in massima precisione

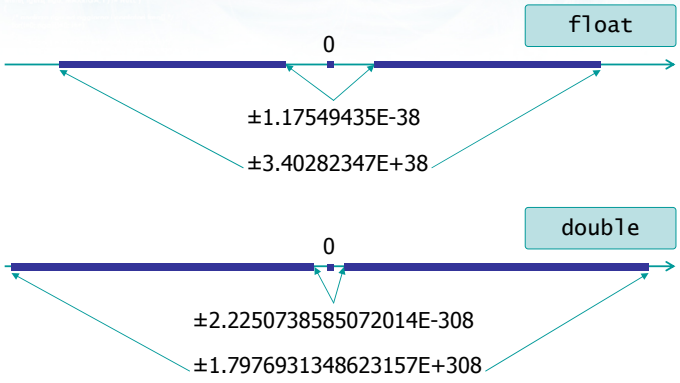
$$A = \overbrace{\pm}^{segno}\ \underbrace{1.mmmmm}_{mantissa} \times 2^{\overbrace{\pm eeeeee}^{esponente}}$$

Numero di bit

Tipo	Dimensione	Mantissa	Esponente
float	32 bit	23 bit	8 bit
double	64 bit	53 bit	10 bit
long double	≥ double		

$$A = \overbrace{\pm}^{segno}\ \underbrace{1.mmmmm}_{mantissa} \times 2^{\overbrace{\pm eeeeee}^{esponente}}$$

Intervallo di rappresentazione



Tipi di dato

Input/output dei tipi scalari

Input/output

- I diversi tipi scalari visti sono utilizzabili con le normali funzioni `scanf`/`printf`, adottando degli specifici indicatori di formato
- Utilizzando la funzione `gets` per l'input, si possono usare le funzioni di conversione `ato...`

Specificatori di formato

Tipo	scanf	printf
char	%c %[...]	%c %d
int	%d	%d
short int	%hd	%hd %d
long int	%ld	%ld
unsigned int	%u %o %x	%u %o %x
unsigned short int	%hu	%hu
unsigned long int	%lu	%lu
float	%f	%f %g
double	%lf	%f %g

Funzioni di conversione

```
char line[80] ;
int x ;

gets(line) ;
x = atoi(line) ;
```

```
char line[80] ;
float x ;

gets(line) ;
x = atof(line) ;
```

```
char line[80] ;
long int x ;

gets(line) ;
x = atol(line) ;
```

```
char line[80] ;
double x ;

gets(line) ;
x = atof(line) ;
```


Tipi di dato

Conversioni di tipo

Conversioni di tipo (1/2)

- Nel linguaggio C è possibile combinare, nella stessa espressione, variabili di tipo diverso
  - I due operandi di un operatore aritmetico possono avere tipi diversi

```
int a ;
long int b, c ;
c = b + a ;
```

```
double prod ;
float v[N] ;
prod = prod * v[i] ;
```

Conversioni di tipo (2/2)

- La variabile di destinazione di un'assegnazione può avere tipo diverso dal tipo dell'espressione

```
int a ;
long int b ;
b = a ;
```

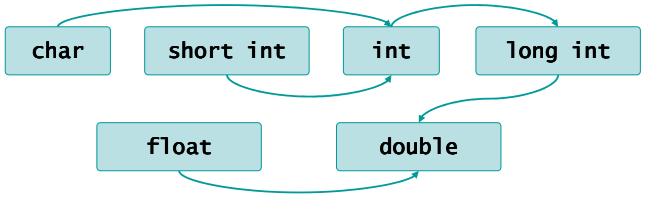
```
double prod ;
float v[N] ;
prod = v[0] ;
```


Tipologie di conversioni

- Per calcolare tali tipi di espressioni, il linguaggio C applica tre tipi di conversioni:
  - Conversioni "automatiche" verso il tipo più capiente, basate sul principio di **promozione del tipo**
  - Arrotondamenti e troncamenti, in caso di assegnazioni "forzate" a tipi meno capienti
  - Conversioni "esplicite", basate sull'operatore di **typecasting**

Promozione del tipo

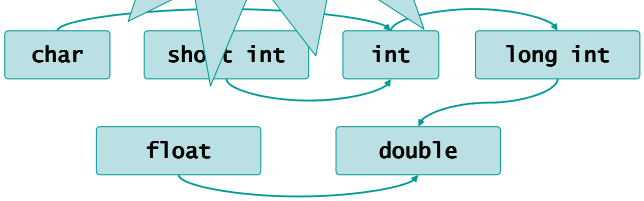
- Se i due operandi di un operatore aritmetico hanno tipo diverso, l'operando del tipo più limitato viene convertito al tipo dell'operando più esteso



Promozione del tipo

- Se i d ore aritmetico
  lim po più
  est dell'operando più

Non si perde mai precisione.
Il C converte automaticamente verso i tipi più capienti



Troncamento del risultato

- Nell'operatore di assegnazione ci possono essere 3 casi:

```
var = expr ;
```

  - La variabile destinazione ha lo stesso tipo dell'espressione calcolata
  - La variabile destinazione ha un tipo più ampio del tipo dell'espressione calcolata
    - Si promuove il tipo dell'espressione al tipo della variabile destinazione
  - La variabile destinazione ha un tipo più ristretto del tipo dell'espressione calcolata
    - Si approssima il tipo dell'espressione al tipo della variabile destinazione, perdendo precisione

Conversioni esplicite

- Qualora si vogliano modificare le regole predefinite di promozione e troncamendo dei tipi, il C mette a disposizione un operatore di **conversione esplicita di tipo**
  - Typecasting

```
(nuovotipo)expr
```

  - Converte l'espressione `expr` dal suo tipo nativo, al tipo desiderato `nuovotipo`
    - Più capiente
    - Meno capiente: troncamento o approssimazione

Esempio 1

```
double media ;
int somma, N ;

media = somma / N ; /* no */

media = (double)somma / N ;
```


Esempio 2

```
int voto ;
float parte1, parte2, parte3 ;


voto = (int) ((parte1 +
        parte2 + parte3)/3) ;
```


Esempio 3

```
int voto ;
float parte1, parte2, parte3 ;
float media ;

/* arrotondamento all'intero
   più vicino */

media = (parte1 +
         parte2 + parte3)/3 ;

voto = (int) (media + 0.5) ;
```

Funzioni

Funzioni in C

Funzioni in C

- Il concetto di funzione
- Parametri formali e attuali
- Il valore di ritorno
- Definizione e chiamata di funzioni
- Passaggio dei parametri
- Corpo della funzione

Funzioni in C

Il concetto di funzione

Strategie di programmazione

- Riuso di codice esistente
  - Funzionalità simili in programmi diversi
  - Funzionalità ripetute all'interno dello stesso programma
  - Minore tempo di sviluppo
  - Frammenti di codice già verificati
  - Utilizzo di parti di codice scritte da altri
    - Funzioni di libreria
    - Sviluppo collaborativo

Come riusare il codice? (1/3)

- Copia-e-incolla
  - Semplice, ma poco efficace
  - Occorre adattare il codice incollato, ritoccando i nomi delle variabili e costanti utilizzati
  - Se si scopre un errore, occorre correggerlo in tutti i punti in cui è stato incollato
  - Nel listato finale non è evidente che si sta riutilizzando la stessa funzionalità
  - Occorre disporre del codice originario
  - Occorre capire il codice originario

Come riusare il codice? (2/3)

- Definizione di funzioni
  - Dichiarazione esplicita che una certa funzionalità viene utilizzata in più punti
  - Si separa la definizione della funzionalità rispetto al punto in cui questa viene utilizzata
  - La stessa funzione può essere usata più volte con parametri diversi

Come riusare il codice? (3/3)

- Ogni miglioramento o correzione è automaticamente disponibile in tutti i punti in cui la funzione viene usata
- Nel listato finale è evidente che si sta riutilizzando la stessa funzionalità
- Non occorre disporre del codice originario
- Non occorre capire il codice originario

Principio di funzionamento (1/3)

```
int main(void)
{
  int x, y ;

/* leggi un numero
   tra 50 e 100 e
   memorizzalo
   in x */
/* leggi un numero
   tra 1 e 10 e
   memorizzalo
   in y */

  printf("%d %d\n",
    x, y ) ;
}
```


Principio di funzionamento (2/3)

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```


Principio di funzionamento (3/3)

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

```
int leggi(int min,
          int max)
{
  int v ;

  do {
    scanf("%d", &v) ;
  } while( v<min ||
           v>max) ;

  return v ;
}
```


Principio di funzionamento (3/3)

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

min=50 max=100

```
int leggi(int min,
          int max)
{
  int v ;

  do {
    scanf("%d", &v) ;
  } while( v<min ||
           v>max) ;

  return v ;
}
```


Principio di funzionamento (3/3)

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

Chiamante

min=50 max=100

```
int leggi(int min,
          int max)
{
  int v ;

  do {
    scanf("%d", &v) ;
  } while( v<min ||
           v>max) ;

  return v ;
}
```

Chiamato

Principio di funzionamento (3/3)

```
int main(void)
{
  int x, y ;

  x = leggi(50, 100) ;
  y = leggi(1, 10) ;

  printf("%d %d\n",
    x, y ) ;
}
```

min=1 max=10

```
int leggi(int min,
          int max)
{
  int v ;

  do {
    scanf("%d", &v) ;
  } while( v<min ||
           v>max) ;

  return v ;
}
```


Sommario

- La definizione di una **funzione** delimita un frammento di codice riutilizzabile più volte
- La funzione può essere **chiamata** più volte
- Può ricevere dei **parametri** diversi in ogni chiamata
- Può restituire un **valore di ritorno** al **chiamante**
  - Istruzione `return`

Miglioramento della funzione

```
int leggi(int min, int max)
{
  char riga[80] ;
  int v ;

  do {
    gets(riga) ;
    v = atoi(riga) ;
    if(v<min) printf("Piccolo: min %d\n", min) ;
    if(v>max) printf("Grande: max %d\n", max) ;
  } while( v<min || v>max) ;

  return v ;
}
```


Funzioni in C

Parametri formali e attuali

Parametri di una funzione

- Le funzioni possono ricevere dei parametri dal proprio chiamante
- Nella funzione:
  - Parametri **formali**
  - Nomi "interni" dei parametri

```
int leggi(int min,
          int max)
{
   ...
}
```


Parametri di una funzione

- Le funzioni possono ricevere dei parametri dal proprio chiamante
- Nella funzione:
  - Parametri **formali**
  - Nomi "interni" dei parametri
- Nel chiamante:
  - Parametri **attuali**
  - Valori effettivi (costanti, variabili, espressioni)

```
int leggi(int min,
          int max)
{
   ...
}
```

```
int main(void)
{
   ...
   y = leggi(1, 10);
   ...
}
```

Parametri formali (1/2)

- Uno o più parametri
- Tipo del parametro
  - Tipo scalare
  - Vettore o matrice
- Nome del parametro

```
int leggi(int min,
          int max)
{
   ...
}
```

- Nel caso in cui la funzione non abbia bisogno di parametri, si usa la parola chiave `void`

```
int stampa_menu(void)
{
   ...
}
```


Parametri formali (2/2)

- Per parametri vettoriali esistono 3 sintassi alternative
  - `int v[]`
  - `int v[MAX]`
  - `int *v`
- Per parametri matriciali
  - `int m[RIGHE][COL]`
  - `int m[][COL]`

```
int leggi(int v[])
{
   ...
}
```

```
int sudoku(int m[9][9])
{
   ...
}
```


Avvertenza (1/2)

- Il valore della dimensione del vettore (es. `MAX`)
  - Viene totalmente ignorato dal meccanismo di chiamata
  - Non sarebbe comunque disponibile alla funzione chiamata
  - Meglio per chiarezza ometterlo
  - Si suggerisce di passare un ulteriore parametro contenente l'occupazione del vettore

Avvertenza (2/2)

- Nel caso di matrici
  - Il secondo parametro (es. `COL`) è obbligatorio e deve essere una costante
  - Il primo parametro viene ignorato e può essere omesso
  - Per matrici pluri-dimensionali, occorre specificare tutti i parametri tranne il primo

Parametri attuali (1/2)

- Uno o più valori, in esatta corrispondenza con i parametri formali
- Tipi di dato compatibili con i parametri formali
- È possibile usare
  - Costanti
  - Variabili
  - Espressioni

```
int main(void)
{
   y = leggi(1, 10);
}
```

```
int main(void)
{
   y = leggi(a, b);
}
```

```
int main(void)
{
   y = leggi(a+b+1,
             c*2);
}
```

Parametri attuali (2/2)

- I nomi delle variabili usate non hanno alcuna relazione con i nomi dei parametri formali

```
int main(void)
{
   ...
   y = leggi(a, b);
   ...
}
```

min=a max=b

- Le parentesi sono sempre necessarie, anche se non vi sono parametri

```
int main(void)
{
   ...
   y = stampa_menu();
   ...
}
```

Funzioni in C

Il valore di ritorno

Valore di ritorno

- Ogni funzione può ritornare un valore al proprio chiamante
- Il tipo del valore di ritorno deve essere **scalare**
- L'istruzione `return`
  - Termina l'esecuzione della funzione
  - Rende disponibile il valore al chiamante

```
int leggi(int min,
          int max)
{
  int v ;

  scanf("%d", &v) ;

  return v ;
}
```

```
int main(void)
{
  ...
  y = leggi(a, b) ;
  ...
}
```

Tipo del valore di ritorno

- Valore scalare
  - `char`, `int` (o varianti), `float`, `double`
- Tipi avanzati
  - Puntatori, struct
- Nessun valore ritornato
  - `void`

```
double sqrt(double a)
{
   ...
}
```

```
void stampa_err(int x)
{
   ...
}
```

L'istruzione `return` (1/2)

- Restituisce il valore
  - Costante
  - Variabile
  - Espressione
- Il tipo deve essere compatibile con il tipo dichiarato per il valore di ritorno
- Sintassi
  - `return x ;`
  - `return(x) ;`
- L'esecuzione della funzione viene interrotta

L'istruzione `return` (2/2)

- Per funzioni che non ritornano valori (`void`):
  - `return ;`
- Il raggiungimento della fine del corpo della funzione } equivale ad un'istruzione `return` senza parametri
  - Permesso solo per funzioni `void`
- Per funzioni non-`void`, è obbligatorio che la funzione ritorni **sempre** un valore

Nel chiamante...

- Il chiamante può:
  - Ignorare il valore ritornato
    - `scanf("%d", &x) ;`
  - Memorizzarlo in una variabile
    - `y = sin(x) ;`
  - Utilizzarlo in un'espressione
    - `if ( sqrt(x*x+y*y)>z ) ...`

Convenzioni utili

- Le funzioni di tipo matematico ritornano sempre un valore `double`
- Le funzioni che non devono calcolare un valore (ma effettuare delle operazioni, per esempio) ritornano solitamente un valore `int`
  - Valore di ritorno == 0 ⇒ tutto ok
  - Valore di ritorno != 0 ⇒ si è verificato un errore
- Molte eccezioni importanti: `strcmp`, `scanf`, ...

Funzioni in C

Definizione e chiamata di funzioni

Sintassi C per le funzioni

- Il linguaggio C prevede 3 distinti momenti :
  - La dichiarazione (prototipo o *function prototype*)
    - L'interfaccia della funzione
    - Solitamente: prima del `main()`
  - La definizione
    - L'implementazione della funzione
    - Solitamente: al fondo del file, dopo il `main()`
  - La chiamata
    - L'utilizzo della funzione
    - Solitamente: dentro il corpo del `main()` o di altre funzioni

Dichiarazione o prototipo

```
int leggi(int min, int max) ;
```

- Tipo del valore di ritorno
- Nome della funzione
- Tipi e nomi dei parametri formali
- Punto-e-virgola

Scopo del prototipo

- Dichiarare che esiste una funzione con il nome dato, e dichiarare i tipi dei parametri formali e del valore di ritorno
- Dal prototipo in avanti, si può chiamare tale funzione dal corpo di qualsiasi funzione (compreso il `main`)
- Non è errore se la funzione non viene chiamata
- I file `.h` contengono centinaia di prototipi delle funzioni di libreria
- I prototipi sono posti solitamente ad inizio file

Definizione o implementazione

- Tipo del valore di ritorno
- Nome della funzione
- Tipi e nomi dei parametri formali

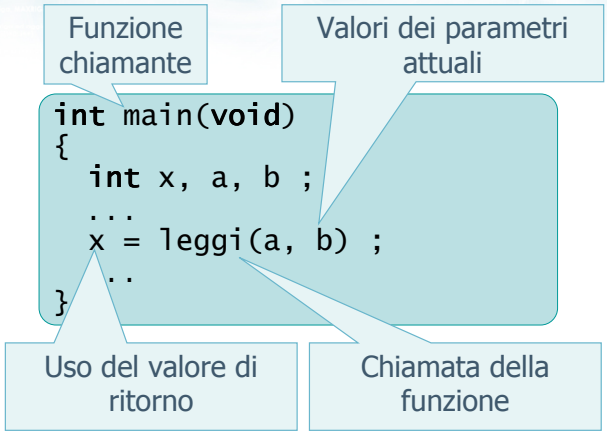
```
int leggi(int min, int max)
{
   . . . codice della funzione . . .
}
```

- Corpo della funzione { ... }
- Nessun punto-e-virgola

Scopo della definizione

- Contiene il codice vero e proprio della funzione
- Può essere posizionata ovunque nel file (al di fuori del corpo di altre funzioni)
- Il nome della funzione ed i tipi dei parametri e del valore di ritorno devono coincidere con quanto dichiarato nel prototipo
- Il corpo della funzione può essere arbitrariamente complesso, e si possono chiamare altre funzioni

Chiamata o invocazione



Meccanismo di chiamata

- Le espressioni corrispondenti ai parametri attuali vengono valutate (e ne viene calcolato il valore numerico)
  - Compaiono le variabili del chiamante
- I valori dei parametri attuali vengono copiati nei parametri formali
- La funzione viene eseguita
- All'istruzione `return`, il flusso di esecuzione torna al chiamante
- Il valore di ritorno viene usato o memorizzato

Riassumendo...

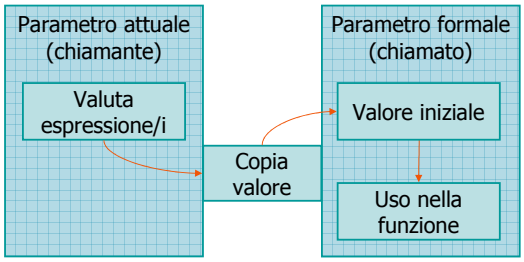
```
int leggi(int min, int max) ;

int main(void)
{
  int x, a, b ;
  ...
  x = leggi(a, b) ;
  ...
}

int leggi(int min, int max)
{
  ... codice della funzione ...
}
```


Funzioni in C

Passaggio dei parametri

Passaggio dei parametri

- Ogni volta che viene chiamata una funzione, avviene il trasferimento del valore corrente dei parametri attuali ai parametri formali



Conseguenza

- La funzione chiamata non ha assolutamente modo di
  - Conoscere il nome delle variabili utilizzate come parametri attuali
    - Ne conosce solo **il valore** corrente
  - Modificare il valore delle variabili utilizzate come parametri attuali
    - Riceve solamente una **copia** del valore
- Questo meccanismo è detto passaggio "by value" dei parametri
  - È l'unico possibile in C

Errore frequente

- Immaginare che una funzione possa modificare i valori delle variabili

```
void azzera(int x)
{
  x = 0 ;
}
```


Parametri di tipo vettoriale

- Il meccanismo di passaggio "by value" è chiaro nel caso di parametri di tipo scalare
- Nel caso di parametri di tipo array (vettore o matrice), il linguaggio C prevede che:
  - Un parametro di tipo array viene passato trasferendo una copia dell'indirizzo di memoria in cui si trova l'array specificato dal chiamante
  - Passaggio "by reference"

Conseguenza

- Nel passaggio di un vettore ad una funzione, il chiamato utilizzerà l'indirizzo a cui è memorizzato il vettore di partenza
- La funzione potrà quindi modificare il contenuto del vettore del chiamante
- Maggiori dettagli nella prossima lezione

Funzioni in C

Corpo della funzione

Variabili locali

- All'interno del corpo di una funzione è possibile definire delle **variabili locali**

```
int leggi(int min,
          int max)
{
  int v ;

  scanf("%d", &v) ;

  return v ;
}
```

Caratteristiche

- Le variabili locali sono accessibili solo dall'interno della funzione
- Le variabili locali sono indipendenti da eventuali variabili di ugual nome definite nel `main`
  - In ogni caso, dal corpo della funzione è impossibile accedere alle variabili definite nel `main`
- Le variabili locali devono avere nomi diversi dai parametri formali

Istruzioni eseguibili

- Il corpo di una funzione può contenere qualsiasi combinazione di istruzioni eseguibili
- Ricordare l'istruzione `return`

```
int leggi(int min,
          int max)
{
  int v ;

  scanf("%d", &v) ;

  return v ;
}
```

Funzioni

Parametri "by reference"

Parametri "by reference"

- Introduzione
- Operatori & e *
- Passaggio "by reference"
- Passaggio di vettori
- Esercizio "`strcpy`"

Parametri "by reference"

Introduzione

Passaggio dei parametri

- Il linguaggio C prevede il passaggio di parametri "by value"
  - Il chiamato non può modificare le variabili del chiamante
  - Il parametro formale viene inizializzato con una copia del valore del parametro attuale

Problemi

- Il passaggio "by value" risulta inefficiente qualora le quantità di dati da passare fossero notevoli
  - Nel caso del passaggio di vettori o matrici, il linguaggio C non permette il passaggio "by value", ma copia solamente l'indirizzo di partenza
  - Esempio: `strcmp`
- Talvolta è necessario o utile poter modificare il valore di una variabile nel chiamante
  - Occorre adottare un meccanismo per permettere tale modifica
  - Esempio: `scanf`

Soluzione

- La soluzione ad entrambi i problemi è la stessa:
  - Nel passaggio di vettori, ciò che viene passato è solamente l'indirizzo
  - Per permettere di modificare una variabile, se ne passa l'indirizzo, in modo che il chiamato possa modificare direttamente il suo contenuto in memoria
- Viene detto passaggio "by reference" dei parametri
  - Definizione impropria, in quanto gli indirizzi sono, a loro volta, passati "by value"

Parametri "by reference"

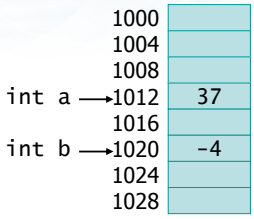
Operatori & e *

Operatori sugli indirizzi

- Per gestire il passaggio "by reference" dei parametri occorre
  - Conoscere l'indirizzo di memoria di una variabile
    - Operatore &
  - Accedere al contenuto di una variabile di cui si conosce l'indirizzo ma non il nome
    - Operatore *
- Prime nozioni della aritmetica degli indirizzi, che verrà approfondita in Unità successive

Operatore &

- L'operatore **indirizzo-di** restituisce l'indirizzo di memoria della variabile a cui viene applicato
  - &a è l'indirizzo 1012
  - &b è l'indirizzo 1020



Osservazioni

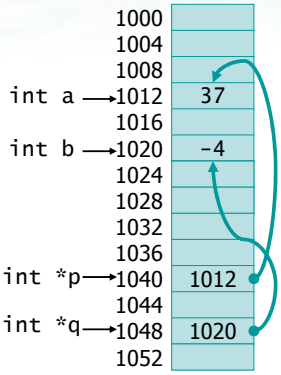
- L'indirizzo di una variabile viene deciso dal compilatore
- L'operatore & si può applicare solo a variabili singole, non ad espressioni
  - Non ha senso &(a+b)
  - Non ha senso &(3)
- Conoscere l'indirizzo di una variabile permette di leggerne o modificarne il valore senza conoscerne il nome

Variabili "puntatore"

- Per memorizzare gli indirizzi di memoria, occorre definire opportune variabili di tipo "indirizzo di..."
- Nel linguaggio C si chiamano **puntatori**
- Un puntatore si definisce con il simbolo *
  - `int *p ; /* puntatore ad un valore intero */`
  - `float *q ; /* puntatore ad un valore reale */`

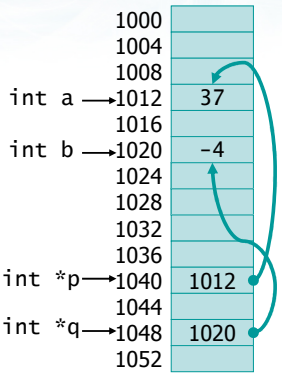
Esempio

```
int main(void)
{
  int a, b ;
  int *p, *q ;

  a = 37 ;
  b = -4 ;

  p = &a ;
     /* p "punta a" a */

  q = &b ;
     /* q "punta a" b */

}
```



Operatore *

- L'operatore di **accesso indiretto** permette di accedere, in lettura o scrittura, al valore di una variabile di cui si conosce l'indirizzo
  - `*p` equivale ad a
  - `*q` equivale a b
  - `*p = 0 ;`
  - `if( *q > 0 ) ...`



Costrutti frequenti

Costrutto	Significato
`int x ;`	x è una variabile intera
`int *p ;`	p è un puntatore a variabili intere
`p = &x ;`	p punta ad x
`*p = 0 ;`	Azzera la variabile puntata da p (cioè x)
`b = *p ;`	Leggi il contenuto della variabile puntata da p e copialo in b

Parametri "by reference"

Passaggio "by reference"

Passaggio "by reference"

- Obiettivo: passare ad una funzione una variabile, in modo tale che la funzione la possa modificare
- Soluzione:
  - Definire un parametro attuale di tipo puntatore
  - Al momento della chiamata, passare l'indirizzo della variabile (anziché il suo valore)
  - All'interno del corpo della funzione, fare sempre accesso indiretto alla variabile di cui è noto l'indirizzo

Esempio: "Azzera"

- Scrivere una funzione `azzera`, che riceve un parametro di tipo intero (by reference) e che azzera il valore di tale parametro

Soluzione

```
void azzera( int *v ) ;
```

```
int main( void )
{
  int x ;
  ...
  azzera(&x) ;
  ...
}
```

```
void azzera( int *v )
{
  *v = 0 ;
}
```

Esempio: "Scambia"

- Scrivere una funzione `scambia`, che riceve due parametri di tipo intero (by reference) e che scambia tra di loro i valori in essi contenuti

Soluzione

```
void scambia( int *p, int *q ) ;
```

```
int main( void )
{
  int a,b ;
  ...
  scambia(&a, &b) ;
  ...
}
```

```
void scambia( int *p, int *q )
{
  int t ;
  t = *p ;
  *p = *q ;
  *q = t ;
}
```


Osservazione

- Il meccanismo di passaggio by reference spiega (finalmente!) il motivo per cui nella funzione `scanf` è necessario specificare il carattere & nelle variabili lette
- Le variabili vengono passate by reference alla funzione `scanf`, in modo che questa possa scrivervi dentro il valore immesso dall'utente

Parametri "by reference"

Passaggio di vettori

Passaggio di vettori e matrici

- Nel linguaggio C, il nome di un array (vettore o matrice) è automaticamente sinonimo del puntatore al suo primo elemento

```
int main(void)
{
  int v[10] ;
  int *p ;

  p = & v[0] ;

}
```

p e v sono del tutto equivalenti

Conseguenze

- Quando il parametro di una funzione è di tipo array (vettore o matrice)
  - L'array viene passato direttamente "by reference"
  - Non è necessario l'operatore & per determinare l'indirizzo
    - È sufficiente il nome del vettore
  - Non è necessario l'operatore * per accedere al contenuto
    - È sufficiente l'operatore di indicizzazione []
  - Non è possibile, neppure volendolo, passare un array "by value"

Esercizio "Duplicati"

- Scrivere una funzione che, ricevendo due parametri
  - Un vettore di `double`
  - Un intero che indica l'occupazione effettiva di tale vettore

  possa determinare se vi siano valori duplicati in tale vettore
- La funzione ritornerà un intero pari a 1 nel caso in cui vi siano duplicati, pari a 0 nel caso in cui non ve ne siano

Soluzione (1/3)

```
int duplicati(double v[], int N) ;
/*
Riceve in ingresso il vettore v[] di double
che contiene N elementi (da v[0] a v[N-1])

Restituisce 0 se in v[] non vi sono duplicati
Restituisce 1 se in v[] vi sono duplicati

Il vettore v[] non viene modificato
*/
```


Soluzione (2/3)

```
int duplicati(double v[], int N)
{
  int i, j ;

  for(i=0; i<N; i++)
  {
    for(j=i+1; j<N; j++)
    {
      if(v[i]==v[j])
        return 1 ;
    }
  }
  return 0 ;
}
```


Soluzione (3/3)

```
int main(void)
{
  const int MAX = 100 ;
  double dati[MAX] ;
  int Ndati ;
  int dupl ;

  ...
  dupl = duplicati(dati, Ndati) ;
  ...
}
```


Errore frequente

- Nel passaggio di un vettore occorre indicarne solo il nome

```
dupl = duplicati(dati, Ndati) ;
```

~~`dupl = duplicati(dati[], Ndati) ;`~~

~~`dupl = duplicati(dati[MAX], Ndati) ;`~~

~~`dupl = duplicati(dati[Ndati], Ndati) ;`~~

~~`dupl = duplicati(&dati, Ndati) ;`~~

Osservazione

- Nel caso dei vettori, il linguaggio C permette solamente il passaggio by reference
  - Ciò significa che il chiamato ha la possibilità di modificare il contenuto del vettore
- Non è detto che il chiamato effettivamente ne modifichi il contenuto
  - La funzione duplicati analizza il vettore senza modificarlo
  - Esplicitarlo sempre nei commenti di documentazione

Parametri "by reference"

Esercizio "strcpy"

Esercizio "strcpy"

- Si implementi, sotto forma di funzione, la ben nota funzione di libreria `strcpy` per la copia di due stringhe

Soluzione (1/2)

```
void strcpy(char *dst, char *src) ;
/*
Copia il contenuto della stringa src
nella stringa dst

Assume che src sia 0-terminata, e restituisce
dst in forma 0-terminata.

Assume che nella stringa dst vi sia spazio
sufficiente per la copia.

La stringa src non viene modificata.
*/
```


Soluzione (2/2)

```
void strcpy(char *dst, char *src)
{
  int i ;

  for(i=0; src[i]!=0; i++)
  {
    dst[i] = src[i] ;
  }
  dst[i] = 0 ;
}
```


Osservazione

- La funzione può essere dichiarata in due modi:
  - `void strcpy(char *dst, char *src)`
  - `void strcpy(char dst[], char src[])`
- Sono forme assolutamente equivalenti
- Tutte le funzioni di libreria che lavorano sulle stringhe accettano dei parametri di tipo `char *`

Funzioni

La funzione `main()`

La funzione `main()`

- Interfaccia con il sistema operativo
- Argomenti sulla linea di comando
- Parametri `argc` e `argv`
- Valore di ritorno del programma
- La funzione `exit`
- Esercizio "Calcolatrice"

La funzione `main()`

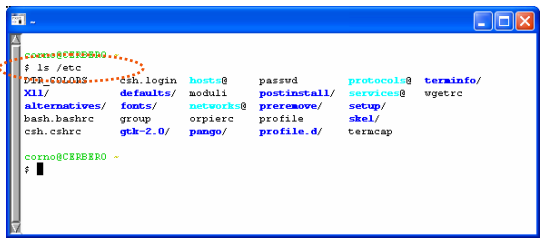
Interfaccia con il sistema operativo

La funzione `main()`

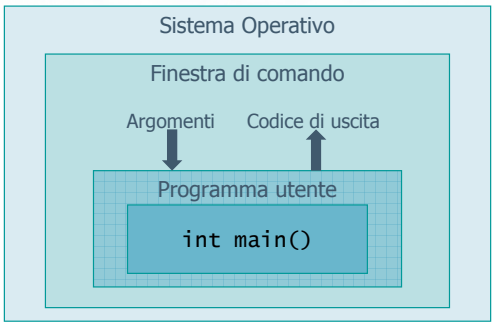
- La funzione `main()`, presente in tutti i programmi C, è una funzione come tutte le altre
- Unica particolarità: viene chiamata automaticamente dal Sistema Operativo, appena il programma viene avviato
  - Non esiste mai una chiamata esplicita a `main()`
  - L'interfaccia della funzione viene definita dalle caratteristiche del sistema operativo

Interfaccia in modalità "console"

- Ricordiamo che il linguaggio C si è evoluto con interfacce "a caratteri"
- L'attivazione di un programma avviene digitandone il nome in una finestra di comando



Il modello "console"



Interfaccia del programma

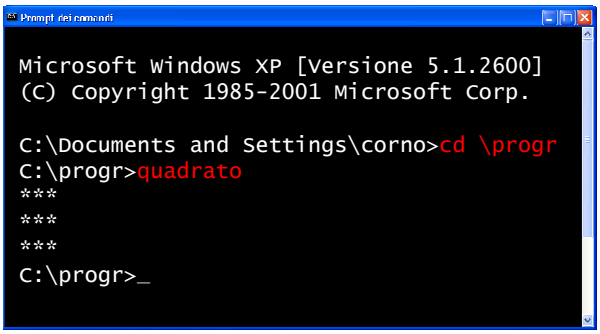
- La finestra di comando permette di passare al programma una serie di **argomenti**
  - Zero, una o più stringhe di testo
  - Utilizzate dal programma come dati in ingresso
- Al termine dell'esecuzione, il programma restituisce un **codice di uscita**
  - Numero intero
  - Indica eventuali condizioni di errore
- Durante l'esecuzione, il programma può accedere all'input (tastiera) e all'output (schermo)

La funzione `main()`

Argomenti sulla linea di comando

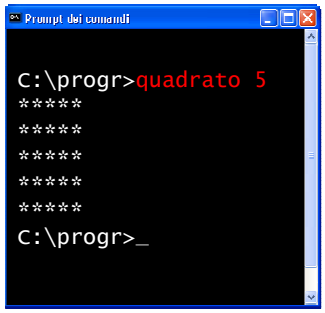
La linea di comando (1/2)

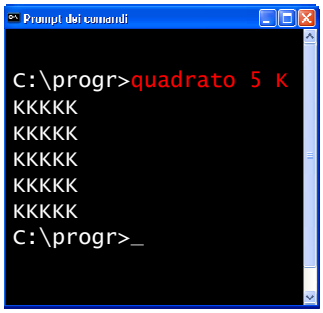
- L'attivazione di un programma avviene digitando il suo nome in una finestra di comando



La linea di comando (2/2)

- È possibile passare dei parametri all'attivazione del programma





Nomenclatura

- Parametri sulla linea di comando
  - Command line parameters
- Argomenti del programma
  - Program arguments
- Parametri di attivazione
- Parametri del main
- Argomenti del main
- Opzioni [sulla linea di comando]

Caratteristiche

- Numero variabile di parametri
  - Anche nessuno
- Tipo variabile
  - Numeri
  - Caratteri o Stringhe
- Il chiamante (sistema operativo) non ha modo di sapere quanti parametri servono al programma né di che tipo
  - Verranno trattati in modo standardizzato

Standardizzazione dei parametri

- Gli argomenti sulla linea di comando vengono trattati come un **vettore di stringhe**
- Il programma riceve
  - Una copia del vettore di stringhe
  - Un valore numerico che indica quante stringhe sono presenti

Esempi

```
C:\progr>quadrato
```

- Numero argomenti = 0

```
C:\progr>quadrato 5
```

- Numero argomenti = 1
- Argomento 1 = "5"

```
C:\progr>quadrato 5 K
```

- Numero argomenti = 2
- Argomento 1 = "5"
- Argomento 2 = "K"

La funzione `main()`

Parametri `argc` e `argv`

Parametri formali del `main`

- I parametri sulla linea di comando sono disponibili al `main` attraverso i suoi parametri formali
- La definizione completa della funzione `main` è:

```
int main(int argc, char *argv[]) ;
```

Argument count

Argument values

Il parametro `argc`

- `int argc`
- Numero di parametri sulla linea di comando
  - Incrementato di uno, in quanto il nome del programma viene considerato come un parametro
- Se non vi sono argomenti effettivi, vale 1
- Se vi sono k argomenti effettivi, vale k+1

Il parametro `argv`

- `char *argv[]`
- È un vettore di stringhe
- Ogni stringa è un parametro del programma
- Vi sono `argc` diverse stringhe
- La prima stringa, `argv[0]`, è il nome del programma
- La seconda stringa, `argv[1]`, è il primo argomento (se esiste)
- …
- L'ultimo argomento è in `argv[argc-1]`

Esempi

```
C:\progr>quadrato
```

- argc==1
- argv[0]=="quadrato"

```
C:\progr>quadrato 5
```

- argc==2
- argv[0]=="quadrato"
- argv[1]=="5"

```
C:\progr>quadrato 5 K
```

- argc==3
- argv[0]=="quadrato"
- argv[1]=="5"
- argv[2]=="K"

Per capire meglio...

arg.c

```c
#include <stdio.h>

int main(int argc, char *argv[])
{
    int i ;

    printf("argc = %d\n", argc) ;
    for(i=0; i<argc; i++)
    {
        printf("argv[%d] = \"%s\"\n",
            i, argv[i]) ;
    }
}
```


Osservazione

- Il vettore `argv` contiene i dati sotto forma esclusivamente di stringa
- Qualora uno dei dati richiesti sia di tipo numerico (`int` o `double`), occorre effettuare la conversione da stringa a numero
  - `i = atoi(argv[1]) ;`
  - `r = atof(argv[1]) ;`
- Se il parametro è invece una stringa, conviene copiarlo in una variabile mediante `strcpy`

La funzione `main()`

Valore di ritorno del programma

Valore di ritorno

- Al termine dell'elaborazione il programma restituisce un numero intero al sistema operativo
- Tale valore viene spesso ignorato, ma in caso di esecuzione "batch" è possibile interrogarlo a livello di sistema operativo
  - in MS-DOS, tramite la variabile ERRORLEVEL
    - `echo %errorlevel%`
  - in sistemi Unix, mediante la macro $?
    - `echo $?`

Convenzioni

- Il valore di ritorno è un int, ma per compatibilità si preferisce ritornare degli "interi positivi piccoli"
- Convenzionalmente
  - Il valore di ritorno pari a 0 indica "programma terminato correttamente"
  - Il valore di ritorno diverso da 0 indica "programma terminato anormalmente a causa di un errore"
    - Il valore specifico ritornato (1, 2, 3, ...) può indicare la causa dell'errore

Esempio

La funzione `main()`

La funzione `exit`

Restituzione del valore di ritorno

- Quando il programma termina, deve restituire il valore di ritorno
  - ==0, se tutto OK
  - !=0, se errore
- Il modo più semplice per restituire tale valore è di utilizzare l'istruzione `return` all'interno della funzione `main`
  - L'elaborazione viene immediatamente interrotta
  - Il valore ritornato viene passato al sistema operativo

Esempio

```
#include <stdio.h>

int main(int argc, char *argv[])
{
   . . .
   . . .
   . . .
   return 0 ;
}
```


La funzione `exit`

- Esiste inoltre la funzione di libreria `exit`, dichiarata in `<stdlib.h>`, che assolve alla stessa funzione
  - Interrompe l'esecuzione del programma
  - Ritorna il valore specificato
- Il vantaggio rispetto all'istruzione `return` è che può essere usata all'interno di qualsiasi funzione, non solo del `main`

```
void exit(int value) ;
```


Esempio

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   . . .
   . . .
   . . .
   exit(0) ;
}
```

Suggerimento

- Ricordare sempre di ritornare un valore
- Mettere come ultima istruzione del `main`: `exit(0);`
- Per eventuali condizioni di errore (parametri assenti, valori illegali, ...) che non possano essere corrette dal programma, restituire un valore positivo: `exit(1) ;`
  - Tali errori possono essere controllati dall'interno di qualsiasi funzione: la `exit` interrompe comunque l'intero programma

La funzione `main()`

Esercizio "Calcolatrice"

Esercizio "Calcolatrice"

- Si scriva un programma da utilizzarsi come semplice calcolatrice sulla linea di comando
- Il programma, denominato `calcola`, accetta 3 parametri sulla linea di comando
  - Il primo ed il terzo parametro sono degli operandi, espressi come numeri reali
  - Il secondo parametro è un operatore, scelto tra +, -, * e /
- Il programma stampa il risultato corrispondente all'operazione

Analisi

```
Prompt dei comandi
c:\progr>calcola 3 + 2
Risultato: 5.0000
c:\progr>calcola 3 * 2
Risultato: 6.0000
c:\progr>calcola 3 $ 2
Errore: operatore non riconosciuto
c:\progr>calcola 3 +
Errore: operando mancante
```


Soluzione (1/5)

calcola.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main(int argc, char *argv[])
{
    double v1, v2 ;
    char op[20] ;
```


Soluzione (2/5)

calcola.c

```
if(argc!=4)
{
    printf("Errore: numero parametri
            insufficiente\n");
    exit(1) ;
}

v1 = atof(argv[1]) ;
strcpy(op, argv[2]) ;
v2 = atof(argv[3]) ;
```

Soluzione (3/5)

calcola.c

```
    if(strcmp(op, "+")==0)
        printf("Risultato: %f\n", v1 + v2 ) ;
    else if(strcmp(op, "-")==0)
        printf("Risultato: %f\n", v1 - v2 ) ;
    else if(strcmp(op, "*")==0)
        printf("Risultato: %f\n", v1 * v2 ) ;
```


Soluzione (4/5)

calcola.c

```
    else if(strcmp(op, "/")==0)
    {
        if(v2==0)
        {
            printf("Errore: divisione per zero\n");
            exit(2) ;
        }
        else
            printf("Risultato: %f\n", v1 / v2 ) ;
    }
```


Soluzione (5/5)

calcola.c

```
    else
    {
        printf("Errore: operatore
                non riconosciuto\n") ;
        exit(3) ;
    }
    exit(0) ;
}
```

Funzioni

Esercizi proposti

Esercizi proposti

- Esercizio "Confronto tra date"
- Esercizio "Quadrato"
- Esercizio "Indovina numero"

Esercizi proposti

Esercizio "Confronto tra date"

Esercizio "Confronto tra date"

- Si scriva un programma che chieda all'utente di inserire due date (giorno, mese, anno) e determini:
  - Se le date solo uguali
  - Se la prima data **precede** la seconda
  - Se la prima data **segue** la seconda
- Il programma dovrà porre particolare attenzione a **non accettare** date non valide
  - Esempio: 30/02/1984
  - Esempio: 10/14/2001

Analisi

```
Prompt dei comandi
Confronto tra date

Inserisci la PRIMA data
Giorno: 15
Mese: 3
Anno: 2007
Inserisci la SECONDA data
Giorno: 26
Mese: 4
Anno: 1967

La prima data 15/3/2007 SEGUE
la seconda data 26/4/1967
```


Controlli

- $1 \leq$ giorno ≤ 31
- $1 \leq$ mese ≤ 12
- $1900 <$ anno < 2100
- Se mese = 4, 6, 9, 11, allora giorno ≤ 30
- Se mese = 2, allora giorno ≤ 29
- Se mese = 2 e l'anno non è bisestile, allora giorno ≤ 28
  - L'anno è bisestile se è multiplo di 4

Soluzione

- Scrivere una funzione per la lettura della data, che comprenda al suo interno tutti i controlli:

```
void leggidata( int *giorno,
      int *mese, int *anno ) ;
```

- La funzione restituisce, nelle 3 variabili passate *by reference*, le componenti (giorno, mese, anno) della data
- La funzione **garantisce** che la data restituita è **corretta**

Programma principale

date.c

```
int main(void)
{
    int g1, m1, a1 ;
    int g2, m2, a2 ;

    printf("Confronto tra date\n\n") ;

    printf("Inserisci la PRIMA data\n") ;
    leggidata( &g1, &m1, &a1 ) ;

    printf("Inserisci la SECONDA data\n") ;
    leggidata( &g2, &m2, &a2 ) ;

    /* Confronto delle date */

} /* main */
```

Confronto delle date

date.c

```
if( g1 == g2 && m1 == m2 && a1 == a2 )
    printf("Le date sono uguali\n") ;
else if( a1<a2 ||
         (a1==a2 && m1<m2) ||
         (a1==a2 && m1==m2 && g1<g2) )
{
    printf("La prima data %d/%d/%d "
     "PRECEDE la seconda %d/%d/%d\n",
        g1, m1, a1, g2, m2, a2) ;
}
else
{
    printf("La prima data %d/%d/%d "
     "SEGUE la seconda %d/%d/%d\n",
        g1, m1, a1, g2, m2, a2) ;
}
```


Funzione leggidata (1/5)

date.c

```
void leggidata( int *giorno, int *mese,
                int *anno )
{
  int g, m, a ;
  int ok ;

  do {
    do {
      printf("Giorno: ") ;
      scanf("%d", &g) ;

      if(g<1 || g>31)
        printf("Giorno non valido\n");
    } while( g<1 || g>31 ) ;
```


Funzione leggidata (2/5)

date.c

```
    do {
      printf("Mese: ") ;
      scanf("%d", &m) ;

      if(m<1 || m>12)
        printf("Mese non valido\n");
    } while( m<1 || m>12 ) ;

    do {
      printf("Anno: ") ;
      scanf("%d", &a) ;

      if ( a<=1900 || a>=2100 )
        printf("Anno non valido\n") ;
    } while( a<=1900 || a>=2100 ) ;
```


Funzione leggidata (3/5)

date.c

```
    ok = 1 ;
    if( g>30 && (m==4 || m==6 ||
        m==9 || m==11) )
    {
      ok = 0 ;
      printf("Il mese %d non "
        "ha %d giorni\n", m, g);
    }
    else if ( g>29 && m==2 )
    {
      ok = 0 ;
      printf("Il mese %d non "
        "ha %d giorni\n", m, g);
    }
```

Funzione leggidata (4/5)

date.c

```
else if ( g==29 && m==2 && a%4!=0 )
    {
      ok = 0 ;
      printf("Il mese %d non "
         "ha %d giorni perche' "
         "l'anno %d non e' bisestile\n",
         m, g, a);
    }
```

Funzione leggidata (5/5)

date.c

```
  } while( ok==0 ) ;

  *giorno = g ;
  *mese = m ;
  *anno = a ;

  return ;

} /* leggidata */
```


Esercizi proposti

Esercizio "Quadrato"

Esercizio "Quadrato" (1/2)

- Si scriva un programma, denominato `quadrato`, che stampi a video un quadrato composto di caratteri tutti uguali.
- Il programma riceve due argomenti sulla linea di comando:
  - Il primo argomento indica la **dimensione** del quadrato (ossia il numero di righe e colonne di cui è composto)
  - Il secondo argomento indica il **carattere** di cui è composto il quadrato

Esercizio "Quadrato" (2/2)

- Gli argomenti sono opzionali: se il carattere viene omesso, occorre stampare "*". Se anche la dimensione viene omessa, si assuma pari a 3

Analisi

```
C:\progr>quadrato
```

- Quadrato 3x3 di "*"

```
C:\progr>quadrato 5
```

- Quadrato 5x5 di "*"

```
C:\progr>quadrato 5 K
```

- Quadrato 5x5 di "K"

Dal punto di vista del main

```
C:\progr>quadrato
```

- argc==1
- argv[0]=="quadrato"

```
C:\progr>quadrato 5
```

- argc==2
- argv[0]=="quadrato"
- argv[1]=="5"

```
C:\progr>quadrato 5 K
```

- argc==3
- argv[0]=="quadrato"
- argv[1]=="5"
- argv[2]=="K"

Soluzione (1/4)

quadrato.c

```
int main(int argc, char *argv[])
{
    int dim ;
    char ch ;
    int i, j ;

    if (argc==1)
    {
        dim = 3 ;
        ch = '*' ;
    }
```


Soluzione (2/4)

quadrato.c

```
    else if (argc==2)
    {
       dim = atoi(argv[1]) ;
       if( dim<1 || dim>20 )
       {
         printf("Dimens. non valida\n") ;
         exit(1) ;
       }
       ch = '*' ;
    }
```

Soluzione (3/4)

quadrato.c

```
    else if (argc==3)
    {
        dim = atoi(argv[1]) ;
        if( dim<1 || dim>20 )
        {
          printf("Dimens. non valida\n");
          exit(1) ;
        }
        ch = argv[2][0] ;
        if(strlen(argv[2])!=1)
        {
          printf("Carattere non valido\n");
          exit(1) ;
        }
    }
```

Soluzione (4/4)

quadrato.c

```
    else
    {
        printf("Numero argomenti "
               "non valido\n") ;
        exit(1) ;
    }

    for(i=0; i<dim; i++)
    {
        for(j=0; j<dim; j++)
            putchar(ch) ;
        putchar('\n') ;
    }

    exit(0) ;
}
```


Esercizi proposti

Esercizio "Indovina numero"

Esercizio "Indovina numero"

- Si realizzi un programma in C per permettere a due giocatori umani di giocare ad "indovina il numero"
- Il primo giocatore attiva il programma, denominato `segreto`, passandogli sulla linea di comando un numero intero tra 1 e 100
- Il secondo giocatore farà una serie di tentativi, immettendoli via tastiera
- Ad ogni tentativo il programma dirà se il numero tentato è più alto o più basso del numero da indovinare

Soluzione (1/3)

segreto.c

```
int main(int argc, char *argv[])
{
    int segreto ;
    int cont, tent ;

    /* Acquisisci dalla linea
     di comando il numero "segreto" */
    if( argc != 2 )
    {
        printf("Numero di parametri "
               "errato\n") ;
        exit(1) ;
    }
```

Soluzione (2/3)

segreto.c

```
segreto = atoi( argv[1] ) ;

if( segreto<1 || segreto>100 )
{
    printf("Numero segreto "
           "errato\n") ;
    exit(1) ;
}

printf("INDOVINA IL NUMERO\n\n");
```

Soluzione (3/3)

segreto.c

```
    cont = 1 ;
    do {
        printf("Tentativo %d: ", cont );
        scanf("%d", &tent) ;

        if( tent < segreto )
            printf("Basso...\n") ;
        else if (tent > segreto )
            printf("Alto...\n") ;

        cont++ ;
    } while (tent != segreto) ;
    cont-- ;

    printf("Indovinato: %d tentativi\n",
           cont) ;
```

Funzioni

Sommario

Argomenti trattati

- Definizione di funzioni in C
  - Prototipo
  - Implementazione
- Passaggio di parametri alle funzioni
  - By value
  - By reference
  - Vettori
- Interfaccia del `main`
  - Parametri sulla linea di comando
  - Valore di ritorno

Tecniche di programmazione

- Decomporre il programma in più funzioni, implementando ciascuna individualmente
- Identificare le parti ripetitive e racchiuderle in apposite funzioni
- Permette il passaggio di parametri al programma
- Analizzare i parametri passati dall'utente

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo

Programmazione in C

Unità
I/O Avanzato e File

I/O Avanzato e File

- Definizione di file
- File di testo in C
- Input robusto
- Formattazione avanzata
- Esercizi proposti
- Sommario

Riferimenti al materiale

- Testi
  - Kernighan & Ritchie: capitolo 7, appendice B
  - Cabodi, Quer, Sonza Reorda: capitoli 3, 8
  - Dietel & Dietel: capitoli 9, 11
- Dispense
  - Scheda: "I/O Avanzato in C"
  - Scheda: "Gestione dei file in C"

I/O Avanzato e File

Definizione di file

Definizione di file

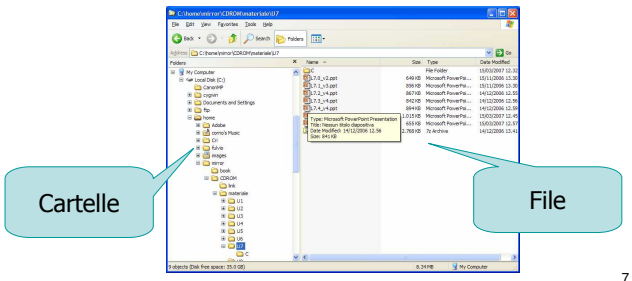
- Directory e file
- File binari e file di testo

Definizione di file

Directory e file

Directory e file

- Tutti i sistemi operativi permettono di organizzare le informazioni su hard disk secondo la metafora di cartelle (directory) e file



Definizioni

- File:
  - Una sequenza di byte
  - Memorizzata su un disco
  - Caratterizzata da uno specifico nome
  - Contenuta all'interno di una specifica directory
- Directory:
  - Un contenitore di file e di altre directory
  - Caratterizzata da uno specifico nome
  - Contenuta all'interno di un'altra directory

Identificazione di un file

- Per identificare un file occorre dunque conoscere:
  - Il nome assegnato al file
  - Il nome della directory in cui esso è memorizzato
- Una volta identificato un file, è possibile accedere al suo contenuto
  - Leggere la sequenza di byte di cui è composto
  - Modificare la sequenza di byte di cui è composto

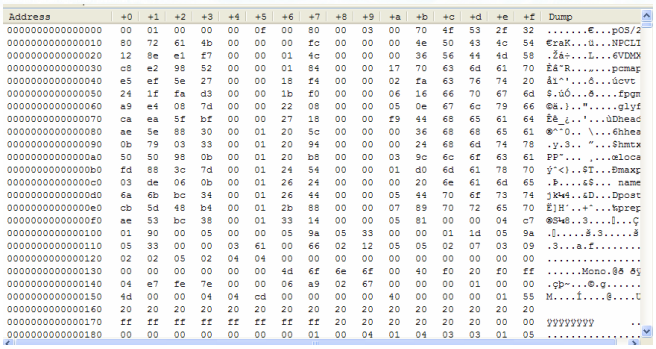
Operazioni permesse sui file

- Operazioni generiche
  - Cancellazione di un file esistente
  - Rinominazione di un file esistente
  - Copia di un file, creando un nuovo file con lo stesso nome in una diversa directory
  - Spostamento di un file, equivalente alla copia con cancellazione dell'originale
- Operazioni specifiche
  - Creazione di un nuovo file
  - Modifica del contenuto del file

Operazioni generiche o specifiche

- Le operazioni generiche sono solitamente svolte interagendo con il sistema operativo, e si possono applicare a qualsiasi file
- Le operazioni specifiche invece coinvolgono il contenuto del file, pertanto richiedono programmi specifici per ogni tipologia di file

File = Sequenza di byte



Address	+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+a	+b	+c	+d	+e	+f	Dump
0000000000000000	00	01	00	00	00	0f	00	80	00	03	00	70	4f	53	2f	32	.....€...pOS/2
0000000000000010	80	72	61	4b	00	00	00	fc	00	00	00	4e	50	43	4c	54	€raK...ü...NPCLI
0000000000000020	12	8e	e1	f7	00	00	01	4c	00	00	00	36	56	44	4d	58	.Žá÷...L...6VDMX
0000000000000030	c8	e2	98	52	00	00	01	84	00	00	17	70	63	6d	61	70	Èâ˜R.....„....pcmap
0000000000000040	e5	ef	5e	27	00	00	18	f4	00	00	02	fa	63	76	74	20	åï^'...ô...úcvt
0000000000000050	24	1f	fa	d3	00	00	1b	f0	00	00	06	16	66	70	67	6d	$.úÓ...ð....fpgm
0000000000000060	a9	e4	08	7d	00	00	22	08	00	00	05	0e	67	6c	79	66	©ä.}.."......glyf
0000000000000070	ca	ea	5f	bf	00	00	27	18	00	00	f9	44	68	65	61	64	Êê_¿..'...ùDhead
0000000000000080	ae	5e	88	30	00	01	20	5c	00	00	00	36	68	68	65	61	®^ˆ0.. \...6hhea
0000000000000090	0b	79	03	33	00	01	20	94	00	00	00	24	68	6d	74	78	.y.3.. "...$hmtx
00000000000000a0	50	50	98	0b	00	01	20	b8	00	00	03	9c	6c	6f	63	61	PP˜... ¸...œloca
00000000000000b0	fd	88	3c	7d	00	01	24	54	00	00	01	d0	6d	61	78	70	ý‹<}...$T...Ðmaxp
00000000000000c0	03	de	06	0b	00	01	26	24	00	00	00	20	6e	61	6d	65	.Þ....&$... name
00000000000000d0	6a	6b	bc	34	00	01	26	44	00	00	05	44	70	6f	73	74	jk¼4..&D...Dpost
00000000000000e0	cb	5d	48	b4	00	01	2b	88	00	00	07	89	70	72	65	70	Ë]H´..+ˆ...‰prep
00000000000000f0	ae	53	bc	38	00	01	33	14	00	00	05	81	00	00	04	c7	®S¼8..3....Ç
0000000000000100	01	90	00	05	00	00	05	9a	05	33	00	00	01	1d	05	9a	.....š.3.....š
0000000000000110	05	33	00	00	03	61	00	66	02	12	05	05	02	07	03	09	.3...a.f........
0000000000000120	02	02	05	02	04	04	00	00	00	00	00	00	00	00	00	00	....
0000000000000130	00	00	00	00	00	00	4d	6f	6e	6f	00	40	f0	20	f0	ff	......Mono.@ð ðÿ
0000000000000140	04	e7	fe	7e	00	00	06	a9	02	67	00	00	00	01	00	00	.çþ~...©.g......
0000000000000150	4d	00	00	04	04	cd	00	00	00	00	40	00	00	00	01	55	M....Í....@....U
0000000000000160	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	20	
0000000000000170	ff	ff	ff	ff	ff	ff	ff	ff	20	20	20	20	20	20	00	00	ÿÿÿÿÿÿÿÿ ...
0000000000000180	00	00	00	00	00	00	00	01	00	04	01	04	03	03	01	05	.....

Tipi di file

- Il significato dei byte memorizzati all'interno dei file è noto, solitamente, solamente al programma che lo ha creato
- Si parla di **tipi di file**, che sono solitamente indicati dall'**estensione** (ultime lettere del nome del file)
  - File prodotto da Microsoft Powerpoint → `.ppt`
  - File prodotto da Microsoft Word → `.doc`
  - File prodotto da Openoffice.Org Writer → `.odt`
  - ...

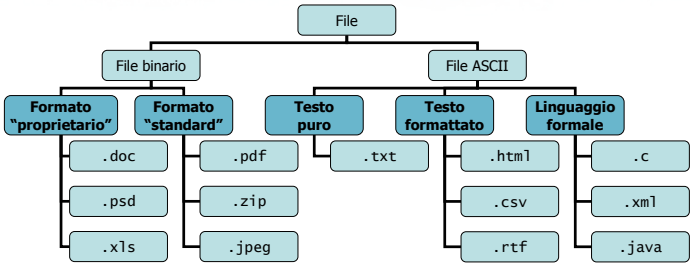
Conseguenza

- Non è possibile lavorare con file gestiti da altri programmi, a meno di non conoscere il formato del file
- **Eccezione**: se il formato del file è pubblicamente documentato, allora sarà possibile leggerlo e scriverlo interpretando correttamente i byte
  - Esempio: file di testo (ASCII)
    - La codifica ASCII è utilizzata in molti campi: testi (.txt), programmi in C (.c), pagine HTML (.html), ...
  - Esempio: file Acrobat (.pdf)
    - Struttura molto complessa, ma documentata

Definizione di file

File binari e file di testo

Vista d'insieme dei formati di file



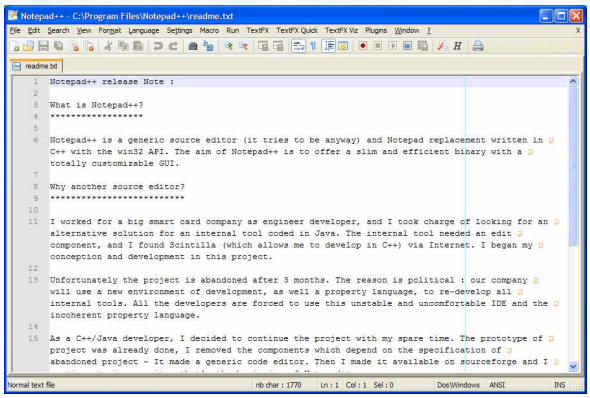
Differenze operative

- A livello di programmazione, esistono
  - Funzioni generali per accedere ai file
  - Funzioni specifiche per la lettura e la scrittura del contenuto di file binari
  - Funzioni specifiche per la lettura e la scrittura del contenuto di file ASCII
- Il corretto funzionamento di un programma dipende dalla perfetta conoscenza del formato del file stesso

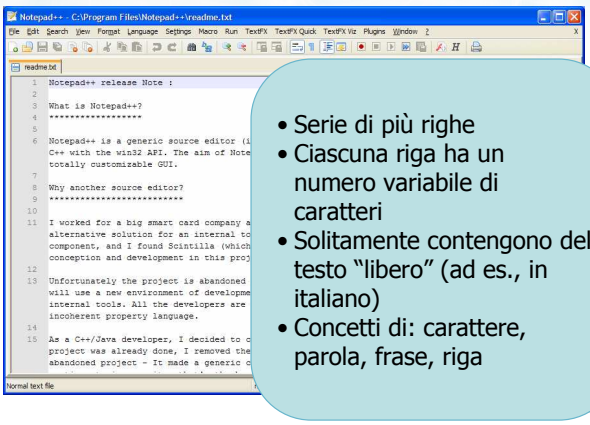
Scelta operativa

- In questo corso si tratteranno esclusivamente i file ASCII
  - Più semplici da comprendere
  - Facili da visualizzare e da creare con qualsiasi editor di testi
    - Notepad o molte sue alternative
    - L'ambiente di sviluppo in C
- Si userà indifferentemente la denominazione "file ASCII" o "file di testo"

File di "testo puro"

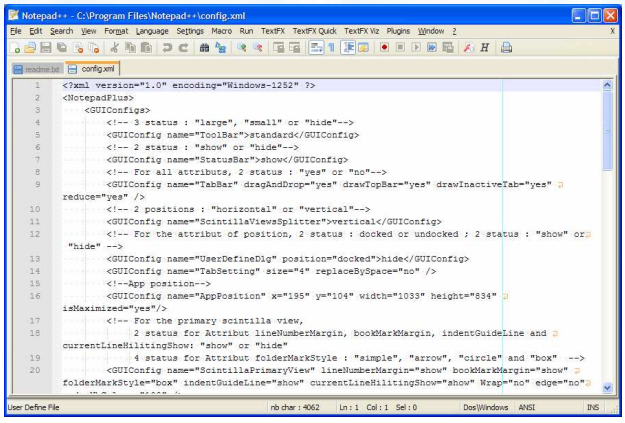


File di "testo puro"

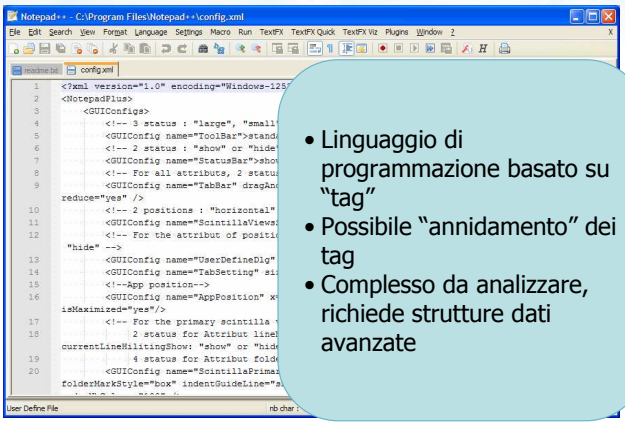


- Serie di più righe
- Ciascuna riga ha un numero variabile di caratteri
- Solitamente contengono del testo "libero" (ad es., in italiano)
- Concetti di: carattere, parola, frase, riga

File di "testo formattato" (es: XML)

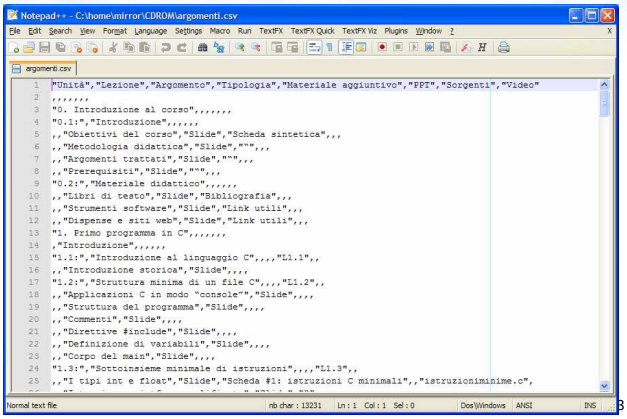


File di "testo formattato" (es: XML)

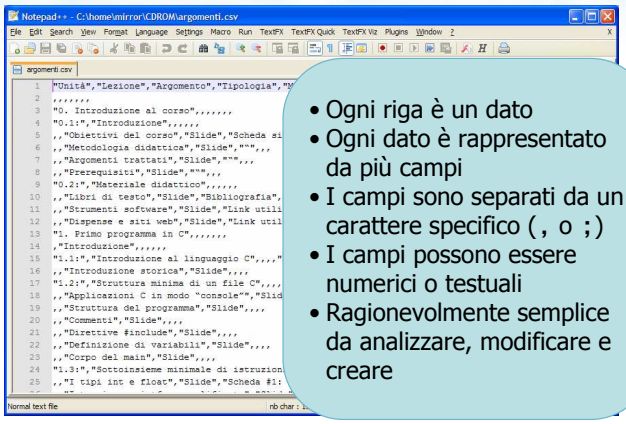


- Linguaggio di programmazione basato su "tag"
- Possibile "annidamento" dei tag
- Complesso da analizzare, richiede strutture dati avanzate

File di "testo formattato" (es: CSV)



File di "testo formattato" (es: CSV)

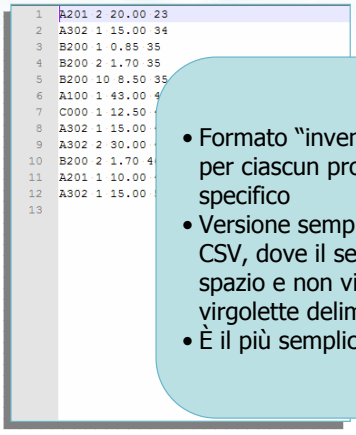


- Ogni riga è un dato
- Ogni dato è rappresentato da più campi
- I campi sono separati da un carattere specifico (, o ;)
- I campi possono essere numerici o testuali
- Ragionevolmente semplice da analizzare, modificare e creare

File di "testo formattato" (custom)

```
A201 2 20.00 23
A302 1 15.00 34
B200 1 0.85 35
B200 2 1.70 35
B200 10 8.50 35
A100 1 43.00 43
C000 1 12.50 44
A302 1 15.00 45
A302 2 30.00 46
B200 2 1.70 46
A201 1 10.00 47
A302 1 15.00 50
```


File di "testo formattato" (custom)



- Formato "inventato" ad hoc per ciascun programma specifico
- Versione semplificata del CSV, dove il separatore è lo spazio e non vi sono virgolette delimitatrici
- È il più semplice da gestire

I/O Avanzato e File

File di testo in C

File di testo in C

- Accesso ai file
- Funzioni `fopen/fclose`
- Funzioni `fget*/fput*`
- Funzioni `fprintf/fscanf`
- Condizione `feof`

File di testo in C

Accesso ai file

Accesso ai file (1/4)

- Un programma C può accedere ad alcuni file presenti sui dischi del calcolatore
  - File aperto: file al quale attualmente il programma ha accesso
  - File chiuso: file residente su disco al quale attualmente il programma non ha accesso

Accesso ai file (2/4)

- All'atto dell'apertura di un file, il programma deve dichiarare la modalità di accesso
  - Modalità di **lettura**: il programma può leggere il contenuto del file, ma non modificarlo
  - Modalità di **scrittura**: il programma può riscrivere da zero il contenuto del file
  - Modalità di **aggiunta**: il programma può aggiungere nuove informazioni al file
  - Modalità di **lettura/scrittura**: tutte le precedenti
- I successivi accessi al file devono essere compatibili con la modalità di accesso dichiarata

Accesso ai file (3/4)

- L'accesso ai file di testo è rigorosamente sequenziale
  - La lettura avviene dalla prima riga all'ultima, dal primo carattere all'ultimo
  - In scrittura, ogni riga o carattere scritto vengono posizionati dopo le righe o caratteri scritti in precedenza
    - A partire dal primo carattere, in modalità di scrittura
    - A partire dall'ultimo carattere esistente, in modalità di aggiunta

Accesso ai file (4/4)

- All'atto dell'apertura di un file, il programma deve dichiarare se il file è di tipo binario oppure di testo
  - La differenza consiste solamente nel trattamento "speciale" del carattere `'\n'` nel caso dei file di testo
  - In questo corso useremo sempre la modalità testuale

Stream associato ad un file

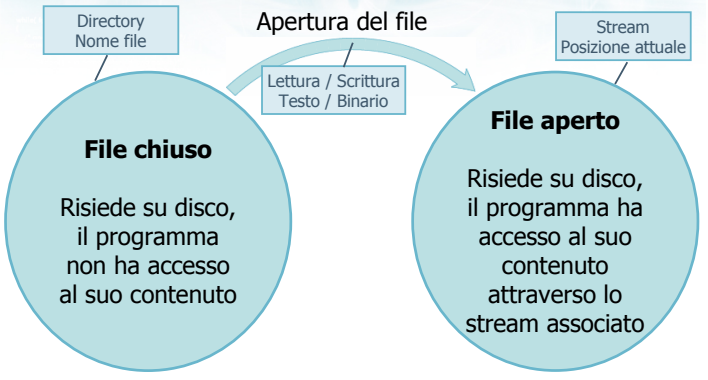
- In un programma C, esiste un tipo di dato specifico per rappresentare le informazioni relative ad un file aperto
  - Denominato: **file stream** (flusso associato ad un file)
  - Tipo di dato: `FILE *` (definito in `<stdio.h>`)
- "Aprire" un file significa quindi creare un nuovo stream ed associarlo ad uno specifico file sul disco

Significato di stream

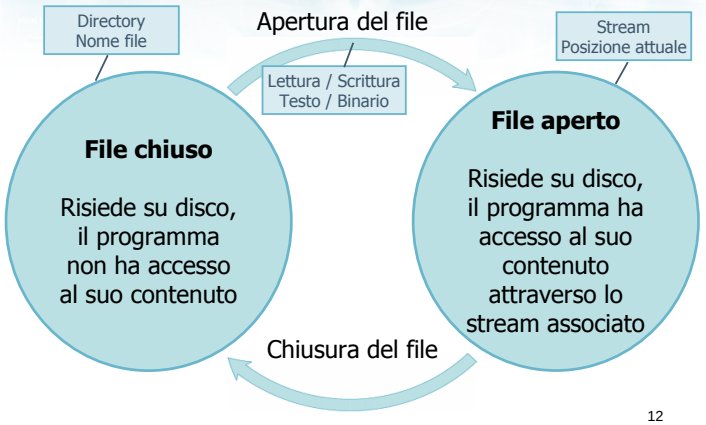
- Una volta che il file è aperto, il suo stream rappresenta
  - Un "collegamento" mediante il quale poter compiere delle operazioni sul contenuto del file
  - Le modalità di accesso scelte (testo/binario, lettura/scrittura/...)
  - La posizione attuale a cui si è arrivati nello scrivere o nel leggere il file
- Ogni operazione sul file avviene chiamando una funzione che riceve lo stream come parametro

Stati di un file

Stati di un file

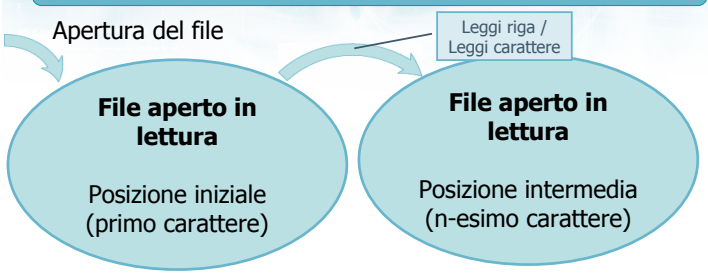


Stati di un file

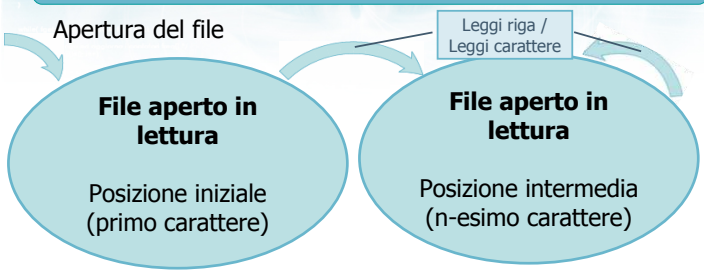


Lettura di un file

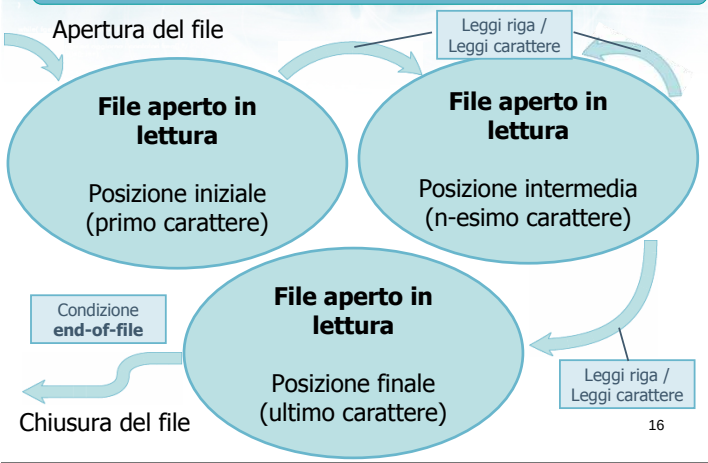
Lettura di un file



Lettura di un file

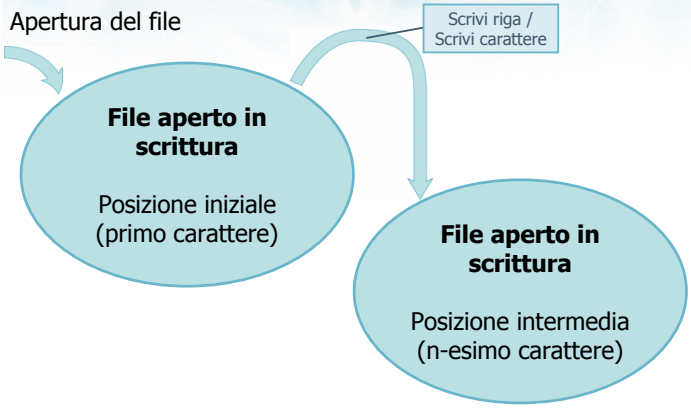


Lettura di un file

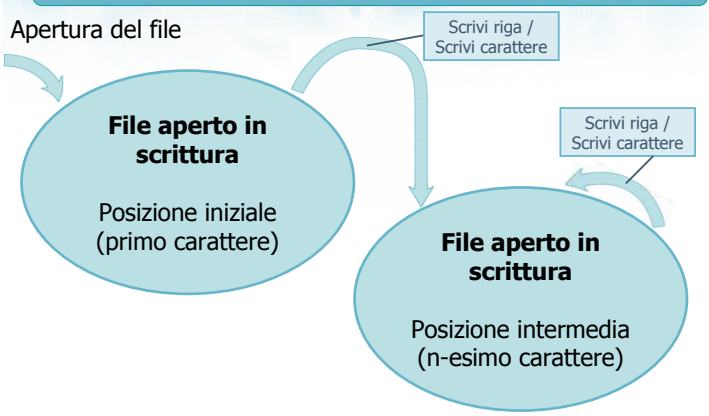


Scrittura di un file

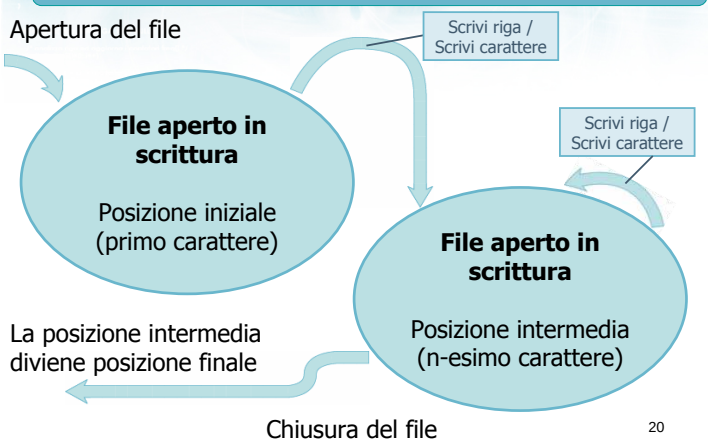
Scrittura di un file



Scrittura di un file

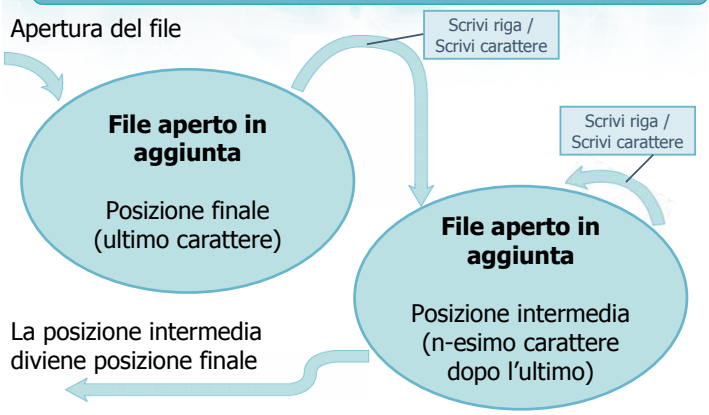


Scrittura di un file



Chiusura del file

Aggiunta ad un file



Chiusura del file

Funzioni C

- Tutte le funzioni per l'accesso ai file sono contenute in `<stdio.h>`
- Funzioni per apertura e chiusura: `fopen`, `fclose`
- Funzioni per la lettura: `fgetc`, `fgets`, `fscanf`
- Funzioni per la scrittura: `fputc`, `fputs`, `fprintf`
- Funzioni per lo stato del file: `feof`

File di testo in C

Funzioni `fopen/fclose`

Funzioni `fopen` e `fclose`

- Apertura del file (`fopen`):
  - La funzione `fopen` apre un file e restituisce una variabile stream
  - Richiede il nome del file e le modalità di apertura
  - Restituisce una nuova variabile di tipo `FILE *`
- Chiusura del file (`fclose`)
  - Quando il file non è più richiesto, si chiama la funzione `fclose` che chiude il file
  - Richiede come parametro lo stream, precedentemente restituito da `fopen`

fopen: sintassi

```
FILE * f ;

...

f = fopen( "nomefile", "modo" ) ;
```

Variabile stream di tipo FILE *

Stringa contenente il nome del file

Modalità di apertura (stringa)

Nome del file

```
f = fopen( "nomefile", "modo" ) ;
```

```
f = fopen( "dati.txt", "modo" ) ;
```


Nome del file

```
f = fopen( "nomefile", "modo" ) ;
```

```
f = fopen( "dati.txt", "modo" ) ;
```

```
f = fopen( "c:\\prog\\dati.txt",
           "modo" ) ;
```


Nome del file

```
f = fopen( "nomefile", "modo" ) ;
```

```
f = fopen( "dati.txt", "modo" ) ;
```

```
f = fopen( "c:\\prog\\dati.txt",
           "modo" ) ;
```

```
char nome[20] ;
gets(nome) ;
f = fopen( nome, "modo" ) ;
```


Nome del file

```
f = fopen( "nomefile", "modo" ) ;
```

```
f = fopen( "dati.txt", "modo" ) ;
```

```
f = fopen( "c:\\prog\\dati.txt",
           "modo" ) ;
```

```
char nome[20] ;
gets(nome) ;
f = fopen( nome, "modo" ) ;
```

```
f = fopen( argv[1], "modo" ) ;
```

Modo di apertura

```
f = fopen( "nomefile", "modo" ) ;
```

Modalità **lettura**, file di testo → `"rt"` `"r"`

Modalità **scrittura**, file di testo → `"wt"` `"w"`

Modalità **aggiunta**, file di testo → `"at"` `"a"`

Effetto della fopen (1/3)

- Modalità "r"
  - Se il file esiste, viene aperto ed f punta allo stream relativo, posizionato in lettura al primo carattere
  - Se il file non esiste, non viene creato nessuno stream e f==NULL

Effetto della fopen (2/3)

- Modalità "w"
  - Se il file non esiste, viene creato da zero ed f punta allo stream relativo, posizionato in scrittura al primo carattere
  - Se il file esiste già, viene innanzitutto cancellato e poi ricreato da zero, f punta allo stream relativo, posizionato in scrittura al primo carattere
  - Se non è possibile creare il file (perché la directory non esiste, o il disco è protetto in scrittura, ...), non viene creato nessuno stream e f==NULL

Effetto della fopen (3/3)

- Modalità "a"
  - Se il file non esiste, viene creato da zero ed f punta allo stream relativo, posizionato in scrittura al primo carattere
  - Se il file esiste già, non viene modificato, f punta allo stream relativo, posizionato in scrittura dopo l'ultimo carattere
  - Se non è possibile creare o modificare il file (perché la directory non esiste, o il disco è protetto in scrittura, ...), non viene creato nessuno stream e f==NULL

Controllo dell'errore

```
FILE * f ;

...

f = fopen( "nomefile", "r" ) ;
if( f == NULL )
{
  printf("Impossibile aprire file\n");
  exit(1) ;
}
```


Suggerimento

- Ogniqualvolta viene chiamata la funzione fopen, è indispensabile subito dopo controllare se il valore ritornato non è NULL
- È da considerarsi **errore** una chiamata ad fopen di cui non venga controllato il risultato
- In caso di errore, solitamente conviene interrompere il programma segnalando un codice di errore
  - Esempio: exit(1) ;

fclose: sintassi

```
FILE * f ;
...
f = fopen( "nomefile", "modo" ) ;

.../* accesso al file */

fclose(f) ;
```

Variabile stream

Avvertenze

- La funzione `fclose` può essere chiamata solamente su stream correttamente aperti
  - Mai chiamare `fclose` se `f==NULL`
- Dopo la chiusura del file, non è più possibile accedere allo stream
  - Eventualmente, ri-aprirlo nuovamente

Controllo dell'errore

```
int ris ;
...

ris = fclose(f) ;
if(ris!=0)
{
  printf("Impossibile chiudere\n") ;
  exit(1) ;
}
```


Suggerimento

- Conviene definire due funzioni aggiuntive, che chiamino le funzioni di libreria `fopen` e `fclose` e addizionalmente compiano i controlli d'errore
- Chiameremo `myfopen` e `myfclose` tali funzioni
- Nei programmi chiameremo sempre `myfopen` e `myfclose`, e mai direttamente le funzioni di libreria

Funzione `myfopen`

```
FILE * myfopen(char *name, char *mode)
{
  FILE * f ;

  f = fopen( name, mode ) ;
  if (f==NULL)
  {   printf("Impossibile aprire %s\n",
              name) ;
      exit(1) ;
  }
  return f ;
}
```

Funzione `myfclose`

```
int myfclose(FILE *f)
{ int ris ;

  if (f==NULL)
  { printf("ERRORE INTERNO\n") ;
    exit(1) ;
  }
  ris = fclose(f) ;
  if( ris!=0 )
  { printf("Impossibile chiudere\n") ;
    exit(1) ;
  }
  return ris ;
}
```

File di testo in C

Funzioni `fget*`/`fput*`

Lettura e scrittura su file

	Lettura	Scrittura
Carattere singolo	`fgetc`	`fputc`
Riga intera	`fgets`	`fputs`

Lettura e scrittura su file

	Lettura	Scrittura
Carattere singolo	`fgetc`	`fputc`
Riga intera	`fgets`	`fputs`

Legge prossimo elemento, fino alla fine del file

Scrive o aggiunge

Lettura e scrittura su file

	Lettura	Scrittura
Carattere singolo	`fgetc`	`fputc`
Riga intera	`fgets`	`fputs`

Parametro: `char`

Parametro: stringa

Legge prossimo elemento, fino alla fine del file

Scrive o aggiunge

`fgetc`: sintassi

```
int ch ;

ch = fgetc(f) ;
```

Stream aperto in lettura

Prossimo carattere del file; `EOF` se il file è finito

`fputc`: sintassi

```
int ch ;

fputc(ch, f) ;
```

Stream aperto in scrittura o in aggiunta

Carattere da aggiungere al file

`fgets`: sintassi

```
char str[80] ;

fgets(str, 79, f) ;
```

Max numero di caratteri letti

Stream aperto in lettura

Stringa nella quale viene letta la prossima riga del file (fino al \n compreso)

Fine del file

- La funzione `fgets` restituisce un valore `NULL` se ha tentato di leggere oltre la fine del file

```
char str[80] ;

while( fgets(str, 79, f) != NULL )
{
    /* elabora str */
}
```


fputs: sintassi

```
char str[80] ;

fputs(str, f) ;
```

Stream aperto in scrittura o in aggiunta

Stringa da aggiungere al file (solitamente termina con \n)

Esercizio: "Frequenza lettere"

- Sia dato un file di testo, contenente dei brani scritti da un utente
- Si scriva un programma in C che acquisisca sulla linea di comando il nome di tale file, e che stampi le frequenze con cui compaiono le varie lettere dell'alfabeto
- Si considerino equivalenti le maiuscole e le minuscole, e si ignorino i caratteri di spaziatura e punteggiatura

Analisi (1/2)

`manzoni.txt`

```
Quel ramo del lago di Como,
che volge a mezzogiorno,
tra due catene non interrotte di monti,
tutto a seni e a golfi,
a seconda dello sporgere e del
rientrare di quelli, vien, quasi
a un tratto, a ristringersi, e a
prender corso e figura di fiume,
tra un promontorio a destra,
e un'ampia costiera dall'altra parte
```


Analisi (2/2)

`manzoni.txt`

```
Quel ramo del lago di Como,
che volge a mezzogiorno,
tra due catene non interrotte di monti,
tutto a seni e a golfi,
a seconda dello sporgere e del
rientrare di quelli, vien, quasi
a un tratto, a ristringersi, e a
prender corso e figura di fiume,
tra un promontorio a destra,
e un'ampia costiera dall'altra parte
```

Prompt dei comandi

```
C:\prog>frequenza manzoni.txt
A : 26
B : 0
C : 6
D : 12
E : 32
F : 3
G : 7
H : 1
I : 21
J : 0
```


Soluzioni possibili

- Occorre calcolare un vettore di frequenze, in cui ciascuna posizione rappresenti la frequenza di ciascuna delle lettere alfabetiche
- Ci sono due approcci possibili alla lettura del file:
  - Soluzione **per caratteri**: il file viene letto un carattere alla volta, usando la funzione `fgetc`
  - Soluzione **per righe**: il file viene letto una riga alla volta, usando la funzione `fgets`, e tale riga viene poi esaminata con un ciclo interno al programma

Soluzione 1: per caratteri (1/3)

```
const int LETT = 26 ;
int freq[LETT] ; /* frequenze lettere */

FILE * f ;
int ch, i ;

if (argc!=2)
{
  printf("Numero argomenti errato\n") ;
  exit(1) ;
}

for(i=0; i<LETT; i++)
    freq[i] = 0 ;
```

frequenza.c

Soluzione 1: per caratteri (2/3)

```
f = myfopen( argv[1], "r" ) ;

ch = fgetc( f ) ;
while( ch!=EOF )
{
  if(isalpha(ch))
  {
    i = toupper(ch)-'A' ;
    /* posizione 0..25 della lettera */
    freq[i]++ ;
  }
  ch = fgetc( f ) ;
}
myfclose( f ) ;
```

frequenza.c

Soluzione 1: per caratteri (3/3)

```
for(i=0; i<LETT; i++)
{
  printf("%c : %d\n", i+'A', freq[i]) ;
}

exit(0) ;
```

frequenza.c

Soluzione 2: per righe

```
const int LUN = 200 ;
char riga[LUN+1];
...

while( fgets( riga, LUN, f ) != NULL )
{
  for(i=0; riga[i]!=0; i++)
  {
    if(isalpha(riga[i]))
    {
      freq[toupper(riga[i])-'A']++ ;
    }
  }
}
```

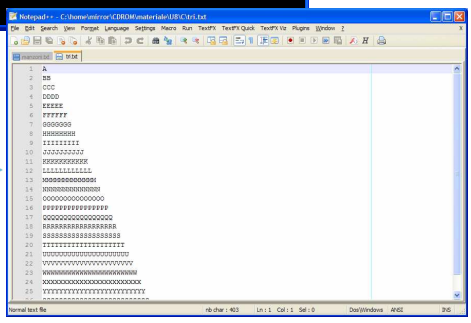
frequenza.c

Esercizio: "Triangolo alfabetico"

- Si realizzi un programma in C che crei un file di testo contenente tutte le lettere dell'alfabeto, con una disposizione "a triangolo"
  - La prima riga contiene una volta la lettera A
  - La seconda riga contiene 2 volte la lettera B
  - La terza riga contiene 3 volte la lettera C
  - ...
- Il nome del file viene passato come primo argomento sulla linea di comando

Analisi

```
c:\prog>triangolo tri.txt
```



Soluzione (1/2)

triangolo.c

```
FILE * f ;
int i, j ;
char ch ;

if (argc!=2)
{
  printf("Numero argomenti errato\n") ;
  exit(1) ;
}

f = myfopen( argv[1], "w" ) ;
```

Soluzione (2/2)

triangolo.c

```
for(i=0; i<26; i++)
{
  ch = i+'A' ;

  for(j=0; j<=i; j++)
    fputc( ch, f ) ;

  fputc( '\n', f ) ;
}

myfclose( f ) ;

exit(0) ;
```

File di testo in C

Funzioni fprintf/fscanf

Output formattato

- Qualora sia necessario creare file con più campi nella stessa riga, è scomodo ricorrere alle funzioni `fputc`/`fputs`
- È possibile utilizzare una variante della funzione `printf`, operante su uno stream aperto in scrittura
  - `fprintf(f, "formato", x, y, z) ;`

fprintf: sintassi

```
FILE * f ;

fprintf(f, "formato", variabili ) ;
```

Stream aperto in scrittura o in aggiunta

Elenco delle variabili da scrivere

Formato dei dati da stampare, usando gli stessi specificatori validi per `printf`

Input formattato

- Qualora sia necessario leggere file con più campi nella stessa riga
  - È scomodo ricorrere alla funzione `fgetc`
  - Il risultato della funzione `fgets` deve successivamente essere analizzato
- È possibile utilizzare una variante della funzione `scanf`, operante su uno stream aperto in lettura
  - `fscanf(f, "formato", &x, &y, &z) ;`

fscanf: sintassi

```
FILE * f ;

fscanf(f, "formato", &variabili ) ;
```

Stream aperto in lettura

Puntatori alle variabili da leggere

Formato dei dati da leggere, usando gli stessi specificatori validi per `scanf`

fscanf: una funzione pericolosa

- Nonostante la funzione `fscanf` sia prevista dalla libreria standard C, è considerata una funzione **pericolosa** nella lettura di file in generale
- In particolare, qualora il file non sia nel formato corretto (file contenente errori), allora il meccanismo di funzionamento di `fscanf` rende **impossibile** acquisire i dati in modo **affidabile**
- Suggerimento: **non usare mai** `fscanf`
- Nella prossima lezione vedremo una **soluzione robusta** al problema

File di testo in C

Condizione `feof`

End-of-File

- Un file aperto in lettura è inizialmente posizionato al primo carattere
- Ad ogni successiva lettura, avanza la posizione corrente all'interno del file
- Quando è stato letto l'ultimo carattere (o l'ultima riga) del file, non sono possibili ulteriori letture
  - In questo caso si dice che si è verificata una condizione di End-of-File (`EOF`)
  - Ulteriori tentativi di lettura genererebbero una condizione di errore

Tentativi di lettura

- Se si tenta di leggere oltre l'End-of-File
  - `fgets` restituisce `NULL`
  - `fgetc` restituisce `EOF`
  - `fscanf` restituisce `EOF`
- È possibile controllare tali valori di ritorno per controllare la fine del file
  - In tali casi, l'errore è già avvenuto, e l'operazione di lettura non è andata a buon fine

Funzione `feof`

- La funzione `feof` è specificatamente utile per verificare se uno stream `f` è già nella condizione di End-of-File **prima** di tentare operazioni di lettura
  - `if ( !feof( f ) ) { ... }`
- La funzione, partendo dallo stream `f`, restituisce:
  - **Vero**, se lo stream **è** già in End-of-File, e quindi le successive letture falliranno
  - **Falso**, se lo stream **non è** ancora in End-of-File, e quindi sono possibili ulteriori letture

Esempio

```
ch = fgetc( f ) ;

while( ch!=EOF )
{
   .../* elabora ch */

   ch = fgetc( f ) ;
}
```

```
while( !feof(f) )
{
   ch = fgetc( f ) ;

   .../* elabora ch */
}
```

I/O Avanzato e File

Input robusto

Input robusto

- Problemi nella lettura da file
- Soluzione basata su `fgetc`
- Funzione `sscanf`
- Soluzione basata su `fgets`

Input robusto

Problemi nella lettura da file

Lettura da file

- I file di testo contengono dati secondo un certo **formato**
- È semplice scrivere un programma in grado di leggere un file formattato correttamente
- Diviene molto complesso gestire eventuali **errori** di formato del file

Errori di formato

- Righe contenenti un numero errato di elementi
  - Elementi in eccesso
    - In fondo
    - All'inizio o in mezzo
  - Elementi in difetto
- Tipi di dato errati
  - Caratteri, stringhe, interi, reali
- Errori di coerenza interna

Esempio

- Un file di testo contiene i PIN dei bancomat dei membri di una famiglia
- Il file di testo contiene sulla prima riga il numero di bancomat descritti nel file
- Le righe successive contengono le informazioni su ciascun bancomat, uno per riga
- Ciascuna riga contiene 3 campi separati da spazi:
  - Il nome del proprietario del bancomat
  - Il numero del bancomat
  - Il PIN segreto (5 cifre)

Esempio di file corretto

bancomat.txt

```
3
Aldo 123456789 12762
Giovanni 334422445 97864
Giacomo 887868083 32552
```


Definizioni

```
const char nomefile[] = "banco.txt" ;
const int MAX = 20 ;
          /* numero max di bancomat */
const int LUN = 15 ;
          /* lunghezza del nome */

int N ;
char nome[MAX][LUN+1] ;
int numero[MAX] ;
int pin[MAX] ;

FILE * f ;
int i ;
```

banco-bad.c

Lettura del file (solo se corretto)

```
f = myfopen(nomefile, "r") ;

fscanf(f, "%d", &N) ;

for(i=0; i<N; i++)
{
  fscanf(f, "%s %d %d",
   nome[i], &numero[i], &pin[i]) ;
}

myfclose(f) ;
```

banco-bad.c

Possibili errori nel file (1/3)

```
3
Aldo 123456789 12762
334422445 97864
Giacomo 887868083 32552
```

Campo mancante

Possibili errori nel file (1/3)

```
3
Aldo 123456789 12762
334422445 97864
Giacomo 887868083 32552
```

Campo mancante

```
3
Aldo 3212 123456789 12762
Giovanni 334422445 97864
Giacomo 887868083 32552
```

Campo extra nella riga

Possibili errori nel file (2/3)

```
3
Aldo 123456789 12762 3212
Giovanni 334422445 97864
Giacomo 887868083 32552
```

Campo extra a fine riga

Possibili errori nel file (2/3)

```
3
Aldo 123456789 12762 3212
Giovanni 334422445 97864
Giacomo 887868083 32552
```

Campo extra a fine riga

```
3
Aldo 123456789 12762
Giovanni 334422445 97864
Giacomo A32Z4324 32552
```

Tipi di dato errati

Possibili errori nel file (3/3)

```
3
Pier Aldo 123456789 12762
Giovanni 334422445 97864
Giacomo 887868083 32552
```

Spazi

Possibili errori nel file (3/3)

```
3
Pier Aldo 123456789 12762
Giovanni 334422445 97864
Giacomo 887868083 32552
```

Spazi

```
3
Aldo 123456789 12762
Giacomo 887868083 32552
```

Incoerenza interna

Osservazioni

- Di fronte a qualsiasi errore di formato, la funzione `fscanf`
  - Perde il "sincronismo" con le righe del file
  - "Blocca" la lettura in caso di stringhe incontrate quando si aspettava un numero
  - Non si "accorge" dell'End-of-File
- La funzione `fscanf` non è sufficientemente robusta per gestire la lettura da file di testo

Input robusto

Soluzione basata su `fgetc`

Lettura basata su `fgetc`

- Dovendo evitare l'utilizzo della funzione `fscanf`, si potrebbe optare per la funzione `fgetc`
- L'adozione di `fgetc` risolve i problemi di sincronizzazione e di lettura dei dati errati, ma introduce spesso una complessità eccessiva nel programma

Soluzioni basate su fgetc (1/4)

- Acquisizione di una stringa

```
char s[MAX] ;

i = 0 ;
ch = fgetc(f) ;
while( ch != EOF  && ch != '\n'
       && ch != ' ' && i < MAX-1 )
{
    s[i] = ch ;
    i++ ;
    ch = fgetc(f) ;
}

s[i] = 0 ; /* terminatore nullo */
```

Soluzioni basate su fgetc (2/4)

- Saltare tutti gli spazi (ma non gli a-capo)

```
ch = fgetc(f) ;
while( ch != EOF  && ch != '\n' &&
       ch == ' ' )
{
    ch = fgetc(f) ;
}
```

Soluzioni basate su fgetc (3/4)

- Acquisizione di un intero positivo

```
char s[MAX] ;

i = 0 ;
ch = fgetc(f) ;
while( ch != EOF  && isdigit(ch)
       && i < MAX-1 )
{
    s[i] = ch ;
    i++ ;
    ch = fgetc(f) ;
}
s[i] = 0 ; /* terminatore nullo */

x = atoi(s) ; /* converti in int */
```

Soluzioni basate su fgetc (4/4)

- Sono possibili tutti i controlli personalizzati, su
  - Lunghezza minima e massima dei campi
  - Tipo di caratteri permessi
- Alcuni tipi di dati sono complessi da acquisire
  - Intero relativo: -124
  - Numero reale: -3.14e+21
- Soluzione in generale completa, ma molto lavoro manuale
- Rischio: dimenticare alcuni casi particolari

Suggerimento

- Utilizzare la funzione fgetc quando occorre leggere dei testi in "formato libero"
  - Esempio: statistiche sul testo
  - Esempio: file di una sola riga
- Per file in formato custom, contenenti campi prefissati e/o di tipo numerico, occorre una soluzione più comoda

Input robusto

Funzione sscanf

Funzione sscanf

- La risposta a molti dei problemi sollevati viene da una nuova funzione di libreria: `sscanf`
- Tale funzione si può usare per analizzare il contenuto di una stringa, estraendone vari campi e memorizzandoli in variabili distinte
- Ha tutta la funzionalità di `scanf` e `fscanf`, ma lavora soltanto all'interno dei caratteri contenuti in una stringa
  - Potente e sicura

sscanf: sintassi

```
char str[80] ;

sscanf(str, "formato", &variabili ) ;
```

Stringa di caratteri

Puntatori alle variabili da leggere

Formato dei dati da leggere, usando gli stessi specificatori validi per `scanf`

Esempio

```
char str[80] ;
char nome[80] ;
int numero, pin ;

strcpy(str, "Aldo 91243213 1234\n");

sscanf(str, "%s %d %d",
       nome, &numero, &pin ) ;
```


Gestione degli errori

- La funzione `sscanf` non potrà mai leggere le righe successive di un file, in quanto la sua visibilità è confinata alla stringa passata
- Gli eventuali campi in eccesso a fine riga vengono quindi ignorati automaticamente
- Gli eventuali campi mancanti o di formato errato causano il mancato riconoscimento di quelli successivi
  - Condizione di errore facile da verificare analizzando il valore di ritorno di `sscanf`

Valore di ritorno

- La funzione sscanf restituisce al chiamante un valore intero:
  - Il valore è pari al numero di argomenti (specificatori %) correttamente riconosciuti e memorizzati nelle rispettive variabili

```
r = sscanf(str, "%s %d %d",
       nome, &numero, &pin ) ;
```


Esempio

```
char str[80] ;
char nome[80] ;
int numero, pin ;
int r ;

strcpy(str, "Aldo 91243213 1234\n");

r = sscanf(str, "%s %d %d",
       nome, &numero, &pin ) ;

if( r != 3 )
{ ... errore ... }
```

Suggerimenti

- Utilizzare **sempre** `sscanf` per analizzare una stringa
- Controllare **sempre** il valore di ritorno
- Non utilizzare più la funzione `atoi`, sostituirla con `sscanf(..."%d"...)`
- Per acquisire dati da tastiera, combinare con `gets`
- Per acquisire dati da file, combinare con `fgets`
- Nella prossima lezione vedremo come "istruire" `sscanf` a riconoscere formati più complessi

Input robusto

Soluzione basata su `fgets`

Input robusto da file di testo

- Affidiamo diversi ruoli alle varie funzioni
- `fgets`
  - Lettura del file riga per riga
  - Limite alla lunghezza max delle righe
  - Riconoscimento End-of-File
- `sscanf`
  - Analisi dei campi presenti in una riga
  - Controllo della correttezza del formato
  - Trasferimento nelle variabili/vettori del programma

Schema consigliato

```
const int LUNRIGA = 200 ;
int r, nr ;
char riga[LUNRIGA+1] ;

f = myfopen(nomefile, "r") ;

 /* Ciclo di lettura */

myfclose(f) ;
```


Ciclo di lettura

```
nr = 0 ;
while( fgets(riga, LUNRIGA, f)!=NULL )
{
  r = sscanf(riga, "%s %d %d",
             nome, &numero, &pin) ;
  if( r == 3 )
  {
    /* ...elabora la riga... */
  }
  else
    printf("Riga %d ignorata\n", nr+1);
  nr++ ;
}
```


Soluzione corretta (1/6)

```
const char nomefile[]="banco.txt";
const int MAX = 20 ;
const int LUN = 15 ;
const int LUNRIGA = 200 ;

int N ;
char nome[MAX][LUN+1] ;
int numero[MAX] ;
int pin[MAX] ;

FILE * f ;
int i, r, nr ;
char riga[LUNRIGA+1] ;
```

banco-ok.c

Soluzione corretta (2/6)

```
f = myfopen(nomefile, "r") ;
if(fgets(riga, LUNRIGA, f)==NULL)
{
  printf("Errore: file vuoto\n") ;
  exit(1) ;
}
r = sscanf(riga, "%d", &N) ;
if(r!=1)
{
  printf("Errore: La prima riga "
     "non contiene il numero\n");
  exit(1) ;
}
```

banco-ok.c

Soluzione corretta (3/6)

```
if( N<1 || N>MAX )
{
  printf("Errore: Num. bancomat "
      "%d non valido\n", N) ;
  printf("Valori ammessi: "
      "da 1 a %d\n", MAX) ;
  exit(1) ;
}
```

banco-ok.c

Soluzione corretta (4/6)

```
i = 0 ;
nr = 0 ;
while( fgets( riga, LUNRIGA, f )
          != NULL )
{
  if(i==N)
  {
    printf("Errore: troppe "
          "righe nel file\n" ) ;
    exit(1) ;
  }
```

banco-ok.c

Soluzione corretta (5/6)

```
  r = sscanf(riga, "%s %d %d",
    nome[i], &numero[i], &pin[i]);

  if( r == 3 )
    i++ ;
  else
  {
    printf("Riga %d ignorata\n",
          nr) ;
  }
  nr++ ;
}
```

banco-ok.c

Soluzione corretta (6/6)

```
if( i != N )
{
  printf("Errore: poche righe "
    " nel file\n" ) ;
  exit(1) ;
}

myfclose(f) ;
```

banco-ok.c

Conclusioni

- Prevedere tutti i possibili errori è difficile e pesante
  - La maggior parte delle linee di codice è dedicata alla gestione degli errori o delle anomalie
- Gli strumenti offerti dalla "coppia" `fgets` + `sscanf` sono validi ed efficaci

I/O Avanzato e File

Formattazione avanzata

Formattazione avanzata

- Modificatori di formato in output
- Modificatori di formato in input
- Stream predefiniti

Formattazione avanzata

Modificatori di formato in output

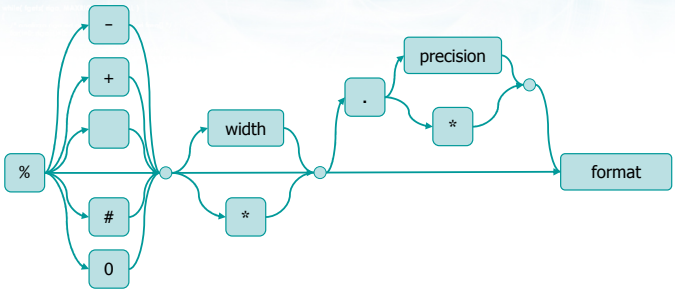
Formattazione dell'output

- L'output (su schermo o su file) viene formattato solitamente mediante la funzione `printf` (o `fprintf`)
- Ogni dato viene stampato attraverso un opportuno specificatore di formato (codici %)
- Ciascuno di questi codici dispone di ulteriori opzioni per meglio controllare la formattazione
  - Stampa incolonnata
  - Numero di cifre decimali
  - Spazi di riempimento
  - ...

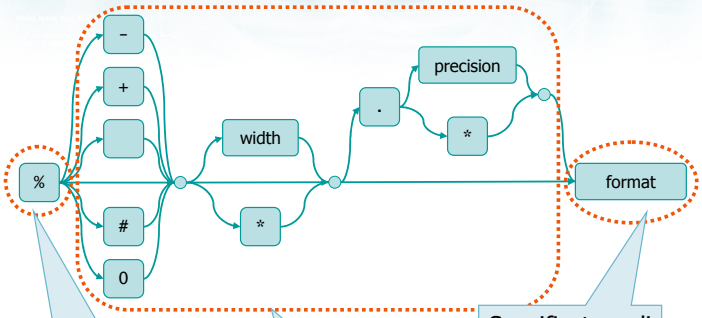
Specificatori di formato

Tipo	printf
char	%c %d
int	%d
short int	%hd %d
long int	%ld
unsigned int	%u %o %x
unsigned short int	%hu
unsigned long int	%lu
float	%f %e %g
double	%f %e %g
char []	%s

Forma completa degli specificatori



Forma completa degli specificatori

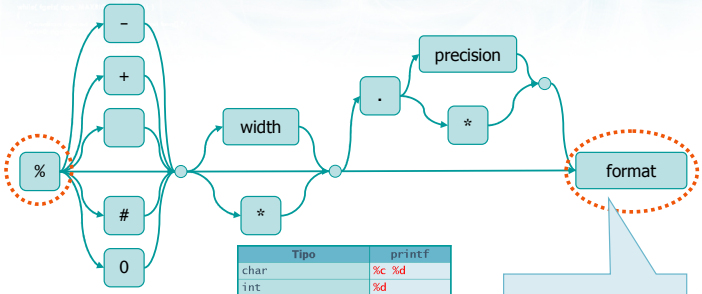


% obbligatorio

Modificatori opzionali

Specificatore di formato obbligatorio

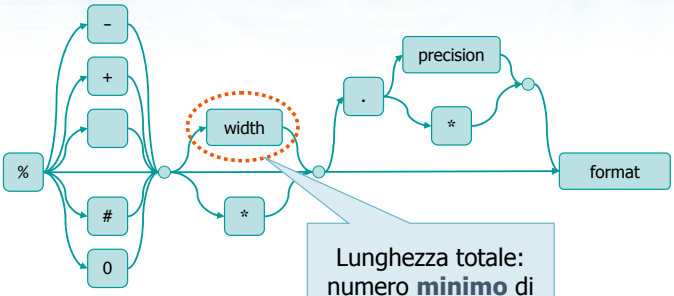
Forma completa degli specificatori



Tipo	printf
char	%c %d
int	%d
short int	%hd %d
long int	%ld
unsigned int	%u %o %x
unsigned short int	%hu
unsigned long int	%lu
float	%f %e %g
double	%f %e %g
char []	%s

Specificatori già noti

Forma completa degli specificatori

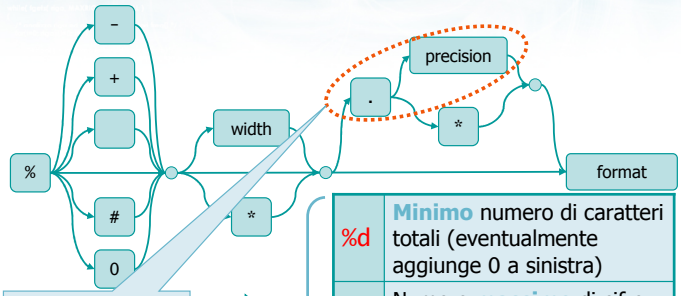


Lunghezza totale: numero **minimo** di caratteri stampati

Esempi

Istruzione	Risultato
`printf("%d",  13) ;`	13
`printf("%1d", 13) ;`	13
`printf("%3d", 13) ;`	␣13
`printf("%f",  13.14) ;`	13.140000
`printf("%6f", 13.14) ;`	13.140000
`printf("%12f", 13.14) ;`	␣␣13.140000
`printf("%6s", "ciao") ;`	␣␣ciao

Forma completa degli specificatori



Precisione (dipende...)

%d	**Minimo** numero di caratteri totali (eventualmente aggiunge 0 a sinistra)
%f	Numero **massimo** di cifre dopo la virgola
%s	**Massimo** numero di caratteri (stringhe più lunghe vengono troncate)

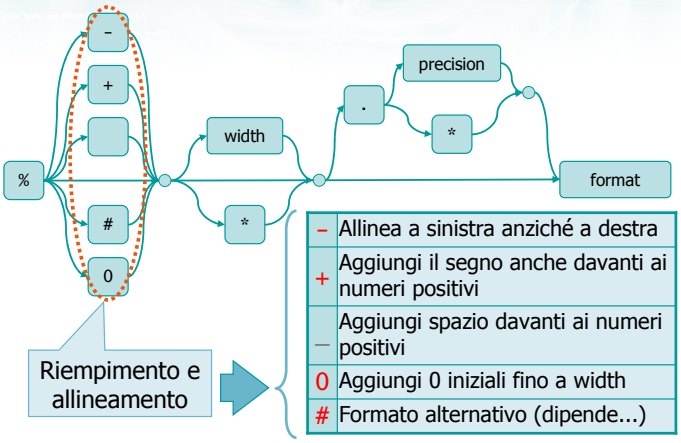
Esempi (1/2)

Istruzione	Risultato
`printf("%.1d",  13) ;`	13
`printf("%.4d", 13) ;`	0013
`printf("%6.4d", 13) ;`	␣␣0013
`printf("%4.6d", 13) ;`	000013
`printf("%.2s", "ciao") ;`	ci
`printf("%.6s", "ciao") ;`	ciao
`printf("%6.3s", "ciao") ;`	␣␣␣cia

Esempi (2/2)

Istruzione	Risultato
`printf("%.2f", 13.14) ;`	13.14
`printf("%.4f", 13.14) ;`	13.1400
`printf("%6.4f", 13.14) ;`	13.1400
`printf("%9.4f", 13.14) ;`	␣13.1400

Forma completa degli specificatori



Riempimento e allineamento

-	Allinea a sinistra anziché a destra
+	Aggiungi il segno anche davanti ai numeri positivi
␣	Aggiungi spazio davanti ai numeri positivi
0	Aggiungi 0 iniziali fino a width
#	Formato alternativo (dipende...)

Esempi (1/2)

Istruzione	Risultato
`printf("%6d", 13) ;`	␣␣␣␣13
`printf("%-6d", 13) ;`	13␣␣␣␣
`printf("%06d", 13) ;`	000013
`printf("%6s", "ciao") ;`	␣␣ciao
`printf("%-6s", "ciao") ;`	ciao␣␣

Esempi (2/2)

Istruzione	Risultato
`printf("%d", 13) ;`	13
`printf("%d", -13) ;`	-13
`printf("%+d", 13) ;`	+13
`printf("%+d", -13) ;`	-13
`printf("% d", 13) ;`	␣13
`printf("% d", -13) ;`	-13

Formattazione avanzata

Modificatori di formato in input

Approfondimenti su scanf

- Tipologie di caratteri nella stringa di formato
- Modificatori degli specificatori di formato
- Valore di ritorno
- Specificatore %[]

Stringa di formato (1/2)

- Caratteri stampabili:
  - `scanf` si aspetta che tali caratteri compaiano esattamente nell'input
  - Se no, interrompe la lettura
- Spaziatura ("whitespace"):
  - Spazio, tab, a capo
  - `scanf` "salta" ogni (eventuale) sequenza di caratteri di spaziatura
  - Si ferma al primo carattere non di spaziatura (o End-of-File)

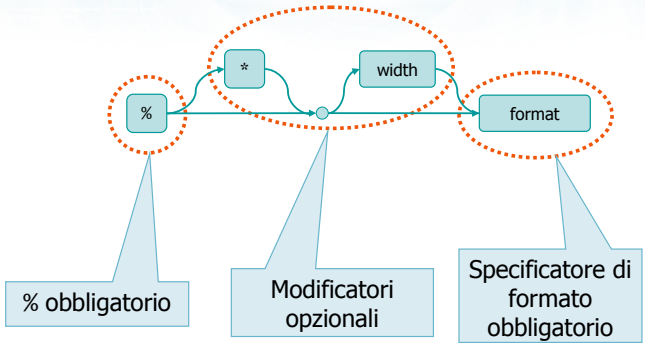
Stringa di formato (2/2)

- Specificatori di formato (%-codice):
  - Se il codice non è %c, innanzitutto `scanf` "salta" ogni eventuale sequenza di caratteri di spaziatura
  - `scanf` legge i caratteri successivi e *cerca* di convertirli secondo il formato specificato
  - La lettura si interrompe al primo carattere che non può essere interpretato come parte del campo

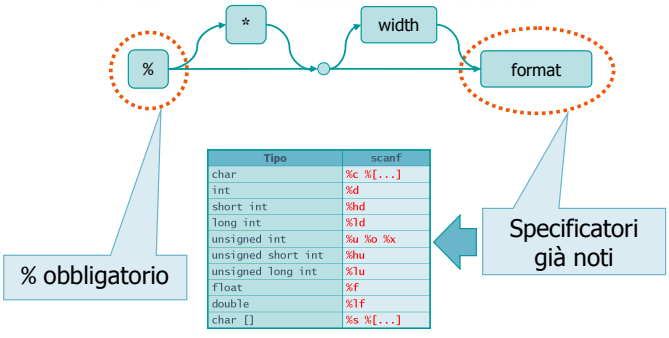
Specificatori di formato

Tipo	scanf
char	%c %[...]
int	%d
short int	%hd
long int	%ld
unsigned int	%u %o %x
unsigned short int	%hu
unsigned long int	%lu
float	%f
double	%lf
char []	%s %[...]

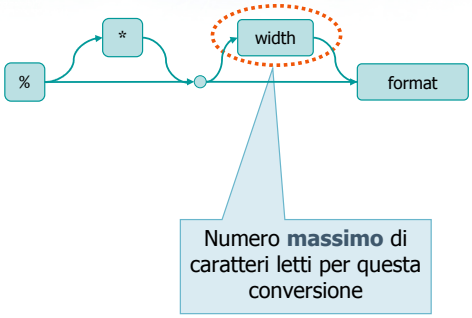
Forma completa degli specificatori



Forma completa degli specificatori



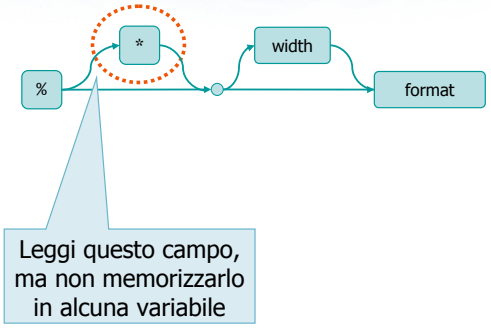
Forma completa degli specificatori



Esempi

Istruzione	Input	Risultato
`scanf("%d", &x) ;`	134xyz	x = 134
`scanf("%2d", &x) ;`	134xyz	x = 13
`scanf("%s", v) ;`	134xyz	v = "134xyz"
`scanf("%2s", v) ;`	134xyz	v = "13"

Forma completa degli specificatori



Esempi

Istruzione	Input	Risultato
`scanf("%d %s", &x, v) ;`	10 Pippo	x = 10 v = "Pippo"
`scanf("%s", v) ;`	10 Pippo	x immutato v = "10"
`scanf("%*d %s", v) ;`	10 Pippo	x immutato v = "Pippo"

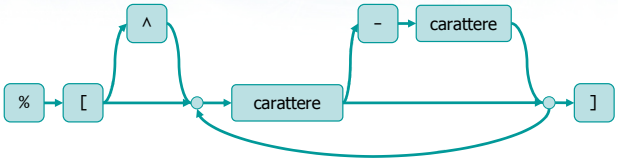
Valore di ritorno

- La funzione `scanf` ritorna un valore intero:
  - Numero di elementi (%) effettivamente letti
    - Non conta quelli "saltati" con %*
    - Non conta quelli non letti perché l'input non conteneva i caratteri desiderati
    - Non conta quelli non letti perché l'input è finito troppo presto
      - End-of-File per `fscanf`
      - Fine stringa per `sscanf`
  - `EOF` se l'input era già in condizione End-of-File all'inizio della lettura

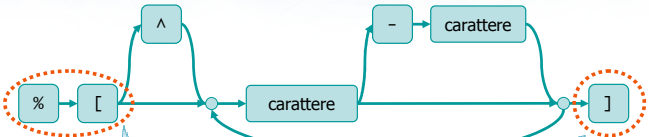
Lettura di stringhe

- La lettura di stringhe avviene solitamente con lo specificatore di formato %s
  - Salta tutti i caratteri di spaziatura
  - Acquisisci tutti i caratteri seguenti, fermandosi al primo carattere di spaziatura (senza leggerlo)
- Qualora l'input dei separatori diversi da spazio, è possibile istruire `scanf` su quali siano i caratteri leciti, mediante lo specificatore %[*pattern*]

Struttura di un pattern

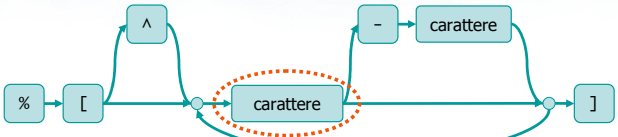


Struttura di un pattern



Definizione di pattern nella stringa di formato

Struttura di un pattern

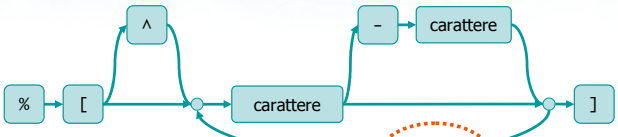


Carattere di cui può essere composta la stringa da leggere

Esempi

Pattern	Effetto
`%[r]`	Legge solo sequenze di `'r'`

Struttura di un pattern

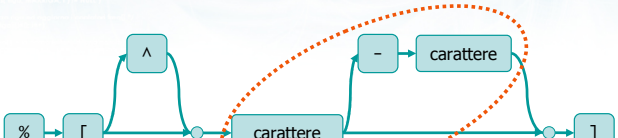


Caratteri o intervalli possono essere ripetuti più volte

Esempi

Pattern	Effetto
`%[r]`	Legge solo sequenze di `'r'`
`%[abcABC]`	Legge sequenze composte da a, b, c, A, B, C, in qualsiasi ordine e di qualsiasi lunghezza

Struttura di un pattern

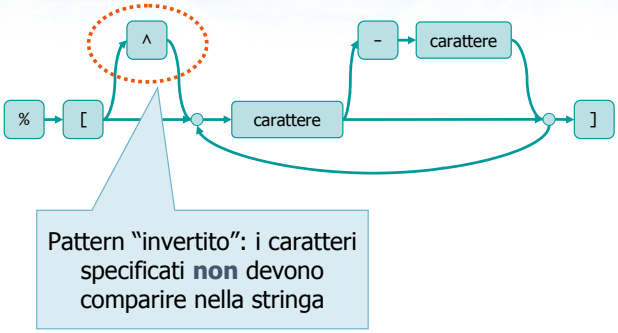


Intervallo di caratteri di cui può essere composta la stringa da leggere

Esempi

Pattern	Effetto
%[r]	Legge solo sequenze di 'r'
%[abcABC]	Legge sequenze composte da a, b, c, A, B, C, in qualsiasi ordine e di qualsiasi lunghezza
%[a-cA-C]	Idem come sopra
%[a-zA-Z]	Sequenze di lettere alfabetiche
%[0-9]	Sequenze di cifre numeriche
%[a-zA-Z0-9]	Sequenze alfanumeriche

Struttura di un pattern



Pattern "invertito": i caratteri specificati **non** devono comparire nella stringa

Esempi

Pattern	Effetto
%[r]	Legge solo sequenze di 'r'
%[abcABC]	Legge sequenze composte da a, b, c, A, B, C, in qualsiasi ordine e di qualsiasi lunghezza
%[a-cA-C]	Idem come sopra
%[a-zA-Z]	Sequenze di lettere alfabetiche
%[0-9]	Sequenze di cifre numeriche
%[a-zA-Z0-9]	Sequenze alfanumeriche
%[^x]	Qualunque sequenza che non contiene 'x'
%[^\n]	Legge fino a file riga
%[^,;.!?]	Si ferma alla punteggiatura o spazio

Osservazioni

- Ricordare che i pattern devono sempre essere associati a dati di tipo stringa (vettori di caratteri)
- Il comune specificatore "%s" equivale al pattern "%[^ \t\n]"

Esempio

- Il file /etc/passwd, presente in tutti i sistemi operativi derivati da Unix, contiene i dati degli utenti nel seguente formato:

```
corno:w3tce34:501:401:Fulvio Corno:/home/corno:/bin/bash
```

  - Campi separati da ':'
  - Nome utente, password: stringhe prive di spazi
  - User ID, Group ID: interi
  - Nome reale: stringa generica (con spazi)
  - Home directory e shell: stringhe generiche

Soluzione

```
f = myfopen("/etc/passwd", "r") ;

while(fgets(riga, MAX, f)!=NULL)
{
  sscanf(riga,
    "%[^:]:[%^:]:%d:%d:"
    "%[^:]:%[^:]:%[^:]",
    login, pass, &uid, &gid,
    realname, home, shell ) ;
  /* elabora i dati ... */
}
myfclose(f) ;
```

Formattazione avanzata

Stream predefiniti

Stream predefiniti

- L'istruzione `fopen` permette di aprire nuovi stream, associati a file esistenti sui dischi dell'elaboratore
- All'avvio di un programma in C, sono già stati aperti in modo automatico 3 stream predefiniti
  - `stdin`
  - `stdout`
  - `stderr`

`stdin`

- `stdin` è detto lo "standard input" di un programma
- Normalmente è associato al canale di input del terminale (o della console) nel quale il programma è avviato
  - In pratica, la tastiera del P.C.
- L'input può essere rediretto, prendendolo da un file anziché dalla tastiera, avviando il programma da linea di comando con l'operatore <
  - `prog < file.txt`

`stdout`

- `stdout` è detto lo "standard output" di un programma
- Normalmente è associato al canale di output del terminale (o della console) nel quale il programma è avviato
  - In pratica, il video del P.C.
- L'output può essere rediretto, salvandolo su un file anziché su video, avviando il programma da linea di comando con l'operatore >
  - `prog > file.txt`

`stderr`

- `stderr` è detto lo "standard error" di un programma
- Normalmente è associato al canale di output del terminale (o della console) nel quale il programma è avviato
  - In pratica, il video del P.C.
- È uno stream distinto ed indipendente da `stdout`
- Solitamente l'output di `stderr` non viene rediretto, per permettere ai messaggi di errore di essere sempre visti dall'utilizzatore

Uso comune

- `stdin` viene usato per acquisire i dati
  - Può essere rediretto da un file
- `stdout` viene usato per presentare i risultati
  - Può essere rediretto su un file
- `stderr` viene usato esclusivamente per i messaggi di errore
  - Rimane visibile sulla console

Equivalenze

La funzione…	è equivalente a…
`scanf("formato", …);`	`fscanf(stdin, "formato", …);`
`printf("formato", …);`	`fprintf(stdout, "formato", …);`
`ch=getchar();`	`ch=fgetc(stdin);`
`putchar(ch);`	`fputc(ch, stdout);`

Stampa dei messaggi di errore

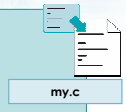
```
if(...condizione errore fatale...)
{
  fprintf(stderr,
    "Messaggio di errore\n") ;
  exit(1) ;
}
```


Suggerimento

- Tutti i messaggi di errore devono essere stampati sullo stream `stderr`
- Conviene definire una funzione `myerror`
  - Stampa un messaggio di errore
  - Interrompe il programma

```
void myerror(char *message) ;
```


Funzione `myerror`




```
int myerror(char *message)
{
    fputs( message, stderr ) ;
    exit(1) ;
}
```

I/O Avanzato e File

Esercizi proposti

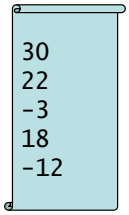
- Esercizio "Somma numeri"
- Esercizio "Bersagli"
- Esercizio "Consumi toner"

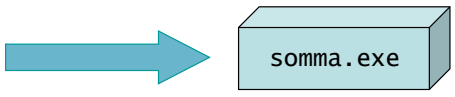
Esercizi proposti

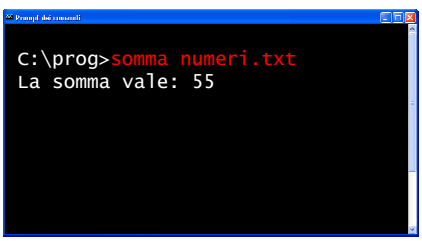
Esercizio "Somma numeri"

- Un file di testo contiene una serie di numeri interi (positivi o negativi), uno per riga
- Si scriva un programma C che:
  - Acquisisca da linea di comando il nome del file
  - Calcoli la somma di tutti i numeri presenti nel file
  - Stampi in output il valore di tale somma

numeri.txt







sommafile.c

```
int main(int argc, char *argv[])
{
  const int MAX = 80 ;

  FILE * f ;
  char nomefile[MAX] ;
  char riga[MAX] ;
  int r, num ;

  int somma ;
```

Soluzione (2/4)

sommafile.c

```
if(argc != 2)
  myerror("Num. argomenti errato\n") ;

strcpy(nomefile, argv[1]) ;

f = myfopen( nomefile, "rt" ) ;

somma = 0 ;
```

Soluzione (3/4)

sommafile.c

```
while( fgets( riga, MAX, f ) != NULL)
{
  r = sscanf( riga, "%d", &num ) ;

  if(r==1)
    somma = somma + num ;
  else
    printf("Riga ignorata\n") ;
}
```

Soluzione (4/4)

sommafile.c

```
  myfclose(f) ;

  printf( "La somma vale: %d\n", somma ) ;

  exit(0) ;
}
```

Esercizi proposti

Esercizio "Bersagli"

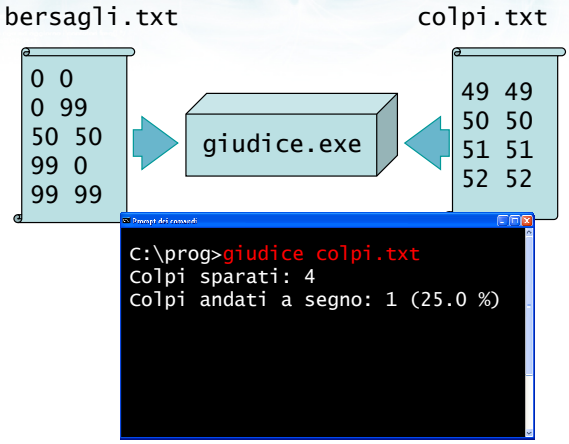
Esercizio "Bersagli" (1/2)

- Si desidera creare un programma in grado di calcolare il numero di colpi andati a segno in un'esercitazione di tiro
- I bersagli sono descritti tramite le coordinate cartesiane del punto in cui sono posizionati all'interno di una griglia 100 × 100. Le coordinate sono rappresentate solo da numeri interi, compresi tra 0 e 99. La posizione dei bersagli è contenuta nel file di testo `bersagli.txt`: ogni riga di tale file contiene le coordinate X e Y di un singolo bersaglio

Esercizio "Bersagli" (2/2)

- I colpi sparati sono descritti anch'essi tramite le loro coordinate X e Y e sono memorizzati in un file di caratteri il cui nome è passato come primo parametro sulla linea di comando. Ogni riga di tale file contiene le coordinate X e Y del punto in cui è stato sparato un colpo
- Si scriva un programma che legga dai file succitati la posizione dei bersagli ed i colpi sparati e quindi calcoli il numero di colpi andati a segno, sia come valore assoluto sia come percentuale dei colpi sparati

Analisi



Algoritmo

- Acquisire dal file `bersagli.txt` tutte le coordinate, memorizzandole in due vettori paralleli `Bx[]` e `By[]`. Lunghezza dei vettori: `Nb`
- Acquisire dal file `argv[1]` le coordinate dei vari colpi `Cx`, `Cy`. Numero colpi: `Nc`
  - Per ciascun colpo, verificare se le coordinate coincidono con quelle di almeno un bersaglio
  - Se sì, incrementare `Ncc`
- Stampare `Ncc` e `Ncc/Nc*100`

Soluzione (1/4)

bersagli.c

```
int main(int argc, char *argv[])
{
  const int MAXB = 100 ;
      /* massimo numero di bersagli */
  const int MAX = 80 ;
      /* lunghezza riga del file */
  const char FILEB[] = "bersagli.txt";

  int Nb ; /* numero di bersagli */
  int Bx[MAXB], By[MAXB] ;
          /* coordinate dei bersagli */

  int Nc ; /* numero colpi sparati */
  int Ncc ; /* numero di colpi centrati */
```

Soluzione (2/4)

bersagli.c

```
FILE *f ;
char riga[MAX] ;
int Cx, Cy ;
int i, r, trovato ;

/* 1: acquisizione coordinate bersagli */
f = myfopen( FILEB, "rt") ;
Nb = 0 ;
while( fgets(riga, MAX, f) != NULL )
{
  r=sscanf(riga,"%d %d",&Bx[Nb],&By[Nb]);
  if( r!=2 )
    myerror("Formato errato\n") ;
  Nb ++ ;
}
myfclose(f);
```

Soluzione (3/4)

bersagli.c

```
/* 2: analisi coordinate dei colpi */
if( argc != 2 )
  myerror("ERR: manca nome file\n");
f = myfopen( argv[1], "rt") ;

Nc = 0 ;
Ncc = 0 ;
while( fgets(riga, MAX, f) != NULL )
{
  r = sscanf( riga, "%d %d", &Cx, &Cy );
  if(r!=2) myerror("Formato errato\n") ;
  Nc ++ ;
  /* Ricerca del bersaglio */
}
myfclose(f);
```

Soluzione (3/4)

bersagli.c

```
/* 2: analisi coordinate dei colpi */
    trovato = 0 ;
    for(i=0; i<Nb && trovato==0; i++)
      if( Cx==Bx[i] && Cy==By[i] )
        trovato = 1 ;

    if(trovato==1)
      Ncc ++ ;

  Nc ++ ;
  /* Ricerca del bersaglio */
}
myfclose(f);
```

Soluzione (4/4)

bersagli.c

```
  /* 3: stampa risultati */
  printf("Colpi sparati: %d\n", Nc) ;
  printf("Colpi andati a segno: %d ", Ncc);
  if(Nc!=0)
    printf("(%.2f%%)", Ncc*100.0/Nc) ;
  printf("\n");

  exit(0) ;
}
```

Esercizi proposti

Esercizio "Consumi toner"

Esercizio "Consumi toner" (1/3)

- Si desidera analizzare la statistica dei consumi di toner di un'azienda per ottimizzare gli acquisti futuri
- La quantità di cartucce di toner prelevate dal magazzino ogni giorno è riportata all'interno di un file di testo il cui nome è passato come primo parametro sulla riga di comando

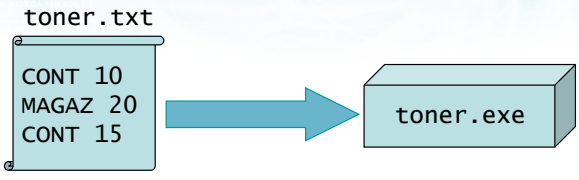
Esercizio "Consumi toner" (2/3)

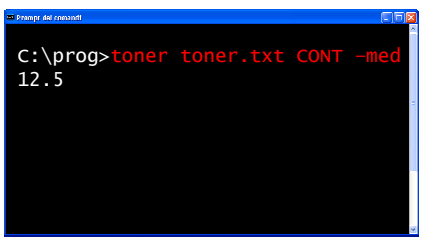
- Il file contiene una riga per ogni giorno. Ogni riga contiene in sequenza:
  - Il nome del dipartimento che ha prelevato il toner (una stringa lunga al massimo 5 caratteri)
  - Un numero intero (valore minimo 1 e massimo 99) che indica la quantità di cartucce di toner prelevate in quel giorno da quel dipartimento
- Non è noto il numero di righe presenti nel file

Esercizio "Consumi toner" (3/3)

- Il programma riceve inoltre come secondo argomento sulla linea di comando il nome di un dipartimento per il quale calcolare l'indicatore statistico dato come terzo argomento sulla linea di comando secondo la seguente codifica:
  - `-min` indica che si desidera il valore minimo
  - `-max` indica che si desidera il valore massimo
  - `-med` indica che si desidera il valore medio (da stamparsi in output con un cifra dopo la virgola)

Analisi





Argomenti del programma

```
C:\prog>toner toner.txt CONT -med
```

argv[1]
Nome del file contenente i consumi

argv[2]
Dipartimento da analizzare

argv[3]
Operazione statistica:
-min
-med
-max

Soluzione (1/4)

toner.c

```
int main(int argc, char *argv[])
{
  const int LUNDIP = 5 ;
  const int MAX = 80 ;

  char dip[LUNDIP+1], dipf[LUNDIP+1] ;
  int stat ;
    /* tipo di statistica:
         1=min, 2=max, 3=med */
  FILE * f ;
  int qtaf, r ;
  int min, max, tot, cont ;
  char riga[MAX+1] ;
```

Soluzione (2/4)

toner.c

```
if(argc!=4)
  myerror("Numero parametri errato\n");
/* Acquisisci il nome del dipartimento */
strcpy(dip, argv[2]) ;
/* Acquisisci tipo statistica */
if( strcmp( argv[3], "-min") == 0 )
  stat = 1 ;
else if ( strcmp( argv[3], "-max") == 0 )
  stat = 2 ;
else if ( strcmp( argv[3], "-med") == 0 )
  stat = 3 ;
else
  myerror("Statistica sconosciuta\n");
```

Soluzione (3/4)

toner.c

```
f = myfopen(argv[1], "rt") ;
tot = 0 ;
cont = 0 ;
min = 100 ;
max = 0 ;
while( fgets(riga, MAX, f) != NULL )
{
  r = sscanf(riga, "%s %d", dipf, &qtaf);
  if(r!=2)
    printf("Riga ignorata\n");
  else
  {
    /* Aggiorna statistiche */
  }
}
myfclose(f) ;
```

Soluzione (3/4)

toner.c

```
    if(strcmp(dip, dipf)==0)
    {
      if( qtaf < min )
        min = qtaf ;
      if( qtaf > max )
        max = qtaf ;
      tot = tot + qtaf ;
      cont++ ;
    }
  else
  {
    /* Aggiorna statistiche */
  }
}
myfclose(f) ;
```

Soluzione (4/4)

toner.c

```
/* Stampa il valore della statistica */
if(cont==0)
  printf("Nessun elemento\n");
else if( stat==1 )
  printf("%d\n", min) ;
else if( stat ==2 )
  printf("%d\n", max) ;
else if( stat==3 )
  printf("%.1f\n", (float)tot/cont) ;

exit(0) ;
}
```

I/O Avanzato e File

Sommario

Argomenti trattati (1/2)

- File
  - File binari
  - File di testo
- Gestione dei file in C
  - Apertura/chiusura
  - Lettura/scrittura
  - Gestione degli errori
  - Il problema degli errori di formattazione

Argomenti trattati (2/2)

- Formattazione avanzata
  - Funzione `sscanf`
  - Opzioni degli specificatori di formato
    - In output
    - In input
    - Pattern di input
  - Stream predefiniti
- Input robusto
  - Utilizzo combinato di `fgets` e `sscanf`

Tecniche di programmazione

- Gestire i file, in lettura e scrittura
- Verificare gli errori che possono incorrere nelle operazioni di I/O
- Utilizzare le funzioni `myfopen`, `myfclose`, `myerror`
- Utilizzare `sscanf` per analizzare righe anche dal formato complesso
- Utilizzare `printf`/`fprintf` per controllare l'ampiezza dei campi di output

Materiale aggiuntivo

- Sul CD-ROM
  - Testi e soluzioni degli esercizi trattati nei lucidi
  - Scheda sintetica
  - Esercizi risolti
  - Esercizi proposti
- Esercizi proposti da altri libri di testo