

12BHDxx Informatica

Programmazione in C

Settimana n.1

Obiettivi

- Problem solving
- Diagrammi di flusso e pseudo codice

Contenuti

- Cenni storici
- Concetto di programma
- Diagrammi di flusso
- Pseudo codice
- Alcuni problemi di esempio

2

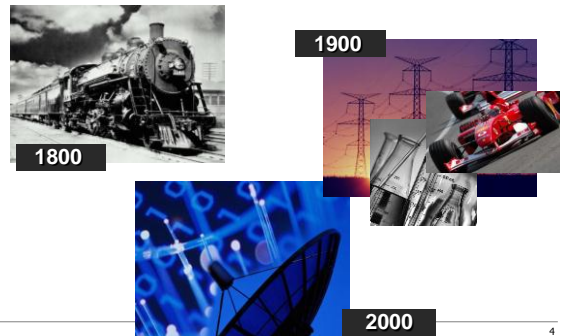
Definizione

- “L'informatica è la scienza che rappresenta e manipola le informazioni”



3

Le tecnologie come fattore abilitante dei cambiamenti industriali e sociali



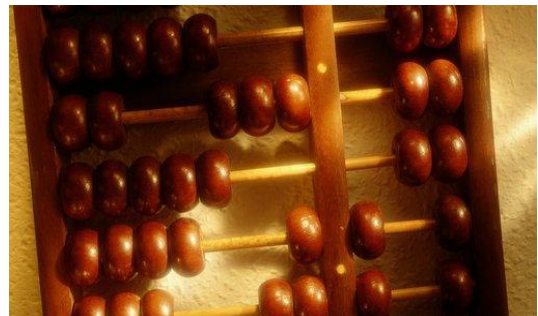
4

La pervasività



5

Un po' di storia: Abacus



6

B. Pascal (1642)



7

J.M. Jacquard (punched card loom- 1801)



- Il software per la computazione meccanica

8

C. Babbage (analytical engine – 1833)



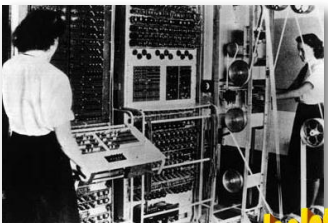
9

Hollerith (punched card -1890)



10

ENIAC (Eckert-Mauchly – 1943-1945)



Il primo calcolatore elettronico!

11

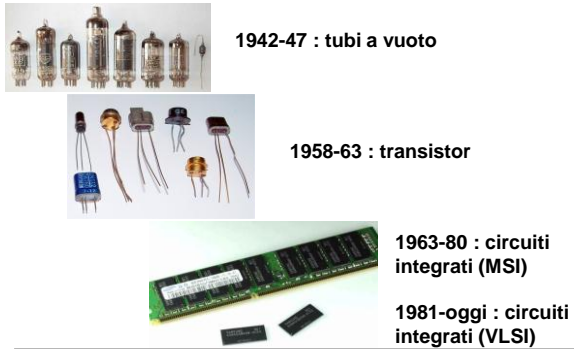
Il computer moderno

- 1942-'57, 1a gen. = tubi a vuoto
- 1958-'63, 2a gen. = transistori
- 1964-'80, 3a gen. = circuiti integrati
- 1980-oggi, 4a gen. = circuiti VLSI
- (futuro) 5a gen. = ?

“Penso che nel mondo ci sia mercato per quattro o cinque computer”
Thomas Watson (Presidente IBM, 1943)

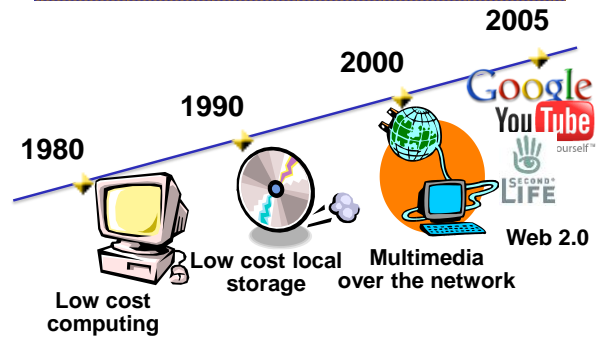
12

Il computer moderno (Cont.)



13

La storia recente



14

I dati digitali: tutto diventa "bit"



15

Le grandi convergenze



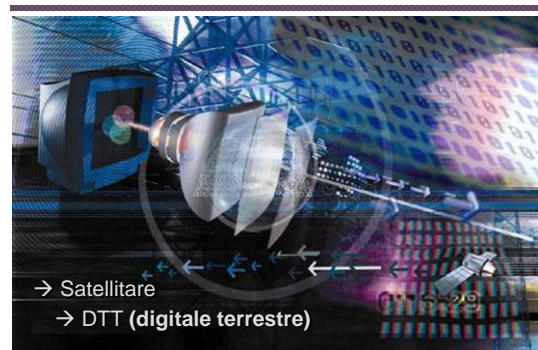
16

Il telefono via Internet (VOIP)



17

TV digitale



18

I tipi di computer

- Esistono due grandi classi di elaboratori:
 - Elaboratori di uso generale (general-purpose computer)
 - Elaboratori dedicati (special-purpose computer)

19

Special - purpose (embedded, dedicated) computer

- Un elaboratore dedicato (embedded system) è un elaboratore programmato per svolgere funzioni specifiche definite a priori in fase di progetto/produzione
- Esempi sono: telefoni cellulari, lettori MP3, computer che controllano aerei, auto, elettrodomestici...

20

Le razze degli elaboratori (general purpose)



Personal (client)



Workstation



Server

Mainframe (host)



21

Server

- Un server è un elaboratore che fornisce dei "servizi" a altri elaboratori (chiamati clients) attraverso una rete (computer network)



22

Server Farm

- Con il termine server farm si fa riferimento all'insieme di elaboratori server collocati in un apposito locale (centro di calcolo) presso una media o grande azienda



23

Mainframe

- Mainframes (colloquialmente indicati anche come Big Iron) sono elaboratori di grandi prestazioni usati principalmente da grandi imprese per rilevanti applicazioni software (mission critical application)



IBM z890
mainframe

24

Supercomputer (novembre 2011)

K computer
RIKEN Advanced
Institute for
Computational
Science (AICS)



Potenza: 11 PFLOPS (PETA FLOPS)

11 280 384 000 000 000 moltiplicazioni secondo

25

Cosa impariamo in questo corso?

Dalla specifica di un problema alla sua realizzazione come programma da eseguire su un elaboratore

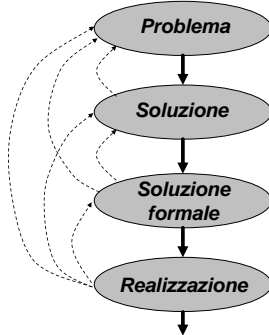


Costruzione di
un programma



26

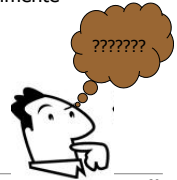
Progettare



27

Difficoltà

- I punti critici nello sviluppo di un progetto risiedono essenzialmente in:
 - Sviluppo di una soluzione "informale"
 - Formalizzazione di una soluzione
 - Permette una più semplice "traduzione" nelle regole di realizzazione
- La soluzione di un problema passa generalmente attraverso lo sviluppo di un *algoritmo*



28

Algoritmo

Con il termine di algoritmo si intende la descrizione precisa (formale) di una sequenza finita di azioni che devono essere eseguite per giungere alla soluzione di un problema



29

Algoritmo

Il termine deriva dal tardo latino "algorismus" che a sua volta deriva dal nome del matematico persiano Muhammad ibn Mūsa 'l-Khwārizmī (780-850), che scrisse un noto trattato di algebra



30

Algoritmi e vita quotidiana

1. metti l'acqua
2. accendi il fuoco
3. aspetta
4. se l'acqua non bolle torna a 3
5. butta la pasta
6. aspetta un po'
7. assaggia
8. se è cruda torna a 6
9. scola la pasta



31

Algoritmo

- Algoritmo: Sequenza di operazioni atte a risolvere un dato problema

- Esempi:

- Una ricetta di cucina
- Istruzioni di installazione di un elettrodomestico



- Spesso non è banale!

- Esempio:

- MCD?
- Quale algoritmo seguiamo per ordinare un mazzo di carte?



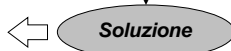
32

Esempio di flusso

- Problema: Calcolo del massimo tra due valori A e B

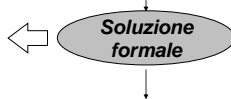


- Soluzione: Il massimo è il più grande tra A e B...



- Soluzione formale:

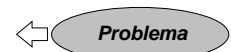
1. inizialmente: $max = 0$
2. se $A > B$ allora $max = A$; stop
3. altrimenti $max = B$; stop



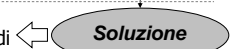
33

Altro esempio di flusso

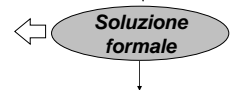
- Problema: Calcolo del massimo comun divisore (MCD) fra due valori A e B



- Soluzione: Usiamo la definizione di MCD: è il **numero naturale** più grande per il quale possono entrambi essere divisi.



- Soluzione formale: ???



34

Stadi di sviluppo di un programma

1. Scrittura di un programma

- File "sorgente"
- Scritto utilizzando un linguaggio di programmazione

Scrittura
del programma

2. Traduzione di un programma in un formato comprensibile al calcolatore

- Articolato in più fasi
- Gestito automaticamente da un programma chiamato traduttore

Traduzione
del programma

- In questo corso ci occuperemo del primo punto

- Ma è importante sapere cosa succede nella fase successiva!

35

Stadi di sviluppo di un programma

- Problema

- Idea
 - Soluzione

- Algoritmo
 - Soluzione formale

- Programma
 - Traduzione dell'algoritmo in una forma comprensibile ad un elaboratore elettronico

- Test

- Documentazione

36

Formalizzazione della soluzione

- La differenza tra una soluzione informale ed una formale sta nel modo di rappresentare un algoritmo:
 - Informale: Descrizione a parole
 - Formale: Descrizione in termini di sequenza di operazioni elementari
- Esistono vari strumenti per rappresentare una soluzione in modo formale
 - Più usati:
 - Pseudo-codice
 - Diagrammi di flusso

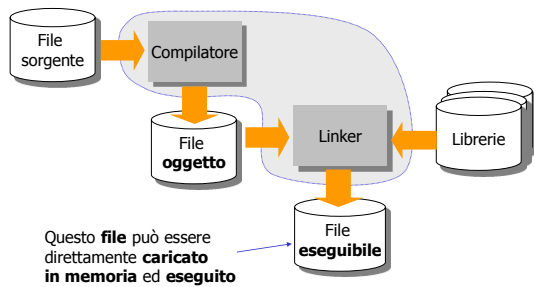
37

Formalizzazione della soluzione (Cont.)

- Pseudo-codice
 - Vantaggi
 - Immediato
 - Svantaggi
 - Descrizione dell'algoritmo poco astratta
 - Interpretazione più complicata
- Diagrammi di flusso
 - Vantaggi
 - Più intuitivi perchè utilizzano un formalismo grafico
 - Descrizione dell'algoritmo più astratta
 - Svantaggi
 - Richiedono l'apprendimento della funzione dei vari tipi di blocco

38

Traduzione di un programma



39

Cosa vuol dire "programmare"?

- La programmazione consiste nella scrittura di un "documento" (*file sorgente*) che descrive la soluzione del problema in oggetto
 - Esempio: *Massimo comun divisore tra due numeri*
- In generale non esiste "la" soluzione ad un certo problema
 - La programmazione consiste nel trovare la soluzione più efficiente, secondo una certa metrica, al problema di interesse

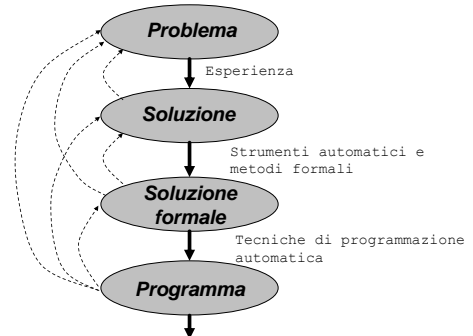
40

Cosa vuol dire "programmare"? (Cont.)

- Programmare è un'operazione "creativa"!
 - Ogni problema è diverso da ogni altro
 - Non esistono soluzioni analitiche o "universali"!
- Programmare è un'operazione complessa
 - Impensabile un approccio "diretto" (dal problema al programma sorgente definitivo)
 - Tipicamente organizzata per *stadi successivi*

41

Stadi di sviluppo di un programma (Cont.)



42

Stadi di sviluppo di un programma (Cont.)

- La costruzione di un programma è generalmente un'operazione iterativa
- Sono previsti passi a ritroso che rappresentano le reazioni a risultati non rispondenti alle esigenze nelle diverse fasi
- La suddivisione in più fasi permette di mantenere i passi a ritroso più brevi possibile (e quindi meno dispendiosi)
- E' necessario quindi effettuare dei test tra una fase e la successiva affinché i ricicli siano i più corti possibili

43

Stadi di sviluppo di un programma (Cont.)

- Una volta scritto e collaudato il programma, possono verificarsi le seguenti situazioni:
 - Il programma è stato scritto non correttamente:
Torno indietro di un livello
 - Il programma è descritto male in termini formali, ma corretto concettualmente:
Torno indietro di due livelli
 - Il programma è errato concettualmente, e necessita di una differente soluzione:
Torno all'inizio

44

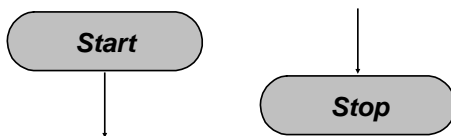
Diagrammi di flusso

Diagrammi di flusso (flow-chart)

- Sono strumenti grafici che rappresentano l'evoluzione logica della risoluzione del problema
- Sono composti da:
 - Blocchi elementari per descrivere azioni e decisioni (esclusivamente di tipo binario)
 - Archi orientati per collegare i vari blocchi e per descrivere la sequenza di svolgimento delle azioni

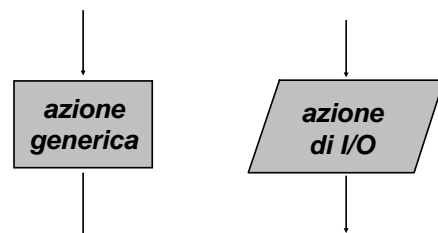
46

Diagrammi di flusso: Blocchi di inizio/fine



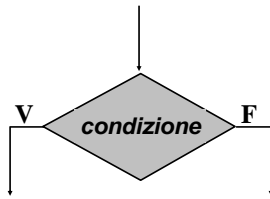
47

Diagrammi di flusso: Blocchi di azione



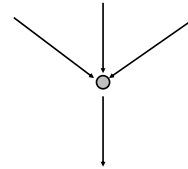
48

Diagrammi di flusso: Blocco di decisione



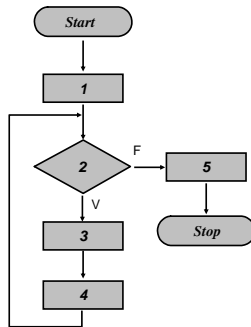
49

Diagrammi di flusso: Blocco di connessione



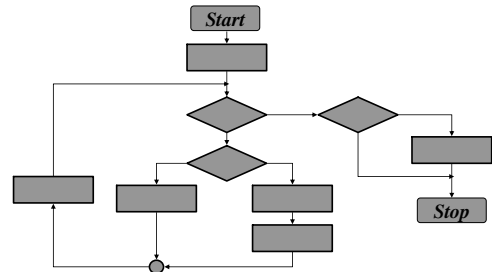
50

Diagramma di flusso : Esempio



51

Diagramma di flusso : Esempio



52

Costruzione di diagrammi di flusso

- Per mezzo dei diagrammi di flusso si possono rappresentare flussi logici complicati a piacere
- E' però preferibile scrivere diagrammi di flusso *strutturati*, che seguano cioè le regole della programmazione strutturata
- Così facendo si ottiene:
 - Maggiore formalizzazione dei problemi
 - Riusabilità del codice
 - Maggiore leggibilità

53

Diagrammi di flusso strutturati

- Definizione formale:
 - Diagrammi di flusso strutturati: Composti da *strutture elementari* indipendenti tra loro
 - Struttura elementare = Composizione particolare di blocchi elementari
 - Sempre riducibili ad un diagramma di flusso elementare costituito da un'unica azione
- Rispetto a diagrammi di flusso non strutturati questo implica l'assenza di *salti incondizionati* all'interno del flusso logico del diagramma

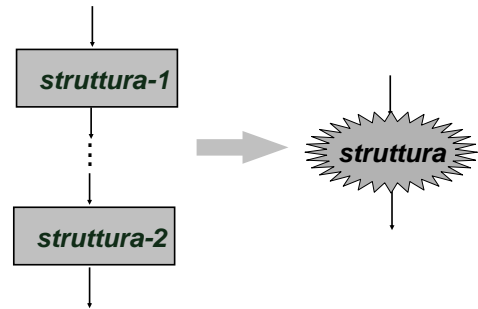
54

Diagrammi di flusso strutturati (Cont.)

- Un diagramma di flusso è detto strutturato se contiene solo un insieme predefinito di *strutture elementari*:
 - Uno ed *uno solo* blocco Start
 - Uno ed *uno solo* blocco Stop
 - Sequenza di blocchi (di azione e/o di input-output)
 - If - Then - (Else)
 - While - Do
 - Repeat - Until

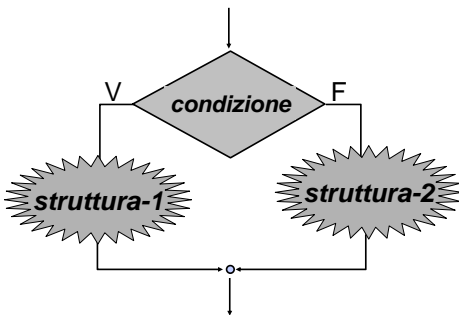
55

Sequenza



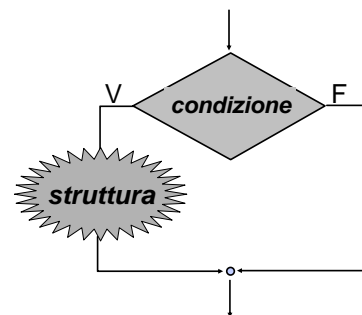
56

If-Then-Else



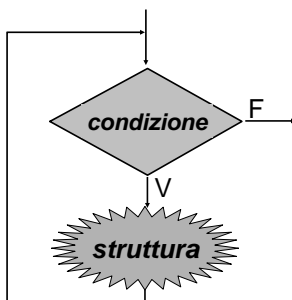
57

If-Then



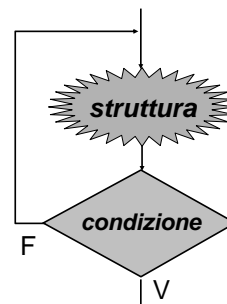
58

While-Do



59

do - while



60

Teorema di Böhm - Jacopini

Qualunque diagramma di flusso è sempre trasformabile in un diagramma di flusso strutturato equivalente a quello dato

- Quindi, qualunque flusso logico può essere realizzato utilizzando solamente due strutture di controllo:
 - Meccanismo di decisione
 - Meccanismo di ripetizione (*loop*)

61

Settimana n.2

Obiettivi

- Utilizzo del compilatore e ciclo scrittura-compilazione-esecuzione.

Contenuti

- Linguaggi di programmazione
- Dati e istruzioni
- Architettura di un elaboratore
- Il linguaggio C
- Uso del compilatore
- Variabili (tipo int, float)
- Rappresentazione dei dati numerici e non numerici

62

Dalla soluzione al programma

- La scrittura del programma vero e proprio è praticamente immediata a partire dalla soluzione formale
- I linguaggi di programmazione forniscono infatti costrutti di diversa complessità a seconda del tipo di linguaggio

63

Quali linguaggi?

- Diversi livelli (di astrazione)
 - Linguaggi ad alto livello
 - Elementi del linguaggio hanno complessità equivalente ai blocchi dei diagrammi di flusso strutturati (condizionali, cicli,...)
 - Esempio: C, C++, Basic, Pascal, Fortran, Java, etc.
 - Indipendenti dall' hardware
 - Linguaggi "assembler"
 - Elementi del linguaggio sono istruzioni microarchiteturali
 - Dipendenti dall' hardware
 - Esempio: Assembler del microprocessore Intel Pentium

64

Quali linguaggi? (Cont.)

Esempi:

- Linguaggi ad alto livello

```
...
if (x > 3) then x = x+1;
...
```

- Linguaggi assembler

```
...
LOAD Reg1, Mem[1000]
ADD Reg1, 10
...
```

Specifico per una specifica architettura (microprocessore)

65

Elementi del linguaggio

- Essendo il linguaggio un'astrazione, esistono alcuni fondamentali elementi sintattici essenziali per l'uso del linguaggio stesso:
 - **Parole chiave** (*keyword*)
 - **Dati**
 - **Identificatori**
 - **Istruzioni**
- Gli elementi sintattici definiscono la struttura formale di tutti i linguaggi di programmazione

66

Parole chiave (keyword)

- Vocaboli "riservati" al traduttore per riconoscere altri elementi del linguaggio
 - Le istruzioni sono tutte identificate da una keyword
 - Esempio: La keyword `PRINT` in alcuni linguaggi identifica il comando di visualizzazione su schermo
- Non possono essere usate per altri scopi
- Costituiscono i "mattoni" della sintassi del linguaggio

67

Dati

- Vista calcolatore:
 - *Dato = Insieme di bit memorizzato in memoria centrale*
- Vista utente:
 - *Dato = Quantità associata ad un certo significato*
- Il linguaggio di programmazione supporta la vista utente
- Dato individuato da:
 - Un **nome** (*identificatore*)
 - Una **interpretazione** (*tipo*)
 - Una **modalità di accesso** (costante o variabile)

68

Identificatore

- Indica il nome di un dato (e di altre entità) in un programma
- Permette di dare nomi intuitivi ai dati
- Esempio:
 - `X`, `raggio`, `dimensione`,...
- Nome unico all'interno di un preciso "ambiente di visibilità"
 - Dati diversi = Nomi diversi!

69

Tipo

- Indica l'interpretazione dei dati in memoria
- Legato *allo spazio occupato da un dato*
- Permette di definire tipi "primitivi" (numeri, simboli) indipendentemente dal tipo di memorizzazione del sistema

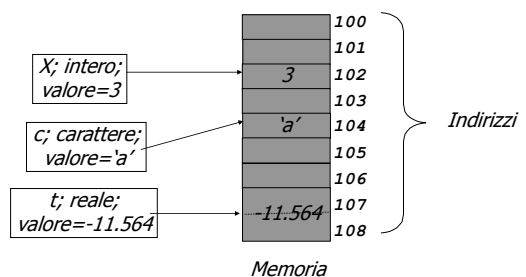
70

Tipo di accesso

- Indica la modalità di accesso ai dati:
 - Variabili
 - Dati modificabili
 - Valore modificabile in qualunque punto del programma
 - Costanti
 - Dati a sola lettura
 - Valore assegnato una volta per tutte

71

Astrazione dei dati



72

Istruzioni

- Indicano le operazioni che il linguaggio permette di eseguire (traducendole) a livello macchina:
 - **Pseudo-istruzioni**
 - Direttive non eseguibili
 - **Istruzioni elementari**
 - Operazioni direttamente corrispondenti ad *operazioni hardware*
 - Esempio: Interazione con i dispositivi di I/O, modifica/accesso a dati
 - **Istruzioni di controllo del flusso**
 - Permettono di eseguire delle combinazioni di operazioni complesse

73

Esempio di programma

```

PROGRAM prova;
// programma di prova ← Pseudo-istruzione
CONSTANTS
  pi = 3.14159
  coeff = 0.19
VARIABLES
  x: INTEGER
  y: REAL
  c: CHARACTER
BEGIN PROGRAM
  x = 2;
  IF (y > 2) THEN y = x * pi;
  PRINT x, y;
END PROGRAM
    
```

Identificatori
PAROLE CHIAVE

Specifica di celle di memoria

Specifica di tipo

Istruzione di controllo del flusso

Istruzioni elementari

Parte "operativa"

74

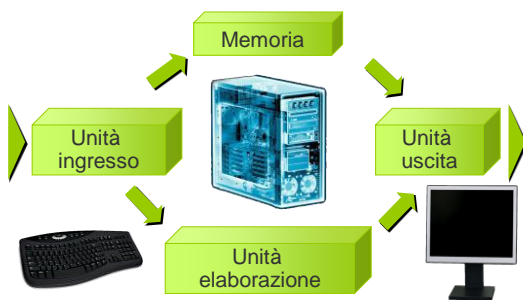
Linguaggio di programmazione

- Imparare un linguaggio significa conoscere:
 - Le parole chiave
 - I tipi predefiniti
 - Le istruzioni e la loro sintassi
- In questo corso:
 - Linguaggio C
- Estensione dei concetti a linguaggi analoghi è immediata

75

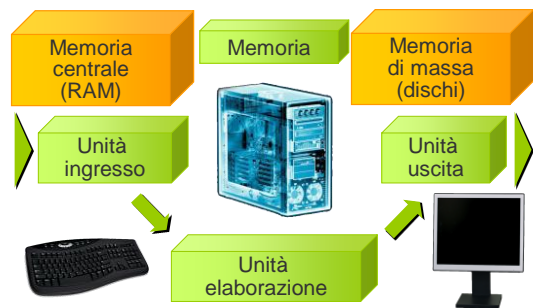
Architettura degli elaboratori

I blocchi fondamentali dell'elaboratore



77

I blocchi fondamentali dell'elaboratore



78

I chip fondamentali



Microprocessore



Memoria centrale - RAM

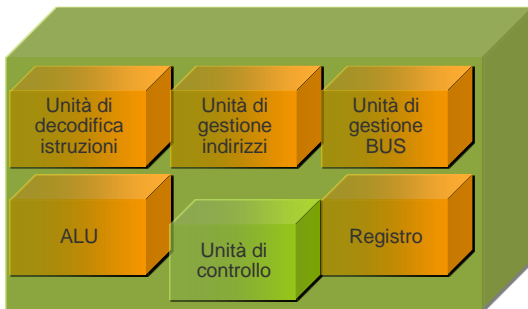
79

Microprocessore

- Un microprocessore (sovente abbreviato come μP) è un chip che realizza le funzioni di una "central processing unit (CPU)" in un computer o in un sistema digitale

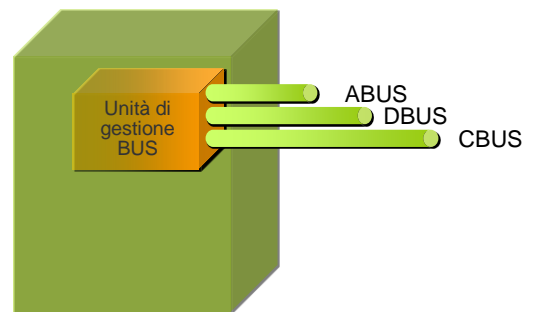
80

CPU (Central Processing Unit)



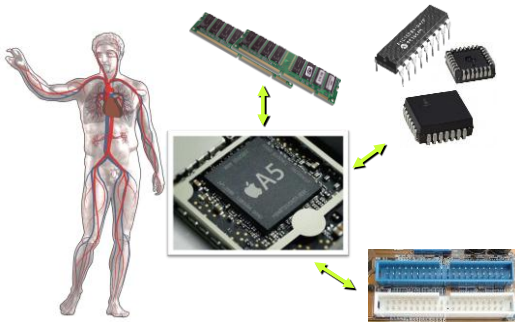
81

Microprocessore



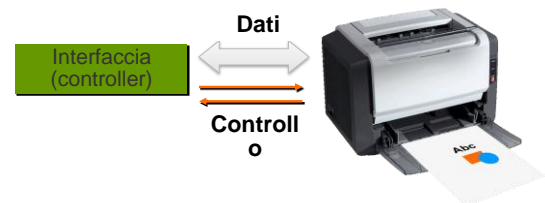
82

I Bus (sistema circolatorio del PC)



83

Dispositivi periferici



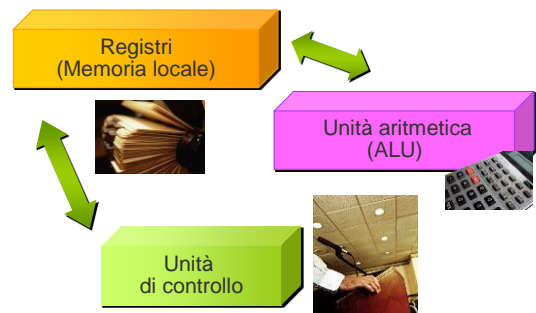
84

Unità di input/output

- Trasformano informazioni dal mondo umano a quello del computer e viceversa:
 - umano = diversi tipi di segnali fisici, analogici, asincroni
 - computer = segnali solo elettronici, digitali, sincroni

85

CPU



86

Registri

- Elementi di memoria locale usati per conservare temporaneamente dei dati (es. risultati parziali).
- Pochi (8...128)
- Dimensione di una word (8...64 bit)

87

Unità operativa

- Svolge tutte le elaborazioni richieste (aritmetiche, logiche, grafiche, ...).
- E' composta di:
 - ALU
 - flag
 - registri

88

ALU (Arithmetic-Logic Unit)

- Svolge tutti i calcoli (aritmetici e logici)
- Solitamente composta da circuiti combinatori

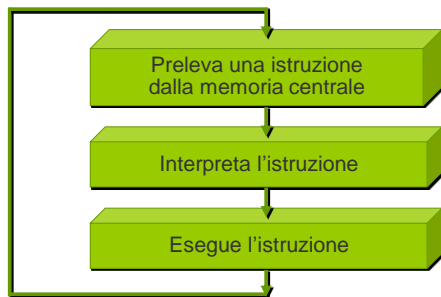
89

Unità di controllo

- E' il cervello dell'elaboratore:
 - in base al programma fornito ...
 - ed allo stato di tutte le unità ...
 - decide l'operazione da eseguire ...
 - ed emette gli ordini relativi

90

Ciclo base di un elaboratore



91

CPU e FPU

- Central Processing Unit (CPU):
 - CPU = UO + UC
 - microprocessore (mP) = CPU + "frattaglie"
- Floating Point Unit (FPU):
 - UO dedicata ai numeri reali
 - alias "coprocessore matematico"

92

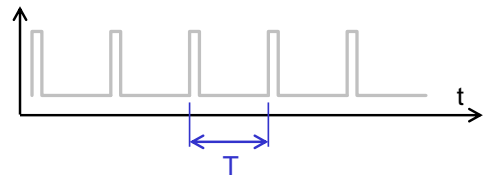
Il clock

- Ogni elaboratore contiene un elemento di temporizzazione (detto clock) che genera un riferimento temporale comune per tutti gli elementi costituenti il sistema di elaborazione.

93

Il clock

- $T = \text{periodo di clock}$
 - unità di misura = s
- $f = \text{frequenza di clock} (= 1 / T)$
 - unità di misura = s⁻¹ = Hz (cicli/s)



94

Nota

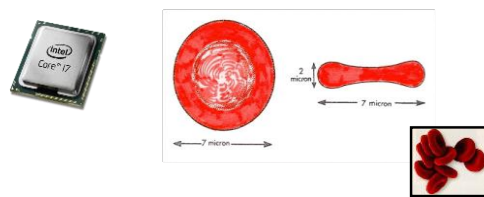
- In un Intel Core i7-2700 la frequenza di clock è 3.5 GHz
 - In 3.5 miliardesimi di secondo la luce percorre 1 metro (104.93 cm)



95

Nota

- Diversi Intel Core i7 e i5 sono costruiti con tecnologia a 32 nm
 - Il diametro di un atomo di cesio è 0.5 nm
 - Un globulo rosso è alto 2 000 nm e largo 7 000 nm
 - Un capello è spesso 100 000 nm



96

Tempistica delle istruzioni

- Un ciclo-macchina è l'intervallo di tempo in cui viene svolta una operazione elementare ed è un multiplo intero del periodo del clock
- L'esecuzione di un'istruzione richiede un numero intero di cicli macchina, variabile a seconda del tipo di istruzione

97



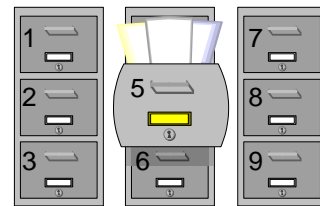
Memoria

- Memorizza i dati e le istruzioni necessarie all'elaboratore per operare.
- Caratteristiche:
 - indirizzamento
 - parallelismo
 - accesso (sequenziale o casuale)

99

Indirizzamento

- La memoria è organizzata in celle (minima unità accessibile direttamente). Ad ogni cella di memoria è associato un indirizzo (numerico) per identificarla univocamente.



100

Parallelismo

- Ogni cella di memoria contiene una quantità fissa di bit:
 - identica per tutte le celle (di una certa unità di memoria)
 - accessibile con un'unica istruzione
 - è un multiplo del byte
 - minimo un byte (tipicamente una word per la memoria principale a supporto dell'UO)

101

Memoria interna

- All'interno dell'elaboratore
- E' allo stato solido (chip)
- Solitamente è volatile
- Veloce (nanosecondi, 10-9s)
- Quantità limitata (qualche GB)
- Non rimovibile
- Costosa (0.1 € / MB)

102

Memoria esterna

- All' esterno dell' elaboratore
- Talvolta rimovibile
- Non elettronica (es. magnetica)
- Permanente
- Lenta (millisecondi, 10-3 s)
- Grande quantità (qualche TB)
- Economica (0.1 € / GB)

103

Memoria RAM (Random Access Memory)

- Circuiti integrati
- Il tempo di accesso è costante (indipendente dalla cella scelta)
- $T_a = \text{costante}$
- Ormai sinonimo di memoria interna volatile casuale a lettura e scrittura

104

La memoria RAM



105

La memoria centrale

Sistema Operativo	RAM
Programmi	RAM
Memoria Video	RAM video
Programma d'avvio (boot program)	ROM

106

Memoria RAM

- Le memorie RAM possono essere di due tipi
 - SRAM: RAM statiche
 - Veloci (10 ns)
 - Minor impaccamento
 - Elevato costo per bit
 - DRAM: RAM dinamiche
 - Meno veloci (60 ns)
 - Maggior impaccamento (64 Mbit/chip)
 - Minor costo per bit

107

La Famiglia delle DRAM

- EDO RAM
- BEDO RAM
- SD RAM
- DDR2 - DDR3
- DRAM (Rambus RAM)

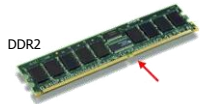
Rambus



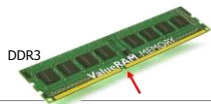
EDO RAM



DDR2



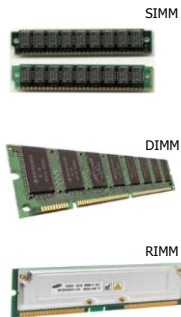
DDR3



108

Le schede delle DRAM

- SIMM
single in-line memory modules
canale di trasferimento a 32 bit
- DIMM
dual in-line memory modules
canale di trasferimento a 32 bit
- RIMM
Rambus in-line memory module



109

Memoria ROM (Read-Only Memory)

- E' un concetto (memorie a sola lettura) ... ma anche una classe di dispositivi allo stato solido (memorie a prevalente lettura = molto più veloce o facile della scrittura).
- ROM
 - dati scritti in fabbrica
- PROM (Programmable ROM)
 - dati scritti dall'utente tramite un apparecchio speciale (programmatore)

110

Memoria ROM (Cont.)

- EPROM (Erasable PROM)
 - PROM cancellabile tramite UV
- EAROM (Electrically Alterable ROM)
 - PROM cancellabile tramite circuito elettronico speciale
- EEPROM, E2PROM (Electrically Erasable PROM)
 - scrivibile/cancellabile mediante specifiche istruzioni mentre è installata sul sistema
- Flash memory
 - EEPROM veloce nella cancellazione (un blocco/tutta invece di un byte alla volta)

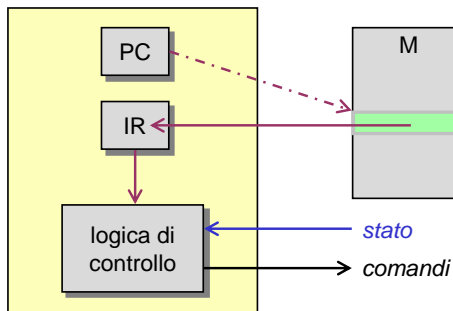
111

Unità di controllo

- E' il cuore dell'elaboratore:
 - in base al programma fornito ...
 - ed allo stato di tutte le unità ...
 - decide l'operazione da eseguire ...
 - ed emette gli ordini relativi

112

Unità di controllo: schema funzionale



113

Componenti dell' UC

- PC (Program Counter)
registro che indica sempre l'indirizzo della cella di memoria che contiene la prossima istruzione da eseguire
- IR (Instruction Register)
registro che memorizza temporaneamente l'operazione corrente da eseguire
- Logica di controllo
interpreta il codice macchina in IR per decidere ed emette gli ordini che le varie unità devono eseguire

114

Esecuzione di un'istruzione

- Tre fasi distinte:
 - fetch $IR \rightarrow M[PC]$
 $PC \rightarrow PC + 1$
 - decode $ordini \rightarrow decode(IR)$
 - execute ready? go!

115

MIPS (Million Instructions Per Second)

- f = frequenza di clock [Hz = cicli/s]
- T = periodo di clock = $1 / f$ [s]
- C = cicli macchina / istruzione
- $IPS = f / C = 1 / (T \cdot C)$
- $MIPS = IPS / 10^6$

116

MFLOPS (Million Floating-point Operations Per Second)

- Velocità di elaborazione per problemi di tipo scientifico

117

Nota

- Calcolatrice tascabile: 10 FLOPS
- Tianhe-1 (天河一号): 2 566 petaFLOPS
i.e., 2 566 000 000 000 000 FLOPS



118

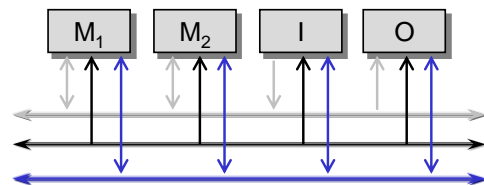
Caratteristiche di un bus

- Trasporto di un solo dato per volta
- Frequenza = n. di dati trasportati al secondo
- Ampiezza = n. di bit di cui è costituito un singolo dato
- Se mal dimensionato, potrebbe essere un collo di bottiglia

119

Tipi fondamentali di bus

- Un singolo bus è suddiviso in tre "sotto bus", detti:
 - bus dati (DBus)
 - bus degli indirizzi (ABus)
 - bus di controllo (CBus)



120

Vecchie CPU Intel per PC

CPU	DBus	ABus	Cache	FPU
8088	8 bit	20 bit	No	No
8086	16 bit	20 bit	No	No
80286	16 bit	24 bit	No	No
80386	32 bit	32 bit	No	No
80486	32 bit	32 bit	8 KB	Sì
Pentium	64 bit	32 bit	8+8KB	Sì
Pentium 3	64 bit	32 bit	8+8/256	Sì

121

Massima memoria interna (fisicamente presente)

- La dimensione dell' Abus determina il max numero di celle di memoria indirizzabili
- La dimensione del Dbus "indica" la dimensione di una cella di memoria
- $\text{max mem} = 2^{|\text{Abus}|} \times |\text{DBus}| \text{ bit}$
- Esempio (Abus da 20 bit, Dbus da 16 bit):
 - $\text{max mem} = 2^{20} \times 2 \text{ byte} = 2 \text{ MB}$
 - ossia 1 M celle di memoria, ognuna da 2 byte

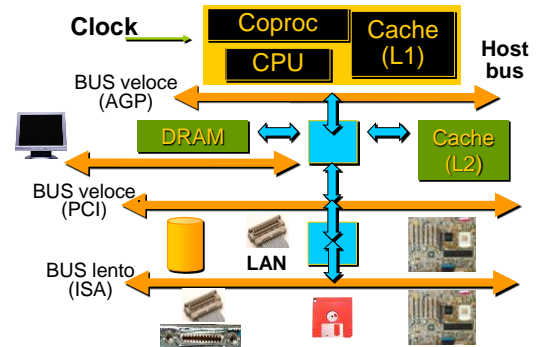
122

Massima memoria esterna

- La memoria esterna (es. dischi) non dipende dall' Abus perché viene vista come un periferico (di input e/o di output)
- La massima quantità di memoria esterna dipende dal bus di I/O (quello su cui sono collegati i periferici)

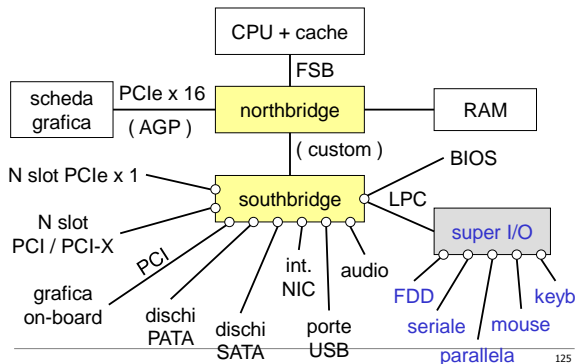
123

Architettura del PC

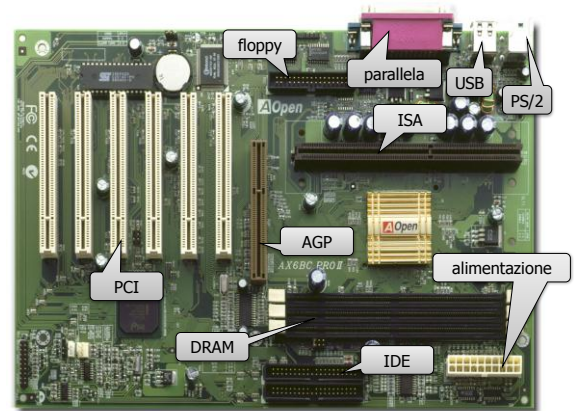


124

Architettura di un PC tradizionale



125



126

CPU multi-core

- Aumentare le prestazioni aumentando la frequenza di clock è difficile (interferenze EM, dissipazione di calore, velocità dei componenti):

- $\text{MIPS} \propto f$ ma anche $\text{Potenza (e Temperatura)} \propto f$

quindi:

$$f_0 \rightarrow 2f_0 \Rightarrow I_0 \rightarrow 2I_0 \text{ ma } T_0 \rightarrow 2T_0$$

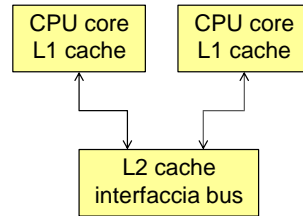
- Più facile aumentare il numero di operazioni svolte simultaneamente (CPU multi-core):

$$1 \text{ core}(f_0) = (I_0, T_0) \Rightarrow 2 \text{ core}(f_0) = (2I_0, T_0)$$

127

Esempio: CPU dual-core

- Nota: un singolo processo non può usufruire di più di un core (a meno che il programma, la CPU ed il Sistema Operativo siano multi-thread).



128

Pentium IV

- Dual core (due cpu sullo stesso chip)
- Architettura a 32 bit
- Tecnologia CMOS da 90 nm a 130 nm
- Cache
 - L1: 8K dati, 12K istruzioni
 - L2: da 256K a 2MB
 - L3: 2MB (versioni di punta)
- Clock da 1.5 a 3.8 GHz



129

AMD Athlon 64

- Dual core (due cpu sullo stesso chip)
- Architettura a 32/64 bit
- Tecnologia CMOS da 90 nm a 130 nm
- Cache
 - L1: 64K dati, 64K istruzioni
 - L2: da 512K a 1MB
 - L3: 2MB (versioni di punta)
- Clock da 1.8 a 2.4 GHz



130

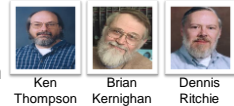
Linguaggio C

Genesi del linguaggio C

- Sviluppato tra il 1969 ed il 1973 presso gli AT&T Bell Laboratories (Ken Thompson, B. Kernighan, Dennis 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"



132

Caratteristiche generali del linguaggio C

- Il C è un linguaggio:
 - Imperativo ad alto livello
 - ... ma anche poco astratto
 - Strutturato
 - ... ma con eccezioni
 - Tipizzato
 - Ogni oggetto ha un tipo
 - Elementare
 - Poche *keyword*
 - *Case sensitive*
 - Maiuscolo diverso da minuscolo negli identificatori!
 - Portabile
 - Standard ANSI

133

Storia

- Sviluppo
 - 1969-1973
 - Ken Thompson e Dennis Ritchie
 - AT&T Bell Labs
- Versioni del C e Standard
 - K&R (1978)
 - C89 (ANSI X3.159:1989)
 - C90 (ISO/IEC 9899:1990)
 - C99 (ANSI/ISO/IEC 9899:1999, INCITS/ISO/IEC 9899:1999)
- Non tutti i compilatori sono standard!
 - GCC: *Quasi* C99, con alcune mancanze ed estensioni
 - Borland & Microsoft: *Abbastanza* C89/C90

134

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

135

Un esempio

```
#include <stdio.h>

int main(void)
{
    printf("hello, world\n");
    return 0;
}
```

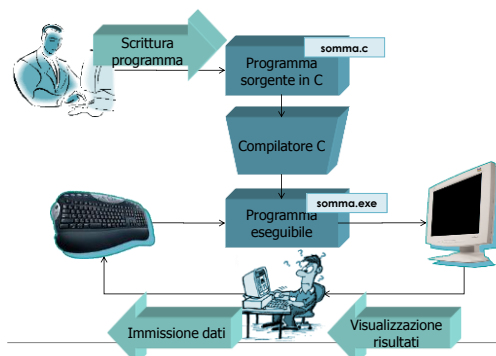
136

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

137

Modello di applicazioni "console"



138

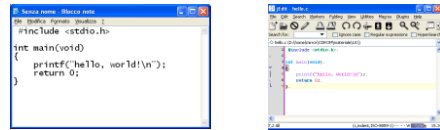
Compilatore C

- 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

139

Scrittura del programma

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



140

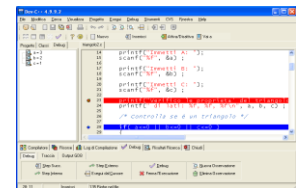
Editor per programmatori

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

141

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



142

Identificatori

- Si riferiscono ad una delle seguenti entità:
 - Costanti
 - Variabili
 - Tipi
 - Sottoprogrammi
 - File
 - Etichette
- Regole:
 - Iniziano con carattere alfabetico o "_"
 - Contengono caratteri alfanumerici o "_"

143

Identificatori (Cont.)

- Caratteristiche:
 - Esterni: Gli oggetti del sistema
 - *Case insensitive*
 - Significativi almeno i primi 6 caratteri
 - Interni: Le entità del programma
 - *Case sensitive*
 - Significativi almeno i primi 31 caratteri
 - Riservati:
 - Parole chiave del linguaggio
 - Elementi della libreria C standard

144

Commenti

- Testo libero inserito all'interno del programma
- Non viene considerato dal compilatore
- Serve al programmatore, non al sistema!
- Formato:
 - Racchiuso tra `/* */`
 - Non è possibile annidarli
 - Da `//` fino alla fine della linea
- Esempi:

```
/* Questo è un commento ! */
/* Questo /* risulterà in un */ errore */
// Questo è un altro commento
```

145

Parole chiave

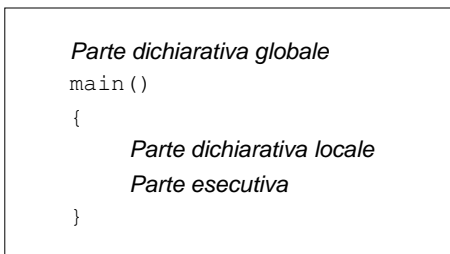
- Riservate!
- Nel C standard sono 32

<code>auto</code>	<code>double</code>	<code>int</code>	<code>struct</code>
<code>break</code>	<code>else</code>	<code>long</code>	<code>switch</code>
<code>case</code>	<code>enum</code>	<code>register</code>	<code>typedef</code>
<code>char</code>	<code>extern</code>	<code>return</code>	<code>union</code>
<code>const</code>	<code>float</code>	<code>short</code>	<code>unsigned</code>
<code>continue</code>	<code>for</code>	<code>signed</code>	<code>void</code>
<code>default</code>	<code>goto</code>	<code>sizeof</code>	<code>volatile</code>
<code>do</code>	<code>if</code>	<code>static</code>	<code>while</code>

146

Struttura di un programma C

- Struttura generale:



147

Struttura di un programma C (Cont.)

- Parte dichiarativa globale
 - Elenco degli oggetti che compongono il programma e specifica delle loro caratteristiche
 - Categoria degli oggetti
 - Tipicamente dati
 - Tipo degli oggetti
 - Numerici, non numerici
- `main`
 - Parola chiave che indica il punto di "inizio" del programma quando viene eseguito dal sistema operativo
 - Contenuto delimitato da parentesi graffe `{ ... }`

148

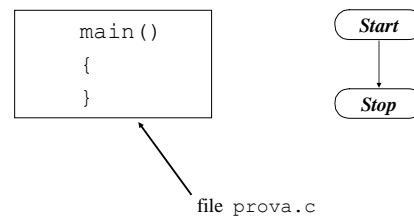
Struttura di un programma C (Cont.)

- Parte dichiarativa locale
 - Elenco degli oggetti che compongono il `main` e specifica delle loro caratteristiche
- Parte esecutiva
 - Sequenza di istruzioni
 - Quella che descriviamo con il diagramma di flusso!

149

Struttura di un programma C (Cont.)

- Programma minimo:



150

Notazione

- Per specificare la sintassi di un'istruzione utilizziamo un formalismo particolare
- Simboli utilizzati
 - `<nome>` Un generico nome
 - Esempio: `<numero>` indica che va specificato un generico valore numerico
 - `[<op>]` Un'operazione opzionale
 - `'c'` Uno specifico simbolo
 - Esempio: `'?'` indica che comparirà il carattere `?` *esplicitamente*
 - `nome` Una parola chiave

151

Pre-processor C

- La compilazione C passa attraverso un passo preliminare che precede la vera e propria traduzione in linguaggio macchina
- Il programma che realizza questa fase è detto *pre-processor*
- Funzione principale: Espansione delle direttive che iniziano con il simbolo `#`
- Direttive principali:
 - `#include`
 - `#define`

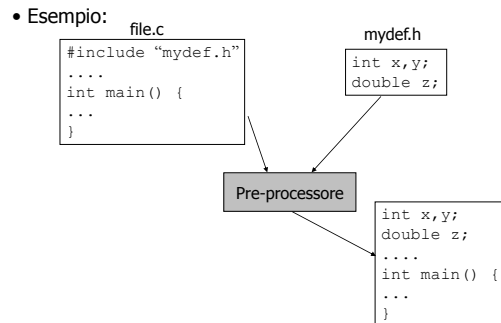
152

Direttiva #include

- Sintassi:
 - `#include <file>`
 - `<file>` può essere specificato come:
 - `'<nomefile>'` per includere un file di sistema
 - Esempio:
`#include <stdio.h>`
 - `"<nomefile>"` per includere un file definito dal programmatore
 - Esempio:
`#include "miofile.h"`
- Significato:
 - `<file>` viene espanso ed incluso per intero nel file sorgente

153

Direttiva #include



154

Dati

I dati numerici

- Sono quelli più usati in ambito scientifico nei moderni sistemi di elaborazione ... tutti gli altri tipi di dato sono trasformati in dati numerici
- Tutti i tentativi di elaborare direttamente dati non numerici o sono falliti o si sono mostrati molto più inefficienti che non effettuare l'elaborazione solo dopo aver trasformato i dati in forma numerica

156

Come contiamo?

- Il sistema di numerazione del mondo occidentale (sistema indo-arabo) è:
 - decimale
 - posizionale

$$\begin{aligned}252 &= 2 \times 100 + 5 \times 10 + 2 \times 1 \\ &= 2 \times 10^2 + 5 \times 10^1 + 2 \times 10^0\end{aligned}$$

157

Sistemi di numerazione

- Non posizionali (additivi):
 - egiziano
 - romano
 - greco

158

Sistemi di numerazione

- Posizionali:
 - babilonese (2 cifre, sessagesimale)
 - inuit, seltsi, maya (ventesimale)
 - indo-arabo (decimale)

159

Sistemi di numerazione

- Ibridi:
 - cinese

160

Sistema di numerazione posizionale

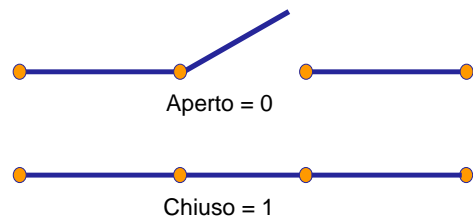
- Occorre definire la base B da cui discendono varie caratteristiche:
 - cifre = $\{0, 1, 2, \dots, B-1\}$
 - peso della cifra i-esima = B^i
 - rappresentazione (numeri naturali) su N cifre

$$A = \sum_{i=0}^{N-1} a_i \cdot B^i$$

161

Bit e interruttori

Interruttore ha due stati
(aperto-chiuso, ON_OFF)



162

Il sistema binario

- Base = 2
- Cifre = { 0, 1 }
- Esempio:

$$101_2 = 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 1 \times 4 + 1 \times 1$$

$$= 5_{10}$$

163

Alcuni numeri binari

0	...	0	1000	...	8
1	...	1	1001	...	9
10	...	2	1010	...	10
11	...	3	1011	...	11
100	...	4	1100	...	12
101	...	5	1101	...	13
110	...	6	1110	...	14
111	...	7	1111	...	15

164

Alcune potenze di due

2^0	...	1	2^9	...	512
2^1	...	2	2^{10}	...	1024
2^2	...	4	2^{11}	...	2048
2^3	...	8	2^{12}	...	4096
2^4	...	16	2^{13}	...	8192
2^5	...	32	2^{14}	...	16384
2^6	...	64	2^{15}	...	32768
2^7	...	128	2^{16}	...	65536
2^8	...	256			

165

Conversione di numeri naturali da binario a decimale

- Si applica direttamente la definizione effettuando la somma pesata delle cifre binarie:

$$1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$$

$$= 8 + 4 + 0 + 1$$

$$= 13_{10}$$

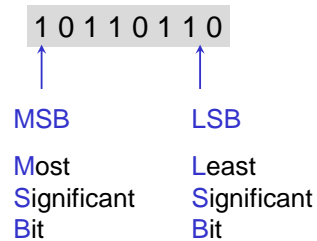
166

Terminologia

- Bit
- Byte
- Word
- Double word/Long word

167

Terminologia



168

Limiti del sistema binario (rappresentazione naturale)

- Consideriamo numeri naturali in binario:
 - 1 bit \sim 2 numeri $\sim \{0, 1\}_2 \sim [0 \dots 1]_{10}$
 - 2 bit \sim 4 numeri $\sim \{00, 01, 10, 11\}_2 \sim [0 \dots 3]_{10}$
- Quindi in generale per numeri naturali a N bit:
 - combinazioni distinte 2^N
 - intervallo di valori

0	$\leq x \leq$	$2^N - 1$	[base 10]
$(000 \dots 0)$	$\leq x \leq$	$(111 \dots 1)$	[base 2]

169

Limiti del sistema binario (rappresentazione naturale)

<i>bit</i>	<i>simboli</i>	<i>min₁₀</i>	<i>max₁₀</i>
4	16	0	15
8	256	0	255
16	65 536	0	65 535
32	4 294 967 296	0	4 294 967 295

170

Somma in binario

- Regole base:

$0 + 0 = 0$
$0 + 1 = 1$
$1 + 0 = 1$
$1 + 1 = 0$ (carry = 1)

171

Somma in binario

- Si effettuano le somme parziali tra i bit dello stesso peso, propagando gli eventuali riporti:

$$\begin{array}{r}
 11 \\
 0110 + \\
 0111 = \\
 \hline
 1101
 \end{array}$$

172

Sottrazione in binario

- Regole base:

$0 - 0 = 0$
$0 - 1 = 1$ (borrow = 1)
$1 - 0 = 1$
$1 - 1 = 0$

173

Sottrazione in binario

- Si effettuano le somme parziali tra i bit dello stesso peso, propagando gli eventuali riporti:

$$\begin{array}{r}
 1 \\
 0001 - \\
 0110 = \\
 \hline
 0011
 \end{array}$$

174

Overflow

- Si usa il termine *overflow* per indicare l'errore che si verifica in un sistema di calcolo automatico quando il risultato di un'operazione non è rappresentabile con la medesima codifica e numero di bit degli operandi.

175

Overflow

- Nella somma in binario puro si ha overflow quando:
 - si lavora con numero fisso di bit
 - si ha carry sul MSB

176

Overflow - esempio

- Ipotesi: operazioni su numeri da 4 bit codificati in binario puro

$$\begin{array}{r} 0101 + \\ 1110 = \\ \hline 10011 \end{array}$$

↑
overflow

177

Il sistema ottale

- base = 8 (talvolta indicata con Q per Octal)
 - cifre = { 0, 1, 2, 3, 4, 5, 6, 7 }
 - utile per scrivere in modo compatto i numeri binari (3:1)

$$\begin{array}{cccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} & & & & & \\ 2 & & 7 & & & 1 & & 8 \end{array}$$

178

Il sistema esadecimale

- base = 16 (talvolta indicata con H per Hexadecimal)
 - cifre = { 0, 1, ..., 9, A, B, C, D, E, F }
 - utile per scrivere in modo compatto i numeri binari (4:1)

$$\begin{array}{cccc} 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 \\ \underbrace{\hspace{1.5cm}} & \underbrace{\hspace{1.5cm}} & & & & & & \\ B & & 9 & & & & & \end{array}$$

179

Rappresentazione dei numeri interi relativi



$$\begin{array}{r} + 25 \\ \text{° C} \end{array}$$



$$\begin{array}{r} - 9 \\ \text{° C} \end{array}$$

180

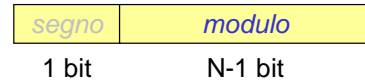
I numeri con segno

- Il segno dei numeri può essere solo di due tipi:
 - positivo (+)
 - negativo (-)
- E' quindi facile rappresentarlo in binario ... ma non sempre la soluzione più semplice è quella migliore!
 - Modulo e segno
 - Complemento a uno
 - Complemento a due
 - Eccesso X

181

Codifica "modulo e segno"

- un bit per il segno (tipicamente il MSB):
 - 0 = segno positivo (+)
 - 1 = segno negativo (-)
- N-1 bit per il valore assoluto (anche detto modulo)



182

Modulo e segno: esempi

- Usando una codifica su quattro bit:

$+3_{10}$	\rightarrow	$0011_{M\&S}$
-3_{10}	\rightarrow	$1011_{M\&S}$
$0000_{M\&S}$	\rightarrow	$+0_{10}$
$1000_{M\&S}$	\rightarrow	-0_{10}

183

Modulo e segno

- Svantaggi:
 - doppio zero (+0, -0)
 - operazioni complesse
 - es. somma A+B

	$A > 0$	$A < 0$
$B > 0$	$A + B$	$B - A $
$B < 0$	$A - B $	$- (A + B)$

184

Modulo e segno: limiti

- In una rappresentazione M&S su N bit:

$$- (2^{N-1} - 1) \leq x \leq + (2^{N-1} - 1)$$

- Esempi:

- 8 bit	=	[-127 ... +127]
- 16 bit	=	[-32 767 ... +32 767]

185

Codifica in complemento a due

- In questa codifica per un numero a N bit:
 - il MSB ha peso negativo (pari a -2^{N-1})
 - gli altri bit hanno peso positivo
- Ne consegue che MSB indica sempre il segno:
 - 0 = + 1 = -
- Esempi (complemento a due su 4 bit):
 - $1000_{CA2} = -2^3 = -8_{10}$
 - $1111_{CA2} = -2^3 + 2^2 + 2^1 + 2^0 = -8 + 4 + 2 + 1 = -1_{10}$
 - $0111_{CA2} = 2^2 + 2^1 + 2^0 = 7_{10}$

186

Complemento a 2

- La rappresentazione in complemento a due è oggi la più diffusa perché semplifica la realizzazione dei circuiti per eseguire le operazioni aritmetiche
- Possono essere applicate le regole binarie a tutti i bit

187

Somma e sottrazione in CA2

- La somma e sottrazione si effettuano direttamente, senza badare ai segni degli operandi:

$$\begin{aligned} \bullet A_{CA2} + B_{CA2} &\rightarrow A_{CA2} + B_{CA2} \\ \bullet A_{CA2} - B_{CA2} &\rightarrow A_{CA2} - B_{CA2} \end{aligned}$$

188

Somma e sottrazione in CA2

- La sottrazione si può effettuare sommando al minuendo il CA2 del sottraendo:

$$A_{CA2} - B_{CA2} \rightarrow A_{CA2} + \overline{\overline{B_{CA2}}}$$

(nota: nella sottrazione B è già codificato in CA2 ma gli si applica l'operazione CA2)

189

Somma in CA2 - esempio

$$00100110 + 11001011$$

$$\begin{array}{r} 00100110 + \\ 11001011 = \\ \hline 11110001 \end{array}$$

$$\text{verifica: } 38 + (-53) = -15$$

190

Sottrazione in CA2 - esempio

$$00100110 - 11001011$$

$$\begin{array}{r} 00100110 - \\ 11001011 = \\ \hline 01011011 \end{array}$$

$$\text{verifica: } 38 - (-53) = 91$$

191

Overflow nella somma in CA2

- Operandi con segno discorde: non si può mai verificare overflow.
- Operandi con segno concorde: c'è overflow quando il risultato ha segno discorde.
- In ogni caso, si trascura sempre il carry sul MSB.

192

Esempio overflow in CA2 (somma)

$\begin{array}{r} 0101 + \\ 0100 = \\ \hline 1001 \\ \uparrow \\ \text{overflow!} \end{array}$ <ul style="list-style-type: none"> • $(5 + 4 = 9)$ • impossibile in CA2 su 4 bit 	$\begin{array}{r} 1110 + \\ 1101 = \\ \hline 11011 = \\ 1011 \\ \text{OK} \end{array}$ <p>$(-2 + -3 = -5)$</p>
--	---

193

Overflow nella sottrazione in CA2

- Poiché la differenza in CA2 è ricondotta ad una somma, in generale per l'overflow valgono le stesse regole della somma.
- Fa eccezione il caso in cui il sottraendo è il valore più negativo possibile (10...0); in questo caso la regola è:
 - minuendo negativo: non si può mai verificare overflow
 - minuendo positivo: c'è sempre overflow
- Nota: come si giustifica questa regola?

194

Esempio overflow in CA2 (sottrazione)

$\begin{array}{r} 3 - (-8) \\ 0011 + \\ 1000 = \\ \hline 1011 \\ \text{overflow!} \end{array}$ <p>$+11$ è impossibile in CA2 su 4 bit</p>	$\begin{array}{r} -3 - (-8) \\ 1101 + \\ 1000 = \\ \hline 10101 = \\ 0101 \\ \text{OK (+5)} \end{array}$
--	--

195

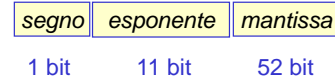
Formato IEEE-754

- Mantissa nella forma "1,..." (valore max < 2)
- Base dell'esponente pari a 2

- IEEE 754 SP:

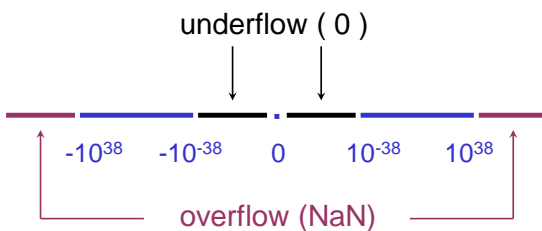


- IEEE 754 DP:



196

IEEE-754 SP: intervallo di valori



197

Problemi

- Numero fisso del numero di bit
- I numeri sono rappresentati da sequenze di cifre
- Problemi:
 - Intervallo di rappresentazione
 - Overflow
 - Precisione

198

Errore assoluto (ϵ)

- Dato un numero A composto da N cifre, l'errore assoluto della sua rappresentazione è non superiore alla quantità – non nulla – più piccola (in valore assoluto) rappresentabile con N cifre
- Nota: talvolta l'errore assoluto è anche detto "precisione (assoluta)" di un numero

199

Errore assoluto - esempi

- Qualunque sia la base ed il numero di cifre, l'errore assoluto dei numeri interi è sempre non superiore a 1:

$$5_{10} \rightarrow \epsilon \leq 110 \quad 27_{10} \rightarrow \epsilon \leq 110$$

- L'errore assoluto dei numeri razionali dipende dal numero di cifre usate per rappresentarli:
 - es. 0.510 $\epsilon \leq 0.110$
 - es. 0.5010 $\epsilon \leq 0.0110$
 - es. 1.010 $\epsilon \leq 0.110$

200

Interi o floating-point?

- Interi:
 - precisione = 1
 - intervallo di valori limitato
 - es. (32 bit) $\sim \pm 2\,000\,000\,000$
 - es. (64 bit) $\sim \pm 9\,000\,000\,000\,000\,000\,000$
- Floating-point:
 - precisione = variabile
 - es. (32 bit) $\sim \pm 9\,999\,999 \times 10^{\pm 38}$
 - es. (64 bit) $\sim \pm 9\,999\,999\,999\,999\,999 \times 10^{\pm 308}$

201

Elaborazione dell'informazione non numerica



202

Informazione non numerica

- Se in quantità finita, si può mettere in corrispondenza coi numeri interi.



203

Rappresentazioni numeriche

- Dati N bit ...
 - si possono codificare 2^N "oggetti" distinti
 - usabili per varie rappresentazioni numeriche

- Esempio (usando 3 bit):

"oggetti" binari	000	001	010	011	100	101	110	111
num. naturali	0	1	2	3	4	5	6	7
num. relativi (M&S)	+0	+1	+2	+3	-0	-1	-2	-3
num. relativi (CA2)	+0	+1	+2	+3	-4	-3	-2	-1

204

Caratteri

- Occorre una codifica standard perché è il genere di informazione più scambiata:
 - codice ASCII (American Standard Code for Information Interchange)
 - codice EBCDIC (Extended BCD Interchange Code)

205

Codice ASCII

- Usato anche nelle telecomunicazioni.
- Usa 8 bit (originariamente 7 bit per US-ASCII) per rappresentare:
 - 52 caratteri alfabetici (a...z A...Z)
 - 10 cifre (0...9)
 - segni di interpunzione (,;!?...)
 - caratteri di controllo

206

Caratteri di controllo

CR	(13)	Carriage Return
LF,	NL (10)	New Line, Line Feed
FF,	NP (12)	New Page, Form Feed
HT	(9)	Horizontal Tab
VT	(11)	Vertical Tab
NUL	(0)	Null
BEL	(7)	Bell
EOT	(4)	End-Of-Transmission
...

207

Codice ASCII - esempio

01000001	A	00100000	
01110101	u	01110100	t
01100111	g	01110101	u
01110101	u	01110100	t
01110010	r	01110100	t
01101001	i	01101001	i
00100000		00100001	!
01100001	a		

208

UNICODE e UTF-8

- Unicode esprime tutti i caratteri di tutte le lingue del mondo (più di un milione).
- UTF-8 è la codifica di Unicode più usata:
 - 1 byte per caratteri US-ASCII (MSB=0)
 - 2 byte per caratteri Latini con simboli diacritici, Greco, Cirillico, Armeno, Ebraico, Arabo, Siriano e Maldiviano
 - 3 byte per altre lingue di uso comune
 - 4 byte per caratteri rarissimi
 - raccomandata da IETF per e-mail

209

Rappresentazione di un testo in formato ASCII

- Caratteri in codice ASCII
- Ogni riga terminata dal terminatore di riga:
 - in MS-DOS e Windows = CR + LF
 - in UNIX = LF
 - in MacOS = CR
- Pagine talvolta separate da FF

210

Codifiche o formati di testo/stampa

- Non confondere il formato di un file word, con codice ASCII!!
- Un testo può essere memorizzato in due formati
 - Formattato: sono memorizzate sequenze di byte che definiscono l'aspetto del testo (e.g., font, spaziatura)
 - Non formattato: sono memorizzati unicamente i caratteri che compongono il testo

211

Formato testi

NEL MEZZO DEL ... ←(11U← s0p12.00s0b3T NEL MEZZO

NEL MEZZO DEL ... ←(15U← s0p12.00s0b3N NEL MEZZO

NEL MEZZO DEL ... ←(16U← s0p9.00s0b3Q NEL MEZZO



Caratteri
di controllo



Caratteri
di stampa(ASCII)

212

PDF

- Il PDF (Portable Document Format) è un formato open di file basato su un linguaggio di descrizione di pagina sviluppato da Adobe Systems per rappresentare documenti in modo indipendente dall'hardware e dal software utilizzati per generarli o per visualizzarli
- Un file PDF può descrivere documenti che contengono testo e/o immagini a qualsiasi risoluzione

213

Dichiarazione di dati

- In C, tutti i dati devono essere dichiarati prima di essere utilizzati!
- La dichiarazione di un dato richiede:
 - L'**allocazione** di uno spazio in memoria atto a contenere il dato
 - L'**assegnazione** di un nome a tale spazio in memoria
- In particolare, occorre specificare:
 - Nome (identificatore)
 - Tipo
 - Modalità di accesso (variabile o costante)

214

Tipi base (primitivi)

- Sono quelli forniti direttamente dal C
- Identificati da parole chiave!
 - `char` caratteri ASCII
 - `int` interi (complemento a 2)
 - `float` reali (floating point singola precisione)
 - `double` reali (floating point doppia precisione)
- La dimensione precisa di questi tipi dipende dall'architettura (non definita dal linguaggio)
 - `|char| = 8 bit = 1 Byte` sempre

215

Modificatori dei tipi base

- Sono previsti dei modificatori, identificati da parole chiave da premettere ai tipi base
 - Segno:
 - `signed/unsigned`
 - Applicabili ai tipi `char` e `int`
 - » `signed`: Valore numerico con segno
 - » `unsigned`: Valore numerico senza segno
 - Dimensione:
 - `short/long`
 - Applicabili al tipo `int`
 - Utilizzabili anche senza specificare `int`

216

Modificatori dei tipi base (Cont.)

- Interi

- [signed/unsigned] short [int]
- [signed/unsigned] int
- [signed/unsigned] long [int]

- Reali

- float
- double

217

Variabili

- Locazioni di memoria destinate alla memorizzazione di dati il cui valore è modificabile
- Sintassi:

<tipo> <variabile> ;

<variabile>: Identificatore che indica il nome della variabile

- Sintassi alternativa (dichiarazioni multiple):

<tipo> <lista di variabili> ;

<lista di variabili>: Lista di identificatori separati da ','

218

Variabili (Cont.)

- Esempi:

```
int x;  
char ch;  
long int x1, x2, x3;  
double pi;  
short int stipendio;  
long y, z;
```

- Usiamo nomi significativi!

- Esempi:
 - int x0all; /* NO */
 - int valore; /* SI */
 - float raggio; /* SI */

219

Esempi di nomi

a	b	a1	a2	
num	n	N	somma	max
area	perimetro	perim		
n_elementi	Nelementi	risultato		
trovato	nome	risposta		

220

Esempi

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

i	0
j	2
N	100
x	-3124

221

Esempi

```
float a, b ;  
float pigr ;  
float Nav, Qe ;  
  
a = 3.1 ;  
b = 2.0 ;  
pigr = 3.1415926 ;  
Nav = 6.02e23 ;  
Qe = 1.6e-19 ;
```

a	3.1
b	2.0
pigr	3.1415
Nav	6.02×10^{23}
Qe	1.6×10^{-19}

222

Valore contenuto

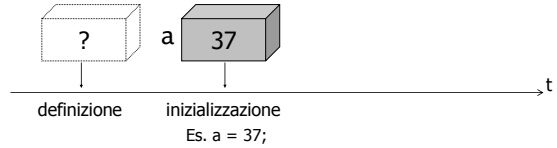
- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
 - Variabili non inizializzate
- In momenti diversi il valore può cambiare



223

Valore contenuto

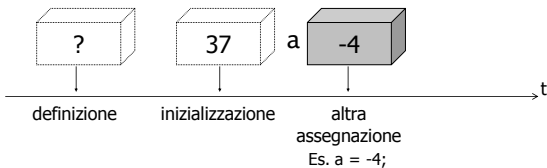
- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
 - Variabili non inizializzate
- In momenti diversi il valore può cambiare



224

Valore contenuto

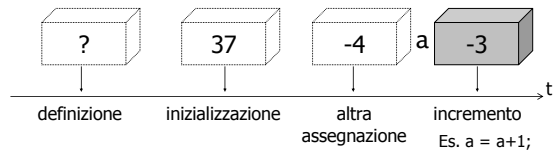
- Ogni variabile, in ogni istante di tempo, possiede un certo valore
- Le variabili appena definite hanno valore ignoto
 - Variabili non inizializzate
- In momenti diversi il valore può cambiare



225

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



226

Costanti

- Locazioni di memoria destinate alla memorizzazione di dati il cui valore non è modificabile
- Sintassi:

```
const <tipo> <costante> = <valore> ;
```

<costante> : Identificatore che indica il nome della costante
<valore> : Valore che la costante deve assumere

- Esempi:
 - const double PIGRECO = 3.14159;
 - const char SEPARATORE = '\$';
 - const float ALIQUOTA = 0.2;

- Convenzione:
 - Identificatori delle costanti tipicamente in MAIUSCOLO

227

Costanti (Cont.)

- Esempi di valori attribuibili ad una costante:

- Costanti di tipo char:
 - 'f'
- Costanti di tipo int, short, long
 - 26
 - 0x1a, 0X1a
 - 26L
 - 26u
 - 26UL
- Costanti di tipo float, double
 - -212.6
 - -2.126e2, -2.126E2, -212.6f

228

Costanti speciali

- Caratteri ASCII non stampabili e/o "speciali"
- Ottenibili tramite "sequenze di escape"
`\<codice ASCII ottale su tre cifre>`
- Esempi:
 - `'\007'`
 - `'\013'`
- Caratteri "predefiniti"
 - `'\b'` backspace
 - `'\f'` form feed
 - `'\n'` line feed
 - `'\t'` tab

229

Visibilità delle variabili

- Ogni variabile è utilizzabile all'interno di un preciso ambiente di visibilità (*scope*)
- Variabili *globali*
 - Definite all'esterno del `main()`
- Variabili *locali*
 - Definite all'interno del `main()`
 - Più in generale, definite all'interno di un blocco

230

Struttura a blocchi

- In C, è possibile raccogliere istruzioni in blocchi racchiudendole tra parentesi graffe
- Significato: Delimitazione di un ambiente di visibilità di "oggetti" (variabili, costanti)
- Corrispondente ad una "sequenza" di istruzioni
- Esempio:

```
{  
    int a=2;  
    int b;  
    b=2*a;  
}
```

a e b sono definite solo all'interno del blocco!

231

Visibilità delle variabili: Esempio

```
int n;  
double x;  
main() {  
    int a,b,c;  
    double y;  
    {  
        int d;  
        double z;  
    }  
}
```

- `n, x`: Visibili in tutto il file
- `a, b, c, y`: Visibili in tutto il `main()`
- `d, z`: Visibili nel blocco delimitato dalle parentesi graffe

232

Settimana n.3

Obiettivi

- Struttura base di un programma in C.
- Costrutti condizionali semplici
- Condizioni complesse
- Costrutti condizionali annidati

Contenuti

- `scanf` e `printf` a livello elementare
- Espressioni aritmetiche ed operatori base (+ - * / %)
- Operatori relazionali
- Algebra di Boole
- Costrutto `if` e `if-else`
- Operatori logici e di incremento
- `if` annidati



233

Istruzioni elementari

Istruzioni elementari

- Corrispondono ai blocchi di azione dei diagrammi di flusso:

- Due categorie:

- Assegnazione 
- Input/output (I/O) 

235

Assegnazione

- Sintassi:

`<variabile> = <valore>`

- Non è un'uguaglianza!
 - Significato: `<valore>` viene assegnato a `<variabile>`
 - `<variabile>` e `<valore>` devono essere di tipi "compatibili"
 - `<variabile>` deve essere stata dichiarata precedentemente!

- Esempi:

```
int x;  
float y;  
x = 3;  
y = -323.9498;
```

- Può essere inclusa nella dichiarazione di una variabile

- Esempi:

- `int x = 3;`
- `float y = -323.9498;`

236

Istruzioni di I/O

- Diverse categorie in base al tipo di informazione letta o scritta:

- I/O formattato
- I/O a caratteri
- I/O "per righe"
 - Richiede la nozione di stringa. Come tale, sarà trattata in seguito

- Nota:

In C, le operazioni di I/O non sono gestite tramite vere e proprie istruzioni, bensì mediante opportune funzioni.

- Il concetto di funzione verrà introdotto successivamente; in questa sezione le funzioni di I/O saranno impropriamente chiamate istruzioni

237

I/O formattato

- Output

- Istruzione `printf()`

- Input

- Istruzione `scanf()`

- L'utilizzo di queste istruzioni richiede l'inserimento di una direttiva

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

all'inizio del file sorgente

- Significato: "includi il file `stdio.h`"
- Contiene alcune dichiarazioni

238

Istruzione `printf()`

- Sintassi:

```
printf(<formato>, <arg1>, ..., <argn>);
```

`<formato>`: Sequenza di caratteri che determina il formato di stampa di ognuno dei vari argomenti

- Può contenere:

- Caratteri (stampati come appaiono)
- Direttive di formato nella forma `%<carattere>`
 - > %d intero
 - > %u unsigned
 - > %s stringa
 - > %c carattere
 - > %x esadecimale
 - > %o ottale
 - > %f float
 - > %g double

- `<arg1>`, ..., `<argn>`: Le quantità (espressioni) che si vogliono stampare
 - Associati alle direttive di formato nello stesso ordine!

239

Istruzione `printf()`: Esempi

```
int x=2;  
float z=0.5;  
char c='a';
```

```
printf("%d %f %c\n", x, z, c);
```

output

```
2 0.5 a
```

```
printf("%f***%c***%d\n", z, c, x);
```

output

```
0.5***a***2
```

240

Istruzione scanf()

• Sintassi:

```
scanf(<formato>, <arg1>, ..., <argn>);
```

<formato>: come per printf

<arg1>, ..., <argn>: le variabili cui si vogliono assegnare valori

• IMPORTANTE:

I nomi delle variabili vanno precedute dall'operatore & che indica l'indirizzo della variabile (vedremo più avanti il perchè)

• Esempio:

```
int x;
float z;
scanf("%d %f", &x, &z);
```

241

Significato di scanf()

• Istruzioni di input vanno viste come assegnazioni dinamiche:

- L'assegnazione dei valori alle variabili avviene al tempo di esecuzione e viene deciso dall'utente

• Assegnazioni tradizionali = Assegnazioni statiche

- L'assegnazione dei valori alle variabili è scritta nel codice!

242

I/O formattato avanzato

• Le direttive della stringa formato di printf e scanf sono in realtà più complesse

- printf:

```
%[flag][min dim][.precisione][dimensione]<carattere>
```

• [flag]: Più usati

- Giustificazione della stampa a sinistra
- + Premette sempre il segno

• [min dim]: Dimensione minima di stampa in caratteri

• [precisione]: Numero di cifre frazionarie (per numeri reali)

• [dimensione]: Uno tra:

- h argomento è short
- l argomento è long

• carattere: Visto in precedenza

243

I/O formattato avanzato (Cont.)

- scanf:

```
%[*][max dim][dimensione]<carattere>
```

• [*]: Non fa effettuare l'assegnazione (ad es., per "saltare" un dato in input)

• [max dim]: Dimensione massima in caratteri del campo

• [dimensione]: Uno tra:

- h argomento è short
- l argomento è long

• carattere: Visto in precedenza

244

printf() e scanf(): Esempio

```
#include <stdio.h>
main()
{
    int a;
    float b;

    printf("Dammi un numero intero (A): ");
    if(scanf("%d", &a) != 1)
    {
        printf("Errore!\n");
        return 1;
    }
    printf("Dammi un numero reale (B): ");
    if(scanf("%f", &b) != 1)
    {
        printf("Errore!\n");
        return 1;
    }
    printf("A= %d\n", a);
    printf("B= %f\n", b);
}
```

245

Espressioni

• Combinazioni di variabili, costanti ed operatori

• Il valore di un'espressione può essere assegnato ad una variabile:

<variabile> = <espressione>

- Significato:

<espressione> è "valutata" ed il valore ottenuto è assegnato a <variabile>

- <variabile> e <espressione> devono essere di tipi "compatibili"

• Esistono varie categorie di operatori, applicabili a tipi di dato diversi:

- Operatori aritmetici
- Operatori relazionali
- Operatori logici
- Operatori su bit
- Operatori di modifica del tipo (cast)
- Operatori di calcolo della dimensione di un tipo: sizeof()

246

Operatori aritmetici

- Quattro operatori (per numeri reali e interi):
+ - * /
- Per numeri interi, esiste l'operatore % che ritorna il resto della divisione intera

- Stesse regole di precedenza dell'aritmetica ordinaria

- Esempi:

```
int x=5;
int y=2;
int q, r;
q = x / y; // (q = 2, troncamento)
r = x % y; // (r = 1)
```

247

Divisione tra interi: Esempio

```
#include <stdio.h>

main()
{
    int a, b;

    printf("Dammi un numero intero (A): ");
    scanf("%d", &a);
    printf("\n");
    printf("Dammi un numero intero (B): ");
    scanf("%d", &b);
    printf("\n");
    printf("A div B = %d\n", a/b);
    printf("A mod B = %d\n", a%b);
}
```

248

Quesito

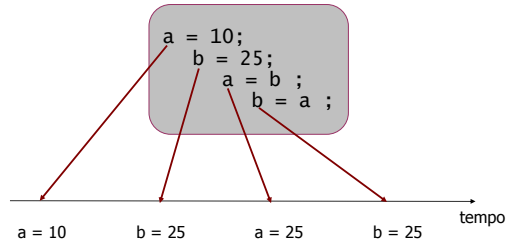
- Che operazione svolge il seguente frammento di programma?

```
a = 10;
b = 25;
a = b;
b = a;
```

249

Soluzione

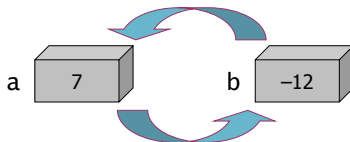
- Che operazione svolge il seguente frammento di programma?



250

Quesito

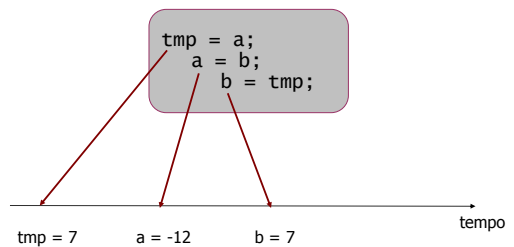
- Come fare a scambiare tra di loro i valori di due variabili?



251

Soluzione

- E' necessario utilizzare una variabile di appoggio



252

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: $a <= b$

253

Operatori relazionali

- Operano su quantità numeriche o `char` e forniscono un risultato "booleano":
 $< \leq > \geq == !=$
- Il risultato è sempre di tipo `int`
 - risultato = 0 FALSO
 - risultato \neq 0 VERO

254

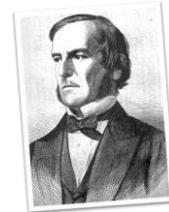
La logica degli elaboratori elettronici



255

La logica Booleana

- Nel 1847 George Boole introdusse un nuovo tipo di logica formale, basata esclusivamente su enunciati di cui fosse possibile verificare in modo inequivocabile la verità o la falsità.



256

Variabili Booleane

- Variabili in grado di assumere solo due valori:
 - VERO
 - FALSO
- In ogni problema è importante distinguere le variabili indipendenti da quelle dipendenti.

257

Operatori Booleani

- Operatori unari (es. Not)
 $op : B \rightarrow B$
- Operatori binari (es. And)
 $op : B^2 \rightarrow B$
- Descritti tramite una tavola della verità (per N operandi, la tabella ha 2^N righe che elencano tutte le possibili combinazioni di valori delle variabili indipendenti ed il valore assunto dalla variabile dipendente)

258

Tavola della verità (truth table)

A	B	A op B
falso	falso	falso
falso	vero	falso
vero	falso	falso
vero	vero	vero

259

Espressioni Booleane

• Un' espressione Booleana è una combinazione di variabili ed operatori Booleani.

• Ad esempio:

$$A \text{ e } (\text{ non } B)$$

260

Funzioni Booleane

• Una funzione Booleana è un' applicazione multi-a-uno:

$$f: B^N \rightarrow B$$

• Ad esempio:

$$f(A, B) = A \text{ e } (\text{ non } B)$$

261

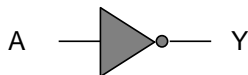
Operatore NOT

A	\bar{A}
falso	vero
vero	falso

Nota: per comodità grafica talvolta la negazione è indicata con un apice dopo la variabile o l' espressione negata (es. A')

262

La porta INV / NOT



$$Y = A'$$

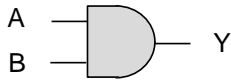
263

Operatore AND

A	B	$A \times B$
falso	falso	falso
falso	vero	falso
vero	falso	falso
vero	vero	vero

264

La porta AND



$$Y = A \times B$$

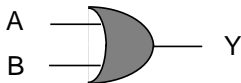
265

Operatore OR

A	B	A + B
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	vero

266

La porta OR



$$Y = A + B$$

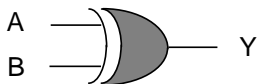
267

Operatore XOR

A	B	A Å B
falso	falso	falso
falso	vero	vero
vero	falso	vero
vero	vero	falso

268

La porta XOR



$$Y = A \oplus B = A \times B' + A' \times B$$

269

Proprietà commutativa e associativa

$$A \times B = B \times A$$

$$A + B = B + A$$

$$A \times B \times C = (A \times B) \times C = A \times (B \times C) = (A \times C) \times B$$

$$A + B + C = (A + B) + C = A + (B + C) = (A + C) + B$$

270

Proprietà distributiva

$$A \times (B + C) = A \times B + A \times C$$

$$A + (B \times C) = (A + B) \times (A + C)$$

271

Teorema di De Morgan

- Teorema:

$$f'(a, b, \dots, z; +, \times) = f(a', b', \dots, z'; \times, +)$$

- ovvero (negando entrambi i membri):

$$f'(a, b, \dots, z; +, \times) = f(a', b', \dots, z'; \times, +)$$

- Ad esempio:

$$- A + B = (A' \times B')$$

$$- (A + B)' = A' \times B'$$



Dimostrazioni in algebra Booleana

- Siccome l'algebra Booleana contempla solo due valori è possibile effettuare le dimostrazioni (di proprietà o teoremi) considerando esaustivamente tutti i casi possibili:

- 2 variabili → 4 combinazioni
- 3 variabili → 8 combinazioni
- 4 variabili → 16 combinazioni
- ecc.

273

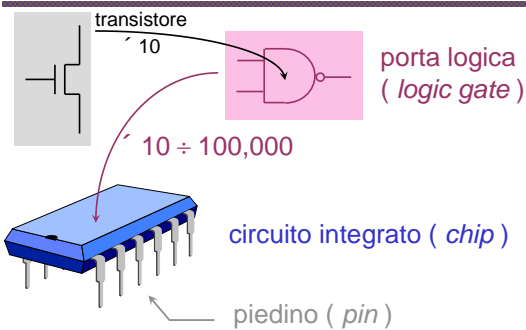
Dimostrazioni: un esempio

$$A + (B \times C) = (A + B) \times (A + C) ?$$

A	B	C	$A + B \times C$	$(A + B) \times (A + C)$
0	0	0	$0 + 0 \times 0 = 0$	$(0+0) \times (0+0) = 0$
0	0	1	$0 + 0 \times 1 = 0$	$(0+0) \times (0+1) = 0$
0	1	0	$0 + 1 \times 0 = 0$	$(0+1) \times (0+0) = 0$
0	1	1	$0 + 1 \times 1 = 1$	$(0+1) \times (0+1) = 1$
1	0	0	$1 + 0 \times 0 = 1$	$(1+0) \times (1+0) = 1$
1	0	1	$1 + 0 \times 1 = 1$	$(1+0) \times (1+1) = 1$
1	1	0	$1 + 1 \times 0 = 1$	$(1+1) \times (1+0) = 1$
1	1	1	$1 + 1 \times 1 = 1$	$(1+1) \times (1+1) = 1$

274

Dal transistor al chip



275

Dal problema al circuito

- Dato un problema per ottenere il circuito corrispondente si applicano i seguenti passi:

1. Individuare le variabili booleane
2. Creare la tabella di verità
3. Generare la funzione F a partire dalla tabella di verità
4. Progettare il circuito usando le porte logiche coerentemente con F

276

Memoria

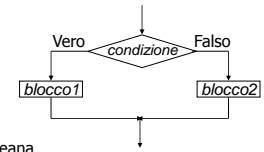
- E' importante non solo fare calcoli, ma anche memorizzare dati (es. i risultati parziali di una lunga sequenza di operazioni).
- A questo fine si usa un elemento logico speciale: il flip-flop.
 - elemento base dei circuiti con memoria
 - memorizza un bit

277

Istruzione if

- Sintassi:

```
if (<condizione>)
    <blocco1>
[else
    <blocco2>]
```



<condizione>: Espressione booleana

<blocco1>: Sequenza di istruzioni

- Se la sequenza contiene più di una istruzione, è necessario racchiuderle tra parentesi graffe

- Significato:

- Se è vera <condizione>, esegui le istruzioni di <blocco1>, altrimenti esegui quelle di <blocco2>

278

Istruzione if : Esempio

- Leggere due valori A e B, calcolarne la differenza in valore assoluto D = |A-B| e stamparne il risultato

```
main()
{
    int A,B,D;

    scanf("%d %d", &A, &B);
    if (A > B)
        D = A-B;
    else
        D = B-A;
    printf("%d\n", D);
}
```

279

Operatori logici

- Operano su espressioni "booleane" e forniscono un risultato "booleano":

! && ||
NOT AND OR

- Equivalenti agli operatori booleani di base

- Stesse regole di precedenza

- NOT > AND > OR

- Esempi:

- (x>0) && (x<10) (x compreso tra 0 e 10)
- (x1>x2) || (x1 == 3)

- Le espressioni "logiche" sono valutate da sinistra a destra

- La valutazione viene interrotta non appena il risultato è univocamente determinato

280

Operatori logici (Cont.)

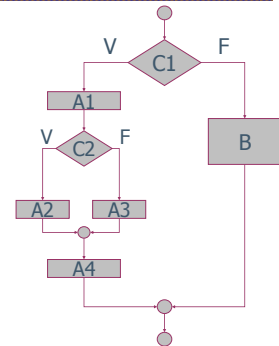
Operatore booleano	Sintassi in C	Esempio
AND	&&	(x>=a)&&(x<=b)
OR		(v1>=18) (v2>=18)
NOT	!	!(a>b)

281

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



282

Settimana n.4

Obiettivi

- Concetto di ciclo
- Cicli semplici
- Cicli annidati

Contenuti

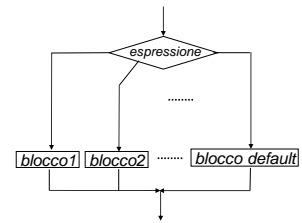
- Costrutto switch
- Cast e sizeof
- Costrutto while
- Ciclo for
- Ciclo Do-while
- Istruzioni break e continue
- Concetto di ciclo annidato ed esempio
- Problem solving su dati scalari

283

Istruzione switch

• Sintassi:

```
switch (<espressione>
{
  case <costante1>:
    <blocco1>
    break;
  case <costante2>:
    <blocco2>
    break;
  ...
  default:
    <blocco default>
}
```



< *espressione* >: Espressione a valore numerico

< *blocco1* >, < *blocco2* >, ... : Sequenza di istruzioni (no parentesi graffe!)

284

Istruzione switch (Cont.)

• Significato:

- In base al valore di < *espressione* >, esegui le istruzioni del *case* corrispondenti
- Nel caso nessun *case* venga intercettato, esegui le istruzioni corrispondenti al *case default*

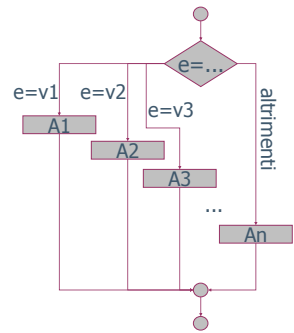
• NOTE:

- I vari *case* devono rappresentare condizioni mutualmente ESCLUSIVE!
- I vari *case* vengono eseguiti in sequenza
 - Per evitare questo, si usa l'istruzione *break* all'interno di un blocco

285

Istruzione switch (Cont.)

```
switch ( e )
{
  case v1:
    A1 ;
    break ;
  case v2:
    A2 ;
    break ;
  case v3:
    A3 ;
    break ;
  .....
  default:
    An ;
}
```



286

Istruzione switch: Esempio

```
int x;
...
switch (x) {
  case 1:
    printf("Sono nel caso 1\n");
    break;
  case 2:
    printf("Sono nel caso 2\n");
    break;
  default:
    printf("Né caso 1 né caso 2\n");
    break;
}
```

287

Operatori di incremento

- Per le assegnazioni composte più comuni sono previsti degli operatori espliciti:

++ --

- Casi particolari degli operatori composti dei precedenti

• Significato:

- Operatore ++ -> +=1
- Operatore -- -> -=1

• Esempi:

- x++;
- valore--;

288

Operatori di incremento (Cont.)

- Possono essere utilizzati sia in notazione *prefissa* che in notazione *postfissa*
- Prefissa: La variabile viene modificata prima di essere utilizzata in un'espressione
- Postfissa: La variabile viene modificata dopo averla utilizzata in un'espressione
- Esempio: Assumendo $x=4$:
 - Se si esegue $y=x++$, si otterrà come risultato $x=5$ e $y=4$;
 - Se si esegue $y=++x$, si otterrà come risultato $x=5$ e $y=5$;

289

Rango delle espressioni aritmetiche

- In C, è possibile lavorare con operandi non dello stesso tipo
- Le operazioni aritmetiche avvengono dopo aver promosso tutti gli operandi al tipo di rango più alto:

```
_Bool
char
short
unsigned short
int
unsigned int
long
unsigned long
long long
unsigned long long
float
double
long double
```

290

Operatori di cast

- In alcuni casi, può essere necessario convertire esplicitamente un'espressione in uno specifico tipo
 - Quando le regole di conversione automatica non si applicano
 - Esempio: `int i; double d;`
l'assegnazione `i = d;` fa perdere informazione
- Sintassi:
'(<tipo>)' <espressione>;
 - Significato: Forza <espressione> ad essere interpretata come se fosse di tipo <tipo>
- Esempio:

```
...
double f;
f = (double) 10;
```

291

Operatori di cast: Esempio

```
#include <stdio.h>

main()
{
    int a, b;

    printf("Dammi un numero intero (A): ");
    scanf("%d", &a);
    printf("Dammi un numero intero (B): ");
    scanf("%d", &b);
    if (b==0)
        printf("Errore: divisione per zero!!\n");
    else
        printf("A / B = %f\n", ((float)a)/b);
}
```

292

Operatore sizeof()

- E' possibile calcolare il numero di byte utilizzato dai tipi di dato di base utilizzando l'operatore `sizeof`
- Sintassi:
`sizeof(<tipo>)`
- Ritorna il numero di byte occupato da <tipo>
- Esempio:

```
unsigned int size;
size = sizeof(float); /* size = 4 */
```
- L'uso dell'operatore `sizeof()` può essere esteso al calcolo dello spazio occupato da espressioni, vettori e strutture

293

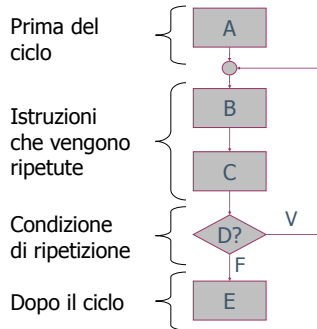
Operatore sizeof(): Esempio

```
#include <stdio.h>

main()
{
    printf("tipo          n.byte\n");
    printf("-----\n");
    printf("char          %d\n", sizeof(char));
    printf("int           %d\n", sizeof(int));
    printf("long          %d\n", sizeof(long));
    printf("long long     %d\n", sizeof(long long));
    printf("float         %d\n", sizeof(float));
    printf("double        %d\n", sizeof(double));
    printf("long double   %d\n", sizeof(long double));
}
```

294

Flusso di esecuzione ciclico



295

Istruzione while

Sintassi:

```
while (<condizione>)  
<blocco>
```

<condizione>: Una condizione Booleana

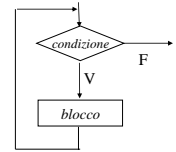
<blocco>: Sequenza di istruzioni

- Se più di una istruzione, va racchiuso tra graffe

Realizza la struttura di tipo while

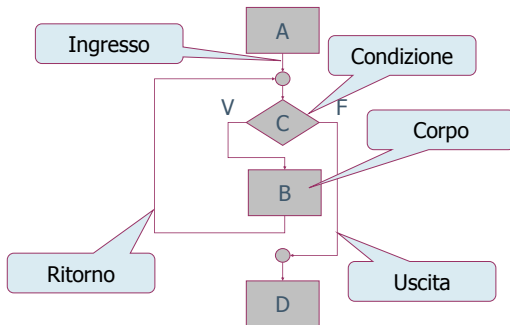
Significato:

- Ripeti <blocco> finché <condizione> è vera



296

Notazione grafica (while)



297

Istruzione while: Esempio

- Leggere un valore N, calcolare la somma S dei primi N numeri interi e stamparla

```
#include <stdio.h>  
main() {  
    int N, i, S;  
    i = 1; S = 0; /* inizializzazioni */  
    scanf ("%d", &N);  
    while (i <= N) {  
        S = S+i; /* operazione iterativa */  
        i++; /* aggiornamento condizione */  
    }  
    printf ("Somma = %d\n", S); /* output */  
}
```

298

Anatomia di un ciclo

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

299

Istruzione for

Sintassi:

```
for (<inizializzazioni>; <condizione>; <incremento>)  
<blocco>
```

<inizializzazioni>: Le condizioni iniziali prima del ciclo

<condizione>: Una condizione booleana

<incremento>: Incremento della variabile di conteggio

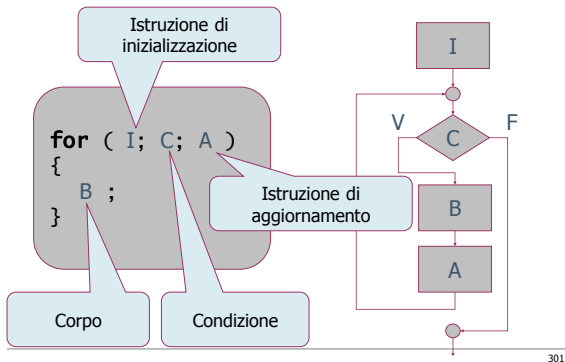
<blocco>: Sequenza di istruzioni

- Se contiene più di una istruzione, va racchiuso tra graffe

- Tutti i campi possono essere vuoti!

300

Istruzione for (Cont.)



301

Istruzione for (Cont.)

- Significato:
 - Equivalente a:

```
<inizializzazioni>
while (<condizione>) {
  <blocco>
  <incremento>
}
```

- Realizza un ciclo basato su conteggio
 - Tipicamente contiene una variabile indice che serve da iteratore:
 - Parte da un valore iniziale (**inizializzazione**)
 - Arriva ad un valore finale (**condizione**)
 - Attraverso uno specifico incremento (**incremento**)

302

Istruzione for (Cont.)

- Esempio:
 - Leggere un carattere `ch` ed un intero `N`, e stampare una riga di `N` caratteri `ch`
 - Esempio: `N=10, ch='*'` output = `*****`
 - Formulazione iterativa:
 - Ripeti `N` volte l'operazione "stampa `ch`"
 - Soluzione:

```
#include <stdio.h>
main() {
  int N, i;
  char ch;

  scanf("%d %c", &N, &ch);
  for (i=0; i<N; i++)
    printf("%c", ch); /*senza '\n' !!!!*/
  printf("\n");
}
```

303

Esercizio

- Introdurre da tastiera 100 numeri interi, e *calcolarne la media*. Si controlli che ogni numero inserito sia compreso tra 0 e 30; in caso contrario, il *numero deve essere ignorato*
- Analisi:
 - Problema iterativo
 - Media=?
 - Controllo del valore inserito

304

Esercizio: Soluzione

```
#include <stdio.h>
main() {
  int valore, i, Totale=0, M=0;
  const int N = 100;
  for (i=0; i<N; i++) /* per ogni valore introdotto */
  {
    scanf("%d", &valore);
    if (valore < 0 || valore > 30) /* controllo validità */
      printf("Valore non valido");
    else
    { /* caso normale */
      Totale += valore; /* accumula nuovo valore in Totale */
      M++; /* ho letto un dato in più */
    }
  }
  printf("La media è: %f\n", (float)Totale/M);
}
```

305

for e while

- Il ciclo `for` può essere considerato un caso particolare del ciclo `while`
- In generale si usa:
 - `for` per cicli di conteggio
 - Numero di iterazioni note a priori
 - Condizione di fine ciclo tipo "conteggio"
 - `while` per cicli "generali"
 - Numero di iterazioni non note a priori
 - Condizione di fine ciclo tipo "evento"

306

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)
{
    .....
}
```

307

Cicli annidati

- Alcuni problemi presentano una struttura "bidimensionale"
 - L'operazione iterativa stessa può essere espressa come un'altra iterazione

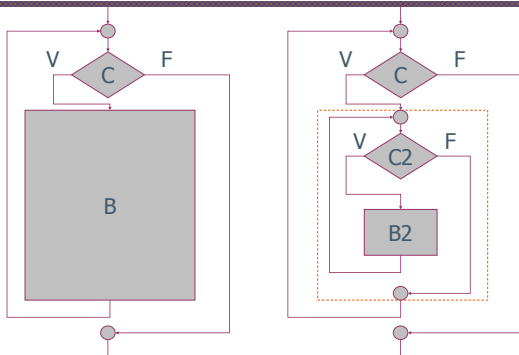
- Realizzazione: Un ciclo che contiene un altro ciclo

• Struttura:

```
for ( ... )
{
    for ( ... )
    {
        ...
    }
}
```

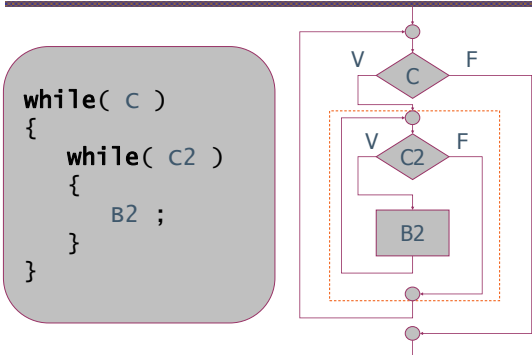
308

Cicli while annidati



309

Cicli while annidati



310

Esempio

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

311

Istruzione do

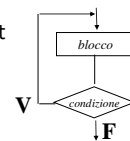
- Sintassi:


```
do
    <blocco>
while ( <condizione> );
```

 - <condizione>: Una condizione booleana
 - <blocco>: Sequenza di istruzioni
 - Se più di una istruzione, va racchiuso tra graffe

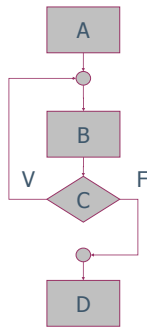
- Realizza la struttura di tipo repeat

- Significato:
 - Ripeti <blocco> finché <condizione> è vera



312

Istruzione do (Cont.)



313

Istruzione do (Cont.)

• Esempio:

- Leggere un valore N controllando che il valore sia positivo. In caso contrario, ripetere la lettura

```
#include <stdio.h>
main() {
    int n;
    do
        scanf ("%d", &n);
    while (n <= 0);
}
```

314

Istruzione do (Cont.)

- È sempre possibile trasformare un ciclo di tipo do in un ciclo di tipo while semplice, anticipando e/o duplicando una parte delle istruzioni

• Esempio:

```
#include <stdio.h>
main() {
    int n;
    scanf ("%d", &n);
    while (n <= 0)
        scanf ("%d", &n);
}
```

315

Interruzione dei cicli

- Il linguaggio C mette a disposizione due istruzioni per modificare il normale flusso di esecuzione di un ciclo:

- break:

- Termina il ciclo
- L'esecuzione continua dalla prima istruzione dopo la fine del ciclo

- continue:

- Termina l'iterazione corrente
- L'esecuzione continua con la prossima iterazione del ciclo

316

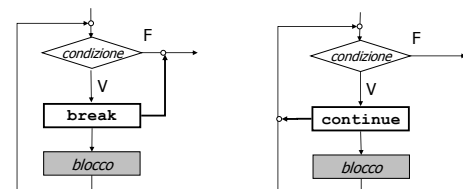
Interruzione dei cicli (Cont.)

- Trasformano i cicli in blocchi non strutturati
 - Usare con cautela (e possibilmente non usare...)
 - **Si può sempre evitare l'uso di break/continue!**
- Usabili in ogni tipo di ciclo (while, for, do)

317

break e continue

- In termini di diagrammi di flusso (esempio: ciclo while):



318

break : Esempio

- Acquisire una sequenza di numeri interi da tastiera; terminare l'operazione quando si legge il valore 0.

- Versione con break

```
int valore;
while (scanf("%d", &valore))
{
    if (valore == 0)
    {
        printf("Valore non consentito\n");
        break; /* esce dal ciclo */
    }
    /* resto delle istruzioni del ciclo */
}
```

319

break : Esempio (Cont.)

- Versione senza break (strutturata)

```
int valore, finito = 0;
while (scanf("%d", &valore) && !finito)
{
    if (valore == 0)
    {
        printf("Valore non consentito\n");
        finito = 1;
    }
    else
    {
        /* resto delle istruzioni del ciclo */
    }
}
```

320

continue : Esempio

- Acquisire una sequenza di numeri interi da tastiera; ignorare i numeri pari al valore 0.

- Versione con continue

```
int valore;
while (scanf("%d", &valore))
{
    if (valore == 0)
    {
        printf("Valore non consentito\n");
        continue; /* va a leggere un nuovo valore */
    }
    /* resto delle istruzioni del ciclo */
}
```

321

continue : Esempio (Cont.)

- Versione senza continue (strutturata)

```
int valore;
while (scanf("%d", &valore))
{
    if (valore == 0)
    {
        printf("Valore non consentito\n");
    }
    else {
        /* resto delle istruzioni del ciclo */
    }
}
```

322

Settimana n.5

Obiettivi

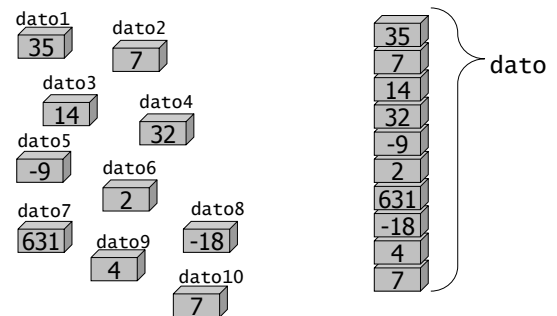
- Vettori

Contenuti

- Definizione di vettori
- Dimensionamento statico dei vettori
- Operazioni elementari: lettura, stampa, copia, confronto di vettori

323

Variabili e vettori




324

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) ;
}
```

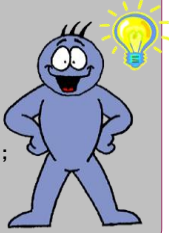


325

...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]) ;
}
```



326

Vettori

- Insiemi di variabili *dello stesso tipo* aggregate in un'unica entità
 - Identificate globalmente da un nome
 - Singole variabili (*elementi*) individuate da un *indice*, corrispondente alla loro posizione rispetto al primo elemento
 - L'indice degli elementi parte da 0
 - Gli elementi di un vettore sono memorizzati in celle di memoria contigue!



327

Dichiarazione di un vettore

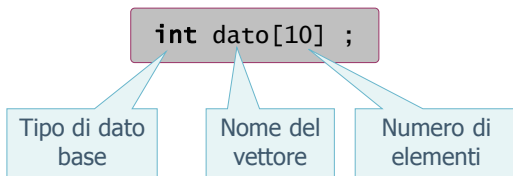
- Sintassi:
`<tipo> <nome vettore> [<dimensione>];`
- Accesso ad un elemento:
`<nome vettore> [<posizione>]`
- Esempio:

```
int v[10];
```

 - Definisce un insieme di 10 variabili intere
`v[0], v[1], v[2], v[3], v[4], v[5], v[6], v[7], v[8], v[9]`

328

Dichiarazione di un vettore (Cont.)



329

Inizializzazione di un vettore

- E' possibile assegnare un valore iniziale ad un vettore (solo) al momento della sua dichiarazione
- Equivalente ad assegnare OGNI elemento del vettore
- Sintassi (vettore di N elementi):
`{<valore 0>, <valore 1>, ..., <valore N-1>};`
- Esempio:

```
int lista[4] = {2, 0, -1, 5};
```
- NOTA: Se vengono specificati meno di N elementi, l'inizializzazione assegna a partire dal primo valore. I successivi vengono posti a zero.
 - Esempio:

```
int s[4] = {2, 0, -1};
/* s[0]=2, s[1]=0, s[2]=-1, s[3]=0 */
```

330

Vettori e indici

- L'indice che definisce la posizione di un elemento di un vettore DEVE essere intero!

- Non necessariamente costante!
 - Può essere un'espressione complessa (purché intera)

- Esempi:

```
double a[100]; /* a vettore di double */
double x;
int i, j, k;
... ..
x = a[2*i+j-k]; /* è corretto! */
```

331

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])` ;

332

Vettori e cicli

- I cicli sono particolarmente utili per "scandire" un vettore

- Utilizzo tipico: Applicazione iterativa di un'operazione sugli elementi di un vettore

- Schema:

```
...
int data[10];
for (i=0; i<10; i++)
{
    // operazione su data[i]
}
...
```

333

Direttiva #define

- Sintassi:

```
#define <costante> <valore>
```

<costante>: Identificatore della costante simbolica

- Convenzionalmente indicato tutto in maiuscolo

<valore>: Un valore da assegnare alla costante

- Utilizzo:

- Definizione di costanti simboliche
- Maggiore leggibilità
- Maggiore flessibilità
 - Il cambiamento del valore della costante si applica a tutto il file!

334

Direttiva #define (Cont.)

- Esempio:

```
- #define PI 3.1415
- #define N 80
- ...
- double z = PI * x;
- int vect[N];
```

335

Direttiva #define (Cont.)

```
#define N 10

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

Definizione della costante

Uso della costante

336

Modificatore const

```
int main(void)
{
    const int N = 10 ;
    int dato[N] ;
    . . .
}
```

Definizione della costante

Uso della costante

337

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 ;
```

338

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]) ;
}
```

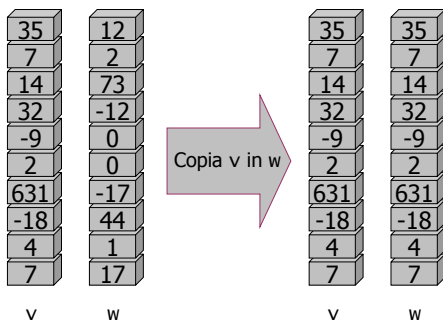
339

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]) ;
}
```

340

Copia di un vettore



341

Copia di un vettore

```
/* copia il contenuto di v[] in w[] */
for( i=0; i<N; i++ )
{
    w[i] = v[i] ;
}
```

342

Esercizio 1

- Leggere 10 valori interi da tastiera, memorizzarli in un vettore e calcolarne il minimo ed il massimo

• Analisi:

- Il calcolo del minimo e del massimo richiedono la scansione dell'intero vettore
- Il generico elemento viene confrontato con il minimo corrente ed il massimo corrente
 - Se minore del minimo, aggiorni il minimo
 - Se maggiore del massimo, aggiorni il massimo
- Importante l'inizializzazione del minimo/massimo corrente!

343

Esercizio 1: Soluzione

```
#include <stdio.h>
main()
{
    int v[10];
    int i, max, min;

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

    /* uso il primo elemento per inizializzare min e max*/
    max = v[0];
    min = v[0];

    for (i=1; i<10; i++) {
        if (v[i] > max)
            max = v[i];
        if (v[i] < min)
            min = v[i];
    }
    printf("Il massimo e': %3d\n", max);
    printf("Il minimo e' : %3d\n", min);
}
```

344

Esercizio 2

- Scrivere un programma che legga un valore decimale minore di 1000 e lo converta nella corrispondente codifica binaria

• Analisi:

- Usiamo l'algoritmo di conversione binaria visto a lezione
 - Divisioni successive per 2
 - Si memorizzano i resti nella posizione del vettore di peso corrispondente
 - La cifra meno significativa è l'ultima posizione del vettore!
- Essenziale determinare la dimensione massima del vettore
 - Per codificare un numero < 1000 servono 10 bit ($2^{10}=1024$)

345

Esercizio 2: Soluzione

```
#include <stdio.h>
main()
{
    int v[10] = {0};
    int i=9; /* ultima posizione del vettore */
    unsigned N; /* unsigned perchè positivo */

    printf("Inserire un numero positivo (<1000): ");
    scanf("%d", &N);

    if (N > 1000)
        printf("Errore: il numero deve essere < 1000\n");
    else {
        while(N != 0) {
            v[i] = (N % 2); /* resto della divisione per 2! */
            N = N/2; /* divido N per 2 */
            i--;
        }

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

346

Settimana n.6

Obiettivi

- Ricerche in Vettori
- Flag
- Funzioni

Contenuti

- Ricerca di esistenza
- Ricerca di universalità
- Ricerca di duplicati
- Problem solving su dati vettoriali
- Definizione di Funzioni
- Passaggio di parametri e valore di ritorno

347

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"

348

Ricerca di un elemento: Esempio (1/3)

```
int dato ; /* dato da ricercare */
int trovato ; /* flag per ricerca */
int pos ; /* posizione elemento */
...
printf("Elemento da ricercare? ");
scanf("%d", &dato) ;
```

349

Ricerca di un elemento: Esempio (2/3)

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

350

Ricerca di un elemento: Esempio (3/3)

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

351

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
 - ...

352

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

353

Ricerca del massimo: Esempio (1/2)

```
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 ;
    }
}
```

354

Ricerca del massimo: Esempio (2/2)

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

355

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?

356

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

357

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?
 - $\exists x : \text{not } P(x)$

358

Realizzazione (1/2)

- | | |
|---|---|
| <ul style="list-style-type: none">• Esistenza: $\exists x : P(x)$<ul style="list-style-type: none">- Inizializzo flag $F = 0$- Ciclo su tutte le x<ul style="list-style-type: none">• Se $P(x)$ è vera<ul style="list-style-type: none">- Pongo $F = 1$- Se $F = 1$, l'esistenza è dimostrata- Se $F = 0$, l'esistenza è negata | <ul style="list-style-type: none">• Universalità: $\forall x : P(x)$<ul style="list-style-type: none">- Inizializzo flag $F = 1$- Ciclo su tutte le x<ul style="list-style-type: none">• Se $P(x)$ è falsa<ul style="list-style-type: none">- Pongo $F = 0$- Se $F = 1$, l'universalità è dimostrata- Se $F = 0$, l'universalità è negata |
|---|---|

359

Realizzazione (2/2)

- | | |
|--|--|
| <ul style="list-style-type: none">• Esistenza: $\exists x : \text{not } P(x)$<ul style="list-style-type: none">- Inizializzo flag $F = 0$- Ciclo su tutte le x<ul style="list-style-type: none">• Se $P(x)$ è falsa<ul style="list-style-type: none">- Pongo $F = 1$- Se $F = 1$, l'esistenza è dimostrata- Se $F = 0$, l'esistenza è negata | <ul style="list-style-type: none">• Universalità: $\forall x : \text{not } P(x)$<ul style="list-style-type: none">- Inizializzo flag $F = 1$- Ciclo su tutte le x<ul style="list-style-type: none">• Se $P(x)$ è vera<ul style="list-style-type: none">- Pongo $F = 0$- Se $F = 1$, l'universalità è dimostrata- Se $F = 0$, l'universalità è negata |
|--|--|

360

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");
```

361

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 ;
}
```

362

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 ;
}
```

363

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 ;
}
```

364

Ricerca di duplicati

- Un vettore v contiene elementi duplicati?

```
for( i=0 ; i<N ; i++ )
{
    duplicato = 0 ;
    for( j = 0 ; j<N ; j++ )
    {
        if( (v[i]== v[j]) && (i != j) )
        {
            duplicato = 1 ;
        }
    }
    if( duplicato == 1 )
        printf( "v[%d] è duplicato\n", i);
}
```

365

Sottoprogrammi

- Un programma realistico può consistere di migliaia di istruzioni
- Sebbene fattibile, una soluzione "monolitica" del problema:
 - Non è molto produttiva:
 - Riuso del codice?
 - Comprensione del codice?
 - Non è intuitiva:
 - Tendenza ad "organizzare" in modo strutturato
 - Struttura gerarchica a partire dal problema complesso fino a sottoproblemi sempre più semplici
- Approccio *top-down*

366

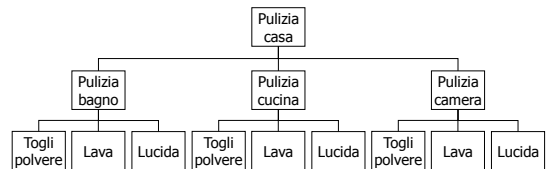
Approccio top-down

- Decomposizione del problema in sottoproblemi più semplici
- Ripetibile su più livelli
- Sottoproblemi "terminali" = Risolvibili in modo "semplice"

367

Approccio top-down (Cont.)

- Esempio: Pulizia di una casa



368

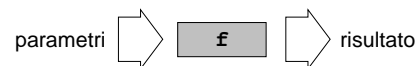
Approccio top-down (Cont.)

- I linguaggi di programmazione permettono di suddividere le operazioni in modo simile tramite **sottoprogrammi**
 - Detti anche **funzioni** o **procedure**
- La gerarchia delle operazioni si traduce in una gerarchia di sottoprogrammi
- `main()` è una funzione!

369

Funzioni e procedure

- Procedure:
 - Sottoprogrammi che NON ritornano un risultato
- Funzioni:
 - Sottoprogrammi che ritornano un risultato (di qualche tipo primitivo o non)
- In generale, procedure e funzioni hanno dei *parametri* (o *argomenti*)
 - Vista funzionale:



370

Funzioni e procedure in C

- Nel C K&R:
 - Esistono solo funzioni (**tutto** ritorna un valore)
 - Si può ignorare il valore ritornato dalle funzioni
- Dal C89 (ANSI) in poi:
 - Funzioni il cui valore di ritorno deve essere ignorato (`void`)
 - Funzioni `void` ↔ procedure

371

Definizione di una funzione

- Stabilisce un "nome" per un insieme di operazioni
- Sintassi:

```
<tipo risultato> <nome funzione> (<parametri formali >)  
{  
  <istruzioni>  
}
```

 - Se la funzione non ha un risultato, **<tipo risultato>** deve essere `void`
 - Per ritornare il controllo alla funzione *chiamante*, nelle **<istruzioni>** deve comparire una istruzione
 - `return <valore>;` **se non** `void`
 - `return;` **se** `void`

372

Definizione di una funzione (Cont.)

- Tutte le funzioni sono definite allo stesso livello del `main()`
 - NON si può definire una funzione dentro un'altra
- `main()` è una funzione!
 - Tipo del valore di ritorno: `int`
 - Parametri: Vedremo più avanti!

373

Prototipi

- Così come per le variabili, è buona pratica dichiarare all'inizio del programma le funzioni prima del loro uso (**prototipi**)
- Sintassi:
 - Come per la definizione, ma si omette il contenuto (istruzioni) della funzione

374

Prototipi: Esempio

```
#include <stdio.h>

int func1(int a);
int func2(float b);
...

main ()
{
  ...
}

int func1(int a)
{
  ...
}

int func2(float b)
{
  ...
}
```

375

Funzioni e parametri

- Parametri e risultato sono sempre associati ad un tipo
- Esempio:
`float media(int a, int b)`

- I tipi di parametri e risultato devono essere rispettati quando la funzione viene utilizzata!
- Esempio:
`float x; int a,b;`
`x = media(a, b);`

376

Utilizzo di una funzione

- Deve rispettare l'interfaccia della definizione
- Utilizzata come una normale istruzione
`<variabile> = <nome funzione> (<parametri attuali>);`
- Può essere usata ovunque
 - Una funzione può anche invocare se stessa (funzione ricorsiva)

377

Utilizzo di una funzione: Esempio

```
#include <stdio.h>

int modabs(int v1, int v2); //prototipo

main() {
  int x,y,d;
  scanf("%d %d",&x,&y);
  d = modabs(x,y); // utilizzo
  printf("%d\n",d);
}

int modabs (int v1, int v2) // definizione
{
  int v;
  if (v1>v2) {
    v = v1-v2;
  } else {
    v = v2-v1;
  }
  return v;
}
```

378

Parametri formali e attuali

- E' importante distinguere tra:
 - Parametri formali:
Specificati nella definizione di una funzione
 - Parametri attuali:
Specificati durante il suo utilizzo

- Esempio:

- funzione Func
 - Definizione: `double Func(int x, double y)`
 - Parametri formali: `(x, y)`
 - Utilizzo: `double z = Func(a*2, 1.34);`
 - Parametri attuali: (Risultato di `a*2`, `1.34`)

379

Parametri formali e attuali (Cont.)

- Vista funzionale:

- Definizione:



- Utilizzo:



380

Passaggio dei parametri

- In C, il passaggio dei parametri avviene *per valore*
 - Significato: Il valore dei parametri attuali viene copiato in variabili locali della funzione
- Implicazione:
 - I parametri attuali non vengono **MAI** modificati dalle istruzioni della funzione

381

Passaggio dei parametri: Esempio

```
#include <stdio.h>

void swap(int a, int b);

main() {
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d %d\n", x, y);
    swap(x, y);
    /* x e y NON VENGONO MODIFICATI */

    printf("%d %d\n", x, y);
}

void swap(int a, int b)
{
    int tmp;
    tmp = a;
    a = b;
    b = tmp;
    return;
}
```

382

Passaggio dei parametri (Cont.)

- E' possibile modificare lo schema di passaggio per valore in modo che i parametri attuali vengano modificati dalle istruzioni della funzione
- Passaggio *per indirizzo (by reference)*
 - Parametri attuali = indirizzi di variabili
 - Parametri formali = puntatori al tipo corrispondente dei parametri attuali
 - Concetto:
 - Passando gli indirizzi dei parametri formali posso modificarne il valore
 - La teoria dei puntatori verrà ripresa in dettaglio più avanti
 - Per il momento è sufficiente sapere che:
 - `'&<variabile>` fornisce l'indirizzo di memoria di `<variabile>`
 - `'*<puntatore>` fornisce il dato contenuto nella variabile puntata da `<puntatore>`

383

Passaggio dei parametri: Esempio

```
#include <stdio.h>

void swap(int *a, int *b);

main() {
    int x, y;
    scanf("%d %d", &x, &y);
    printf("%d %d\n", x, y);
    swap[&x, &y];
    /* x e y SONO ORA MODIFICATI */
    printf("%d %d\n", x, y);
}

void swap(int *a, int *b)
{
    int tmp;
    tmp = *a;
    *a = *b;
    *b = tmp;
    return;
}
```

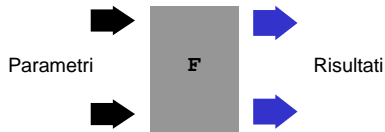
Passo l'indirizzo di x e y

Uso *a e *b come "interi"

384

Passaggio dei parametri (Cont.)

- Il passaggio dei parametri per *indirizzo* è indispensabile quando la funzione deve ritornare più di un risultato



385

Vettori e funzioni

- Le funzioni possono avere come parametri dei vettori o matrici:
 - Parametri formali
 - Si indica il nome del vettore, con "[]" senza dimensione
 - Parametri attuali
 - Il nome del vettore SENZA "[]"
- Il nome del vettore indica l'indirizzo del primo elemento, quindi il vettore è passato per indirizzo!

386

Vettori e funzioni (Cont.)

- Conseguenza:
 - Gli elementi di un vettore passato come argomento vengono SEMPRE modificati!
- **ATTENZIONE:** Dato che il vettore è passato per indirizzo, la funzione che riceve il vettore come argomento **non ne conosce la lunghezza!!!!**
- Occorre quindi passare alla funzione anche la dimensione del vettore!

387

Esercizio

- Scrivere una funzione `nonnull()` che ritorni il numero di elementi non nulli di un vettore di interi passato come parametro

- Soluzione:

```
int nonnull(int v[], int dim)
{
    int i, n=0;
    for (i=0; i<dim; i++) {
        if (v[i] != 0)
            n++;
    }
    return n;
}
```

All'interno della funzione bisogna sapere la dimensione del vettore

Se `v[]` fosse modificato dentro la funzione, il valore sarebbe modificato anche nella funzione chiamante

388

Settimana n.7

Obiettivi

- Caratteri
- Vettori di caratteri
- Stringhe

Contenuti

- Funzioni `<math.h>`
- Il tipo `char`
- Input/output di caratteri
- Operazioni su variabili `char`
- Funzioni `<ctype.h>`
- Stringhe come vettori di `char`
- Il terminatore nullo
- Stringhe come tipo gestito dalla libreria
- Funzioni di I/O sulle stringhe

389

Funzioni matematiche

- Utilizzabili includendo in testa al programma

```
#include <math.h>
```

- **NOTA:** Le funzioni trigonometriche (sia dirette sia inverse) operano su angoli espressi in radianti

390

math.h

funzione	definizione
double sin (double x)	sin (x)
double cos (double x)	cos (x)
double tan (double x)	tan (x)
double asin (double x)	asin (x)
double acos (double x)	acos (x)
double atan (double x)	atan (x)
double atan2 (double y, double x)	atan (y / x)
double sinh (double x)	sinh (x)
double cosh (double x)	cosh (x)
double tanh (double x)	tanh (x)

391

math.h (Cont.)

funzione	definizione
double pow (double x, double y)	x^y
double sqrt (double x)	radice quadrata
double log (double x)	logaritmo naturale
double log10 (double x)	logaritmo decimale
double exp (double x)	e^x

392

math.h (Cont.)

funzione	definizione
double ceil (double x)	ceil (x)
double floor (double x)	floor (x)
double fabs (double x)	valore assoluto
double fmod (double x, double y)	modulo
double modf (double x, double *ipart)	restituisce la parte frazionaria di x e memorizza la parte intera di x in ipart

393

Funzioni matematiche: Esempio

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

double log2(double x);

main()
{
    int nogg, nbit;

    printf("Dammi il numero di oggetti: ");
    scanf("%d", &nogg);
    nbit=ceil(log2((double)nogg));
    printf("Per rappresentare %d oggetti servono %d
        bit\n", nogg, nbit);
}

double log2(double x)
{
    return log(x)/log((double)2);
}
```

394

Codice ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	
0	000	NUL	(null)	32	040	#32	Space	64	40	100	#64	0
1	001	SOH	(start of heading)	33	041	#33		65	41	101	#65	A
2	002	STX	(start of text)	34	042	#34		66	42	102	#66	B
3	003	ETX	(end of text)	35	043	#35	#	67	43	103	#67	C
4	004	EOF	(end of transmission)	36	044	#36	!	68	44	104	#68	D
5	005	ENQ	(enquiry)	37	045	#37	%	69	45	105	#69	E
6	006	ACK	(acknowledge)	38	046	#38	@	70	46	106	#70	F
7	007	BEI	(bell)	39	047	#39	*	71	47	107	#71	G
8	010	BS	(backspace)	40	050	#40	(72	48	110	#72	H
9	011	TAB	(horizontal tab)	41	051	#41)	73	49	111	#73	I
10	A 012	LF	(NL line feed, new line)	42	052	#42	*	74	4A	112	#74	J
11	B 013	VT	(vertical tab)	43	053	#43	+	75	4B	113	#75	K
12	C 014	FF	(DP form feed, new page)	44	054	#44	=	76	4C	114	#76	L
13	D 015	CR	(carriage return)	45	055	#45	-	77	4D	115	#77	M
14	E 016	SO	(shift out)	46	056	#46	.	78	4E	116	#78	N
15	F 017	SI	(shift in)	47	057	#47	/	79	4F	117	#79	O
16	10 020	DLX	(data link escape)	48	060	#48	0	80	50	120	#80	P
17	11 021	DC1	(device control 1)	49	061	#49	1	81	51	121	#81	Q
18	12 022	DC2	(device control 2)	50	062	#50	2	82	52	122	#82	R
19	13 023	DC3	(device control 3)	51	063	#51	3	83	53	123	#83	S
20	14 024	DC4	(device control 4)	52	064	#52	4	84	54	124	#84	T
21	15 025	NAK	(negative acknowledge)	53	065	#53	5	85	55	125	#85	U
22	16 026	SYN	(synchronous idle)	54	066	#54	6	86	56	126	#86	V
23	17 027	ETB	(end of trans. block)	55	067	#55	7	87	57	127	#87	W
24	18 030	CAN	(cancel)	56	030	#56	8	88	58	130	#88	X
25	19 031	EM	(end of medium)	57	031	#57	9	89	59	131	#89	Y
26	1A 032	SUB	(substitute)	58	032	#58	:	90	5A	132	#90	Z
27	1B 033	ESC	(escape)	59	033	#59	;	91	5B	133	#91	[
28	1C 034	FS	(file separator)	60	034	#60	<	92	5C	134	#92	\
29	1D 035	GS	(group separator)	61	035	#61	=	93	5D	135	#93]
30	1E 036	RS	(record separator)	62	036	#62	>	94	5E	136	#94	^
31	1F 037	US	(unit separator)	63	037	#63	?	95	5F	137	#95	_

395

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	0	6	A	Z	N
70	117	108	118	105	111	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

396

I/O a caratteri

- Acquisizione/stampa di un carattere alla volta
- Istruzioni:
 - int getchar()
 - Legge un carattere da tastiera
 - Il carattere viene fornito come "risultato" di getchar (valore intero)
 - In caso di errore il risultato è la costante EOF (dichiarata in stdio.h)
 - int putchar(<carattere>)
 - Stampa <carattere> su schermo
 - <carattere>: Un dato di tipo char

397

EOF

- EOF = End-of-File
- Rappresenta in realtà un valore fittizio corrispondente alla fine dell' input
- Indica che non ci sono più dati in input
- EOF può essere prodotto in diversi modi:
 - Automaticamente, se si sta leggendo un file
 - Premendo CTRL+'Z' in MS-DOS o VMS
 - Premendo CTRL+'D' in Unix

398

I/O a caratteri: Esempio

```
#include <stdio.h>

main()
{
    int tasto;

    printf("Premi un tasto...\n");
    tasto = getchar();
    if (tasto != EOF) /* errore ? */
    {
        printf("Hai premuto %c\n", tasto);
        printf("Codice ASCII = %d\n", tasto);
    }
}
```

399

scanf/printf e getchar/putchar

- scanf e printf sono "costruite" a partire da getchar/putchar
- scanf/printf sono utili quando è noto il formato (tipo) del dato che viene letto
 - Esempio: Serie di dati con formato fisso
- getchar/putchar sono utili quando non è noto tale formato
 - Esempio: Un testo

400

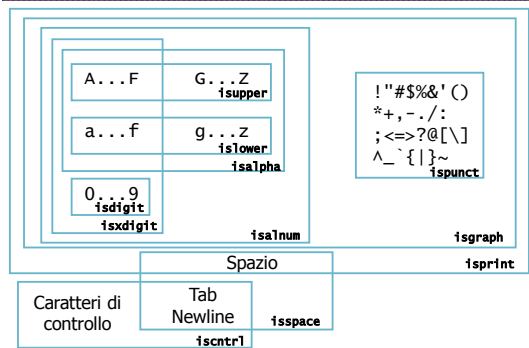
Funzioni di utilità

- Classificazione caratteri (<ctype.h>)

funzione	definizione
int isalnum (char c)	Se c è lettera o cifra
int isalpha (char c)	Se c è lettera
int isascii(char c)	Se c è lettera o cifra
int isdigit (char c)	Se c è una cifra
int islower(char c)	Se c è minuscola
int isupper (char c)	Se c è maiuscola
int isspace(char c)	Se c è spazio,tab,\n
int isctrl(char c)	Se c è di controllo
int isgraph(char c)	Se c è stampabile, non spazio
int isprint(char c)	Se c è stampabile
int ispunct(char c)	Se c è di interpunzione
int toupper (char c)	Converte in maiuscolo
int tolower (char c)	Converte in minuscolo

401

Funzioni di utilità: vista d'insieme



402

Stringhe

- 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
```

403

Stringhe (Cont.)

- Definizione:
Sequenze di caratteri terminate dal carattere '\0' (NULL)
- Tecnicamente:
Vettori di caratteri terminati da un carattere aggiuntivo '\0' (NULL)
- Memorizzate come i vettori
- La lunghezza della stringa può essere definita implicitamente mediante l'assegnazione di una costante stringa, rappresentata da una sequenza di caratteri racchiusa tra doppi apici

Esempio:
char s[] = "ciao!";

```
'c' '!' 'a' 'o' '!' '\0'  
s[0] s[1] s[2] s[3] s[4] s[5]
```

404

Stringhe (Cont.)

- NOTA: La stringa vuota non è un vettore "vuoto"!
- Esempio: char s[] = "";

```
\0'  
s[0]
```
- Attenzione: 'a' è diverso da "a"
- Infatti 'a' indica il carattere a, mentre "a" rappresenta la stringa a (quindi con '\0' finale).
- Graficamente:

```
- "Ciao" ----> '\C' '\!' '\a' '\o' '\!' '\0'  
- "a" ----> '\a' '\0'  
- 'a' ----> '\a'
```

405

Formattazione di stringhe

- Le operazioni di I/O formattato possono essere effettuate anche da/su stringhe
- Funzioni

```
int sscanf(char* <stringa>, char* <formato>, <espressioni>);
```

 - Restituisce EOF in caso di errore, altrimenti il numero di campi letti con successo

```
int sprintf(char* <stringa>, char* <formato>, <variabili>);
```

 - Restituisce il numero di caratteri scritti
- Utili in casi molto particolari per costruire/analizzare stringhe con un formato fisso

406

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

407

Esempio

```
const int MAX = 20 ;  
char nome[MAX+1] ;  
  
printf("Come ti chiami? ") ;  
  
scanf("%s", nome) ;
```

408

Esempio

```
const int MAX = 20 ;
char nome[MAX+1] ;

printf("Come ti chiami? ") ;

gets(nome) ;
```

409

Esempio

```
printf("Buongiorno, ") ;
printf("%s", nome) ;
printf("!\n") ;

printf("Buongiorno, %s!\n", nome) ;
```

410

Esempio

```
printf("Buongiorno, ") ;
puts(nome) ;

/* No!! printf("!\n") ; */
```

411

Settimana n.8

Obiettivi

- Stringhe
- Matrici
- Vettori di Stringhe

Contenuti

- Funzioni <string.h>
- Vettori multidimensionali
- Matrice come estensione dei vettori
- Problem solving su dati testuali

412

Manipolazione di stringhe

- Data la loro natura di tipo "aggregato", le stringhe non possono essere usate come variabili qualunque

- Esempi di operazioni non lecite:

```
char s1[20], s2[10], s3[50];
...
s1 = "abcdefg";
s2 = "hijklmno";
s3 = s1 + s2;
```

↑
NO!
↓

- A questo scopo esistono apposite funzioni per la manipolazione delle stringhe

413

Funzioni di libreria per stringhe

- Utilizzabili includendo in testa al programma
#include <string.h>

funzione	definizione
char* strcat (char* s1, char* s2);	concatenazione s1+s2
char* strchr (char* s, int c);	ricerca di c in s
int strcmp (char* s1, char* s2);	confronto
char* strcpy (char* s1, char* s2);	s1 <= s2
int strlen (char* s);	lunghezza di s
char* strncat (char* s1, char* s2, int n);	concat. n car. max
char* strncpy (char* s1, char* s2, int n);	copia n car. max
char* strncpy (char* dest, char* src, int n);	cfr. n car. max

414

Funzioni di libreria per stringhe (Cont.)

- NOTE:

- Non è possibile usare vettori come valori di ritorno delle funzioni di libreria

- Esempio:

```
char s[20]
...
s = strcat(stringa1, stringa2); /* NO! */
```

- Alcune funzioni possono essere usate "senza risultato"

- Esempio:

```
strcpy(<stringa destinazione>, <stringa origine>)
strcat(<stringa destinazione>, <stringa origine>)
```

- Il valore di ritorno coincide con la stringa destinazione

415

Esercizio 1

- Realizzare un meccanismo di "cifratura" di un testo che consiste nell'invertire le righe del testo stesso

- Esempio:

C'era una volta
un re che ...



atlov anu are`C
ehc er nu

416

Esercizio 1: Soluzione

```
#include <stdio.h>
main()
{
    int i,j, len;
    char s[80], dest[80]; /* una riga */

    while (gets(s) != NULL) {
        len = strlen(s); j = 0;
        for (i=len-1;i>=0;i--) {
            dest[j] = s[i];
            j++;
        }
        dest[j]='\0';
        puts(dest);
    }
}
```

417

Esercizio 2

- Si scriva un programma che legga da tastiera due stringhe e cancelli dalla prima stringa i caratteri contenuti nella seconda stringa

- Esempio:

- str1: "Olimpico"
- str2: "Oio"
- risultato: "Impc"

418

Esercizio 2: Soluzione

```
#include <stdio.h>
#define MAXCAR 128

char *elimina(char str1[], char str2[]);

main()
{
    char str1[MAXCAR], str2[MAXCAR];
    printf("Dammi la stringa str1: ");
    scanf("%s", str1);
    printf("Dammi la stringa str2: ");
    scanf("%s", str2);
    printf("str1-str2= %s\n", elimina(str1,str2));
}
```

419

Esercizio 2: Soluzione (Cont.)

```
char *elimina(char str1[], char str2[])
{
    int i, j, k;

    for(i=j=0;str1[i]!='\0';i++)
    {
        for(k=0;(str2[k]!='\0') && (str1[i]!=str2[k]);k++);
        if(str2[k]=='\0')
            str1[j++]=str1[i];
    }
    str1[j]='\0';
    return str1;
}
```

420

I/O a righe

- Acquisizione/stampa di una riga alla volta
 - Riga = Serie di caratteri terminata da `'\n'`

- Istruzioni:

- `char *gets(<stringa>)`
 - Legge una riga da tastiera (fino al `'\n'`)
 - La riga viene fornita come stringa (<stringa>), senza il carattere `'\n'`
 - In caso di errore, il risultato è la costante `NULL` (definita in `stdio.h`)
- `int puts(<stringa>)`
 - Stampa <stringa> su schermo
 - Aggiunge sempre `'\n'` alla stringa

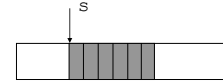
421

I/O a righe (Cont.)

- L'argomento di `gets/puts` è di tipo "puntatore" (discussione più avanti), definito come segue:
`char*`

- Significato: Il puntatore ad una stringa contiene l'indirizzo di memoria in cui il primo carattere della stringa è memorizzato

- Esempio:
 - `char* s;`



422

I/O a righe (Cont.)

- NOTE:

- `puts/gets` sono "costruite" a partire da `getchar/putchar`
- Uso di `gets` richiede l'allocazione dello spazio di memoria per la riga letta in input
 - Gestione dei puntatori che vedremo più avanti
- `puts(s)` è identica a `printf("%s\n", s);`

- Usate meno di frequente delle altre istruzioni di I/O

423

I/O a righe: Esempio

- Programma che replica su video una riga di testo scritta dall'utente

```
#include <stdio.h>

main()
{
    char *s, *res;
    printf("Scrivi qualcosa\n");
    //res = gets(s);
    if (res != NULL) // if (gets(s) != NULL) /* errore ? */
    {
        puts("Hai inserito"); //printf("Hai inserito\n");
        puts(s); //printf("%s\n", s);
    }
}
```

424

Vettori multidimensionali

- E' possibile estendere il concetto di variabile indicizzata a più dimensioni
 - Utilizzo tipico:
Vettori bidimensionali per realizzare tabelle (matrici)

- Implementazione: Vettore di vettori

a	a[0][0]	a[0][1]	a[0][2]	a[0][3]	a[0][4]	a[0]
	a[1][0]	a[1][1]	a[1][2]	a[1][3]	a[1][4]	a[1]
	a[2][0]	a[2][1]	a[2][2]	a[2][3]	a[2][4]	a[2]

```
int a[3][5];
```

425

Vettori multidimensionali (Cont.)

- Sintassi:
`<tipo> <nome vettore> [<dim1>][<dim2>] ... [<dimN>];`

- Accesso ad un elemento:
`<nome vettore> [<pos1>] [<pos2>] ... [<posN>]`

- Esempio:

```
int v[3][2];
```

- Inizializzazione:

- Inizializzazione per righe!

- Esempio:

```
int v[3][2] = {{8,1}, {1,9}, {0,3}}; // vettore 3x2
int w[3][2] = { 8,1, 1,9, 0,3 }; // equivalente
```

426

Matrice bidimensionale

		colonne				
		0	1	2	...	M-1
righe	0	1	2	3	4	5
	1	2	4	6	8	10
	2	3	6	9	12	15
	⋮	4	8	12	16	20
	⋮	5	10	15	20	25
	N-1	6	12	18	24	30

pitagora

427

Vettori multidimensionali e cicli

- Per vettori a più dimensioni, la scansione va applicata a tutte le dimensioni
- Cicli "annidati"

- Esempio:
Accesso agli elementi di una matrice 3x5

```
int x[3][5];
...
for (i=0; i < 3; i++) { /* per ogni riga i */
    for (j=0; j < 5; j++) { /* per ogni colonna j */
        ... // operazione su x[i][j]
    }
}
```

428

Stampa per righe matrice di reali

```
printf("Matrice: %d x %d\n", N, M);
for(i=0; i<N; i++)
{
    for(j=0; j<M; j++) /* Stampa la riga i-esima */
    {
        printf("%f ", mat[i][j]) ;
    }
    printf("\n");
}
```

429

Lettura per righe matrice di reali

```
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]) ;
    }
}
```

430

Somma per righe

```
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]) ;
```

431

Esercizio 1

- Scrivere un programma che acquisisca da tastiera gli elementi di una matrice quadrata 5x5 e che stampi su video la matrice trasposta
- Analisi:
 - Per il caricamento dei dati nella matrice, utilizziamo due cicli `for` annidati
 - Il più interno scandisce la matrice per colonne, utilizzando l'indice `j`
 - Il più esterno scandisce la matrice per righe, utilizzando l'indice `i`
 - Per la stampa della matrice trasposta, utilizziamo due cicli `for` annidati, ma con gli indici di riga (`i`) e colonna (`j`) scambiati rispetto al caso dell'acquisizione dei dati

432

Esercizio 1: Soluzione

```
#include <stdio.h>

main()
{
    int matrice[5][5], i, j;

    printf("Inserire gli elementi per righe:\n");
    for (i=0; i<5; i++)
        for (j=0; j<5; j++)
            scanf("%d", &matrice[i][j]);
    printf("\n\n");

    /* stampa della matrice trasposta */
    for (j=0; j<5; j++) {
        for (i=0; i<5; i++)
            printf("%5d", matrice[i][j]);
        printf("\n");
    }
}
```

433

Esercizio 2

- Scrivere un programma che legga da tastiera una matrice quadrata di dimensioni massime 32x32 e stampi su video la matrice stessa con accanto a destra la somma di ciascuna riga ed in basso la somma di ciascuna colonna
- Le dimensioni della matrice devono essere inserite da tastiera
- Esempio:

4	3	1	2	10
1	7	2	2	12
3	3	5	0	11
8	13	8	4	

434

Esercizio 2 (Cont.)

- **Analisi:**
 - Il caricamento dei valori nella matrice avviene con un doppio ciclo `for` annidato
 - Le somme delle varie righe sono memorizzate in un vettore `vet_righe` avente un numero di elementi pari al numero di righe della matrice
 - Le somme delle varie colonne sono memorizzate in un vettore `vet_col` avente un numero di elementi pari al numero di colonne della matrice
 - Il calcolo delle somme delle righe viene eseguito tramite un doppio ciclo `for` annidato che scandisce la matrice per righe
 - Il calcolo delle somme delle colonne viene eseguito tramite un doppio ciclo `for` annidato che scandisce la matrice per colonne
 - La stampa della matrice e del vettore `vet_righe` avviene con un doppio ciclo `for` annidato
 - La stampa del vettore `col` avviene con un singolo ciclo `for`

435

Esercizio 2: Soluzione

```
#include <stdio.h>
#define MAXDIM 32

main()
{
    int matrice[MAXDIM][MAXDIM],
        vet_righe[MAXDIM],
        vet_col[MAXDIM],
        nrighe, ncol, somma, i, j;

    printf("Inserire le dimensioni della matrice: ");
    scanf("%d %d", &nrighe, &ncol);
    /* caricamento elementi della matrice per righe */
    printf("Inserire gli elementi per righe:\n");
    for (i=0; i<nrighe; i++)
        for (j=0; j<ncol; j++)
            scanf("%d", &matrice[i][j]);
```

436

Esercizio 2: Soluzione (Cont.)

```
/* calcolo della somma delle righe */
for (i=0; i<nrighe; i++) {
    somma = 0;
    for (j=0; j<ncol; j++)
        somma = somma + matrice[i][j];
    vet_righe[i] = somma;
}

/* calcolo della somma delle colonne */
for (j=0; j<ncol; j++) {
    somma = 0;
    for (i=0; i<nrighe; i++)
        somma = somma + matrice[i][j];
    vet_col[j] = somma;
}
```

437

Esercizio 2: Soluzione (Cont.)

```
/* stampa matrice e vettore somma delle righe */
printf("\n\n");
for (i=0; i<nrighe; i++) {
    for (j=0; j<ncol; j++)
        printf("%4d", matrice[i][j]);
    printf("%7d\n", vet_righe[i]);
}
/* stampa vettore somma delle colonne */
printf("\n");
for (j=0; j<ncol; j++)
    printf("%4d", vet_col[j]);
printf("\n\n");
}
```

438

Settimana n.9

Obiettivi

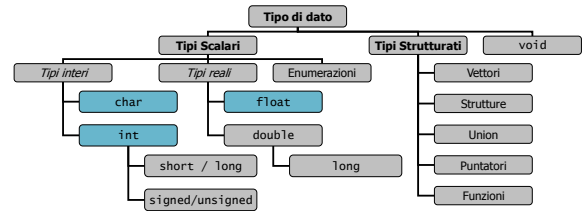
- Tipi di dato scalari (completi)
- Argomenti sulla linea di comando

Contenuti

- Il sistema dei tipi scalari in C (completo)
- Attivazione di programmi da command line
- Argc, argv
- Conversione degli argomenti
- La funzione exit

439

Il sistema dei tipi C



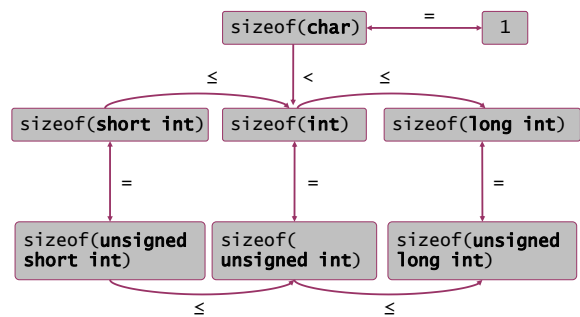
440

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	

441

Specifiche del C



442

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>
```

443

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

444

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

$$A = \overset{\text{segno}}{\pm} \underbrace{1.mmmmm}_{\text{mantissa}} \times 2^{\overset{\text{esponente}}{\pm eeeee}}$$

445

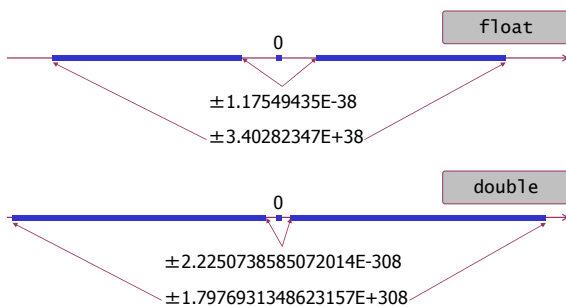
Numero di bit

Tipo	Dimensione	Mantissa	Esponente
float	32 bit	23 bit	8 bit
double	64 bit	53 bit	10 bit
long double	≥ double		

$$A = \overset{\text{segno}}{\pm} \underbrace{1.mmmmm}_{\text{mantissa}} \times 2^{\overset{\text{esponente}}{\pm eeeee}}$$

446

Intervallo di rappresentazione



447

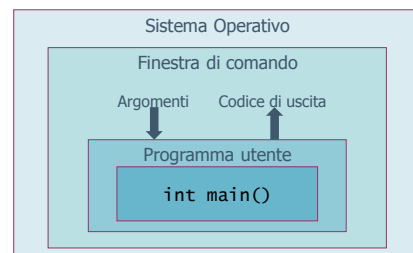
Specificatori di formato

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

448

Argomenti sulla linea di comando

Il modello "console"



450

Argomenti sulla linea di comando

- In C, è possibile passare informazioni ad un programma specificando degli argomenti sulla linea di comando
 - Esempio:

```
C:\> myprog <arg1> <arg2> ... <argN>
```
- Comuni in molti comandi "interattivi"
 - Esempio: MS-DOS

```
C:\> copy file1.txt dest.txt
```
- Automaticamente memorizzati negli argomenti del `main()`

451

Argomenti del `main()`

- Prototipo:

```
main (int argc, char* argv[])
```

- `argc`: Numero di argomenti specificati
 - Esiste sempre almeno un argomento (il nome del programma)
- `argv`: Vettore di stringhe
 - `argv[0]` = primo argomento
 - `argv[i]` = generico argomento
 - `argv[argc-1]` = ultimo argomento

452

Esempi

```
C:\progr>quadrato
```

- ▶ Numero argomenti = 0

```
C:\progr>quadrato 5
```

- ▶ Numero argomenti = 1
- ▶ Argomento 1 = "5"

```
C:\progr>quadrato 5 K
```

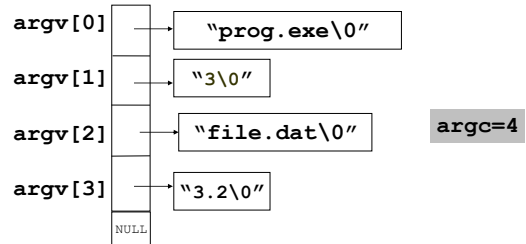
- ▶ Numero argomenti = 2
- ▶ Argomento 1 = "5"
- ▶ Argomento 2 = "K"

453

`argc` e `argv`

- Struttura:

- Esempio: `c:\> prog.exe 3 file.dat 3.2`



454

`argc` e `argv` (Cont.)

- Ciclo per l'elaborazione degli argomenti

```
for (i=0; i<argc; i++) {  
    /*  
    elabora argv[i] come stringa  
    */  
}
```

- NOTA:

- Qualunque sia la natura dell'argomento, è sempre una stringa
- Necessario quindi uno strumento per "convertire" in modo efficiente stringhe in tipi numerici

455

Conversione degli argomenti

- Il C mette a disposizione tre funzioni per la conversione di una stringa in valori numerici

```
int atoi(char *s);  
long atol(char *s);  
double atof(char *s);
```

- Esempio:

```
int x = atoi("2") ; // x=2  
long y = atol("23L"); // y=23  
double z = atof("2.35e-2"); // z=0.0235
```

- Definite in `stdlib.h`

456

atoi, atof, atol: esempi

```
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) ;
```

457

Conversione degli argomenti (Cont.)

- NOTA: Si assume che la stringa rappresenti l'equivalente di un valore numerico:
 - cifre, '+', '-' per interi
 - cifre, '+', '-', 'l', 'L' per long
 - cifre, '+', '-', 'e', 'E', '.' per reali
- In caso di conversione errata o non possibile le funzioni restituiscono il valore 0
 - Necessario in certi casi controllare il valore della conversione!
- NOTA: Importante controllare il valore di ogni `argv[i]`!

458

Conversione degli argomenti (Cont.)

- Esempio:
Programma C che prevede due argomenti sulla linea di comando:
 - Primo argomento: Un intero
 - Secondo argomento: Una stringa
- Schema:

```
int x;
char s[80];
x = atoi(argv[1]);
strcpy(s,argv[2]); /* s=argv[2] è errato! */
```

459

Programmi e opzioni

- Alcuni argomenti sulla linea di comando indicano tipicamente delle modalità di funzionamento "alternative" di un programma
- Queste "opzioni" (dette **flag** o **switch**) sono convenzionalmente specificate come
--<carattere>
per distinguerle dagli argomenti veri e propri
- Esempio
C:\> myprog -x -u file.txt
 opzioni argomento

460

La funzione exit

- Esiste inoltre la funzione di libreria `exit`, dichiarata in `<stdlib.h>`, che:
 - 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) ;
```

461

Esercizio 1

- Scrivere un programma che legga sulla linea di comando due interi N e D, e stampi tutti i numeri minori o uguali ad N divisibili per D

462

Esercizio 1: Soluzione

```
#include <stdio.h>
main(int argc, char* argv[] )
{
    int N, D, i;
    if (argc != 3) {
        fprintf(stderr, "Numero argomenti non valido\n");
        return 1;
    }
    if (argv[1] != NULL) N = atoi(argv[1]);
    if (argv[2] != NULL) D = atoi(argv[2]);

    for (i=1; i<=N; i++) {
        if ((i % D) == 0) {
            printf("%d\n", i);
        }
    }
}
```

Altrimenti le operazioni successive operano su stringhe = NULL

463

Esercizio 2: Soluzione

```
#include <stdio.h>
main(int argc, char* argv[]) {
    int lowercase = 0, uppercase = 0;

    for (i=1; i<argc; i++)
    {
        switch (argv[i][1]) {
            case 'l':
                lowercase = 1;
                break;
            case 'L':
                uppercase = 1;
                break;
            case 'h':
                printf("Uso: m2m [-luh]\n");
                break;
        }
    }
    ...
}
```

465

Esercizio 3: Soluzione

```
main(int argc, char* argv[]) {
    int i, aflag=0, bflag=0;
    char filename[80];

    if (argc >= 2) { /* almeno due argomenti */
        /* copiamo in una stringa, verra' aperto dopo */
        strcpy (filename, argv[argc-1]);

        /* processiamo gli altri (eventuali argomenti) */
        for (i=1; i<= argc-1; i++) {
            if (argv[i][0] == '-') { /* e' un flag */
                switch (argv[i][1]) {
                    case 'a':
                        aflag = 1; break;
                    case 'b':
                        bflag = 1; break;
                    default:
                        fprintf(stderr, "Opzione non corretta.\n");
                }
            }
        }
    }
}
```

467

Esercizio 2

- Scrivere un programma `m2m` che legga da input un testo e converta tutte le lettere maiuscole in minuscole e viceversa, a seconda dei flag specificati sulla linea di comando

-l, -L conversione in minuscolo
-u, -U conversione in maiuscolo

Un ulteriore flag `-h` permette di stampare un help

- Utilizzo:

m2m -l
m2m -L
m2m -u
m2m -U
m2m -h

464

Esercizio 3

- Scrivere un frammento di codice che gestisca gli argomenti sulla linea di comando per un programma `TEST.EXE` il cui comando ha la seguente sintassi:

`TEST.EXE [-a] [-b] <nome file>`

- I flag `-a` e `-b` sono opzionali (e possono essere in un ordine qualunque)
- L'ultimo argomento (`<nome file>`) è obbligatorio (ed è sempre l'ultimo)

- Esempi validi di invocazione:

```
TEST.EXE <nome file>
TEST.EXE -a <nome file>
TEST.EXE -b <nome file>
TEST.EXE -a -b <nome file>
TEST.EXE -b -a <nome file>
```

466

Esercizio 3: Soluzione (Cont.)

```
    }
    else {
        /* Non e' un flag. Esce dal programma */
        fprintf(stderr, "Errore di sintassi.\n");
        return;
    }
}
else {
    /* sintassi errata. Esce dal programma */
    fprintf(stderr, "Errore di sintassi.\n");
    return;
}
}
```

468

Settimana n.10

Obiettivi

- Strutture
- Vettori di strutture

Contenuti

- Struct. Operatore "."
- Definizione vettori di struct
- Definizione di struct contenenti vettori (es. stringhe)

469

Tipi aggregati

Tipi aggregati

- In C, è possibile definire dati composti da elementi eterogenei (detti *record*), aggregandoli in una singola variabile
 - Individuata dalla keyword `struct`
- Sintassi (definizione di tipo):

```
struct <identificatore> {  
    campi  
};
```

I campi sono nel formato:

```
<tipo> <nome campo>;
```

471

Tipi aggregati: esempio

studente

cognome:	Rossi		
nome:	Mario		
matricola:	123456	media:	27.25

472

struct

- Una definizione di `struct` equivale ad una definizione di tipo
- Successivamente, una struttura può essere usata come un tipo per dichiarare variabili
- Esempio:

```
struct complex {  
    double re;  
    double im;  
}  
...  
struct complex num1, num2;
```

473

struct: Esempi

```
struct complex {  
    double re;  
    double im;  
}  
  
struct identity {  
    char nome[30];  
    char cognome[30];  
    char codicefiscale[15];  
    int altezza;  
    char statocivile;  
}
```

474

Accesso ai campi di una struct

- Una struttura permette di accedere ai singoli campi tramite l'operatore '.', applicato a variabili del corrispondente tipo struct

<variabile>.<campo>

- Esempio:

```
struct complex {
    double re;
    double im;
}
...
struct complex num1, num2;
num1.re = 0.33; num1.im = -0.43943;
num2.re = -0.133; num2.im = -0.49;
```

475

Definizione di struct come tipo

- E' possibile definire un nuovo tipo a partire da una struct tramite la direttiva typedef
 - Passabile come parametro
 - Indicizzabile in vettori

- Sintassi:

`typedef <tipo> <nome nuovo tipo>;`

- Esempio:

```
typedef struct complex {
    double re;
    double im;
} compl;
compl z1, z2;
```

476

Definizione di struct come tipo (Cont.)

- Passaggio di struct come argomenti

```
int f1 (compl z1, compl z2)
```

- struct come risultato di funzioni

```
compl f2(.....)
```

- Vettore di struct

```
compl lista[10];
```

- Nota:

La direttiva typedef è applicabile anche non alle strutture per definire nuovi tipi

- Esempio: typedef unsigned char BIT8;

477

Operazioni su struct

- Confronto:

- Non è possibile confrontare due variabili dello stesso tipo di struct usando il loro nome

- Esempio:

```
compl s1, s2 * s1==s2 o s1!=s2 è un errore di sintassi
```

- Il confronto deve avvenire confrontando i campi uno ad uno

- Esempio:

```
compl s1, s2 *(s1.re == s2.re) && (s1.im == s2.im)
```

- Inizializzazione:

- Come per i vettori, tramite una lista di valori tra {}

- Esempio:

```
compl s1 = {0.1213, 2.655};
```

- L'associazione è posizionale: In caso di valori mancanti, questi vengono inizializzati a:

- 0, per valori "numerici"
- NULL per puntatori

478

Esercizio 1

- Data la seguente struct:

```
struct stud {
    char nome[40];
    unsigned int matricola;
    unsigned int voto;
}
```

- Si definisca un corrispondente tipo studente
- Si scriva un main() che allochi un vettore di 10 elementi e che invochi la funzione descritta di seguito
- Si scriva una funzione ContaInsufficienti() che riceva come argomento il vettore sopracitato e ritorni il numero di studenti che sono insufficienti

479

Esercizio 1: Soluzione

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

```
#define NSTUD 10
```

```
typedef struct stud {
    char nome[40];
    unsigned int matricola;
    unsigned int voto;
} studente;
```

```
int ContaInsufficienti(studente vett[], int dim); /*
prototipo */
```

480

Esercizio 1: Soluzione (Cont.)

```
main()
{
    int i, NumIns;
    studente Lista[NSTUD];

    /* assumiamo che il programma riempia
       con valori opportuni la lista */

    NumIns = ContaInsufficienti(Lista, NSTUD);
    printf("Il numero di insufficienti e': %d.\n", NumIns);
}

int ContaInsufficienti(studente s[], int numstud)
{
    int i, n=0;

    for (i=0; i<numstud; i++) {
        if (s[i].voto < 18)
            n++;
    }
    return n;
}
```

481

Esercizio 2

- Data una struct che rappresenta un punto nel piano cartesiano a due dimensioni:

```
struct point {
    double x;
    double y;
};
```

ed il relativo tipo typedef struct point Point; scrivere le seguenti funzioni che operano su oggetti di tipo Point:

```
- double DistanzaDaOrigine (Point p);
- double Distanza (Point p1, Point p2);
- int Quadrante (Point p); /* in quale quadrante */
- int Allineati(Point p1, Point p2, Point p3);
  /* se sono allineati */
- int Interseca(Point p1, Point p2);
  /* se il segmento che ha per estremi p1 e p2
  interseca un qualunque asse*/
```

482

Esercizio 2: Soluzione

```
double DistanzaDaOrigine (Point p)
{
    return sqrt(p.x*p.x + p.y*p.y);
}

double Distanza (Point p1, Point p2)
{
    return sqrt((p1.x-p2.x)*(p1.x-p2.x) +
               (p1.y-p2.y)*(p1.y-p2.y));
}

int Quadrante (Point p)
{
    if (p.x >= 0 && p.y >= 0) return 1;
    if (p.x <= 0 && p.y >= 0) return 2;
    if (p.x <= 0 && p.y <= 0) return 3;
    if (p.x >= 0 && p.y <= 0) return 4;
}
```

483

Esercizio 2: Soluzione (Cont.)

```
int Allineati (Point p1, Point p2, Point p3)
{
    double r1, r2;
    /* verifichiamo che il rapporto tra y e x
       delle due coppie di punti sia identico */
    r1 = (p2.y-p1.y)/(p2.x-p1.x);
    r2 = (p3.y-p2.y)/(p3.x-p2.x);
    if (r1 == r2)
        return 1;
    else
        return 0;
}

int Interseca(Point p1, Point p2)
{
    /* verifichiamo se sono in quadranti diversi */
    if (Quadrante(p1) == Quadrante(p2))
        return 0;
    else
        return 1;
}
```

484

Settimana n.11

Obiettivi

- File di testo

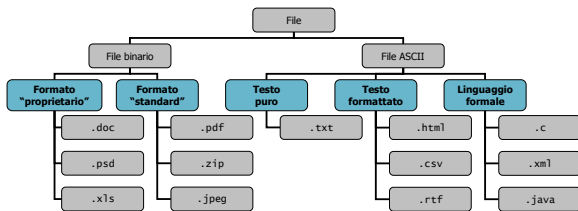
Contenuti

- Concetto di file e funzioni fopen/fclose
- Funzioni fgets+sscanf
- Approfondimenti su printf e scanf

Files

485

Vista d'insieme dei formati di file



487

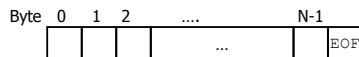
File sequenziali

- Il modo più comune per realizzare I/O da file consiste nell'utilizzo del cosiddetto accesso *bufferizzato*
 - Informazioni prelevate dal file attraverso una memoria interna al sistema (detta *buffer*)
- Vantaggi:
 - Livello di astrazione più elevato
 - Possibilità di I/O formattato
- I/O non bufferizzato:
 - Accesso diretto a livello binario un carattere per volta

488

File sequenziali (Cont.)

- Il C vede i file come un *flusso (stream) sequenziale di byte*
 - **Nessuna struttura particolare:**
La strutturazione del contenuto è a carico del programmatore
 - Carattere terminatore alla fine del file: EOF



- **NOTA:** L'accesso sequenziale implica l'impossibilità di:
 - Leggere all'indietro
 - Saltare ad uno specifico punto del file

489

File sequenziali (Cont.)

- Accesso tramite una variabile di tipo `FILE*`
- Definita in `stdio.h`
- Dichiarazione:

```
FILE* <file>;
```

`FILE*` contiene un insieme di variabili che permettono l'accesso per "tipi"
- Al momento dell'attivazione di un programma vengono automaticamente attivati tre file:
 - `stdin`
 - `stdout`
 - `stderr`

490

File sequenziali (Cont.)

- La struttura dati `FILE` contiene vari campi:

```
- short      level;      /* Fill/empty level of buffer */
- unsigned   flags;      /* File status flags          */
- char       fd;         /* File descriptor            */
- unsigned char hold;    /* Ungetc char if no buffer   */
- short      bsize;      /* Buffer size                 */
- unsigned char *buffer; /* Data transfer buffer       */
- unsigned char *curp;   /* Current active pointer     */
- unsigned   istemp;     /* Temporary file indicator   */
- short      token;      /* Used for validity checking */
```

491

File sequenziali (Cont.)

- `stdin` è automaticamente associato allo standard input (tastiera)
- `stdout` e `stderr` sono automaticamente associati allo standard output (video)
- `stdin`, `stdout`, `stderr` sono direttamente utilizzabili nelle istruzioni per l'accesso a file
 - In altre parole, sono delle variabili predefinite di tipo `FILE*`

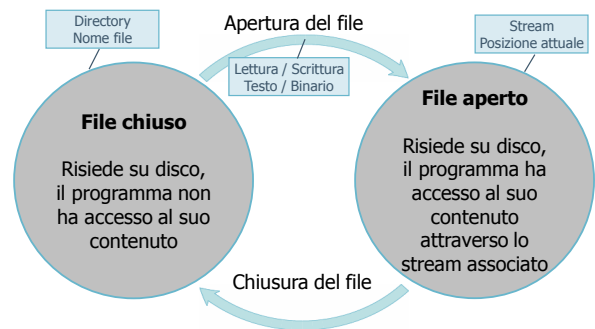
492

File: Operazioni

- L'uso di un file passa attraverso tre fasi fondamentali
 - **Apertura del file**
 - **Accesso al file**
 - **Chiusura del file**
- Prima di aprire un file occorre dichiararlo!

493

Stati di un file



494

Apertura di un file

- Per accedere ad un file è necessario aprirlo:
 - Apertura:
Connessione di un file fisico (su disco) ad un file logico (interno al programma)
- Funzione:

```
FILE* fopen(char* <nomefile>, char* <modo>);
```

<nomefile>: Nome del file fisico

495

Apertura di un file (Cont.)

- <modo>: Tipo di accesso al file
 - "r": sola lettura
 - "w": sola scrittura (cancella il file se esiste)
 - "a": *append* (aggiunge in coda ad un file)
 - "r+": lettura/scrittura su file esistente
 - "w+": lettura/scrittura su nuovo file
 - "a+": lettura/scrittura in coda o su nuovo file
- Ritorna:
 - Il puntatore al file in caso di successo
 - NULL in caso di errore

496

Chiusura di un file

- Quando l'utilizzo del file fisico è terminato, è consigliabile chiudere il file:
 - Chiusura:
Cancellazione della connessione di un file fisico (su disco) ad un file logico (interno al programma)
- Funzione:

```
int fclose(FILE* <file>);
```

<file>: File aperto in precedenza con `fopen()`

 - Ritorna:
 - 0 se l'operazione si chiude correttamente
 - EOF in caso di errore

497

Apertura e chiusura di un file: Esempio

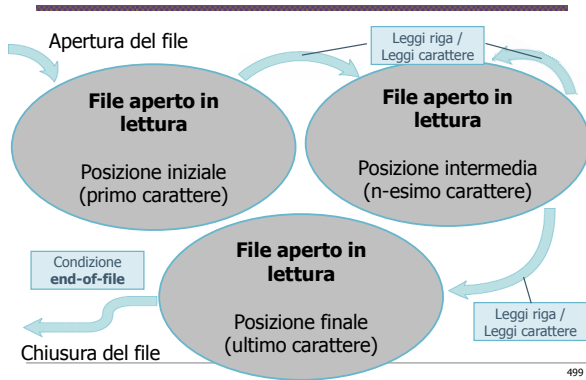
```
FILE *fp; /* variabile di tipo file */
...

/* apro file 'testo.dat' in lettura */
fp = fopen("testo.dat", "r");

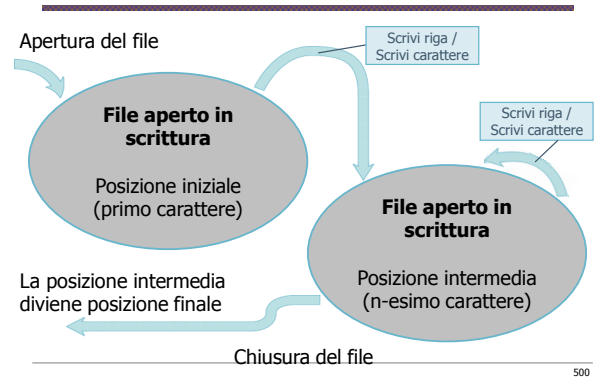
if (fp == NULL)
    printf("Errore nell'apertura\n");
else
{
    /* qui posso accedere a 'testo.dat' usando fp */
}
...
fclose(fp);
```

498

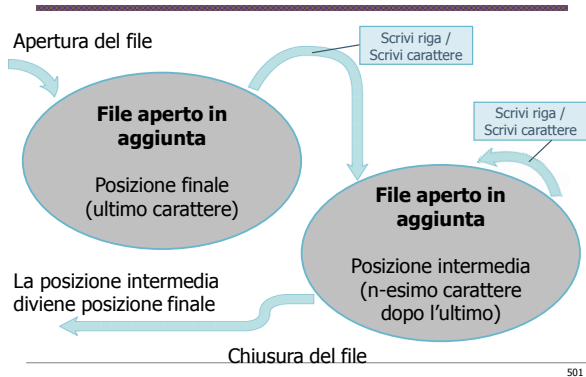
Letture di un file



Scrittura di un file



Aggiunta ad un file



Letture a caratteri

- Lettura:
 - int `getc (FILE* <file>);`
 - int `fgetc (FILE* <file>);`
 - Legge un carattere alla volta dal file
 - Restituisce il carattere letto o EOF in caso di fine file o errore
- NOTA: `getchar()` equivale a `getc(stdin)`

502

Scrittura a caratteri

- Scrittura:
 - int `putc (int c, FILE* <file>);`
 - int `fputc (int c, FILE* <file>);`
 - Scrive un carattere alla volta nel file
 - Restituisce il carattere scritto o EOF in caso di errore
- NOTA: `putchar(...)` equivale a `putc(..., stdout)`

503

Letture a righe

- Lettura:
 - `char* fgets(char* <s>, int <n>, FILE* <file>);`
 - Legge una stringa dal file fermandosi al più dopo n-1 caratteri
 - L'eventuale '\n' NON viene eliminato (diverso da `gets!`)
 - Restituisce il puntatore alla stringa letta o NULL in caso di fine file o errore
- NOTA: `gets(...)` "equivale" a `fgets(..., stdin)`

504

Scrittura a righe

- **Scrittura:**

- ```
int fputs(char* <s>, FILE* <file>);
```
- Scrive la stringa <*s*> nel senza aggiungere '\n' (diverso da puts !)
  - Restituisce l'ultimo carattere scritto, oppure EOF in caso di errore

- **NOTA:** puts(...) "equivalente" a fputs(..., stdout)

505

## Letture formattata

- **Letture:**

- ```
int fscanf(FILE* <file>, char* <formato>, ...);
```
- Come scanf(), con un parametro aggiuntivo che rappresenta un file
 - Restituisce il numero di campi convertiti, oppure EOF in caso di fine file

- **NOTA:** scanf(...) "equivalente" a fscanf(stdin, ...)

506

Scrittura formattata

- **Scrittura:**

- ```
int fprintf(FILE* <file>, char* <formato>, ...);
```
- Come printf(), con un parametro aggiuntivo che rappresenta un file
  - Restituisce il numero di byte scritti, oppure EOF in caso di errore

- **NOTA:** printf(...) "equivalente" a fprintf(stdout, ...)

507

## Altre funzioni

- FILE\* freopen(char\* <*nomefile*>, char\* <*modo*>);

- Come fopen, ma si applica ad un file già esistente
- Restituisce il puntatore al file oppure NULL

- int fflush(FILE\* <*file*>);

- "Cancella" il contenuto di un file
- Restituisce 0 se termina correttamente oppure EOF

- int feof(FILE\* <*file*>);

- Restituisce falso (0) se il puntatore NON è posizionato alla fine del file
- Restituisce vero (!0) se il puntatore è posizionato alla fine del file

508

## Posizionamento in un file

- Ad ogni file è associato un buffer ed un "puntatore" all'interno di questo buffer
- La posizione del puntatore può essere manipolata con alcune funzioni
- Più utile:

```
void rewind (FILE* <file>)
```

- Posiziona il puntatore all'inizio del file
- Utile per "ripartire" dall'inizio nella scansione di un file

509

## Schema generale di lettura da file

```
leggi un dato dal file;
finchè (non è finito il file)
{
 elabora il dato;
 leggi un dato dal file;
}
```

- La condizione "non è finito il file" può essere realizzata in vari modi:

- Usando i valori restituiti dalle funzioni di input (consigliato)
- Usando la funzione feof()

510

## Esempio 1

- Lettura di un file formattato (esempio: Un intero per riga)
  - Uso dei valori restituiti dalle funzioni di input (`fscanf`)

```
res = fscanf (fp, "%d", &val);
while (res != EOF)
{
 elabora val;
 res = fscanf (fp, "%d", &val);
}
```

511

## Esempio 1 (Cont.)

- Versione "compatta" senza memorizzare il risultato di `fscanf()`
  - Usiamo `fscanf()` direttamente nella condizione di fine input

```
while (fscanf (fp, "%d", &val) != EOF)
{
 elabora val;
}
```

512

## Esempio 2

- Lettura di un file formattato (esempio: Un intero per riga)
  - Uso di `feof()`

```
fscanf (fp, "%d", &val);
while (!feof(fp))
{
 elabora val;
 fscanf (fp, "%d", &val);
}
```

513

## Esempio 3

- Lettura di un file non formattato
  - Uso dei valori restituiti dalle funzioni di input (`getc`)

```
c = getc(fp);
while (c != EOF)
{
 elabora c; Versione 1
 c = getc(fp);
}

.....

while ((c=getc(fp)) != EOF)
{
 elabora c; Versione 2
}
```

514

## Esempio 4

- Lettura di un file non formattato
  - Uso dei valori restituiti dalle funzioni di input (`fgets`)

```
s = fgets(s,80,fp);
while (s != NULL)
{
 elabora s; Versione 1
 s = fgets(s,80,fp);
}

.....

while ((s = fgets(s,80,fp)) != NULL)
{
 elabora s; Versione 2
}
```

515

## Letture di un file: `fgets` + `sscanf`

- Lettura di un file formattato in cui ogni riga abbia un dato numero di campi di tipo noto (esempio un intero, ed una stringa)
  - Uso di `fgets` per leggere la riga, e di `sscanf` per leggere i campi

```
while ((s = fgets(s,80,fp)) != NULL)
{
 sscanf(s, "%d %s", &intero, stringa);
}
```

516

## Esercizio

- Leggere un file "estremi.dat" contenente coppie di numeri interi (x,y), una per riga e scrivere un secondo file "diff.dat" che contenga le differenze x-y, una per riga

- Esempio:

| File 1 |     | File 2 |
|--------|-----|--------|
| 23     | 32  | -9     |
| 2      | 11  | -9     |
| 19     | 6   | 13     |
| 23     | 5   | 18     |
| 3      | 2   | 1      |
| ...    | ... | ...    |

517

## Esercizio: Soluzione

```
#include <stdio.h>

main() {
 FILE *fpin, *fpout;
 int x,y;
 /* apertura del primo file */
 if ((fpin = fopen("estremi.dat","r")) == NULL)
 {
 fprintf(stderr,"Errore nell'apertura\n");
 return 1;
 }
}
```

518

## Esercizio: Soluzione (Cont.)

```
/* apertura del secondo file */
if ((fpout = fopen("diff.dat","w")) == NULL)
{
 fprintf(stderr,"Errore nell'apertura\n");
 return 1;
}
/* input */
while (fscanf(fpin,"%d %d",&x,&y) != EOF)
{
 /* ora ho a disposizione x e y */
 fprintf(fpout,"%d\n",x-y);
}
fclose (fpin);
fclose (fpout);
}
```

519

## Avvertenza

- In generale, è errato tentare di memorizzare il contenuto di un file in un vettore
  - La dimensione (numero di righe o di dati) di un file non è quasi mai nota a priori
  - Se la dimensione è nota, tipicamente è molto grande!

520

## 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
  - ...

521

## 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       |

522

## Esempi

| Istruzione                          | Risultato      |
|-------------------------------------|----------------|
| <code>printf("%d", 13);</code>      | 13             |
| <code>printf("%1d", 13);</code>     | 13             |
| <code>printf("%3d", 13);</code>     | _13            |
| <code>printf("%F", 13.14);</code>   | 13.140000      |
| <code>printf("%6F", 13.14);</code>  | 13.140000      |
| <code>printf("%12f", 13.14);</code> | _ _ _13.140000 |
| <code>printf("%6s", "ciao");</code> | _ _ciao        |

523

## Esempi (Cont.)

| Istruzione                             | Risultato |
|----------------------------------------|-----------|
| <code>printf("%.1d", 13);</code>       | 13        |
| <code>printf("%.4d", 13);</code>       | 0013      |
| <code>printf("%.6.4d", 13);</code>     | _ _0013   |
| <code>printf("%.4.6d", 13);</code>     | 000013    |
| <code>printf("%.2s", "ciao");</code>   | ci        |
| <code>printf("%.6s", "ciao");</code>   | ciao      |
| <code>printf("%.6.3s", "ciao");</code> | _ _ _cia  |

524

## Esempi (Cont.)

| Istruzione                            | Risultato  |
|---------------------------------------|------------|
| <code>printf("%.2f", 13.14);</code>   | 13.14      |
| <code>printf("%.4f", 13.14);</code>   | 13.1400    |
| <code>printf("%.6.4f", 13.14);</code> | 13.1400    |
| <code>printf("%.9.4f", 13.14);</code> | _ _13.1400 |

525

## Esempi (Cont.)

| Istruzione                           | Risultato  |
|--------------------------------------|------------|
| <code>printf("%6d", 13);</code>      | _ _ _ _13  |
| <code>printf("%-6d", 13);</code>     | 13 _ _ _ _ |
| <code>printf("%06d", 13);</code>     | 000013     |
| <code>printf("%6s", "ciao");</code>  | _ _ciao    |
| <code>printf("%-6s", "ciao");</code> | ciao _ _   |

526

## Esempi (Cont.)

| Istruzione                       | Risultato |
|----------------------------------|-----------|
| <code>printf("%d", 13);</code>   | 13        |
| <code>printf("%d", -13);</code>  | -13       |
| <code>printf("%+d", 13);</code>  | +13       |
| <code>printf("%+d", -13);</code> | -13       |
| <code>printf("% d", 13);</code>  | _13       |
| <code>printf("% d", -13);</code> | -13       |

527

## Approfondimenti su scanf

- Tipologie di caratteri nella stringa di formato
- Modificatori degli specificatori di formato
- Valore di ritorno
- Specificatore %[]

528



## 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)

529

## 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

530

## 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 %[...] |

531

## 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"     |

532

## Esempi (Cont.)

| Istruzione              | Input    | Risultato                 |
|-------------------------|----------|---------------------------|
| scanf("%d %s", &x, v) ; | 10 Pippo | x = 10<br>v = "Pippo"     |
| scanf("%s", v) ;        | 10 Pippo | x immutato<br>v = "10"    |
| scanf("%*d %s", v) ;    | 10 Pippo | x immutato<br>v = "Pippo" |

533

## 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

534

## Letture 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]

535

## Esempi (Cont.)

| 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-zA-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                                                                    |
| %[^\s;.,!?]  | Si ferma alla punteggiatura o spazio                                                      |

536

## Settimana n.12

### Obiettivi

- Comprensione degli operatori & e \*

### Contenuti

- Cenno ai puntatori
- Puntatori e vettori
- Puntatori e stringhe
- Cenno alla allocazione dinamica di un vettore

537

## Puntatori

## Puntatori

- Puntatori = Variabili che contengono indirizzi di memoria
  - Indirizzi = Indirizzi di una variabile (di qualche tipo)
- Definiti tramite l'operatore unario '\*' posto accanto al nome della variabile
- Sintassi:  
`<tipo> * * <identificatore>`
- Per ogni tipo di variabile "puntata", esiste un "tipo" diverso di puntatore
  - Esempi:  
`int x; * int *px;`  
`double y; * double *py;`

539

## Puntatori (Cont.)

- L'indirizzo di una variabile (da assegnare, per esempio, ad un puntatore) è determinabile tramite l'operatore unario '&' posto a fianco dell'identificatore della variabile
- Sintassi:  
`'&' <identificatore>`
- Esempio:  
`int a, *ptr;`  
`ptr = &a;`

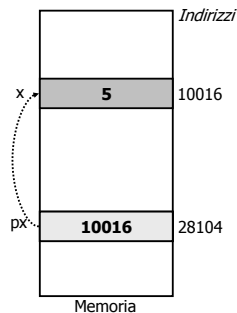
540

## Puntatori: Esempio

```
int x=5, *px;
...
px = &x; /* px = indirizzo di x */

printf("%d\n",x); --> 5
printf("%d\n",px); --> 10016
printf("%d\n",&x); --> 10016
printf("%d\n",*px); --> 5

/* x e *px sono la stessa cosa! */
```



541

## Puntatori (Cont.)

- Gli operatori '\*' e '&' sono l'uno l'inverso dell'altro
  - L'applicazione consecutiva in qualunque ordine ad un puntatore fornisce lo stesso risultato
    - Esempio:
      - \*&px e &\*px sono la stessa cosa!
- In termini di operatori:
  - '\*' = Operatore di *indirizione*
    - Opera su un indirizzo
    - Ritorna il valore contenuto in quell'indirizzo
  - '&' = Operatore di *indirizzo*
    - Opera su una variabile
    - Ritorna l'indirizzo di una variabile
- Quindi:
  - \* (&px): Fornisce il contenuto della casella che contiene l'indirizzo di px
  - & (\*px): Fornisce l'indirizzo della variabile (\*px) puntata da px

542

## Puntatori (Cont.)

- Esempio:

```
main()
{
 int a, *aptr;
 a=7;
 aptr = &a;

 printf("l'indirizzo di a e' %p\n", &a);
 printf("il valore di aptr e' %p\n", aptr);

 printf("il valore di a e' %p\n", a);
 printf("il valore di *aptr e' %p\n", *aptr);

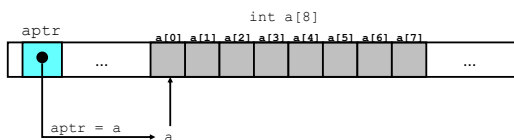
 printf("&*aptr = %p ----- *aptr = %p\n",
 &*aptr, *aptr);
}
```

543

## Puntatori e vettori

### Puntatori e vettori

- La relazione tra puntatori e vettori in C è molto stretta
- Elemento di connessione:
  - Il nome di un vettore rappresenta un indirizzo
    - Indirizzo del primo elemento!
  - Esempio:
    - int a[8], \*aptr;
    - L'assegnazione `aptr = a;` fa puntare `aptr` al primo elemento del vettore `a`



545

### Puntatori e vettori: Analogie

- L'analogia tra puntatori e vettori si basa sul fatto che, dato un vettore `a[]`

$$a[i] \text{ e } *(aptr+i) \text{ sono la stessa cosa}$$
- Due modi per accedere al generico elemento del vettore:
  1. Usando l'array (tramite indice)
  2. Usando un puntatore (tramite scostamento o *offset*)
- Interpretazione:
  - `aptr+i` = Indirizzo dell'elemento che si trova ad `i` posizioni dall'inizio
  - `*(aptr+i)` = Contenuto di tale elemento

546

## Puntatori e vettori: Analogie (Cont.)

- E' possibile equivalentemente:
  - Accedere al vettore tramite offset usando il nome del vettore  
`*(a+i)`
  - Accedere al vettore tramite indice usando il puntatore  
`aptr[i]`
- Riassumendo:
  - Dati `int a[], *aptr=a;` l'accesso al generico elemento `i` del vettore a può avvenire mediante:  
`a[i]`  
`*(a+i)`  
`aptr[i]`  
`*(aptr+i)`

547

## Puntatori e vettori: Esempio

```
main()
{
 int b[] = {1,2,3,4};
 int *bptr = b;
 int i, offset;

 printf("== notazione indicizzata ==\n");
 for(i=0; i<4; i++) {
 printf("b[%d] = %d\n",i, b[i]);
 }

 printf("== notazione tramite offset e vettore ==\n");
 for(offset=0; offset<4; offset++) {
 printf("(b+%d) = %d\n",offset, *(b+offset));
 }

 printf("== notazione tramite indice e puntatore ==\n");
 for(i=0; i<4; i++) {
 printf("bptr[%d] = %d\n",i, bptr[i]);
 }

 printf("== notazione tramite offset e puntatore ==\n");
 for(offset=0; offset<4; offset++) {
 printf("(bptr+%d) = %d\n",offset, *(bptr+offset));
 }
}
```

548

## Puntatori e vettori: Differenze

- Un puntatore è una variabile che può assumere valori differenti
- Il nome di un vettore è fisso e non può essere modificato
- Esempio:

```
int v[10], *vptr;
...
vptr = a; /* OK */
vptr += 3; /* OK */
v = vptr; /* NO, v non si può modificare */
v += 3; /* NO, v non si può modificare */
```

549

## Puntatori e stringhe

### Puntatori e stringhe

- Stringa = Vettore di `char` terminato da `'\0'`
- Differenza sostanziale nella definizione di una stringa come vettore o come puntatore
  - Esempio:

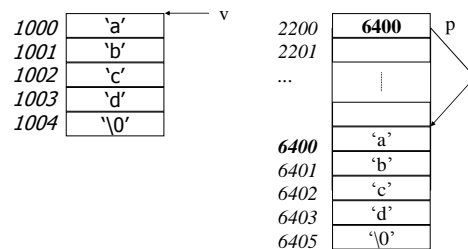
```
char v[] = "abcd";
char *p = "abcd";
```

    - `v` è un vettore:
      - I caratteri che lo compongono possono cambiare
      - Si riferisce sempre alla stessa area di memoria
    - `p` è un puntatore:
      - Si riferisce ("punta") ad una stringa costante ("abcd")
      - Si può far puntare altrove (esempio: Scrivendo `p = ...`)
      - La modifica del contenuto di `p` ha risultato NON DEFINITO
  - Allocate in "zone" di memoria diverse!!!

551

### Puntatori e stringhe: Esempio

```
char v[] = "abcd";
char *p = "abcd";
```



552

## Puntatori e stringhe (Cont.)

- Utilizzando la relazione tra puntatori e vettori è possibile sfruttare appieno la libreria delle stringhe

- Esempio:

```
char* strchr (char* s, int c);
/* Trova la prima occorrenza di c in s */

char *p, *string = "stringa di prova";
int n, i;
if ((p=strchr(string, ' ')) != NULL) {
 /* ho trovato uno spazio, p punta al carattere spazio*/
 printf("La stringa prima dello spazio e': ",);
 n = p - string;
 for (i=0; i<n; i++) putchar(string[i]);
 putchar('\n');
 printf("La stringa dopo lo spazio e': %s\n", p+1);
}
else {
 printf("Non ho trovato ' ' nella stringa %s\n", string);
}
```

553

## Allocazione dinamica della memoria

### Allocazione statica

- Il C permette di allocare esclusivamente variabili in un numero predefinito in fase di definizione (**allocazione statica della memoria**):

- Variabili scalari: Una variabile alla volta
- Vettori:  $N$  variabili alla volta (con  $N$  valore costante)

555

### Allocazione statica: Esempio

```
#define MAX 1000
```

```
struct scheda {
 int codice;
 char nome[20];
 char cognome[20];
};
```

```
struct scheda vett[MAX];
```

vett è dimensionato in eccesso, in modo da soddisfare in ogni caso le esigenze del programma, anche se questo usa un numero minore di elementi. Quindi in ogni caso vett sarà composto da 1000 elementi da 42 byte ciascuno, occupando cioè 42.000 byte

556

### Memoria dinamica

- Questa limitazione può essere superata allocando esplicitamente la memoria (**allocazione dinamica della memoria**):

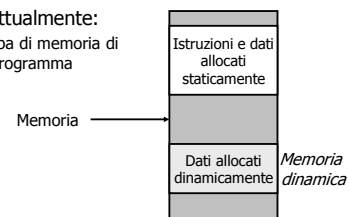
- Su richiesta:
  - Il programma è in grado di determinare, ogni volta che è lanciato, di quanta memoria ha bisogno
- Per quantità definite al tempo di esecuzione:
  - Il programma usa ad ogni istante soltanto la memoria di cui ha bisogno, provvedendo periodicamente ad allocare la quantità di memoria da usare e a liberare (deallocare) quella non più utilizzata
  - In tal modo si permette ad eventuali altri processi che lavorano in parallelo sullo stesso sistema di meglio utilizzare la memoria disponibile

557

### Memoria dinamica (Cont.)

- L'allocazione dinamica assegna memoria ad una variabile in un'area apposita, fisicamente separata da quella in cui vengono posizionate le variabili dichiarate staticamente

- Concettualmente:
  - Mappa di memoria di un programma



558

## Memoria dinamica: malloc

- La memoria viene allocata dinamicamente tramite la funzione `malloc()`

- Sintassi:

```
void* malloc(<dimensione>)
```

<dimensione>: Numero (intero) di **byte** da allocare

- Valore di ritorno:

Puntatore a cui viene assegnata la memoria allocata

- In caso di errore (esempio, memoria non disponibile), ritorna **NULL**

- **NOTA:**

Dato che viene ritornato un puntatore `void*`, è obbligatorio forzare il risultato al tipo del puntatore desiderato

559

## Memoria dinamica: malloc (Cont.)

- Esempio:

```
char *p;
p = (char*)malloc(10); // Alloca 10 caratteri a p
```

- L'allocazione dinamica permette di superare la limitazione del caso statico

- Esempio:

```
int n;
char p[n]; // Errore!!

int n;
char *p = (char*) malloc (n); // OK!!!
```

560

## malloc: Esempio

```
int *punt;
int n;
...
punt = (int *)malloc(n);
if (punt == NULL)
{
 printf ("Errore di allocazione\n");
 exit();
}
...
```

Richiede l'allocazione di una zona di memoria di n byte.

Trasforma il puntatore generico ritornato da malloc in un puntatore a int.

Verifica che l'allocazione sia avvenuta regolarmente.

561

## Settimana n.13

### Obiettivi

- Preparazione all'esame

### Contenuti

- Svolgimento temi d'esame

562

## Settimana n.14

### Obiettivi

- Preparazione all'esame

### Contenuti

- Svolgimento temi d'esame

563