Funzioni

Parametri "by reference"

Parametri "by reference"

- Introduzione
- Operatori & e *
- Passaggio "by reference"
- Passaggio di vettori
- Esercizio "`strcpy`"

Parametri "by reference"

Introduzione

Passaggio dei parametri

- Il linguaggio C prevede il passaggio di parametri "by value"
  - Il chiamato non può modificare le variabili del chiamante
  - Il parametro formale viene inizializzato con una copia del valore del parametro attuale

Problemi

- Il passaggio "by value" risulta inefficiente qualora le quantità di dati da passare fossero notevoli
  - Nel caso del passaggio di vettori o matrici, il linguaggio C non permette il passaggio "by value", ma copia solamente l'indirizzo di partenza
  - Esempio: `strcmp`
- Talvolta è necessario o utile poter modificare il valore di una variabile nel chiamante
  - Occorre adottare un meccanismo per permettere tale modifica
  - Esempio: `scanf`

Soluzione

- La soluzione ad entrambi i problemi è la stessa:
  - Nel passaggio di vettori, ciò che viene passato è solamente l'indirizzo
  - Per permettere di modificare una variabile, se ne passa l'indirizzo, in modo che il chiamato possa modificare direttamente il suo contenuto in memoria
- Viene detto passaggio "by reference" dei parametri
  - Definizione impropria, in quanto gli indirizzi sono, a loro volta, passati "by value"

Parametri "by reference"

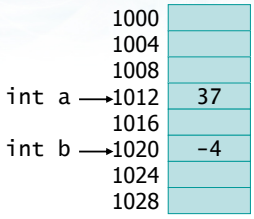
Operatori & e *

Operatori sugli indirizzi

- Per gestire il passaggio "by reference" dei parametri occorre
  - Conoscere l'indirizzo di memoria di una variabile
    - Operatore &
  - Accedere al contenuto di una variabile di cui si conosce l'indirizzo ma non il nome
    - Operatore *
- Prime nozioni della aritmetica degli indirizzi, che verrà approfondita in Unità successive

Operatore &

- L'operatore **indirizzo-di** restituisce l'indirizzo di memoria della variabile a cui viene applicato
  - &a è l'indirizzo 1012
  - &b è l'indirizzo 1020



Osservazioni

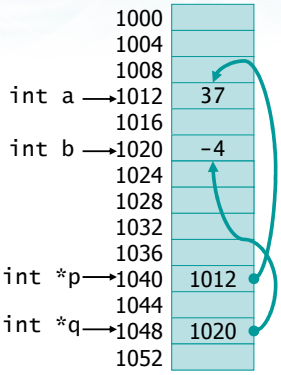
- L'indirizzo di una variabile viene deciso dal compilatore
- L'operatore & si può applicare solo a variabili singole, non ad espressioni
  - Non ha senso &(a+b)
  - Non ha senso &(3)
- Conoscere l'indirizzo di una variabile permette di leggerne o modificarne il valore senza conoscerne il nome

Variabili "puntatore"

- Per memorizzare gli indirizzi di memoria, occorre definire opportune variabili di tipo "indirizzo di..."
- Nel linguaggio C si chiamano **puntatori**
- Un puntatore si definisce con il simbolo *
  - `int *p ; /* puntatore ad un valore intero */`
  - `float *q ; /* puntatore ad un valore reale */`

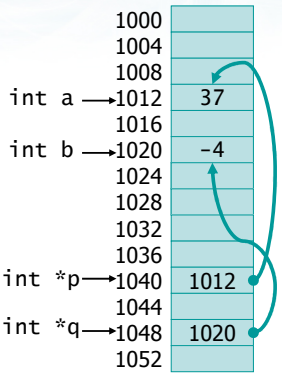
Esempio

```
int main(void)
{
  int a, b ;
  int *p, *q ;

  a = 37 ;
  b = -4 ;

  p = &a ;
     /* p "punta a" a */

  q = &b ;
     /* q "punta a" b */

}
```



Operatore *

- L'operatore di **accesso indiretto** permette di accedere, in lettura o scrittura, al valore di una variabile di cui si conosce l'indirizzo
  - `*p` equivale ad a
  - `*q` equivale a b
  - `*p = 0 ;`
  - `if( *q > 0 ) ...`



Costrutti frequenti

Costrutto	Significato
`int x ;`	x è una variabile intera
`int *p ;`	p è un puntatore a variabili intere
`p = &x ;`	p punta ad x
`*p = 0 ;`	Azzera la variabile puntata da p (cioè x)
`b = *p ;`	Leggi il contenuto della variabile puntata da p e copialo in b

Parametri "by reference"

Passaggio "by reference"

Passaggio "by reference"

- Obiettivo: passare ad una funzione una variabile, in modo tale che la funzione la possa modificare
- Soluzione:
  - Definire un parametro attuale di tipo puntatore
  - Al momento della chiamata, passare l'indirizzo della variabile (anziché il suo valore)
  - All'interno del corpo della funzione, fare sempre accesso indiretto alla variabile di cui è noto l'indirizzo

Esempio: "Azzera"

- Scrivere una funzione `azzera`, che riceve un parametro di tipo intero (by reference) e che azzera il valore di tale parametro

Soluzione

```
void azzera( int *v ) ;
```

```
int main( void )
{
  int x ;
  ...
  azzera(&x) ;
  ...
}
```

```
void azzera( int *v )
{
  *v = 0 ;
}
```

Esempio: "Scambia"

- Scrivere una funzione `scambia`, che riceve due parametri di tipo intero (by reference) e che scambia tra di loro i valori in essi contenuti

Soluzione

```
void scambia( int *p, int *q ) ;
```

```
int main( void )
{
  int a,b ;
  ...
  scambia(&a, &b) ;
  ...
}
```

```
void scambia( int *p, int *q )
{
  int t ;
  t = *p ;
  *p = *q ;
  *q = t ;
}
```


Osservazione

- Il meccanismo di passaggio by reference spiega (finalmente!) il motivo per cui nella funzione `scanf` è necessario specificare il carattere & nelle variabili lette
- Le variabili vengono passate by reference alla funzione `scanf`, in modo che questa possa scrivervi dentro il valore immesso dall'utente

Parametri "by reference"

Passaggio di vettori

Passaggio di vettori e matrici

- Nel linguaggio C, il nome di un array (vettore o matrice) è automaticamente sinonimo del puntatore al suo primo elemento

```
int main(void)
{
  int v[10] ;
  int *p ;

  p = & v[0] ;

}
```

p e v sono del tutto equivalenti

Conseguenze

- Quando il parametro di una funzione è di tipo array (vettore o matrice)
  - L'array viene passato direttamente "by reference"
  - Non è necessario l'operatore & per determinare l'indirizzo
    - È sufficiente il nome del vettore
  - Non è necessario l'operatore * per accedere al contenuto
    - È sufficiente l'operatore di indicizzazione []
  - Non è possibile, neppure volendolo, passare un array "by value"

Esercizio "Duplicati"

- Scrivere una funzione che, ricevendo due parametri
  - Un vettore di `double`
  - Un intero che indica l'occupazione effettiva di tale vettore

  possa determinare se vi siano valori duplicati in tale vettore
- La funzione ritornerà un intero pari a 1 nel caso in cui vi siano duplicati, pari a 0 nel caso in cui non ve ne siano

Soluzione (1/3)

```
int duplicati(double v[], int N) ;
/*
Riceve in ingresso il vettore v[] di double
che contiene N elementi (da v[0] a v[N-1])

Restituisce 0 se in v[] non vi sono duplicati
Restituisce 1 se in v[] vi sono duplicati

Il vettore v[] non viene modificato
*/
```


Soluzione (2/3)

```
int duplicati(double v[], int N)
{
  int i, j ;

  for(i=0; i<N; i++)
  {
    for(j=i+1; j<N; j++)
    {
      if(v[i]==v[j])
        return 1 ;
    }
  }
  return 0 ;
}
```


Soluzione (3/3)

```
int main(void)
{
  const int MAX = 100 ;
  double dati[MAX] ;
  int Ndati ;
  int dupl ;

  ...
  dupl = duplicati(dati, Ndati) ;
  ...
}
```


Errore frequente

- Nel passaggio di un vettore occorre indicarne solo il nome

```
dupl = duplicati(dati, Ndati) ;
```

~~`dupl = duplicati(dati[], Ndati) ;`~~

~~`dupl = duplicati(dati[MAX], Ndati) ;`~~

~~`dupl = duplicati(dati[Ndati], Ndati) ;`~~

~~`dupl = duplicati(&dati, Ndati) ;`~~

Osservazione

- Nel caso dei vettori, il linguaggio C permette solamente il passaggio by reference
  - Ciò significa che il chiamato ha la possibilità di modificare il contenuto del vettore
- Non è detto che il chiamato effettivamente ne modifichi il contenuto
  - La funzione duplicati analizza il vettore senza modificarlo
  - Esplicitarlo sempre nei commenti di documentazione

Parametri "by reference"

Esercizio "strcpy"

Esercizio "strcpy"

- Si implementi, sotto forma di funzione, la ben nota funzione di libreria `strcpy` per la copia di due stringhe

Soluzione (1/2)

```
void strcpy(char *dst, char *src) ;
/*
Copia il contenuto della stringa src
nella stringa dst

Assume che src sia 0-terminata, e restituisce
dst in forma 0-terminata.

Assume che nella stringa dst vi sia spazio
sufficiente per la copia.

La stringa src non viene modificata.
*/
```


Soluzione (2/2)

```
void strcpy(char *dst, char *src)
{
  int i ;

  for(i=0; src[i]!=0; i++)
  {
    dst[i] = src[i] ;
  }
  dst[i] = 0 ;
}
```


Osservazione

- La funzione può essere dichiarata in due modi:
  - `void strcpy(char *dst, char *src)`
  - `void strcpy(char dst[], char src[])`
- Sono forme assolutamente equivalenti
- Tutte le funzioni di libreria che lavorano sulle stringhe accettano dei parametri di tipo `char *`