Le funzioni

Sottoprogrammi e funzioni

- Un programma può essere decomposto in un insieme di sottoprogrammi che risolvono un sottoproblema
- Un sottoprogramma può essere visto come un insieme di istruzioni che realizzano una specifica attività (es. lettura, visualizzazione, elaborazione, ecc)
- I sottoprogrammi permettono di replicare porzioni di codice ottenendo un approccio modulare
- I sottoprogrammi permettono di ottenere codice riusabile aumentando la produttività e la qualità del software
- I sottoprogrammmi in C sono detti *funzioni:* dati più valori in ingresso restituiscono un valore al programma chiamante

Tipi di funzioni

- In C si possono realizzare due tipi di funzioni:
  - Funzioni che restituiscono un valore (funzioni di tipo matematico)
  - Funzioni che non restituiscono alcun valore (funzioni di tipo procedurali).
- Il primo tipo viene usato quando la funzione deve produrre un risultato sulla base del valore dei parametri di ingresso
  - Es. `float media(float x, float y)`
- Il secondo tipo viene usato quando la funzione deve realizzare una qualche operazione che non richiede la restituzione di un valore
  - Es.
```
void stampa ( int Vettore[N])
  { int i;
    for(i=0;i<N;i++) printf("%d",Vettore[i]);
  }
```
- Il tipo `void` indica il fatto che non viene restituito alcun valore.

Dichiarazione di una Funzione

- Una funzione viene dichiarata mediante il seguente prototipo:

```
SpecificatoreTipo NomeFunzione(ElencoParametri);
```

- La dichiarazione di una funzione ne permette l'utilizzo nel programma, ma presuppone che da un'altra parte ne esista la definizione
- In C è possibile dichiarare una funzione e definirla:
  - nello stesso file
  - in file diversi

Definizione di una Funzione

La forma generale di funzione è la seguente:

```
SpecificatoreTipo NomeFunzione(ElencoParametri)
{
  CorpoDellaFunzione;
}
```

- Lo SpecificareTipo specifica il tipo dei dati restituito dalla funzione.
- L'ElencoParametri è un elenco di nomi di variabili separate da virgole e preceduti dai rispettivi tipi.

  Es. `int somma(int a, int b)`

- Tali parametri noti come *parametri formali* ricevono il valore degli argomenti passati (noti come *parametri attuali*) nel momento dell'invocazione della funzione.
- I parametri formali sono viste come delle variabili locali della funzione.

Esempio: definizione della funzione somma

```
#include <stdio.h>
int somma (int x, int y);

main(){
 int a,b,som;
 scanf("%d %d",&a,&b);
 som=somma(a,b);
 printf("%d",som);
}
int somma (int x, int y)
{ int s;
  s=x+y;
  return s;
}
```

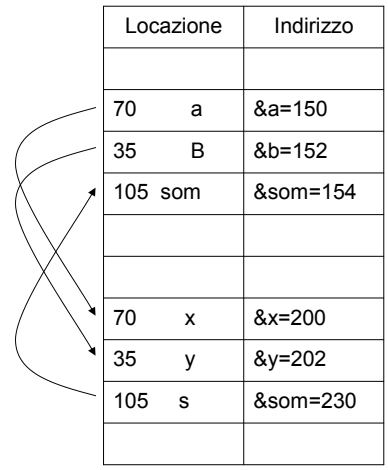
- La funzione somma restituisce un valore intero e ha due parametri formali (x e y) entrambi di tipo int.
  - Il valore viene restituito attraverso l'istruzione **return**.
  - La funzione somma viene invocata passando i parametri attuali **a** e **b**.

Modello della memoria durante l'attivazione di una funzione

```
int somma (int x, int y)
{ int s;
  s=x+y;
  return s;
}
```

In seguito all'attivazione della funzione somma:

1. Vengono riservate le locazioni di indirizzo &x, &y e &s.
2. I valori dei parametri attuali vengono copiati nei parametri formali.
3. Viene calcolato il valore della variabile s.
4. Viene eseguita l'istruzione return e pertanto viene copiato il valore di som nella locazione relativa al parametro attuale som.
5. Le locazioni relative ai parametri formali e alle variabili locali vengono rilasciate.



Locazione	Indirizzo
70 a	&a=150
35 B	&b=152
105 som	&som=154
70 x	&x=200
35 y	&y=202
105 s	&som=230

Funzioni senza parametri

- Una funzione può anche essere senza parametri, in tal caso l'elenco dei parametri sarà vuoto.
- Tuttavia, anche se non vi è alcun parametro, è richiesta la presenza delle parentesi.

Es.

```
void menu()
{ printf("Scegli una delle seguenti opzioni\n");
  printf("i) Inserimento\n");
  printf("c) Cancellazione\n");
  printf("f) Fine\n");
}
```

Campo di azione di una dichiarazione

- Una dichiarazione introduce un nome in un ambito di definizione detto scope
- Un nome è visibile solo in una specifica parte del programma
- Un nome dichiarato all'interno in una funzione è visibile (ed utilizzabile) solo all'interno di quella funzione.
- Un nome definito al di fuori delle funzioni (nome globale) è visibile da tutte le funzione

Chiamata di una funzione all'interno di un'altra funzione

- È possibile chiamare una funzione all'interno di un'altra funzione
- È necessario che la funzione chiamata sia dichiarata o definita prima della funzione chiamante

```
#include <stdio.h>
float f2(int x2, float y2);

float f1(int x1, float y1)
{ float z;
   z=f1(x1,y1);
   return z;
}
main()
{ int a;
 float b,c;
 scanf("%d %f",&a,&b);
 c=f1(a,b);
}

float f2(int x2, float y2) { return x2+3*Y2; }
```

Esempio (errato) di scambio dei valori di variabili mediante la funzione scambia

```
#include <stdio.h>
void scambia (int x, int y);
main ()
{ int a, b;
  scanf("%d %d",&a, &b);
  printf("%d %d\n",a,b);
  scambia(a,b);
  printf("%d %d\n",a,b);
}
void scambia (int x, int y)
{ int temp;
  temp=x;
  x=y;
  y=temp;
}
```

Locazione		Indirizzo
70	a	&a=160
35	b	&b=162

Non appena il programma viene mandato in esecuzione:

-Vengono associate le locazioni di indirizzo &a, &b alle variabili a e b;

-In seguito alla scanf vengono copiati i valori letti dalla tastiera nelle locazioni &a e &b

-Viene chiamata la funzione scambia

Esempio (errato) di scambio dei valori di variabili mediante la funzione scambia

La chiamata della funzione scambia fa sì che:

-vengano riservate le locazioni di indirizzo &x, &y e &temp per i parametri formali x, y e per la variabile locale temp;

-vengano copiati i valori dei parametri attuali **a** e **b**, di indirizzo &a!=&x e &b!=&y, nei parametri formali **x** e **y**;

Locazione		Indirizzo
70	a	&a=160
35	b	&b=162
70		&x=200
35		&y=202
		&temp=230

Esempio (errato) di scambio dei valori di variabili mediante la funzione scambia

La chiamata della funzione scambia fa sì che:

- vengano riservate le locazioni di indirizzo &x, &y e &temp per i parametri formali x, y e per la variabile locale temp;
- vengano copiati i valori dei parametri attuali **a** e **b**, di indirizzo &a!=&x e &b!=&y, nei parametri formali **x** e **y**;
- vengono scambiati i valori di x e y

  i_1: **temp=x;**

  i_2: **x=y;**

  i_3: **y=x;**

Locazione		Indirizzo
70	a	&a=160
35	b	&b=162
~~70~~ **35**		&x=200
~~35~~ i_2	i_1 **70**	&y=202
70	i_3	&temp=230

Esempio (errato) di scambio dei valori di variabili mediante la funzione scambia

- Al termine dell'esecuzione della funzione, i parametri formali e le variabili locali vengono riliasciate ovvero l'associazione variabile-indirizzo viene eliminata e il contenuto delle variabili viene perso
- Il valore dei parametri attuali non è stato modificato.

Locazione		Indirizzo
70	a	&a=160
35	b	&b=162
35		&x=200
70		&y=202
70		&temp=230

Passaggio dei parametri di una funzione

In C il passaggio dei parametri nel momento in cui una funzione viene chiamata può avvenire in due modi:

- ***Passaggio per valore***, in cui il valore del parametro attuale viene copiato nel parametro formale
  - In tutti gli esempi visti fino a questo momento è stato usato sempre il passaggio per valore;
- ***Passaggio per indirizzo***, in cui viene copiato l'indirizzo del parametro attuale nel parametro formale
  - Il parametro formale deve essere dichiarato come puntatore allo stesso tipo del parametro attuale

Passaggio per valore

- Questa modalità può essere utilizzata quando non è necessario modificare il valore del parametro attuale, ad esempio se la funzione deve calcolare un risultato o deve visualizzare uno o più valori.

Es.

```
float media(int x, int y)
{ return x+y; }

void stampa_intero (int x)
{ printf("%d",x);}
```

Passaggio dei parametri per indirizzo

- Per permettere la modifica del valore del parametro attuale ad opera di una funzione,
  - non possiamo passare direttamente il valore del parametro (passaggio per valore),
  - è necessario passare il valore dell'indirizzo del parametro attuale (passaggio per indirizzo).

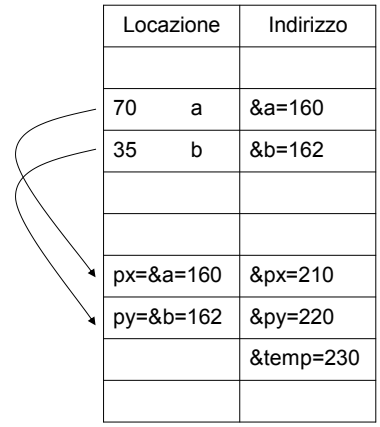
- Il passaggio per indirizzo viene realizzato in C passando il puntatore al parametro attuale.

Passaggio dei parametri per indirizzo

```
#include <stdio.h>

void scambia (int *px, int *py);

main ()
{ int a, b;
  scanf("%d %d",&a, &b);
  printf("%d %d\n",a,b);
  scambia(&a,&b);
  printf("%d %d\n",a,b);
}

void scambia (int *px, int *py)
{ int temp;

/* In temp viene copiato il contenuto della locazione di indirizzo px     */
        temp=*px;
 /*  Nella locazione di indirizzo px viene copiato il contenuto della locazione  py   */
        *px=*py;
/*  Nella locazione di indirizzo py viene copiato il valore di temp */
   *py=temp;
}
```

Chiamata per indirizzo della funzione scambia

- La chiamata della funzione scambia fa sì che:
- vengano riservate le locazioni di indirizzo &px, &py e &temp per i parametri formali px, py e per la variabile locale temp;
- vengano copiati i valori degli indirizzi dei parametri attuali &**a** e &**b**, nei parametri formali **px** e **py**;



Chiamata per indirizzo della funzione scambia

- La chiamata della funzione scambia fa sì che:
- vengano riservate le locazioni di indirizzo &px, &py e &temp per i parametri formali px, py e per la variabile locale temp;
- vengano copiati i valori degli indirizzi dei parametri attuali &**a** e &**b**, nei parametri formali **px** e **py**;
- Venga realizzato lo scambio tra a e b:
  1. Copia in temp del contenuto della locazione di indirizzo px=&a

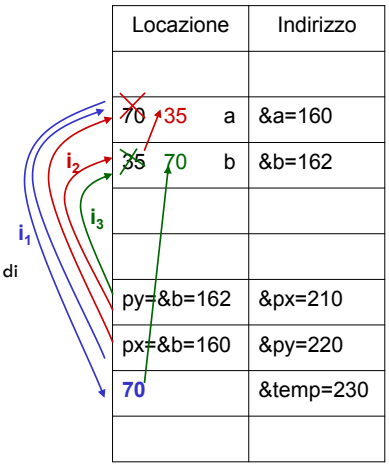
  i_1: `temp=*px;` `/* =*(&a)=a */`

  1. Copia nella locazione di indirizzo px del contenuto della locazione di indirizzo py

  i_2: *px=*py;

  1. Copia nella locazione di indirizzo py del contenuto della variabile temp

  i_3: *py=temp;



Chiamata per indirizzo della funzione scambia

- Al termine dell'esecuzione della funzione, i parametri formali e le variabili locali vengono riliasciate ovvero l'associazione variabile-indirizzo viene eliminata e il contenuto delle variabili viene perso
- Il valore dei parametri attuali è stato modificato.

Locazione	Indirizzo
35 a	&a=160
70 b	&b=162
py=&b=162	&px=210
px=&b=160	&py=220
70	&temp=230

Esempio: lettura e visualizzazione degli elementi di un vettore (1)

```
#include <stdio.h>

void leggi(int *px)
{ /* in px viene copiato l'indirizzo della variabile da leggere*/
  printf("Inserisci un numero: ");
  scanf("%d",px);
 /* nella scanf uso direttamente px e non &px
    poichè px e' gia' l'indirizzo della variabile da leggere */
}

void stampa( int x)
{  printf("%d\n",x);}

main()
{ int V[5],i;
/* passaggio per indirizzo per modificare il valore di V[i] */
 for(i=0;i<5;i++)
    leggi(&V[i]);
/* passaggio per valore poiche' non si altera il valore di V[i] */
 for(i=0;i<5;i++)
    stampa(V[i]);
}
```

Esempio: lettura e visualizzazione degli elementi di un vettore (2)

```
#include <stdio.h>

void leggi(int V[10])
{ int i;
  for(i=0;i<10;i++)
  { printf("Inserisci un numero: ");
    scanf("%d",&V[i]); }
}

void stampa( int V[10])
{int i;
  for(i=0;i<10;i++) printf("%d\n",V[i]);
}

main()
{ int A[5],i;
   /* Nel caso di un vettore, usare come parametro attuale il
   nome del vettore equivale a passare l'indirizzo del 1°
   elemento */
   leggi(A);
   stampa(A);
}
```

Vettore come parametro di una funzione

- Nel caso in cui bisogna passare come parametro di una funzione un vettore, si possono usare indifferentemente le seguenti notazioni:
  1. `tipo_elemento V[dimensione_vettore]`
  2. `tipo_elemento V[ ]`
  3. `tipo_elemento *V`

- La ragione di questa equivalenza e' data dal fatto che, come detto in precedenza, il nome del vettore rappresenta il puntatore al primo elemento del vettore e che, al momento della chiamata della funzione, nel parametro formale viene copiato solo il valore dell'indirizzo del primo elemento e non viene fatto alcun controllo sulla dimensione del vettore.

Lettura e visualizzazione degli elementi di un vettore (3)

```
#include <stdio.h>

void leggi(int V[10])
{ int i;
  for(i=0;i<10;i++)
  { printf("Inserisci un numero: ");
    scanf("%d",&V[i]); }
}

void stampa( int *V)
{int i;
  for(i=0;i<10;i++) printf("%d\n",V[i]);
}

main()
{ int A[5],i;
   /* Nel caso di un vettore, usare come parametro attuale il
   nome del vettore equivale a passare l'indirizzo del 1°
   elemento */
   leggi(A);
   stampa(A);
}
```