Funzioni in C

Definizione di funzione

```
<tipo> <nome_funzione> (<lista_parametri>)
{
   <definizioni di variabili>
   <istruzioni>
}
```

- `<tipo>` è il tipo di dato restituito dalla funzione:
  - può essere omesso, nel qual caso il compilatore assume che sia `int`.
  - è considerata buona norma specificare esplicitamente il tipo di dato della funzione (al limite con l'uso del tipo di dato `void`).

Definizione di funzione: continua

- `<nome_funzione>` è un identificatore.
- `<lista_parametri>` è una lista di definizione del tipo `<tipo>` `<identificatore>` separate da virgole: i **parametri formali**. La lista può essere vuota.

```
int power(int m, double n)
{
...
}
```

```
int main(void)
{
...
}
```

- le variabili definite in `<definizioni di variabile>` sono dette **variabili locali** della funzione. Ad esse è possibile riferirsi solo nel corpo della funzione.

Chiamata di funzione

Le funzioni vengono utilizzate scrivendo il loro nome seguito da un opportuno elenco di **parametri attuali** fra parentesi tonde.

<nome_funzione> (<lista_parametri>)

L'associazione fra parametri formali e parametri attuali è posizionale.

Esecuzione di una funzione

Prima di eseguire le istruzioni di una funzione:

1. L'elaboratore riserva memoria per ogni parametro formale ed ogni variabile locale della funzione.
2. Le espressioni corrispondenti ai parametri attuali vengono valutate.
3. Se necessario il tipo di tali valori viene convertito al tipo dei parametri formali.
4. I valori ottenuti dalla valutazione dei parametri attuali vengono assegnati ai corrispondenti parametri formali.

Per questi motivi diremo che il passaggio di paramteri avviene **per valore**.

Chiamata di funzione: esempio

```
#include <stdio.h> void swap(int a, int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
    printf("Sono in swap a=%d b=%d\n",a,b);
}
int main(void)
{
    int x, y;
    x=2;
    y=9;
    printf("x=%d y=%d\n",x,y);
    swap(x,y);
    printf("x=%d y=%d\n",x,y);
    return 0;
}
```

L'istruzione `return`

L'istruzione `return` è il meccanismo che consente durante l'esecuzione di sostituire la chiamata della funzione con un valore.

return <espressione>;

- Può essere seguita da una qualunque espressione. Per ragioni di chiarezza l'espressione viene in genere racchiusa fra parentesi tonde, che però sono opzionali.
- Se necessario l'espressione viene convertita nel tipo del valore ritornato dalla funzione.

L’istruzione `return`

- L’espressione può essere omessa, in questo caso il chiamante non riceve nessun valore di ritorno.
- Un altro caso in cui il controllo dell’esecuzione ritorna al chiamante senza che gli venga passato alcun valore di ritorno è quello in cui l’esecuzione della funzione termina per il raggiungimento della fine del corpo.

Funzione quadrato

```
#include <stdio.h>
int quad(int x)
{
    int square;
    square = x * x;
    return square;
}
int main(void)
{
    int x, y;
    for(x = 2 ; x < 4 ; x = x + 1)
    {
        y = quad(x);
        printf("Il quadrato di %d: %d\n", x, y);
    }
    return 0;
}
```

Dichiarazioni e definizioni

- Le funzioni dovrebbero essere dichiarate (o definite) prima di essere utilizzate.
- Se una funzione `f` viene usata prima di essere dichiarata il compilatore assume che il suo tipo sia `int f()`. Quindi se in seguito viene definita in modo diverso il compilatore lo segnala (warning).
- Le **dichiarazioni o prototipi di funzione** possono essere usati per informare il compilatore sul numero e il tipo dei parametri di una funzione prima della sua definizione.
- La dichiarazione di una funzione ha la forma <tipo> <nome_fun>(<lista_tipo_parametri>); dove <lista_tipo_parametri> è l'elenco dei tipi dei parametri separati da virgole. Gli identificatori sono opzionali e non influiscono sul prototipo.

Dichiarazioni e definizioni: esempio

```
#include <stdio.h>
int quad(int);
int main(void)
{
. . .
   y = quad(x);
. . .
}
int quad(int input)
{
. . .
}
```

Dichiarazioni e definizioni: esempio

```
#include <stdio.h>
void tabella(int);
int main(void)
{
    tabella(2);
    return 0;
}
void tabella(int n)
{
    int i, j;
    for (i = 1; i <= n; i = i + 1)
        for (j = 1; j <= n; j = j + 1)
            printf("%d\t", i*j);
}
```

Fattoriale

```
#include <stdio.h>
int fattoriale (int n)
{
    int prod;
    prod=1;
    for(;n>1;n = n - 1)
        prod = prod * n;
    return prod;
}
int main(void)
{
    int n;
    printf("Inserisci un numero: ");
    scanf("%d",&n);
    printf("Fattoriale %d\n",fattoriale(n));
    return 0;
}
```

Argomenti di Funzione ed indirizzi

Poichè il C passa alle funzioni gli argomenti per valore, la funzione chiamata non ha un modo diretto per alterare una variabile nella funzione chiamante.

```
. . .
   swap(x,y);
. . .
void swap(int a, int b)
{
   int temp;
   temp=a;
   a=b;
   b=temp;
}
```

Argomenti di Funzione e indirizzi

- A causa della chiamata per valore, la funzione `swap` non può alterare i parametri attuali (`x, y`) nella chiamata.
- La funzione `swap` altera solamente le copie, i parametri formali, di `x` e `y`.
- Il modo per modificare una variabile nella funzione chiamante consiste nel passare alla funzione l'**indirizzo** (o **puntatore**) delle variabili che devono essere modificate.

Argomenti di Funzione ed indirizzi

- L'operatore unario di **indirizzamento** & consente di ottenere l'indirizzo di una variabile (la sua posizione in memoria centrale).
- L'operatore unario di **deriferimento** `*`, quando viene applicato ad un indirizzo, permette di accedere all'area di memoria indirizzata.
- Il tipo di dato **indirizzo (*)** è utilizzato per definire variabili che consentono di memorizzare indirizzi di celle di memoria.

```
int x;
int y;
int *pi; /* pi e' una variabile che
contiene indirizzi di variabili int */
x = 1;
pi = &x; /* pi contiene l'indirizzo di x */
y = *pi; /* y vale 1 */
*pi = 0; /* x vale 0 */
```

Argomenti di Funzione ed indirizzi

- Per consentire alla funzione swap di modificare le variabili x e y della funzione chiamante, è necessario passare alla funzione i puntatori a tali variabili: swap (&x, &y);
- Poichè l'operatore unario & produce l'indirizzo di una variabile, nella definizione della funzione swap, i parametri formali vengono dichiarati come variabili che contengono indirizzi attraverso i quali i parametri attuali vengono acceduti indirettamente.

Argomenti di Funzione ed indirizzi

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

Argomenti di Funzione ed indirizzi

- Il fatto che gli argomenti di una funzione siano dichiarati come parametri per indirizzi, consente alla funzione di accedere a variabili definite fuori falla funzione e di modificarle.
- Ad esempio, la chiamata della funzione scanf(“%d”,&x) consente modificare il contenuto della variabile x con un valore intero letto dallo standard input.

Argomenti di Funzione ed indirizzi: esempio

```
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp;
    temp=*a;
    *a=*b;
    *b=temp;
}
int main(void)
{
    int x, y;
    x=2;
    y=9;
    printf("x=%d\ty=%d\n",x,y);
    swap(&x,&y);
    printf("x=%d\ty=%d\n",x,y);
    return 0;
}
```

Funzioni ricorsive (1)

- Una funzione è una funzione, nella cui definizione ricorre la stessa funzione che si sta definendo. Ad esempio

$$f(n) = \begin{cases} 1 & \text{se } n = 0 \\ n * f(n-1) & \text{altrimenti} \end{cases}$$

- L'apparente circolarità della definizione è superata dai seguenti fatti:
  - Per calcolare f si deve calcolare f per un termine più "semplice" (nell'esempio $n-1$).
  - Si conosce il valore della funzione per gli elementi base (nell'esempio 0).
- Le funzioni ricorsive si compongono di due parti
  - base della ricorsione (nell'esempio $f(0) = 1$)
  - passo della ricorsione (nell'esempio $f(n) = n * f(n-1)$)

Funzioni ricorsive

- Il C consente l'utilizzo della ricorsione per definire le funzioni.
- Una chiamata ricorsiva si ottiene utilizzando una chiamata della funzione nel suo corpo.

```
void f(int a)
{
   . . .
   f(a-1);
}
```

Fattoriale ricorsivo

```
#include <stdio.h>
int fatt(int n)
{
   if (n <= 1)
      return 1; /* base */
   else
      return (n * fatt(n - 1)); /* passo */
}
int main(void)
{
   printf("Il fattoriale di 3: %d\n",fatt(3));
   return 0;
}
```

Fattoriale ricorsivo

```
#include <stdio.h>
void fatt(int n, int *r)
{
    if (n <= 1)
        *r = 1;
    else
    {
        fatt(n - 1, r);
        *r = n * (*r);
    }
}
int main(void)
{
    int x;
    fatt(3, &x);
    printf("Il fattoriale di 3: %d\n",x);
    return 0;
}
```

Esercizio: funzione di Fibonacci

Tasso di riproduzione dei conigli:

$$fb(g) \begin{cases} 0 & \text{se } g = 0 \\ 1 & \text{se } g = 1 \\ fb(g-1) + fb(g-2) & \text{altrimenti} \end{cases}$$

Scrivere la funzione ricorsiva per la funzione di Fibonacci.