Introduzione alla programmazione in C

Testi Consigliati:

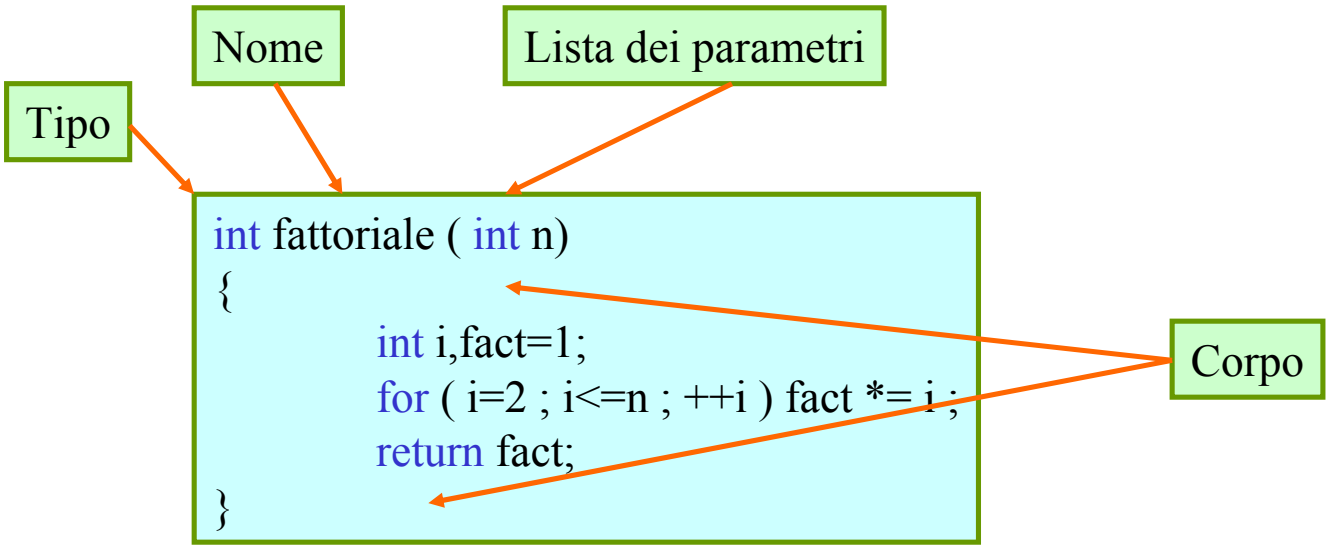
- A. Kelley & I. Pohl "*C didattica e programmazione*"
- B.W. Kernighan & D. M. Ritchie "*Linguaggio C*"

Materiale disponibile sul sito

http://www.fisica.unile.it/~martello/corsi/dottorato/TecnoOO/TecnoOO_03-04/index.html

Funzioni: *fattoriale*

Esempio: una funzione che calcola il fattoriale di un numero.



La definizione di una funzione parte con il tipo della funzione che corrisponde al tipo di valore che la funzione deve ritornare al flusso principale del programma una volta terminata la sua esecuzione. Se non e' necessario ritornare alcun valore allora il tipo e' ***void***.

Funzioni: *void*

Esempio di una funzione che non ritorna alcun valore.

```
void stampa(int n)
{
        int i;
        for (i=0; i<n ; i++)
            std::cout << "Ciao Mondo! " <<std::endl;
}
```

Questa funzione stampa n volte la frase **`Ciao Mondo!`**

```
…
int f;
f=fattoriale(4);
stampa(f);
…
```

Questo pezzo di codice chiama la funzione fattoriale per calcolare il fattoriale di 4 e pone il risultato nella variabile f. Successivamente stampa f volte la frase Ciao Mondo! utilizzando la funzione stampa.

Funzioni: *i parametri*

Esempio di funzioni con due parametri e senza parametri.

```
double somma( double a, double b)
{
        return a+b;
}
```

```
void indirizzo(void)
{
        std::cout<<Via M. Rossi, N. 123”<<std::endl;
}
```

Funzioni: *return*

```
int  fattoriale( int n)
{
      if (n<0) {
        std::cout<<“Il fattoriale e’ definito solo per numeri interi positivi!”<<std::endl;
        return 0;
       }
      int fact=1,i;
      for (i=2; i<=n; ++i) fact *= i;
      return fact;
}
```

Il controllo passa alla funzione fattoriale

```
…
int f;
f=fattoriale(-2);
std::cout<<“Ciao Mondo!” <<std::endl;
…
```

Viene eseguito il corpo della funzione dato che si richiede il fattoriale di un numero negativo la prima condizione e’ vera per cui viene eseguito la prima istruzione return 0;

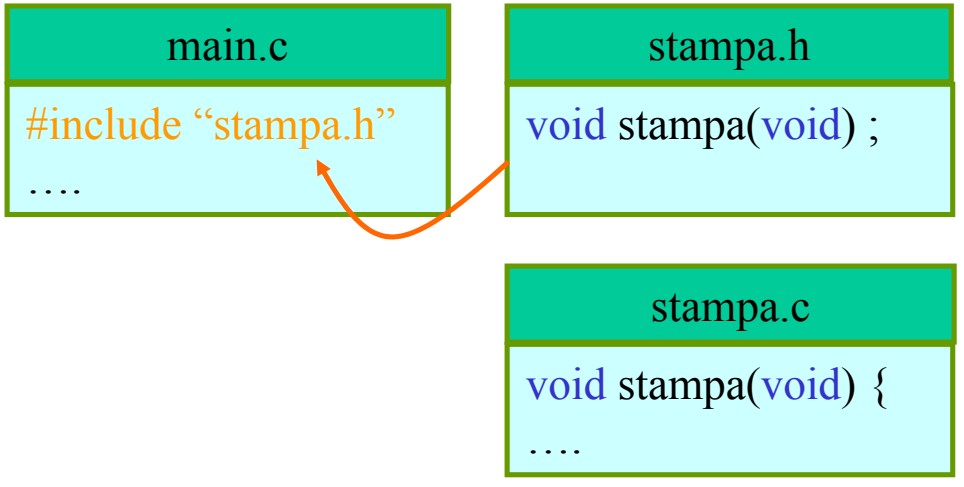
Viene eseguita la stampa di **`Ciao Mondo!`**

Funzioni: *I Prototipi*

Le funzioni devono essere dichiarate prima di venire utilizzate. Questo e' necessario perche' il compilatore deve verificare la correttezza della sintassi utilizzata al momento in cui una specifica funzione viene chiamata (verificare che il tipo ritornato sia corretto e che il numero e il tipo degli argomenti passati sia giusto).
Normalmente, quindi, si inserisce una dichiarazione della funzione (prototipo) in un file di ***header*** (file di solito con estensione .***h***) che viene incluso all'inizio del codice in cui la funzione e' usata.

Funzioni: *I Prototipi*

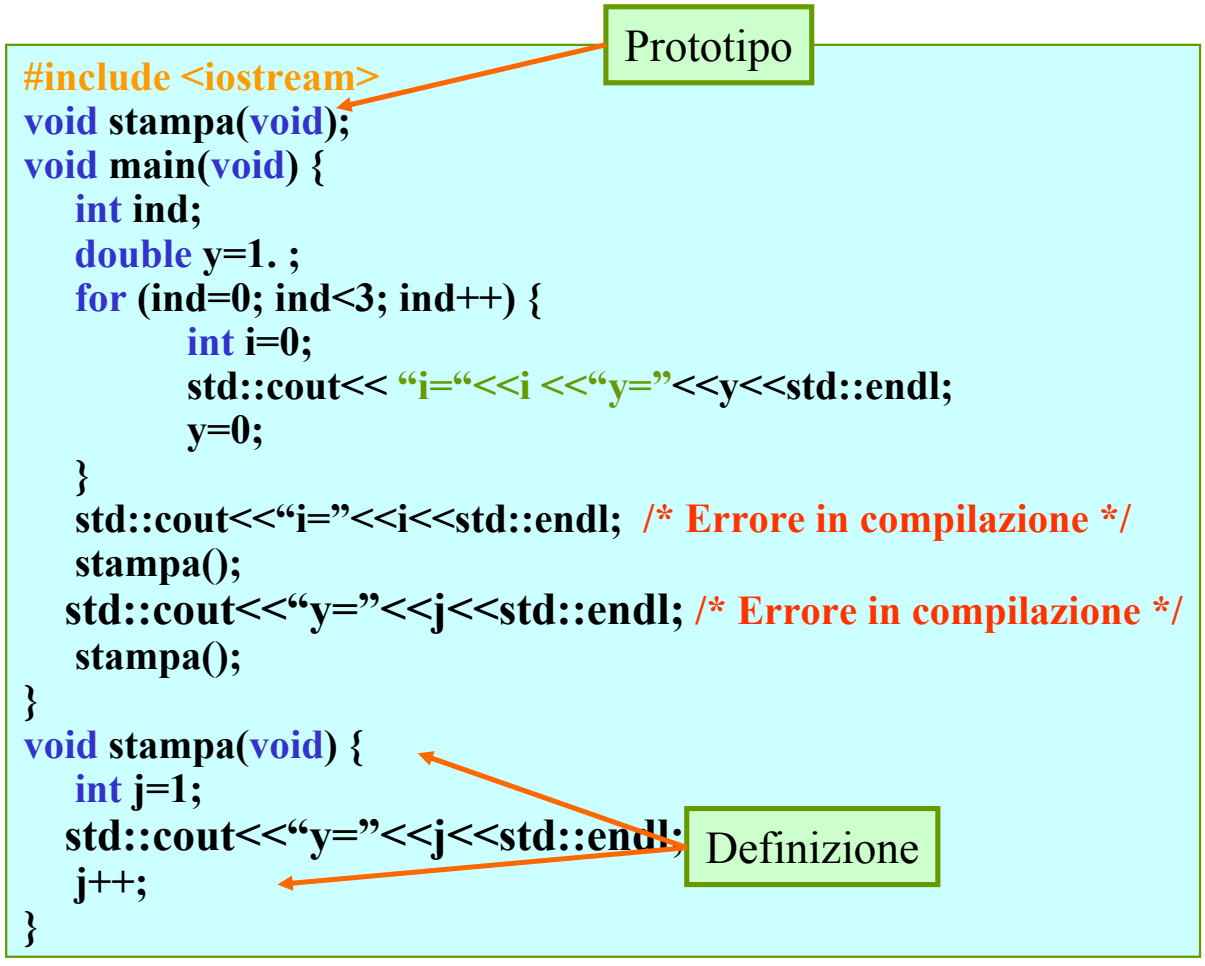
Programmi molto estesi non vengono mai scritti in un unico file, ma vengono divisi in piu' file. In un file e' contenuta la definizione di una o piu' funzioni raggruppate per tipo di operazioni che esse eseguono (es.: funziooni statistiche, funzioni che gestiscono I/O, funzioni grafiche, etc.). I prototipi di queste funzioni vengono salvati in file che hanno lo stesso nome del file contenente le definizioni delle funzioni, ma con estensione .h. Questi file vengono poi inclusi nel codice che deve utilizzare le funzioni. Spesso le definizioni delle funzioni stesse sono compilate e raccolte in librerie.



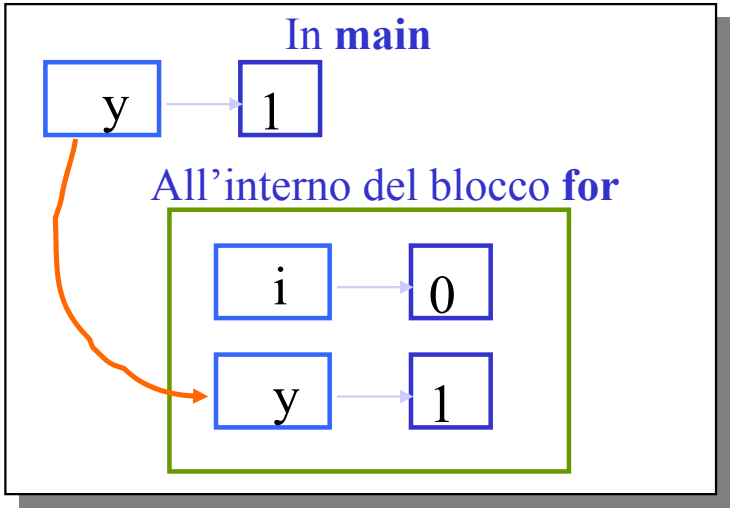
g++ –o esempio main.c stampa.c

g++ –o esempio -lmylib

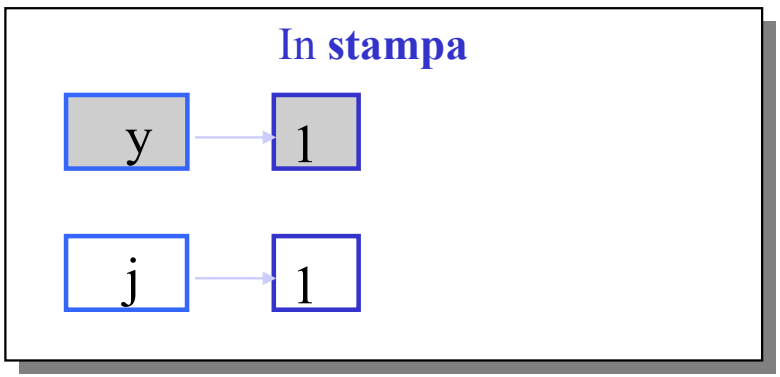
Funzioni: *Regole di scope*



Funzioni: *Regole di scope*




```
double y=1.
for (ind=0; ind<3; ind++) {
    int i=0;
    std::cout<< "i="<<i <<"y="<<y<<std::endl;
    y=0;
}
```




```
void stampa(void) {
    int j=1;
    std::cout<<"y="<<y<<std::endl;
    std::cout<<"j="<<j<<std::endl;
    j++;
}
```

Variabili: *metodi di storage*

In C quando viene definita una variabile occorre specificare il modo con cui la variabile deve essere mantenuta in memoria (***storage***). Esistono quattro modi di registrare in memoria una variabile: ***auto*** ***extern*** ***register*** ***static***
Di default, quando si definisce una variabile senza ulteriori specifiche, questa viene considerata di tipo ***auto*** (automatica).

Le variabili automatiche vengono create in memoria nel momento in cui vengono definite, restano visibili all'interno del blocco in cui sono definite, e vengono automaticamente distrutte (liberando quindi la memoria a loro riservata) nel momento in cui termina l'esecuzione del loro blocco.

```
{
        int i=1;
        std::cout<<i <<std::endl ;
}
/* sono uscito dal blocco, ora i non esiste piu' …. */
```

Variabili: *metodi di storage*

E' possibile definire variabili che siano visibili in piu' blocchi di codice. Tali variabili sono dichiarate ***extern*** (globali). Per rendere una variabile globale basta definirla al di fuori di un qualunque blocco e richiamarla all'interno del blocco in cui la si deve utilizzare usando la parola riservata ***extern*** .

main.c

```
#include <iostream>
void stampa(void);
int a=1;  /* variabile globale */
int main(void) {
      extern int a;
      std::cout<<a<<std::endl;
      stampa();
}
```

stampa.c

```
#include <iostream>
void stampa(void) {
     extern int a;
     std::cout<<a<<std::endl;
}
```

g++ –o esempio main.c stampa.c

Variabili: *metodi di storage*

Cosa succede se….

```
#include <iostream>
void stampa(void);
int main(void) {
        for (int i=0 ; i<3 ; i++)  stampa();
}
void stampa(void) {
        int a=0;
        std::cout<< " a=" << a<<std::endl;
        a++;
}
```

Variabili: *metodi di storage*

In alcune circostanze e' utile che un blocco di codice "*ricordi*" il valore di una variabile e non la crei nel momento in cui viene eseguito e la distrugga appena il blocco termina.
Per fare cio' occorre dichiarare al compilatore che la variabile e' statica (***static***).

```
#include <iostream>
void stampa(void);
int main(void) {
        for (int i=0 ; i<3 ; i++)  stampa();
}
void stampa(void) {
        static int a=0;
        std::cout<< " a= "<<a<<std:.endl;
        a++;
}
```

Variabili: *metodi di storage*

Se una variabile deve essere utilizzata con estrema frequenza in una porzione di codice e si sta scrivendo codice in cui le *performance* sono importanti allora e' possibile definirla in maniera tale che venga salvata in una memoria ad alta velocita' (registri della CPU). Lo spazio disponibile nei registri e' molto limitato, per cui occorre usarlo con estrema parsimonia.

```
#include <iostream>
void stampa(void);
int main(void) {
        register int i;
        for (i=0 ; i<3 ; i++)  stampa();
}
void stampa(void) {
        static int a=0;
        std::cout<<" a= " << a<<std::endl;
        a++;
}
```

Funzioni: *ricorsione*

Molti algoritmi matematici sono di tipo ricorsivo (una funzione che richiama se stessa, pensiamo per esempio ad una successione in cui il calcolo del termine n-esimo e' ottenuto a partire dal termine n-1).
In C e' possibile scrivere funzioni ricorsive.

$$(a+b)^2 = a^2 + 2ab + b^2$$

$$(a+b)^3 = a^3 + 3a^2b + 3ab^2 + b^3$$

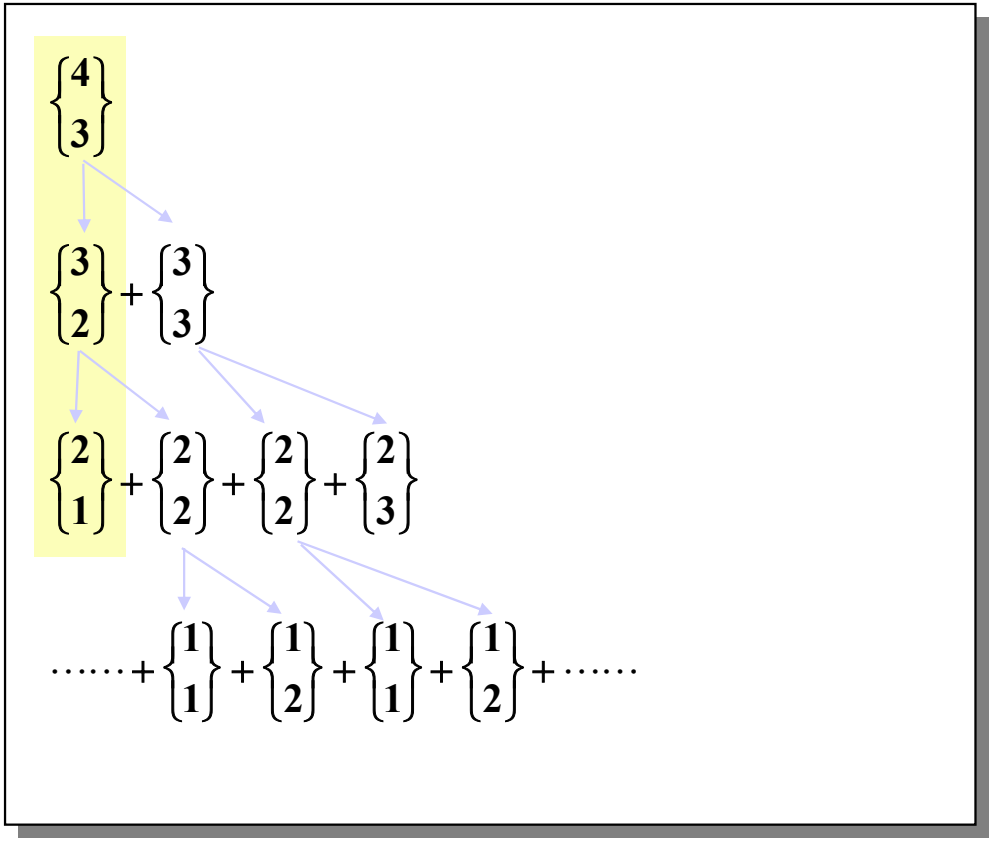
$$\left\{ {n \atop k} \right\} = \begin{cases} err & k < 1, k > n+1, n < 1 \\ 1 & k = 1, k = n+1 \\ \left\{ {n-1 \atop k-1} \right\} + \left\{ {n-1 \atop k} \right\} & altrimenti \end{cases}$$

Funzioni: *ricorsione*

```
/* Calcolo del coefficiente di posto k di un binomio
   elevato alla n*/
int bcoef (int n, int k) {
    if (k<1 || k>n+1|| n<1) {
            std::cout<<“Valori non permessi!”<<std::endl;
            return 0;
    }
    else  if (k==n+1 || k==1) return 1;
    else {
            return bcoef(n-1,k-1)+bcoef(n-1,k);
    }
}
```

Funzioni: *ricorsione*

Consideriamo il caso **n=4, k=3**



Funzioni: *ricorsione*