28. Le funzioni

Le funzioni sono parti di un programma messe a disposizione per strutturare in modo più chiaro e leggibile un programma. Si raggruppano parti del programma per evitare di duplicare parti del codice.

In un programma è sempre opportuno e conveniente strutturare il codice, raggruppandone delle sue parti in sotto-programmi autonomi, detti funzioni, che vengono eseguite in ogni punto in cui è richiesto.

L'organizzazione in funzioni ha moltissimi vantaggi: miglior strutturazione e organizzazione del codice, maggior leggibilità del codice, maggior mantenibilità del codice, riutilizzo di sotto-parti di uno stesso programma più volte, condivisioni di sotto-programmi tra programmi distinti, utilizzo di codice fatto da altri/librerie, sviluppo di un programma in parallelo, tra più autori.

In un programma C++ è possibile definire e chiamare funzioni (è possibile anche chiamare funzioni definite altrove: funzioni definite in altri file o funzioni di libreria).

28.1 Funzioni di libreria

Una funzione è un sotto-programma che può essere utilizzato ripetutamente in un programma, o in programmi diversi.
Una libreria è un insieme di funzioni precompilate (standard o fatte dal programmatore stesso in altri file).
Alcune librerie C++ sono disponibili in tutte le implementazioni e con le stesse funzioni (ad es. cmath, iostream, iomanip, …).

28.1.1 Come è strutturata una libreria

Una libreria è formata da una coppia di file:

- un file di intestazione (header) contenente le dichiarazioni dei sotto-programmi stessi
- un file oggetto contenente le funzioni compilate

Per utilizzare in un programma le funzioni in una libreria bisogna:

- includere il file di intestazione della libreria con la direttiva *#include <nomelibreria>*
- in alcuni casi, indicare al linker il file contenente le funzioni compilate della libreria (*g++ -lm prova.cc* //-lm serve per indicare che uso la libreria cmath, ma si può omettere per librerie famose come questa)
- introdurre nel programma chiamate alle funzioni della libreria

Sotto linux, nel terminale, è possibile con il comando *man <funzione della libreria>* individuare il manuale della funzione per vedere cosa essa fa e altre informazioni sui parametri. Inoltre dice come linkare la libreria.

Quando includiamo una libreria il compilatore prende dal file system un file chiamato *nomelibreria.h* (ci sarà un file *iostream.h* e *cmath.h*). Se includiamo una libreria includiamo il suffisso *.h* o *.hpp* (con le librerie standard il suffisso può venire omesso).
Come specificato poc'anzi anche al compilatore possiamo dare informazioni che useremo una libreria. Quando compiliamo possiamo mettere una flag per indicare la libreria usata al linker. Nel caso della libreria matematica la flag è *-lm* quindi si scriverà: *g++ -lm fact.cc*. Per vedere la flag delle librerie si può andare su *man <comando di una libreria>* e verrà specificata la flag per la seguente libreria.

Infine possiamo trovare le librerie su linux andando in questa cartella dal terminale:

/usr/lib/gcc/x86-64-linux-gnu/9 e troveremo alcuni file oggetto andando nella cartella */include* troveremo alcune librerie. Col comando *cat* o *less* seguito dal nome di un file ne vedremo il contenuto.

28.1.2 Come usare una libreria creata da noi

Se in una nostro file usiamo una libreria da noi creata, esso sarà decomposto in più file.
Ipotizziamo un file in cui viene calcolato il fattoriale di un numero grazie ad una nostra libreria. Il suddetto file verrà chiamato *fact_main.cc*. In un altro file, chiamato *fact.cc*, scriviamo la DEFINIZIONE della nostra funzione *fact* che calcola il fattoriale del numero.

```
#include "fact.h"

long long fact (int n){
long long fatt;
… ;
return fatt; }
```

Anche nel file .cc dove abbiamo scritto la nostra funzione includiamo *fact.h* (al contrario delle librerie precompilate le nostre librerie vanno scritte tra virgolette "" e non tra <>). Infine in un file di testo (che sarà chiamato *fact.h)* scriviamo la DICHIARAZIONE della nostra funzione.

```
long long fact (int);
```

Ora nel nostro *fact_main.cc* potremo usare la funzione *fact* senza averla mai definita, dopo aver incluso la libreria col comando *#include "fact.h"*.

In fase di compilazione il nostro file risulterà decomposto in più file quindi sarà necessario compilare più file insieme.

g++ -c fact.cc //otteniamo un file oggetto *fact.o*
g++ fact_main.cc //otteniamo un errore di compilazione perché il compilatore non trova la definizione della funzione utilizzata

g++ fact.cc fact_main.cc -o fact_main.out // in questo modo otteniamo un file eseguibile
Oppure si può compilare passando da un file oggetto
g++ -c fact.cc
g++ fact.o fact_main.cc //otteniamo un file eseguibile *a.out*

IMPORTANTE! I tre file (*fact.cc* con la definizione, *fact.h* con la dichiarazione e *fact_main.cc* con il programma e la chiamata) devono essere tutti nella stessa cartella!

28.1.3 Esempi di librerie

cmath

Libreria <*cmath*>: funzioni matematiche (da *double* a *double*, ma può essere effettuata con tutti i tipi numerici in quanto viene fatta una conversione esplicita)

fabs(x): valore assoluto di tipo float
sqrt(x): radice quadrata di x
pow(x,y): eleva x alla potenza di y
exp(x): eleva e alla potenza di x
log(x): logaritmo naturale di x
log10(x): logaritmo in base 10 di x
sin(x) e asin(x): seno e arcoseno trigonometrico
cos(x) e acos(x): coseno e arcocoseno trigonometrico

tan(x) e atan(x): tangente e arcotangente trigonometrico

Hint: *log2(x)=log(x)/log(2)* (il logaritmo in base due di x equivale al rapporto tra il logaritmo naturale di x e il logaritmo naturale di due).

cctype
Libreria <*cctype*>: funzioni di riconoscimento (da *char* a *bool*) (NON consentita all'esame)
isalnum(c): carattere alfabetico o cifra decimale
isalpha(c): carattere alfabetico
iscntrl(c): carattere di controllo
isdigit(c): cifra decimale
isgraph(c): carattere grafico, diverso da spazio
islower(c): lettera minuscola
isupper(c): lettera maiuscola
isprint(c): carattere stampabile, anche spazio
isspace(c): spazio, salto pagina, nuova riga o tab.
isxdigit(c): cifra esadecimale

La libreria <*cctype*> permette anche di effettuare delle funzioni di conversione (da *char* a *char*)
tolower(c): se *c* è una lettera maiuscola restituisce la corrispondente lettera
minuscola, altrimenti restituisce *c*
toupper(c): come sopra ma in maiuscolo

28.1.3 Esempio di uso di funzione di libreria
Se in un programma si usa ad esempio la funzione *log(i)* di *cmath* alla chiamata *log(i)*:

- il programma trasferisce il controllo dal codice di *main* al codice di *log* in *cmath*, e lo riprende al termine della funzione
- il valore di *i* viene valutato e passato in input alla funzione *log*
- *log(i)* viene valutata al valore restituito dalla computazione della funzione *log* con il valore *i* in input

28.2 Definizione, dichiarazione e chiamata di funzioni

Definizione.

Sintassi: *tipo id(tipo1 id1, ... , tipoN idN) {...}*
Esempio: *double pow(double x, double y) {...}*
dove:

- *id1,...,idN* sono i parametri formali (sempre presenti) della funzione (gli argomenti della funzione) e devono essere posti tra parentesi tonde precedute dal loro tipo (sono i parametri presi in input dalla funzione, la quale può usare anche altre variabili come le variabili globali o le variabili definite al suo interno). Devo specificare il tipo del parametro per ogni identificativo.
- *tipo* rappresenta il tipo del valore restituito dalla funzione in output
- *id* rappresenta il nome della funzione

! Una funzione può anche essere senza argomenti. In questo caso ritorna un valore sempre costante che non dipende da nessuna variabile.

Dichiarazione.

Sintassi: *tipo id(tipo1 [id1], ... , tipoN [idN]);*
Esempio: *double pow(double, double e);*

Serve per "richiamare" una definizione fatta altrove, e consentirne l'uso! Non sostituisce la definizione di una funzione. Usando la dichiarazione si avvisa il programma che verrà definita una funzione più avanti nel programma. Noi possiamo fare una dichiarazione fuori dal *main*, richiamare la nostra funzione e successivamente fuori dal *main* definire la funzione

Nota: id1,...,idN sono opzionali!

Chiamata.

Sintassi: *id (exp1, ..., expN)*
Esempio: *x = pow(2.0*y,3.0);*
dove:

- exp1, ..., expN sono i parametri attuali della chiamata
- *id* è l'identificativo della funzione.

x deve essere del tipo specificato in fase di definizione della funziona (il tipo che precede l'*id* della funzione) e in *x* verrà memorizzato il valore di ritorno della funzione.

La chiamata può essere fatta anche senza eguagliare ad una variabile.
Ad esempio: *cout << pow(2.0*y,3.0);*

N.B.! I parametri attuali *exp1, ..., expN* della chiamata devono essere **compatibili** per numero, ordine e rispettivamente per tipo ai corrispondenti parametri formali!

28.2.1 L'istruzione *return*

Il corpo di una funzione può contenere una o più istruzioni *return*.
Sintassi: *return expression;*
Esempio: *return 3*x;*
expression deve essere compatibile con il tipo restituito dalla funzione.

L'esecuzione dell'istruzione *return*:

- fa terminare la funzione

- fa sì che il valore della chiamata alla funzione sia il valore dell'espressione *expression* (con conversione implicita se di tipo diverso).

! È buona prassi che una funzione contenga un'unica istruzione *return*!

28.2.1.1 *return* multipli di una funzione

In una funzione è buona prassi evitare l'uso di *return* multipli (in particolare se usati come impliciti if-then-else).

```
int f (...) {                    int f (...) {
...                              int res;
return exp1;    diventa →        ...
...                              res = exp1;
return expN;                     ...
}                                res = expN;
                                 return res;
                                 }
```

```
if (...) {                       if (...) {
return exp1;                     res = exp1;
}else {          diventa →        } else {
return exp2;                     res = exp2;
}                                }
                                 return res;
```

28.2.1.2 L'istruzione *return* in un loop

L'istruzione *return* termina direttamente il ciclo (e l'intera funzione) equivalente ad un *break.*
Da evitare! Infatti si può sempre fare modificando la condizione.

28.2.2 Esempio di funzione

La funzione *mcd*

```
int mcd(int a, int b){
int resto;
while(b!=0) {
resto = a%b;
a = b;
b = resto;
}
return a; }
```

La chiamata *mcd(n1,n2)* viene eseguita nel modo seguente:

1. vengono calcolati i valori dei parametri attuali *n1* e *n2* (l'ordine non è specificato)
2. i valori vengono copiati, nell'ordine, nei parametri formali *a* e *b* (chiamata per valore)
3. viene eseguita la funzione e modificati i valori di *a* e *b* e della variabile locale *resto* (*n1* e *n2* rimangono con il loro valore originale)
4. la funzione *mcd* restituisce al programma chiamante il valore dell'espressione che appare nell'istruzione *return*

Nel *main* le chiamate sono le seguenti:

- *x = mcd (n1,n2)*, *x* assumerà il valore di *a* e potrà essere stampato in questo modo *cout << x;*
- oppure si può direttamente stampare in questo modo *cout << mcd(n1,n2);*

28.3 Le procedure (funzioni *void*)
In C++ c'è la possibilità di definire procedure, cioè funzioni che non ritornano esplicitamente valori (ovvero funzioni il cui valore di ritorno è di tipo *void*).
Esempio definione: *void pippo (int x)* {… … *return;* }
Esempio chiamata: *pippo(n*2);*

Nelle funzioni *void*, l'espressione *return* può mancare, oppure apparire senza essere seguita da espressioni (termina la procedura).
Le funzioni *void*, non restituendo alcun valore, presentano dei *cout* interni per poter stampare a video quello che fanno.