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

Variabili

Un importante tipo di identificatori sono le variabili.
Tutti voi possedete già il concetto di variabile in matematica.
In programmazione una variabile è un elemento del codice che identifica un particolare blocco di memoria. Il contenuto di tale blocco di memoria potrà quindi essere utilizzato e "manipolato" dal programma.

Per poter utilizzare una variabile occorre in primo luogo definirla.
La definizione di una variabile è un punto essenziale in quanto permette al compilatore di *allocare* la quantità di memoria necessaria al tipo di variabile che si deve gestire e di scegliere le istruzioni in linguaggio macchina idonee ad eseguire operazioni su di essa.

In C le definizioni di variabili vanno tutte inserite all'inizio del blocco di istruzioni.
In C++ possono essere definite quando vengono utilizzate.

Variabili

Elemento importante nella definizione di una variabile è il tipo di dato che essa deve contenere. Le quattro parole chiave riportate di seguito identificano i quattro più comuni tipi di dato contenuto in una variabile che potrete trovare in C.

char	int	float	double

char	carattere. Richiede un byte di memoria
int	intero. Richiede (quasi sempre) 4 bytes di memoria
float	reale in singola precisione. 4 bytes di memoria.
double	reale in doppia precisione. 8 bytes di memoria

```
int main() {
    int a;
    int numero_di_studenti;
    double Media_voto,media_voto;
}
```

Variabili: *int*

Le variabili intere (int) vengono memorizzate in una parola della memoria della macchina. Le macchine moderne usano parole da 32 bit per cui il massimo numero intero memorizzabile è

$$+/-2^{31}$$

```
int main() {
    int a=2000000000;
    int b=2000000000;
    int c=a+b;
}
```

Variabili: *float* e *double*

I due formati di dato che più frequentemente vengono usati in C per rappresentare variabili di tipo reale sono double e float

Di *default* (a differenza di molti altri linguaggi) una costante reale in C è di tipo double (doppia precisione).

Nella maggior parte delle macchine un reale di tipo float è registrato in 4 bytes di memoria mentre un reale di tipo double in 8.

Dato che la macchina lavora in binario un reale non è quasi mai registrato in maniera esatta, per cui le operazioni tra reali (a differenza delle operazioni tra interi) non sono necessariamente esatte.

I possibili valori assegnabili ad un reale sono espressi da due attributi: *precisione* e *range*. La precisione identifica il numero di cifre significative rappresentabili, mentre il range identifica il più grande e il più piccolo numero rappresentabile

Variabili: *float* e *double*

Un float normalmente ha una precisione di 6 cifre significative e un range di 10^{-38} -10^{38}

Un float può essere visto come un numero scritto nel formato

$$0.d_1d_2d_3d_4d_5d_6 \times 10^n$$

con n compreso tra -38 e 38

Un double ha invece una precisione di 15 cifre significative e un range che va da -308 a 308

```
int main() {
   float a=1.e+8;
   float c=a+0.1;
}
```

limite in precisione

```
int main() {
   float a=1.e+38;
   float c=a*10;
}
```

limite in range

Variabili: *char*

I caratteri sono un particolare tipo di variabili. Una variabile di tipo char può contenere **uno e un solo** carattere.

Dato che la macchina lavora utilizzando sempre e solo numeri, i caratteri vengono registrati in memoria sotto forma di un numero intero.

Esiste quindi una tabella di conversione (ASCII) che permette di mettere in relazione biunivoca i caratteri con gli interi ad essi associati.

Un solo byte di memoria è sufficiente a registrare 256 valori diversi (2^8). Per cui un solo byte è sufficiente a contenere tutti i caratteri utili.

Espressione

Una espressione e' una successione di identificatori, costanti e operatori che da origine a un risultato (numerico, logico o di tipo carattere).

Non costituiscono una espressione:
- Le parole chiave
- Gli elementi di punteggiatura (normalmente utilizzati per delimitare le espressioni)

Esempi di espressioni
a=b+c;
a == b;
(-i+j*5>= ++k*3)
('A'+c-'a'=='z')

Istruzione e Istruzione composta

Un'istruzione e' una successione di elementi limitata da un punto e virgola (;). Il compilatore, una volta individuati i vari elementi li raccoglie in istruzioni. Le istruzioni vengono valutate una sola volta. Una istruzione puo' coincidere con una espressione (istruzioni di assegnazione).

Un'istruzione composta e' un insieme di istruzioni semplici delimitate da due parentesi graffe ({ … }). Un'istruzione composta puo' contenere ***definizioni***. In questo caso e' spesso detta ***blocco***.

Esempi di istruzioni

```
int a;                          Definizione

c=a;                            Assegnazione

while(a>b) c++;                 Controllo del flusso

if (a>b)
      c=a;

while (a  >  b)
/* commento.. */
      c++;
```

Esempi di istruzioni composte

```
{
        int a=1;
        a++;
}

{
        int h=0;
        {
                int c=3;
                c++;
        }
}
```

Funzioni Matematiche

Per poter utilizzare le funzioni matematiche occorre "*renderle note*" al compilatore

```
#include <math.h>
#include <iostream>
int main(void) {
    double x=0.5;
    double y;
    y=sin(x);
    std::cout << "sin ( " <<x<< ") =" <<y << std::endl;
    return 0;
}
```

sin(x), cos(x), tan(x)	funzioni goniometriche
asin(x), acos(x), atan(x)	funzioni goniometriche inverse
exp(x)	esponenziale
log(x), log10(x)	logaritmo naturale e in base 10
fabs(x)	valore assoluto
pow(x,y)	elevazione a potenza x^y
sqrt(x)	radice quadrata

Operatori Logici

	Operatori	Associativita'
	() ++ (*postfisso*) -- (*postfisso*)	Sin. - Des.
	! + (*unario*) – (*unario*) ++ (*prefisso*) -- (*prefisso*)	Des. - Sin.
	* / %	Sin. – Des.
	+ -	Sin. – Des.
	< <= > >=	Sin. – Des.
	== !=	Sin. – Des.
	&&	Sin. - Des.
	||	Sin. – Des.
	= += -= *= /*	Des. – Sin.

Operatori Logici

Nel linguaggio C falso e' rappresentato dal valore zero mentre vero e' rappresentato da una qualsiasi espressione diversa da zero.
0, 0.0, '\0' e NULL sono esempi di valori considerati falsi.

L'azione di un operatore logico tra due espressioni dara' quindi come risultato l'intero 1 nel caso in cui la relazione sia vera e l'intero 0 nel caso in cui la relazione sia falsa.

Consideriamo alcuni esempi

```
char c='h';
int i=1, j=10, k=-10;
float  x=7e+30, y=1e-8;
```

'a'+3<c	('a'+3)<c	1
-i+j*5>k*2	(-i)+(j*5)>(k*2)	1
3<j<5	???	???
x<x+y	x<(x+y)	???

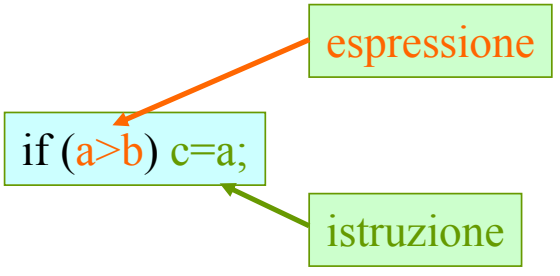
Operatori Logici

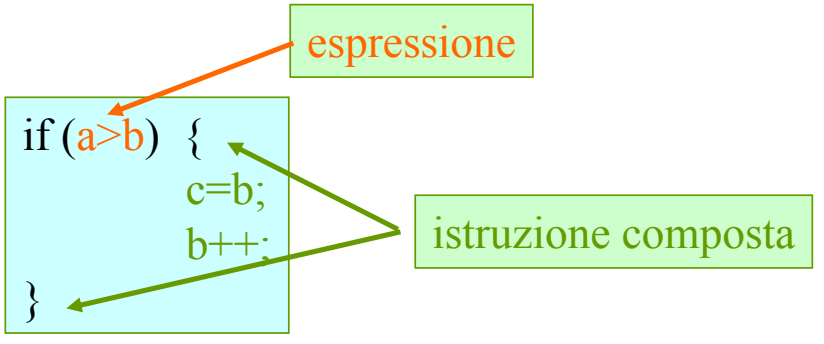
```
char c='h';
int h=0,i=1, j=10, k=-10;
float  x=7e+30, y=1e-8;
```

i && j && k	(i && j) && k	1
!++h		0
!h++		1
!h*!!x		???
!(x<y) && k		???
(h==i) && j		???
(h=i) && j		???

Controllo del Flusso: *if if-else*

La parola chiave if permette l'esecuzione di una istruzione se e solo se una espressione e' vera'.






```
if (a>b) c=a;
else c=b;
```

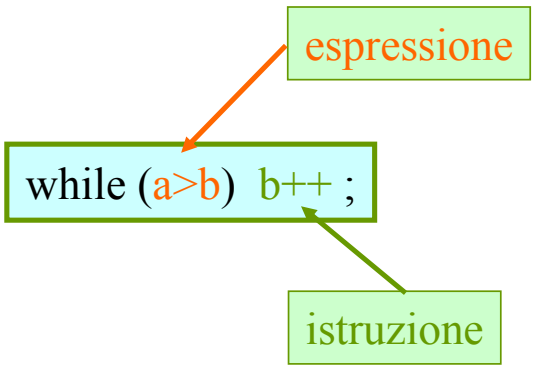
```
if (a>b) {
        c=a;
        b++;
} else {
        c=b;
        a++;
}
```

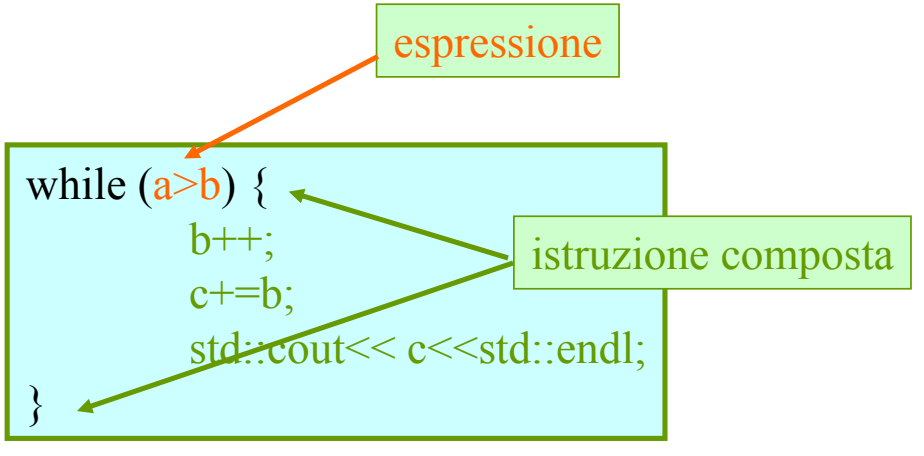
```
if (a>b) {
        c=a;
        b++;
} else c=b;
```

```
if (a>b) {
        c=a;
        b++;
} else if (a==b) a++;
else {
        c=b;
        a++;
}
```

Controllo del Flusso: *while*

La parola chiave while permette di eseguire una istruzione (semplice o composta) sino a quando una espressione e' vera.
Nota: l'istruzione viene eseguita se e solo se l'espressione e' vera.





Controllo del Flusso: *for*

La parola chiave for permette di eseguire la stessa istruzione un numero di volte fissato e introduce la possibilita' di utilizzare all'interno dell'istruzione un indice che identifica il numero di iterazione.
L'istruzione for ha (normalmente) tre argomenti.

```
for (arg1 ; arg2 ; arg3) istruzione ;
```

Se confrontata con la parola chiave while, essa e' equivalente a:

```
arg1;
while (arg2) {
        istruzione;
        arg3;
}
```

Come si puo' vedere dalla precedente sintassi, il primo e il terzo argomento di for sono a loro volta delle istruzioni (normalmente istruzioni di assegnazione o definizione) mentre il secondo argomento e' una espressione che torna una condizione logica (*zero / non-zero*)

Controllo del Flusso: *for*

```
for ( i=1 ; i<= k ; i++ ) fattoriale *=i;
```

La successione delle operazione eseguite dalla precedente istruzione e' la seguente:

1. Viene valutato il primo argomento (eseguita l'espressione di definizione e assegnazione int i=0;)
2. Viene valutato il secondo argomento (espressione i<=k). Se e' vero (*non-zero*) …
3. Viene eseguita l'istruzione (fattoriale *=i;)
4. Viene valutato il terzo argomento (istruzione i++).
5. Ritorna al passo 2

L'istruzione termina quando l'espressione al passo 2 diventa falsa.

Vantaggi:
• Numero di iterazioni fissato.
• Disponibilita di un indice di iterazione per l'istruzione (i nell'esempio) .

Controllo del Flusso: *for*

```
int i=0;
for ( ; i<=10 ; i++)  std::cout<<“Ciao Mondo!”<<std::endl;
```

```
int i=0;
for ( ; i<= 10 ; ) {
   std::cout<<“Ciao Mondo!”<<std::endl;
    i++;
}
```

```
for ( ; ; )  {
    …..
/* questo e’ un loop infinito  */
}
```

Controllo del Flusso: *break* e *continue*

Mentre si e' all'interno di un loop e' possibile saltare una parte dell'istruzione che si sta eseguendo o interrompere bruscamente l'intera esecuzione del loop al verificarsi di particolari condizioni mediante le parole chiave break e continue.

```
/* calcolo la radice quadrata di numeri inseriti da tastiere */
#include <iostream>
#include <math.h>
int main(void) {
    double radicando,radice;
    while (1)  {
            std::cin >> radicando;

            if (radicando< 0.0) break;

            radice=sqrt(radicando);

            std::cout<<"La radice di "<< radicando << " è " << radice<< std::endl;
    }
     return 0;
}
```

???

Pre-compilatore

input mediante cin

Funzione matematica sqrt()

Output mediante cout

Controllo del Flusso: *break* e *continue*

```
/* calcolo la radice quadrata di numeri inseriti da tastiera */
#include <iostream>
#include <math.h>
int main(void) {
    double radicando,radice;
    while (1)  {
            std::cin >> radicando;
            if (radicando< 0.0) continue;

            radice=sqrt(radicando);
             std::cout<<"La radice di "<< radicando << " è " << radice<< std::endl;

    }
   return 0;
}
```

Esempi

1. Leggere un numero da tastiera e stamparlo sullo schermo
2. Leggere due numeri da tastiera, sommarli e stampare il risultato
3. Leggere un numero da tastiera, calcolarne la radice quadrata e stamparla sullo schermo
4. Scrivere un programma che calcoli l'ipotenusa di un triangolo rettangolo leggendo la lunghezza dei cateti da tastiera. Stampare il risultato sullo schermo.
5. Sommare due vettori
6. Calcolare il prodotto scalare/vettoriale tra due vettori nel piano.
7. Come 3 ma verificare se il numero in input è positivo
8. Calcolare il valore assoluto di un numero in input
9. Calcolare la distanza di un punto da una retta
10. Leggere 10 numeri da tastiera sommarli e stampare il risultato
11. Leggere N numeri da tastiera sommarli e stampare il risultato
12. Leggere N numeri da tastiera e sommarli in valore assoluto e stampare il risultato

Esercizio

Scrivere un codice in grado trovare gli zeri di un polinomio di secondo grado

$$ax^2+bx+c=0$$

Il codice deve:

1. Chiedere all'utente il valore numerico dei tre coefficenti.
2. Calcolare e stampare il valore degli zeri.