C++, manuale minimo

1. Compilazione, esecuzione

Su Linux per compilare un programma di nome `prog.cpp` in un terminale si usa il comando `g++ prog.cpp`. Per lanciare l'eseguibile si usa il comando `./a.out`.

2. Tipi standard

numeri interi	`short`, `int`, `long`
numeri reali	`float`, `double`, `long double`
carattere	`char`
booleana	`bool`

Esempio:

```
int x; // dichiara x senza inizializzarla
int y=10; // dichiara y e la inizializza a 10
char a='x'; // dichiara a e la inizializza a x
double b; // dichiara b senza inizializzarla
float c=1.1; // dichiara c e la inizializza a 1.1
```

Array monodimensionale (statico)

Gli array contengono più variabili dello stesso tipo. L'indice del primo elemento è 0. Dichiarazione:

```
tipo nome[dimensione];
```

Esempio:

```
int x[5]; // dichiara un array di 5 int
double y[10]; // dichiara un array di 10 double
long z[]={12,34}; // dichiara un array di 2 long
                  // inizializzate a 12 e 34
cout << z[0] << endl; // stampa 12
cout << z[1] << endl; // stampa 34
cout << z << endl; // stampa l'indirizzo di z[0]
```

Array multidimensionale (statico)

Gli array multidimensionali sono array di array. Dichiarazione:

```
tipo nome[dim_1][dim_2]...[dim_n];
```

Esempio:

```
int x[2][3]; // matrice con 2 righe e 3 colonne
x[1][2]=10; // setta l'elemento nella 2. riga e
            //nella 3. colonna a 10
```

3. Operatori comuni

Assegnamento:

`=`	assegna la destra alla sinistra
`+=`	combinazione di somma e assegnamento
`*=`	combinazione di prodotto e assegnamento
`-=`	combinazione di sottrazione e assegnamento
`/=`	combinazione di divisione e assegnamento

Aritmetica:

`+`	somma
`*`	prodotto
`-`	sottrazione
`/`	divisione
`%`	resto di divisione intera

(La divisione `7/5` restituisce `1` mentre la divisione `7./5` restituisce `1.4`.)

Relazionali:

`<`	minore
`<=`	minore o uguale
`>`	maggiore
`>=`	maggiore o uguale
`==`	uguale
`!=`	non uguale

Logici:

`||`	disgiunzione (or)
`&&`	congiunzione (and)
`!`	negazione (not)

Incremento e decremento:

`++`	incremento
`--`	decremento

Gli operatori `++` e `--` si possono usare in maniera post (prima si utilizza il valore e si modifica il valore dopo) e pre (prima si modifica il valore e dopo si utilizza il valore già modificato).

Esempio:

```
int x=4,y=5,z;
z=x+y; // assegna x+y a z
cout << z << endl; // stampa 9
z+=x; // assegna z+x a z
cout << z << endl; // stampa 13
cout << z%x << endl; // stampa 1
cout << (z<y) << endl; // stampa 0 (falso)
cout << (z>y) << endl; // stampa 1 (vero)
cout << (z>y&&z>x) << endl; //stampa 1 (vero)
cout << (z>y&&x>y) << endl; //stampa 0 (falso)
cout << (z>y||x>y) << endl; //stampa 1 (vero)
cout << x++ << endl; // stampa 4 e incrementa x
cout << x << endl; // stampa 5
cout << ++x << endl; // incrementa x e stampa x (6)
```

4. Strutture di controllo

Selezione

Se `condizione` è vero allora si esegue `istruzione_1` (che può essere composta da più istruzioni) altrimenti `istruzione_2` (che può essere composta da più istruzioni):

```
if(condizione)
  istruzione_1;
else
  istruzione_2;
```

Esempio:

```
int a,b;
cin >> a >> b;
if(a>b)
  cout << a << "␣e'␣maggiore␣di␣" << b;
else
  cout << a << "␣non␣e'␣maggiore␣di␣" << b;
```

Ripetizione

Ciclo `while`:

```
while{condizione}
  istruzione;
```

Funzionamento:

1. Si valuta `condizione`: se è vero si va al punto 2 altrimenti al punto 3.
2. Si esegue `istruzione` (che può essere composta da più istruzioni) e si torna al punto 1.
3. Il ciclo termina.

Esempio:

```
int i=2,s=0;
while(i<=20){
  s+=i;
  i+=2;}
// s e' la somma dei numeri pari fra 2 e 20 (inclusi)
cout << s << endl;
```

Ciclo `do-while`:

```
do
  istruzione;
while{condizione};
```

Funzionamento:

1. Si esegue `istruzione` (che può essere composta da più istruzioni) e si va al punto 2.
2. Si valuta `condizione`: se è vero si va al punto 1 altrimenti al punto 3.
3. Il ciclo termina.

Esempio:

```
int i=2,s=0;
do{
  s+=i;
  i+=2;}while(i<=20);
// s e' la somma dei numeri pari fra 2 e 20 (inclusi)
cout << s << endl;
```

Ciclo `for`:

```
for(inizializzazione;condizione;aggiornamento)
  istruzione;
```

Funzionamento:

1. Si esegue `inizializzazione` (che può essere composta da più istruzioni separate da `,`) e si va la punto 2.
2. Si valuta `condizione`: se è vero si va al punto 3 altrimenti al punto 5.
3. Si esegue `istruzione` (che può essere composta da più istruzioni) e si va al punto 4.
4. Si esegue `aggiornamento` (che può essere composta da più istruzioni separate da `,`) e si torna al punto 2.
5. Il ciclo termina.

Esempio:

```
int i,s;
for(i=2,s=0;i<=20;i+=2)
  s+=i;
// s e' la somma dei numeri pari fra 2 e 20 (inclusi)
cout << s << endl;
```

5. Funzioni

Le funzioni possono prendere parametri e possono restituire valori. Forma generale:

```
tipo_restituito nome(tipo_1 parametro_1,...,
                     tipo_n parametro_n){
...}
```

Passaggio di parametro per valore

I parametri di default vengono passati con passaggio di parametro per valore.

Esempio:
Funzione che restituisce il maggiore fra due interi:

```
int maggiore(int a, int b){
  if(a>b) return a;
  return b;
}
```

Possibile utilizzo della funzione precedente:

```
int e=10,f=100;
cout << maggiore(e,f) << endl;
```

Funzione che stampa il minore fra due numeri reali:

```
void minore(double a, double b){
  if(a<b) cout << a << endl;
  cout << b << endl;
}
```

Possibile utilizzo della funzione precedente:

```
double e=1.1,f=2.2;
minore(e,f);
```

Passaggio di parametro per riferimento

Aggiungendo `&` si può effettuare passaggio di parametro per riferimento. Questo permette di restituire più di un valore.

Esempio:
Funzione che calcola i zeri di un polinomio di secondo grado se essi sono numeri reali (restituisce `true` se i zeri sono numeri reali e i zeri stessi vengono restituiti tramite il quarto e il quinto parametro della funzione):

```
bool zeri(double a, double b, double c,
          double &z1, double &z2){
  double d=b*b-4*a*c;
  if(d<0) return false;
  d=sqrt(d);
  z1=(-b-d)/2/a;
  z2=(-b+d)/2/a;
  return true;
}
```

Possibile utilizzo della funzione precedente:

```
double c2,c1,c0,r1,r2;
cin >> c2 >> c1 >> c0;
if(zeri(c2,c1,c0,r1,r2))
  cout << r1 << "␣" << r2 << endl;
```

Array come parametro

Esempio:
Funzione che ribalta un array.

```
void ribalta(int a[], int n){
  int i=0,j=n-1,t;
  while(i<j){
    t=a[i]; a[i]=a[j]; a[j]=t;
    i++; j--;
  }
}
```

Utilizzo:

```
  int v[5]={1,2,3,4,5};
  ribalta(v,5);
```

6. Puntatori

I puntatori rappresentano la posizione (indirizzo di memoria) di variabili. Per dichiarare un puntatore si utilizza `*`:

```
short *x; // dichiara un puntatore a short
double *y; // dichiara un puntatore a double
```

L'operatore `&` restituisce l'indirizzo di una variabile.
L'operatore `*` restituisce il valore che si trova ad un certo indirizzo.

Esempio:

```
int x1=10,x2;
int *p1;
p1=&x1; // p1 punta a x1
x2=*p1; // il valore puntato da p1 viene
        // assegnato a x2
cout << x2 << endl; // stampa 10
*p1=15; // viene assegnato 15 alla variabile
        // puntata da p1 (cioe' a x1)
cout << x1 << endl; // stampa 15
```

Allocazione dinamica della memoria (array dinamici)

I puntatori si usano per allocare memoria in maniera dinamica (la quantità di memoria utilizzata viene stabilita "run-time"). Per allocare memoria si usa `new`, per deallocare `delete`. Una volta allocata la memoria, si accede agli elementi come se si trattasse di un array statico.

Esempio:

```
// legge un vettore di interi
int* read_vet(int &n){
  cin >> n; // numero di elementi
  int *v=new int[n]; // alloca memoria
  for(int i=0;i<n;i++)
    cin >> v[i];
  return v; // restituisce l'indirizzo degli interi
}

// calcola la somma di un vettore di interi
int sum_vet(int *v, int n){
  int s=0;
  for(int i=0;i<n;i++)
    s+=v[i];
  return s;
}

// utilizzo di read_vet e sum_vet
int *v1,n1;
v1=read_vet(n1);
cout << "Somma:␣" << sum_vet(v1,n1) << endl;
delete[] v1; // dealloca
```

7. Struttura di un semplice programma

Un semplice programma può essere scritto in un singolo file con estensione `.cpp`. Il programma comincia con gli `#include` delle librerie e le dichiarazioni globali (costanti, variabili e `using`). Il resto del programma è un elenco di funzioni. Deve esserci una funzione `main` dove comincia l'esecuzione del programma. Per poter usare una funzione prima di definirla completamente bisogna specificare il suo prototipo.

```
#include<iostream> // input-output
#include<cmath> // funzioni matematiche
// exp, log, log10, pow, sqrt, ceil, floor, round, abs
#include<cstdlib> // numeri casuali
#include<ctime> // per misurare tempo
#include<iomanip> // formato output

using namespace std; // per abbreviare

// prototipi delle funzioni
bool zeri(double, double, double, double &, double &);
int maggiore(int, int);

// modulo principale
int main() {
  double c2,c1,c0,r1,r2;
  cin >> c2 >> c1 >> c0;
  if(zeri(c2,c1,c0,r1,r2))
    cout << setprecision(5) << r1 << "␣" << r2 << endl;

  int e,f;
  cin >> e >> f;
  cout << maggiore(e,f) << endl;

  return 0;
}

int maggiore(int a, int b){
  if(a>b) return a;
  return b;
}

bool zeri(double a, double b, double c,
          double &z1, double &z2){
double d=b*b-4*a*c;
if(d<0) return false;
d=sqrt(d);
z1=(-b-d)/2/a;
z2=(-b+d)/2/a;
return true;
}
```