STRUTTURE DI CONTROLLO DEL C++

Le istruzioni if e else

Le istruzioni condizionali ci consentono di far eseguire in modo selettivo una singola riga di codice o una serie di righe di codice (che viene detto **blocco di istruzioni**).

Il C++ fornisce quattro istruzioni condizionali:

- **if**
- **if-else**
- **?**
- **switch**

Prima di affrontare tali istruzioni una per una, è bene dire che, quando all'istruzione condizionale è associata una sola riga di codice, non è necessario racchiudere l'istruzione da eseguire fra **parentesi graffe**. Se invece all'istruzione condizionale è associata una serie di righe di codice, il blocco di codice eseguibile dovrà essere racchiuso fra parentesi graffe.

Per evitare il rischio di errori (specie quando si è neofiti) e per una buona lettura del codice è consigliabile l'uso delle parentesi graffe anche quando l'istruzione da eseguire è soltanto una.

Le istruzioni if e else

Le istruzioni in oggetto sono molto intuitive se si considera che le parole inglesi **if** ed **else** corrispondono rispettivamente alle italiane *se* e *altrimenti*. La sintassi di `if` è:

```
if (<condizione>)
{
  (<istruzioni da svolgere se la condizione è vera>);
}
```

mentre se `if` è accompagnata da altre istruzioni alternative la sintassi è:

```
if(<condizione>)
{
  (<istruzioni da svolgere se la condizione è vera>);
}
else
{
  (<istruzioni da svolgere se la condizione è falsa>);
}
```

oppure se è necessario condizionare anche le istruzioni rette da else:

```
if(<condizione 1>)
{
  <istruzioni da svolgere solo se la condizione 1 è vera>);
```

```
}
else if(<condizione 2>)
{
  (<istruzioni da svolgere solo se la condizione 1 è falsa e la condizione 2 è
vera>);
}
```

All'interno delle parentesi le istruzioni vanno completate sempre con il punto e virgola, mentre se non utilizziamo blocchi, ma semplici istruzioni facciamo attenzione a non dimenticare il punto e virgola prima dell'istruzione `else`. Ad esempio:

```
if(i>=0)
  <istruzione 1>; // singola istruzione
else
{                 // blocco di istruzioni
  <istruzione 2>;
  <istruzione 3>;
}
```

L'istruzione 1 verrà eseguita solo se `i >= 0` mentre in caso contrario viene eseguito il blocco con le istruzioni 2 e 3.

le istruzioni switch

Le istruzioni switch permettono di evitare noiose sequenze di if. Esse sono particolarmente utili quando in un programma si deve dare all'utente la possibilità di scegliere tra più opzioni. La sintassi di tali istruzioni è la seguente:

switch(<espressione intera>)
{
case (<valore costante 1>):
…(<sequenza di istruzioni>)
break;
case (<valore costante 2>)
…(<sequenza di istruzioni>)
break;
….
….
default:
// è opzionale
…(<sequenza di istruzioni>)
}

Il costrutto precedente esegue le seguenti operazioni:

- Valuta il valore dell'espressione intera passata come parametro all'istruzione switch.

- Rimanda lo svolgimento del programma al blocco in cui il parametro dell'istruzione case ha lo stesso valore di quello dell'istruzione switch.
- Se il blocco individuato termina con un'istruzione break allora il programma esce dallo switch.
- Altrimenti, vengono eseguiti anche i blocchi successivi finchè un'istruzione break non viene individuata oppure non si raggiunge l'ultimo blocco dello switch.
- Se nessun blocco corrisponde ad un valore uguale a quello dell'istruzione switch allora viene eseguito il blocco default, se presente.

Vediamo un paio di esempi per comprendere meglio quanto detto:

```
bool freddo, molto_freddo, caldo, molto_caldo;
switch (temperatura)
{
case(10):
freddo = true;
molto_freddo = false;
caldo = false;
molto_caldo = false
break;
case(2):
freddo = true;
molto_freddo = true;
caldo = false;
molto_caldo = false;
break;
case(28):
freddo = false;
molto_freddo = false;
caldo = true;
molto_caldo = false;
break;
case(40):
freddo = false,
molto_freddo = false;
caldo = true;
molto_caldo = true;
break;
default:
freddo = false;
molto_freddo = false;
caldo = false;
molto_caldo = false;
```

Nell'esempio precedente, viene valutato il valore costante temperatura passato come parametro a switch. A secondo del valore che tale costante assume viene poi eseguito il relativo blocco di istruzioni. Se, invece, nessuno dei blocchi ha un valore uguale a quello passato a switch, verrà eseguito il blocco default.
Si noti, che nel caso seguente:

……..

case(28):
freddo = false;
molto_freddo = false;
caldo = true;
molto_caldo = false;
case(40):
freddo = false,
molto_freddo = false;
caldo = true;
molto_caldo = true;
break;
…….

se il valore della temperatura fosse di 28 gradi, verrebbe giustamente eseguito il blocco corrispondente, ma poichè manca l'istruzione break alla fine di tale blocco, il programma avrebbe continuato ad eseguire anche le istruzioni del blocco successivo (quello con la temperatura uguale a 40 gradi) e quindi il valore finale delle variabili sarebbe stato:

freddo = false;
molto_freddo = false;
caldo = true;
molto_caldo = true;

Il ciclo for

Il linguaggio C++ è dotato di tutte le istruzioni di controllo iterative presenti negli altri linguaggi: cicli **for, while e do-while**. La differenza fondamentale fra un ciclo `for` e un ciclo `while` o `do-while` consiste nella definizione del numero delle ripetizioni.

Normalmente, i cicli for vengono utilizzati quando il numero delle operazioni richieste è ben definito mentre i cicli while e do-while vengono utilizzati quando invece il numero delle ripetizioni non è noto a priori.

Il ciclo for

Il ciclo for ha la seguente sintassi:

```
for(valore_iniziale, condizione_di_test, incremento)
{
    (<istruzioni da eseguire all'interno del ciclo>)
}
```

- Con l'espressione *valore_iniziale* indichiamo quale sarà la variabile contatore nel ciclo e ne impostiamo il valore iniziale;
- L'espressione *condizione_di_test* rappresenta la condizione da verificare ad ogni passo del ciclo per valutare se è necessario continuare oppure uscire dall'iterazione.
- L'espressione *incremento* descrive come modificare, incrementare o decrementare il contatore ad ogni esecuzione.

Facciamo subito uno degli esempi più classici, la somma dei primi 10 numeri interi:

```
/*
 * Somma dei primi 10 numeri interi
 */

int totale = 0; // Questa variabile conterrà la somma totale

// Il ciclo seguente aggiunge i numeri da 1 a 10 alla variabile totale
for (int i=1; i <= 10; i++)
{
    totale += i;
}
```

Ecco come viene impostato il nostro loop in questa semplice porzione di codice:

- Come `valore_iniziale` abbiamo l'espressione `int i=1`, con cui oltre a indicare la variabile `i` come variabile contatore inizializzata a `1`, ne effettuiamo anche la dichiarazione.
- Nella `condizione_di_test` verifichiamo che `i` sia minore o uguale a `10`, per rimanere all'interno del ciclo.
- Nell'espressione `incremento` ci limitiamo ad aggiungere 1 alla variabile `i` ad ogni passo del ciclo (l'istruzione `i++` serve per incrementare di un'unità un intero).

Già all'ingresso del loop la prima volta, la variabile intera `i` assume il vaolre `1`, viene eseguita l'istruzione `totale += i` (che equivale a scrivere `totale = totale+i`) e quindi il valore della variabile `totale` (che all'inzio era 0) diventa `1`.

A questo punto viene eseguito l'incremento e quindi `i` diventa uguale a `2`. Quindi si ritorna a valutare la `condizione_di_test`.

Il test verificherà che `i` è ancora minore o uguale a `10`, quindi sarà di nuovo aggiunto il valore di `i` a `totale`, che ora varrà `1 + 2 = 3`.

Il processo continuerà in questo modo finchè la variabile `i` non supererà il valore di 10. In definitiva avremo eseguito l'operazione:

```
1+2+3+4+5+6+7+8+9+10 = 55
```

Alla fine del ciclo quindi, la variabile `totale` avrà assunto il valore `55`.

l ciclo while

L'istruzione while segue la seguente sintassi:

while(condizione)
{
// Istruzioni da eseguire
}

dove *condizione* rappresenta un controllo booleano che viene effettuato ogni volta al termine del blocco di istruzioni contenuto tra le parentesi graffe.
Se la condizione restituisce true allora il ciclo sarà eseguito ancora mentre se la condizione ritorna un valore uguale a false il ciclo sarà terminato.
Vediamo un esempio. Supponiamo di voler scrivere un programma che stampi sullo schermo tutti i numeri pari tra 11 e 23:

```
// File da includere per operazioni
// di input/output cout
#include <iostream.h>

int main()
{
// Definiamo una variabile che conterrà il valore corrente
int numero_corrente = 12;

// ciclo while che stampa sullo schermo tutti i numeri pari
// tra 11 e 23
while (numero_corrente < 23)
{
cout << numero_corrente << endl;
numero_corrente = numero_corrente + 2;
}

cout << “Fine del Programma!” << endl;
}
```

Il funzionamento del programma precedente è molto semplice: viene definita una variabile numero_corrente che useremo per conservare il valore che di volta in volta il ciclo while modificherà. Inizialmente tale variabile viene inizializzata a 12 (che è il primo valore pari dell’intervallo). A questo punto si entra nel ciclo while e viene testata la condizione che si chiede se numero_corrente è minore di 23. Ovviamente la condizione viene soddisfatta e così viene eseguito il blocco all’interno del ciclo while. La prima istruzione del blocco stampa il valore della variabile numero_corrente (che è attualmente 12) mentre la seconda istruzione incrementa il valore della stessa variabile di 2 unità. Adesso numero_corrente vale 14.
Si ricomincia di nuovo: si testa la condizione, questa è ancora soddisfatta, si stampa il valore della variabile (adesso è 14) e si incrementa numero_corrente di 2 unità (ora vale 16). E così via fino a quando la condizione non è più verificata, ovvero quando accadrà che numero corrente varrà 24.
L’output del programma precedente, sarà allora il seguente:

```
12

14

16

18

20

22
```

Fine del Programma!

Il ciclo do-while

Il ciclo do-while ha la seguente sintassi:

```
do
{
(<istruzioni da eseguire all'interno del ciclo>)
}while (condizione)
```

La differenza fondamentale tra il ciclo **do-while** e i cicli **for** e **while** è che il **do-while** esegue l'esecuzione del ciclo almeno per una volta. Infatti, come si vede dalla sintassi, il controllo della condizione viene eseguito al termine di ogni loop. Se volessimo scrivere lo stesso esempio della visualizzazione dei numeri pari compresi tra 11 e 23 otterremmo:

```
// File da includere per operazioni di
//input/output cout
#include <iostream.h>

int main()
{
// Definiamo una variabile che conterrà il valore corrente
int numero_corrente = 12;

// ciclo do-while che stampa sullo
//schermo tutti i numeri pari
// tra 11 e 23
do
{
cout << numero_corrente << endl;
numero_corrente = numero_corrente + 2;
} while (numero_corrente < 23);

cerr << "Fine del Programma!" << endl;
}
```

Il funzionamento del programma precedente è molto simile a quello visto per il ciclo while con la differenza, come detto, che la condizione viene verificata al termine dell'esecuzione del blocco di istuzioni. Se, per esempio, il valore iniziale della variabile numero_corrente fosse stato 26, il programma avrebbe comunque stampato sul video il valore 26 e poi si sarebbe fermato. In tal caso, avremmo ottenuto un risultato diverso da quello desiderato in quanto il numero 26 è certamente maggiore del 23 che rappresenta il limite dell'intervallo da noi definito. Il ciclo while, invece, non avrebbe stampato alcun numero.
Tale osservazione va tenuta presente per capire che non sempre le tre istruzioni di ciclo sono intercambiabili.