Espressioni logiche

mercoledì 7 ottobre 2020 13:44

Tipo booleano (bool)

in C++ esiste un tipo base chiamato booleano che consiste di due soli valori `true` (vero) e `false` (falso)

```
bool controllo;    // dichiarazione di variabile di tipo bool
controllo = true; // esempio di assegnazione
```

Operatori relazionali

- == uguale a notate bene!
- != diverso da
- < minore, <= minore o uguale
- > maggiore, >= maggiore o uguale

operatori relazionali

- (2==7) è false
- (8<15) è true
- 3.1!=3.0 è true
- 6<=6 è true

Si possono usare anche gli operatori logici:

- Espressioni più complesse possono essere ottenute combinando espressioni logiche con gli operatori logici && (AND), || (OR) e ! (NOT)
- Esempio
  (6<6) && (1==1) è false
  (6<6)|| (1==1) è true
  !(6<6) è true
- Precedenze (l'operatore logico che arriva primo ha la precedenza)
  (5<3)&&(6<=6)||(5!=6) è true

Espressioni logiche e string

Nel confronto fra stringhe, l'ordine di riferimento è quello lessicografico, che comprende anche quello alfabetico. Si ricorda che nel C++ si fa riferimento all'ordine di valori secondo la tavola ASCII.

Espressioni logiche e floating point

- i numeri floating point sono spesso soggetti ad errori di arrotondamento dovuti allo spazio finito in memoria ad essi dedicato
- vanno sempre trattati con cautela, ne vediamo un primo esempio nell'ambito delle espressioni logiche
- 3.0/7.0 + 2.0/7.0 + 2.0/7.0 == 1.0 può essere false

provate le seguenti varianti…

```
float f;
f=3.0/7.0 + 2.0/7.0 + 2.0/7.0;
cout << (f==1.0);
```

```
double d;
d=3.0/7.0 + 2.0/7.0 + 2.0/7.0;
cout << (d==1.0);
```

Questo perché differenti tipi hanno differenti possibilità di approssimazione: se per un float dunque f == 1.0 è uguale, per un double, che trattiene più cifre, questo valore differisce di un valore decimale anche piccolissimo, di cui appunto tiene conto nel confronto a differenza del float.

Tolleranza

Se il numero ottenuto differisce di una quantità inferiore alla tolleranza, è appunto "tollerabile", quindi accettabile nel confronto dell'eventuale uguaglianza.

- E' buona norma includere una *tolleranza* ossia accettare che il numero approssimato sia un po' diverso da quello atteso:

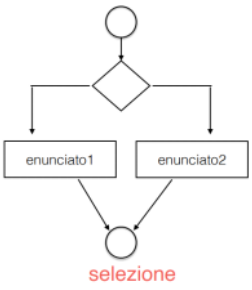
```
#include<cmath>
....
x= 3.0/7.0 + 2.0/7.0 + 2.0/7.0;
y=1.0;
fabs(x-y)<0.00001;
```

Istruzioni di selezione

mercoledì 7 ottobre 2020 14:09

Il caso SE…ALLORA

```
if(espressione)
   enunciato;
```



Il caso SE…ALLORA; ALTRIMENTI

```
if(espressione)
     enunciato1;
else
     enunciato2;
```

Altrimenti ciò significherebbe che dopo la condizione c'è un'espressione vuota. E la riga intesa da noi come enunciato, viene letta come una riga non vincolata da alcuna espressione!

Esempio in un codice:

```
int main(){

    int a, b;

    cout << "Inserisci due interi (stando ben attento a non sbagliare!)" << endl;
    cin >> a >> b;

    if (a==b)
        cout << "Gli interi a e b sono uguali" << endl;
    else
        cout << "Gli interi a e b sono diversi" << endl;

}
```

Nota bene! Nel caso in cui vi siano più espressioni negli enunciati di IF ed ELSE, bisogna mettere delle parentesi graffe per identificare i vari blocchi. Se ciò non viene fatto, può succedere che il compilatore leggerà e farà eseguire le righe dopo la prima come se non fossero vincolate da alcuna condizione, oppure altri errori.

If annidati: ricorda che è possibile concatenare più SE…ALLORA, utile in date circostanze:

Esempio

```
if (temperature >=25)
   if (temperature >=30)
       cout << "Good day for swimming" << endl;
   else
       cout << "Good day for golfing" << endl;
else
   cout<<"Good day to play tennis" << endl;
```

```
if (month==1)
    cout << "January";
if (month==2)
    cout << "February";
if (month==3)
    cout << "March";
...
if (month==12)
    cout << "December";
cout << endl;
```

```
if (month==1)
    cout << "January";
else if (month==2)
        cout << "February";
     else if (month==3)
             cout << "March";
...
             else if (month==12)
                cout << "December";
cout << endl;
```

più efficiente,
fa meno confronti

… non troppo leggibile
(molti nesting, troppe indent)

Il costrutto di destra fa sì che se il mese non è il primo, va a controllare se è il secondo e se non è questo quello successivo, e così via. A differenza del costrutto di sinistra però, quello di destra si ferma non appena soddisfatta la condizione. Mentre il costrutto di sinistra, anche se il mese è, ad esempio, il terzo, continua a controllare anche TUTTI i successivi.

Codice ambiguo

- in alcuni casi il codice può diventare indiscutibilmente ambiguo:

```
if (media >= 18)
 if (media < 19)
     cout << "Passato per il rotto della cuffia";
 else cout << "Esame Fallito";
```

in questo caso no!

- ecco la versione corretta:

```
if (media >= 18) {
 if (media < 19)
     cout << "Passato per il rotto della cuffia";
 }
else cout << "Esame Fallito";
```

Operatore condizionale ternario

E' un altro modo per esprimere SE…ALLORA; ALTRIMENTI.

- L'operatore condizionale ?: è un operatore **ternario** (necessita di 3 operandi)
- `espressione1 ? espressione2 : espressione3`

SE ALLORA ALTRIMENTI

- equivalente a

```
if (espressione1)
        espressione2;
else
        espressione3;
```

- esempio:

```
max=(a>=b) ? a : b;
```

Struttura switch

struttura switch

- struttura di selezione del C++ che permette di scegliere tra molte alternative
- inoltre non richiede la valutazione di un'espressione logica

deve assumere un valore di uno dei tipi di dato semplici

```
switch (espressione)
{
case valore1:
     enunciati1
     break;
case valore2:
     enunciati2
     break;
…
case valoreN:
     enunciatiN
     break;

default:
     enunciati
}
```

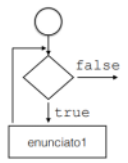
→ SE NESSUN CASO E' SODDISFATTO

Loop (Cicli)

martedì 13 ottobre 2020 11:35

Il loop (dall'inglese "ripetere") serve per far ripetere un determinato enunciato un numero di volte definito o indefinito.

Il while




```
while (espressione)
    enunciato;
```

```
while (espressione)
{
    enunciato1;
    …
    enunciatoN;
}
```

N.B

Non sempre sono a conoscenza del numero di volte per cui il ciclo si ripete.
Dunque è bene utilizzare dei riferimenti per far uscire dal ciclo quando necessario, questi sono:

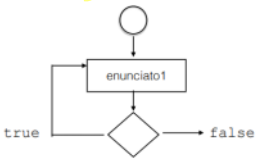
- **Sentinelle**: rimango nel ciclo fino a che la variabile di controllo non raggiunge un valore speciale, prefissato, detto "sentinella" .

```
cin >> variabile;

while (variabile!=sentinella)
    {
    cin >> variabile;
     ….
     }
```

- **Flag** - il flag è una variabile di tipo booleano.

```
found=false;

while (!found)
    {
     ….
     if (espressione)
         found=true;
     }
```

do-while



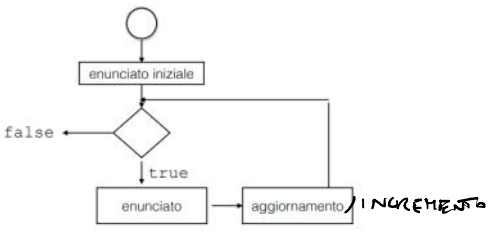

```
do
   enunciato
while (espressione);
```

```
do {
    enunciato1;
    …
    enunciatoN;
}
while (espressione);
```

il ciclo for

- e' una forma particolare di ciclo che semplifica il caso di ciclo con contatore

```
for(enunciato iniziale; condizione; aggiornamento)
        enunciato
```



N.B

Solitamente si preferisce usare il ciclo for negli algoritmi dove vi è necessità di utilizzare un contatore.

```
int contatore=1;
while (contatore <=10)
{
    cout << "BIANCO NERO BIANCO " << endl;
    ++contatore;
}

int contatore;
for (contatore=1; contatore <=10; ++contatore)
{
    cout << "BIANCO NERO BIANCO " << endl;
}
```

O ANCHE:

```
for (int contatore=1; contatore <=10; ++contatore)
{
    cout << "BIANCO NERO BIANCO " << endl;
}
```

MEGLIO USARE IL FOR

Nell'ultimo caso, se la variabile contatore viene dichiarata all'interno delle parentesi del ciclo, essa avrà uno **scope** limitato al solo ciclo, ciò significa che la dichiarazione vale solo per le espressioni nel corpo del ciclo e ogni altro richiamo della variabile al di fuori del ciclo **non** sarà possibile.