Istruzioni di iterazione – cicli

Questi comandi, chiamati anche **cicli**, ci permettono di ripetere più volte una o più operazioni. Nella programmazione sono uno strumento indispensabile per gestire grandi quantità di dati perché ci permettono di ripetere da 10 a milioni di volte dei compiti che ci porterebbero sicuramente alla noia mortale.

while()

La dichiarazione while() è la più classica dei cicli, all'interno delle parentesi il programmatore dichiara una condizione: **finchè** la condizione è vera il programma esegue le istruzioni all'interno delle parentesi graffe in caso contrario il ciclo si conclude ed il programma passa oltre.

La sua dichiarazione è la seguente:

```
while( condizione  ){
        //Se la condizione è vera
}
```

do while()

Il ciclo do-while() si distingue dal "fratello" while perché all'inizio esegue le istruzioni descritte e successivamente controlla la condizione, dopo di che si comporta esattamente come il while: finché la condizione è vera il ciclo continua a reiterarsi.

La sua dichiarazione è la seguente:

```
do{
        //Esegue il codice almeno una volta e si ripete se la condizione è vera.
}while( condizione );
```

for()

Il ciclo for aggiunge una comodità per i cicli che coinvolgono una successione numerica, come ad esempio un contatore, infatti permettono di dichiarare direttamente nella dichiarazione del ciclo una o più variabili e le operazioni da eseguire su di esse.

Il ciclo for ha 4 fasi principali:

1. Dichiara ed inizializza le eventuali variabili.
2. Controlla la condizione.
3. Se la condizione è vera esegue le istruzioni definite al suo interno.
4. Infine, esegue le operazioni di ciclo e ricomincia da capo.

La sua dichiarazione classica è la seguente:

```
for(  dichiarazione e inizializ. ;   condizione  ;   operazioni di ciclo  ){
        //Istruzioni
}
```

Qui presento un ciclo contatore sviluppato con il cicli while, do-while e for in modo da notarne le differenze.

```
Int i = 0;
while(i < 5){
i++;
}
```

```
int i = 0;
do{
        if(i > 0){
                i++;
        }
} while(i <= 5);
```

```
for(int i = 0; i < n; i++){
}
```

Notare che per fare ottenere al do-while lo stesso comportamento degli altri due ho dovuto ritardarlo di una iterazione inserendo l'istruzione if.

Range-for

Questo tipo di ciclo è una sotto-categoria del for che ci permette di definire una variabile intera e un valore massimo, il range-for scorre tutti i valori del range fino al limite impostato, una volta raggiunto il numero limite la dichiarazione si conclude.

```
for(variabile : num_limite){
        //Istruzioni
}
```

Esempio utile

```
Int v = 5;
for(int x : v){
        s+=x;
}
```

Cicli infiniti

In C++ possiamo definire anche dei cicli che non hanno mai fine e per capire come costruirli vi faccio una domanda: dal momento che il ciclo continua ad eseguirsi fintanto che la condizione è vera cosa succede se inseriamo l'espressione 1 == 1 come condizione?

Come state già immaginando la condizione 1 == 1 è sempre vera e quindi il risultato è che il ciclo si esegue infinite volte.+

Esempi coi vari tipi di ciclo

Per ordine ed eleganza è meglio inserire direttamente il valore true dentro la condizione.

```
while(true){
}
```

```
do{
} while(true);
```

```
for( ; true; ){
}
```

Abbiamo definito come si costruisce un ciclo infinito ma adesso si presenta un altro problema: una volta che siamo in un ciclo che non finisce mai come ne usciamo?

Istruzioni di uscita dal ciclo

Il C++ definisce 4 strumenti per permettere al programma di uscire da un ciclo:

1. break;
2. return;
3. throw;
4. goto;

break

Il break è lo strumento più utilizzato e semplice per uscire da un ciclo, si scrive semplicemente

```
for( ;itrue; ){
        break;
}
cout << “Ciclo infinito concluso”;
```

Quando il programma incontra questo comando conclude immediatamente il ciclo e continua con le istruzioni seguenti in questo caso il cout.

N.B. Il break si può utilizzare con qualsiasi ciclo, non solo con quelli infiniti!

return e throw

Sono due funzioni che si utilizzano all’interno delle funzioni, return è un’istruzione che ci permette di ritornare il valore di una funzione mentre throw ci permette di lanciare un errore se si presenta una determinata condizione. Per ora ci basta sapere che se vengono lanciati all’interno di un ciclo ne concludono immediatamente l’esecuzione.

goto

Il goto è un comando pericoloso, si tratta di uno strumento molto potente poiché permette di spostare l’esecuzione del programma in qualsiasi riga del codice, ma è meglio non abusarne. I goto confondono enormemente la struttura del codice e questo porta a problemi di mantenimento e di correzione del codice, quindi attenzione.

La sintassi del goto è la seguente

```
for( int i = 0 ; true ; i++){
        if(i == 7){
                goto outofcycle;
        }
}
outofcycle:
        cout << “Il ciclo è concluso”;
```

continue

Infine vi presento l’istruzione continue, l’ho tenuta per ultima e non l’ho elencata perché non permette di uscire da un ciclo ma bensì di arrivare appena prima della conclusione delle istruzioni. Questo permette di saltare al ciclo di operazioni successive se si presenta una particolare condizione.

```
for(int i=0 ; true ; i++){
        if(i == 4){
            continue;
        }
        cout << “i sicuramente non e’ uguale a 4”;
        cout << “i non e’ 4, giuro”;
        cout << “Infatti i e’ ” << i;

        //< ------ Il continue salta tutte le istruzioni fino a qui
}
//< ---------- il break invece fa saltare il programma qui, fuori dal ciclo
```

In questo simpatico esempio le istruzioni cout vengono processate soltanto se i è diverso da quattro, in quel caso il ciclo salta tutte le istruzioni fino alla freccia e passa al ciclo successivo. Ho messo anche la posizione in cui salta il comando break per palesarne le differenze.