

ANSI C – Costrutti Iterativi

Bentornati nel mondo dell'ANSI C, in questo tutorial aggiungeremo tre nuovi costrutti a quelli presentati. Parleremo in particolare di ripetizione, perché molto spesso vi capiterà di dover ripetere alcuni frammenti di codice più e più volte fino a quando una determinata condizione non è soddisfatta.

Abbiamo quindi tre tipi di costrutti diversi che ci permettono di ripetere del codice in maniera controllata, spesso la scelta di un costrutto invece di un altro è puramente stilistica, ma in alcuni casi le scelte sono obbligate, quindi andremo a vedere bene nel dettaglio il funzionamento dei tre analizzando quando prediligere uno o l'altro.

Iniziamo dal più semplice e più simile ai costrutti che abbiamo preso in esame nel tutorial precedente, ovvero il “WHILE”.

Il ciclo while

Scriviamo subito la sintassi del ciclo, poi passiamo a commentarla, evidenziando i suoi punti di forza e soprattutto vediamo quando usarlo.

```
while (condizione)
{
    blocco di istruzioni
}
```

Come vedete la struttura è simile a quella dei costrutti visti nel precedente tutorial, il seguente codice si legge in questo modo: “Fintanto che la condizione è vera (se la guardia è TRUE) esegui le istruzioni che si trovano nel blocco tra parentesi graffe.

Anche per il while vale lo stesso discorso delle parentesi fatto per l'IF, se non ricordate di cosa sto parlando, tornate al tutorial sui costrutti condizionali.

Quindi quando si usa il ciclo while? Fondamentalmente quando non sappiamo quante volte dobbiamo ripetere il blocco di codice, ma sappiamo che ci interessa farlo fino a quando una determinata condizione è valida, vediamo un esempio, pronti per un pochino di matematica?

ESEMPIO DI CICLO WHILE:

Piccolo incipit: nel XIII secolo un matematico pisano cercò di trovare una legge in grado di descrivere la crescita di una popolazione di conigli nell'arco del tempo. Assunse che una coppia di conigli diventi fertile dal primo mese di vita e che dia alla luce una nuova coppia di conigli al compimento del secondo mese. Naturalmente ogni nuova coppia nata si comporta esattamente come la precedente, diventando fertile al primo mese e dando alla luce una coppia al secondo.

Vedendo l'evoluzione nel corso del tempo notiamo che al mese 0 abbiamo una coppia di conigli, dopo il primo mese la coppia di conigli diventa fertile, alla fine del secondo mese abbiamo due coppie di conigli, la prima ancora fertile e la seconda che deve diventarlo. Alla fine del terzo mese abbiamo quindi 3 coppie di conigli, la prima e la seconda fertili e la terza non ancora, alla fine del quarto mese le coppie sono diventate in questo modo 5. Ovviamente la storia continua all'infinito dando origine ad una delle successioni numeriche più importanti nella storia, ovvero la serie di Fibonacci (dal nome del matematico pisano Leonardo “filio di Bonacci” Fibonacci).

Quindi possiamo notare che un numero della serie di Fibonacci è dato dalla somma dei due numeri che lo precedono, ecco un esempio dei primi numeri della successione:

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, //Fino all'infinito.

Dopo questo bel racconto il nostro esempio utilizzerà proprio questa serie, noi vogliamo fare la somma dei numeri di Fibonacci e metterla dentro una variabile chiamata "totaleSomma", ovviamente vogliamo fermarci, altrimenti il nostro programma andrebbe avanti all'infinito, quindi vogliamo continuare la somma fino a quando totaleSomma non supera 1000000.

In questo esempio non sappiamo quante somme dobbiamo fare, o meglio, in teoria lo sappiamo se ci mettiamo a fare a mano le singole somme, ma la cosa non ha molto senso, l'unica cosa che ci interessa in questo momento è che vogliamo continuare a sommare fino a quando totaleSomma non supera 1000000.

```
while (totaleSomma < 1000000)
```

Abbiamo detto che la guardia del while ci obbliga ad eseguire il blocco di codice fino a quando la condizione è VERA, quindi noi vogliamo continuare la somma fino a quando la variabile con il totale è più piccola del valore che vogliamo raggiungere, appena raggiunge tale valore possiamo uscire dal ciclo.

Vediamo il frammento di codice per risolvere il problema:

```
int totaleSomma          = 0;
int numeroFibonacci     = 0;
int nf_1                 = 1;
int nf_2                 = 0;

while (totaleSomma < 1000000)
{
    numeroFibonacci = nf_1 + nf_2;
    nf_2 = nf_1;
    nf_1 = numeroFibonacci;
    totaleSomma += numeroFibonacci;
}
```

Cerchiamo di spiegare il seguente codice, abbiamo la nostra variabile per la somma e la inizializziamo a zero, poi creiamo 3 variabili per il calcolo della successione di Fibonacci. La prima variabile "numeroFibonacci" contiene il numero calcolato che andremo a sommare, mentre le altre due variabili "nf_1" ed "nf_2" ci servono per il calcolo del numero, in pratica come abbiamo detto precedentemente ogni numero di Fibonacci è dato dalla somma dei due precedenti e le due variabili rappresentano proprio i numeri precedenti.

Come vedete ripetere il codice è molto semplice ed in teoria potreste limitarvi ad utilizzare il `while` per tutto, ma il linguaggio di programmazione ci fornisce altri due costrutti che andremo ad analizzare. Il secondo costrutto è il “do-while”, vediamo il codice:

```
do
{
    blocco di istruzioni
}
while (condizione)
```

Come vedete il codice è molto simile a quello del `while` ed in effetti il funzionamento è esattamente lo stesso, fai qualche cosa fino a quando la condizione è verificata, cosa ha di diverso il “do-while” rispetto al “while”? Nel “do-while” prima viene eseguito il codice e dopo si verifica la condizione, nel “while” esattamente il contrario, prima valuto e in caso sia verificata eseguo il codice. Quindi nel “do-while” il blocco di istruzioni viene **sempre eseguito almeno una volta**. Sinceramente in tanti anni di sviluppo codice credo di aver preferito il “do-while” al “while” solo una volta e per un caso davvero particolare, quindi molto probabilmente userete sempre il “while”, ma anche queste sono questioni di stile.

Arriviamo al costrutto più importante e complesso da usare, prima abbiamo sottolineato come il “while” sia particolarmente indicato per eseguire cicli di codice un numero di volte che non conosciamo a priori, ma se sappiamo esattamente quanti sono i cicli che intendiamo fare, allora possiamo usare il ciclo “for”, anzi direi che la scelta è quasi obbligata. Vediamo il codice e poi iniziamo a spiegarlo:

```
for(inizializzazione; condizione; incremento)
{
    blocco di istruzioni
}
```

Mi rendo conto che la struttura possa spaventare, ma vi assicuro che non è complessa da capire, dato che riprende cose spiegate nei tutorial precedenti. Il “for” ci permette di eseguire il blocco di istruzioni un numero di volte stabilito, ma come facciamo a dire quante volte lo vogliamo eseguire? Abbiamo bisogno di una variabile che ci tenga il conto dei passaggi fatti, quindi prendiamo la prima parte che dice “inizializzazione”, in quel punto inizializziamo la nostra variabile con il valore di partenza per il calcolo del numero dei cicli:

```
int i = 0;
```

La riga scritta sopra vi ricorda qualche cosa? Esattamente una dichiarazione di una variabile e la sua inizializzazione a zero. Ora vediamo che significa “incremento”:

```
i++;
```

E questa riga cosa vi ricorda? Proprio così, è esattamente l'incremento visto nei tutorial

precedenti che equivale al classico:

```
i = i + 1;
```

Giungiamo infine a “condizione”, come per “if”, “while”, “do-while”, abbiamo una condizione che deve essere soddisfatta, per esempio se vogliamo eseguire 100 cicli di codice ci basterà dire: Prendiamo una variabile contatore che chiamiamo “i”, la inizializziamo a zero e la incrementiamo di 1 ogni volta che viene fatto il ciclo fino a che “i” è minore di 100 (esattamente eseguiamo 100 cicli, da 0 a 99), come scriviamo in codice tutto questo?

```
int i;  
for(i = 0; i < 100; i++)  
{  
    blocco di codice  
}
```

Ora sembra complicato, ma usandolo e vedendolo usare, scoprirete che in realtà è molto semplice e utile, basta solo prestare un pochino di attenzione.

Nel prossimo tutorial andremo a prendere in esame gli array, in modo da poter vedere bene alcuni esempi di utilizzo di cicli e array insieme.

Manuel (aka Jekrom)

per segnalazioni : manuel.montini@gmail.com

<http://iphonecodes.wordpress.com/>

LICENZA:

Questo tutorial è rilasciato sotto licenza Creative Commons : “Attribuzione-Non commerciale-Non opere derivate 2.5 Italia”,

<http://creativecommons.org/licenses/by-nc-nd/2.5/it/>