CICLI E ITERAZIONI

Le ripetizioni aiutano, nella vita come nella programmazione. Quando si sviluppano software e algoritmi, la possibilità di ripetere in modo automatico uno o più blocchi di istruzioni è fondamentale per l'efficienza e la versatilità dei propri programmi.

La possibilità di **ripetere ciclicamente una o più parti di codice** è un elemento di fondamentale importanza nello sviluppo del software. Basti pensare a programmi che svolgono operazioni cicliche e periodiche o ad algoritmi che operano in modo sistematico su moli di dati di dimensioni variabili. Per tutte queste e altre esigenze è fondamentale poter usufruire di **costrutti ciclici** (detti anche **'iterativi'**). Vediamo in modo più approfondito di cosa stiamo parlando, partendo da un esempio di tipo pratico. Ipotizziamo di voler scrivere un programma in grado di calcolare la somma dei primi N numeri interi, dove N è un numero inserito arbitrariamente dall'utente. Come è facile intuire, nessuno degli strumenti di cui siamo in possesso ci permette di svolgere questa operazione in modo generico, proprio perché la rigida esecuzione sequenziale usata finora vincola estremamente le potenzialità del software. Dobbiamo ricorrere quindi a costrutti ciclici, che ci permettano di eseguire ripetutamente parti di codice, vincolando la ripetizione a precise condizioni di stato del programma.

IL CICLO 'WHILE' ›››

La prima struttura iterativa che affrontiamo è il **ciclo 'while'**. Esso appartiene alla categoria dei cosiddetti **cicli precondizionati**, ossia **cicli nei quali la condizione di iterazione viene verificata prima di passare all'esecuzione delle istruzioni presenti nel corpo**. Nei paragrafi che seguiranno in questo articolo incontreremo anche altri tipi di cicli, caratterizzati da un approccio iterativo diverso. Come prima cosa soffermiamoci sulla **sintassi** del ciclo 'while':

```
while (condizione)
{
   istruzione1;
   istruzione2;
   ..............
   istruzioneN;
}
```

La **parola chiave 'while'** definisce il ciclo, mentre **'condizione'** è una qualsiasi espressione di tipo logico o matematico (elementare o composta) che si vuole utilizzare per **permettere al programma di decidere se mantenere attivo il ciclo o meno**. Una nota importante deve essere fatta proprio a proposito delle **condizioni di uscita dei cicli**: se i cicli non sono studiati correttamente, infatti, può verificarsi la presenza di **'loop infiniti'**, ossia di situazioni di potenziale stallo per il programma che ne bloccano

```
Esempio1: corretto
.......
i=0;
......
while (i < 10)
{
        Stampa la parola RoboZak e vai a capo;
        incrementa 'i' di 1;
}
```

```
Esempio2: loop infinito
.......
i=0;
......
while (i < 10)
{
        Stampa la parola RoboZak e vai a capo;
}
```

la corretta evoluzione temporale. Per fare un esempio confrontiamo i due esempi in fondo alla pagina precedente, costituiti da due porzioni di pseudocodice che scrivono a video in maniera ciclica la parola 'RoboZak'. Come si può osservare, 'l'ossatura' dei due esempi è la medesima in termini di istruzioni. Ciò che li contraddistingue, però, è **l'assenza all'interno dell'esempio 2 dell'istruzione 'incrementa 'i' di 1;'**. Ma è una differenza così rilevante? La risposta è ovviamente sì: nel primo caso, infatti, l'incremento progressivo della variabile 'i', originariamente inizializzata a 0, la porterà ad assumere il valore numerico 10, facendo così fallire la condizione di permanenza nel ciclo (i<10) e causando l'uscita dal costrutto iterativo. Nel secondo esempio, al contrario, per quante volte venga eseguito il corpo del ciclo la variabile 'i' manterrà sempre il valore iniziale (0), con la conseguenza il ciclo proseguirà all'infinito bloccando il programma stesso. **È quindi fondamentale tenere sempre ben presente le condizioni di iterazione e vagliare ogni possibile stato del ciclo.**

IL CICLO 'DO/WHILE' ›››

Dopo aver visto il costrutto 'while', in questo paragrafo presentiamo una seconda struttura ciclica, nota con il nome **'do/while'**, a causa delle parole chiave che permettono di definirla. La differenza principale dal ciclo 'while' si ritrova nel fatto che **il 'do/while' non appartiene ai cicli pre-condizionati, ma a quelli post-condizionati.** Il funzionamento è, di conseguenza, opposto al precedente in quanto **il corpo del ciclo viene eseguito prima di attuare il test sulla validità della condizione di permanenza.** Osserviamo, anche in questo caso, la sintassi del costrutto.

```
do
{

   istruzione1;
   istruzione2;
   istruzione3;
   ..............
   istruzioneN;

} while (condizione);
```

Come in tutti i cicli, anche per la struttura 'do/while' valgono le considerazioni fatte finora in merito ai rischi di stallo del programma.

'WHILE' E 'DO/WHILE', UN SEMPLICE CONFRONTO ›››

Vediamo, avvalendoci di due semplici esempi concreti, la differenza tra il comportamento dei due cicli appena presentati. I due esempi in alto nella pagina successiva mostrano come l'utilizzo di due corpi di codice analoghi possa produrre risultati differenti in funzione del fatto che vengano impiegati all'interno di un ciclo pre-condizionato o post-condizionato. Partiamo analizzando il primo caso. Lo pseudocodice ci dice che prima di tutto viene inizializzata la variabile 'i' con il valore 10, per poi giungere al ciclo 'while'. A questo punto il computer effettua il test 'i<10' che, fallendo, impone un 'salto' alla linea di programma successiva al ciclo stesso, ignorando completamente le istruzioni contenute nel blocco di codice del costrutto 'while'. Vediamo, invece, cosa accade nel secondo esempio. Anche in questo caso la variabile 'i' viene inizializzata con il valore 10. Successivamente, però, giunge al blocco 'do/while' che, essendo post-condizionato, esegue prima

QUICK REFERENCE ›››

Il ciclo 'while' è un costrutto iterativo pre-condizionato. Ripete il corpo del ciclo finché la condizione di permanenza è verificata.

```
while (condizione_di_permanenza)
{
    corpo del ciclo;
}
```

QUICK REFERENCE ›››

Il ciclo 'do/while' è un costrutto iterativo post-condizionato. Ripete il corpo del codice finché risulta 'validata' a livello logico la condizione di permanenza.

```
do
{
    corpo del ciclo;
} while (condizione_di_permanenza);
```

```
Esempio1: Ciclo while
........
i=10;
.....

while(i<10)
{
        Stampa a video la scritta RoboZak;
        incrementa i di 1;
}
```

```
Esempio2: Ciclo do/while
........
i=10;
.....

do
{
        Stampa a video la scritta RoboZak;
        incrementa i di 1; .
}while(i<10);
```

le istruzioni contenute nel suo 'corpo', per poi effettuare il test di permanenza. Viene quindi stampata a video la parola RoboZak e viene applicato l'incremento della variabile 'i', che al momento del test assumerà valore 11. È evidente la differenza di funzionamento dei due tipi di ciclo. Nello StepbyStep potrai sperimentare ulteriori esempi di utilizzo.

IL CICLO 'FOR' ›››

Il terzo e ultimo ciclo che vedremo in questo articolo è il ciclo **'for'**, forse la struttura iterativa più adottata dai programmatori C. Il ciclo ha una sintassi più complessa rispetto ai costrutti 'while' e 'do/while'. Prima di passare a commentarla nel dettaglio, puoi vederla schematizzata nello pseudocodice che segue.

```
for (inizializzazione_variabili;
condizione; modifica_variabili)
{

   istruzione1;
   istruzione2;
   istruzione3;
  ..............
   istruzioneN;

}
```

Si osserva che il ciclo 'for' è definito sulla base di **tre differenti blocchi di istruzioni, separati dal carattere ';'**. Il primo di essi permette di **inizializzare le variabili utilizzate nel corpo del ciclo** (ad esempio i contatori o gli accumulatori). **È possibile inizializzare una o più variabili, separando ogni singola operazione da una virgola.** Ad esempio:

```
for (var1=valore, var2=valore, ...
, varN=valore; ..... ; .....)
```

Il secondo blocco che definisce l'istruzione 'for' rappresenta, invece, **l'espressione logica di permanenza nel ciclo ed è concettualmente analoga alle condizioni utilizzate nei cicli 'while' e 'do/while'**. Con l'**ultimo parametro**, infine, definiamo le eventuali **operazioni di modifica delle variabili** che devono essere attuate a ogni ripetizione. Nell'esempio sotto vediamo un'applicazione del costrutto 'for' del linguaggio C che ci permette di stampare a video 10 volte la parola 'RoboZak' (è mostrata solo la parte di codice relativa al ciclo).

```
int i;
....
for(i=0; i<10; i++)
{
    printf("\nRoboZak");
}
```

In questo esempio, il ciclo ha inizio con l'assegnamento del valore 0 alla variabile 'i', che viene incrementata (istruzione 'i++') ogni volta che il corpo del codice viene eseguito. Infine, l'espressione 'i<10' afferma che l'iterazione prosegue fino quando 'i' non ha valore 10. Passiamo a fare un po' di pratica.

QUICK REFERENCE ›››

Il costrutto 'for' ripete il corpo del ciclo finché la condizione di permanenza è verificata. Permette di inizializzare variabili e di gestire automaticamente le loro procedure di modifica e incremento.

```
for (inizializzazione variabili; condizione di permanenza; modifica variabili)
{
    corpo del ciclo;
}
```

STEPbySTEP

IL CALCOLO DEL FATTORIALE

In questo StepbyStep potrai sperimentare tre programmi che implementano un semplice algoritmo per il calcolo matematico del 'fattoriale', scritto sfruttando i tre differenti cicli presentati finora. Il fattoriale è innanzitutto un operatore matematico che, preso un parametro N in ingresso, restituisce il prodotto tra tutti i numeri interi inferiori o uguali a N. Più semplicemente:

N! = N * (N-1) * (N-2) * (N-3) * ... * 1

Ad esempio:

4! = 4*3*2*1 = 24 **5! = 5*4*3*2*1 = 120**

CALCOLARE IL FATTORIALE CON IL CICLO 'WHILE'

In questo primo esempio puoi sperimentare il calcolo del fattoriale facendo uso del costrutto iterativo 'while'.

Il fattoriale con il 'while'

```
#include <stdio.h>
int main()
{
     int numeroletto, fattoriale;
     printf("\nInserisci un numero intero positivo");
     scanf("%d", &numeroletto); /* legge il numero */

     if(numeroletto>0)    /* se il numero letto è positivo procede*/
     {
          fattoriale=1; /* inizializziamo fattoriale a 1 */
          while(numeroletto>0)
          {
               fattoriale*=numeroletto; /*moltiplico fattoriale
                                          per il numero letto*/
               numeroletto--;     /* decremento numero letto */
           }

          /* stampa il risultato */
        printf("\nIl fattoriale del numero inserito e': %d ",
                        fattoriale);
     }
     else printf("\nIl numero che hai inserito non è positivo");
getch(); /* attende la pressione di un tasto */
return 1;
}
```

CALCOLARE IL FATTORIALE CON IL CICLO 'DO/WHILE'

Ecco, in questo secondo esempio, come appare il calcolo del fattoriale facendo uso del ciclo 'do/while'.

Il fattoriale con il 'do/while'

```
#include <stdio.h>
int main()
{
     int numeroletto, fattoriale;
     printf("\nInserisci un numero intero positivo");
     scanf("%d", &numeroletto); /* legge il numero */

     if(numeroletto>0)    /* se il numero letto è positivo procede*/
     {
       fattoriale=1; /* inizializziamo fattoriale a 1 */

       do
        {
         fattoriale*=numeroletto;   /* moltiplico fattoriale per
                                       numeroletto */
         numeroletto--;
        }while(numeroletto>0);

        /* stampa il risultato */
        printf("\nIl fattoriale del numero inserito e': %d ",
                     fattoriale);
     }
     else printf("\nIl numero che hai inserito non è positivo");
getch(); /* attende la pressione di un tasto */
return 1;
}
```

CALCOLARE IL FATTORIALE CON IL CICLO 'FOR'

Nel terzo esempio, puoi vedere riscritto il programma per il calcolo del fattoriale facendo uso del ciclo 'for'.

Il fattoriale con il 'for'

```
#include <stdio.h>
int main()
{
     int numeroletto, fattoriale;
     printf("\nInserisci un numero intero positivo");
     scanf("%d", &numeroletto); /* legge il numero */

     if(numeroletto>0)    /* se il numero letto è positivo procede*/
     {
       for(fattoriale=1; numeroletto>0; numeroletto--)
        {
```

```
        fattoriale*=numeroletto;  /* moltiplico fattoriale per
                                     numeroletto */
        }
        /* stampa il risultato */
        printf("\nIl fattoriale del numero inserito e': %d ",
                fattoriale);
    }
    else printf("\nIl numero che hai inserito non è positivo");
getch();  /* attende la pressione di un tasto */
return 1;
}
```

Come puoi osservare confrontando le tre versioni del programma, mentre i cicli 'while' e 'do/while' forniscono sorgenti abbastanza simili tra loro, la versione del programma ottenuta con il ciclo 'for' risulta molto più compatta ed essenziale, anche se lievemente meno 'lineare' (proprio questa compattezza rende il ciclo 'for' molto amato dai programmatori).

LA SOMMA DI 'N' NUMERI FORNITI DALL'UTENTE

In questo esempio scriveremo un programma di somma automatica di 'n' (numero acquisito da tastiera) numeri inseriti dall'utente.

Somma di 'n' numeri

```
#include <stdio.h>

int main()
{
   int N, numeroletto, somma, i;

   /*acquisizione del numero di interi da sommare */
   printf("\nInserisci il numero di numeri da sommare");
   scanf("%d",&N);

   /* inizializzo il ciclo for */
   for(somma=0, i=0; i<N; i++)
   {
     /* acquisisco un numero intero per ogni iterazione del
        ciclo */
     printf("\nInserisci il numero intero da sommare: ");
     scanf("%d", &numeroletto);

     /* sommo il numero acquisito alla variabile somma */
     somma+=numeroletto;
   }

   /* stampo in output la somma calcolata */
   printf("\nLa somma dei %d numeri inseriti e': %d", N, somma);
   getch();  /* attende la pressione di un tasto */
   return 1;
}
```

SOMMA INTERATTIVA DI 'N' NUMERI FORNITI DALL'UTENTE

Questo esempio è per molti versi analogo al precedente, con la differenza che l'utente non dovrà specificare il quantitativo di numeri da sommare subito dopo aver avviato il programma, ma potrà gestire l'inserimento in modo interattivo, comunicando di volta in volta l'aggiunta di ulteriori valori al processo di somma. Per questo esempio, ricorreremo all'uso del ciclo 'do/while'.

Somma interattiva

```
#include <stdio.h>

int main()
{
    int somma=0; /* accumulatore per il calcolo della somma */
    int numero_letto=0; /* numero acquisito */

    /* carattere acquisito: se carattere_letto è uguale a 's' o 'S'
       allora l'utente richiede l'inserimento di un nuovo numero,
       se ha valore diverso l'operazione è conclusa*/
    char carattere_letto;

    /* inizia il ciclo di lavoro del programma */
    do
    {
        /* messaggio di richiesta */
        printf("\nInserisci il numero da sommare: ");

        /* acquisizione del numero dall'utente */
        scanf("%d", &numero_letto);

        /* somma del numero alla variabile di accumulazione */
        somma += numero_letto;

        /* richiesta di proseguimento del programma */
        printf("\nVuoi sommare un altro numero? [scrivi 'S' o 's',
                qualsiasi altro carattere per termianare]");

        /* acquisizione della risposta dell'utente da tastiera */
        scanf("%c", &carattere_letto);

      /* mentre la variabile 'carattere_letto' ha valore 's' o 'S',
         procede con il ciclo di somma */
    } while( carattere_letto=='s' || carattere_letto=='S');


    /* Se l'utente ha inserito un carattere differente da 's' o 'S'
       il programma esce dal ciclo e stampa a video la somma */
    printf("\nLa somma dei numeri inseriti è: %d", somma);

    getch();  /* attende la pressione di un tasto */
    return 1;
}
```

UN CICLO NEL CICLO

In questo ultimo programma vedremo un esempio di 'incapsulamento' dei cicli, ossia di un ciclo definito all'interno di un altro ciclo. Nello specifico, applicheremo tale principio per creare un semplice programma che stampi a video una porzione di tavola pitagorica, che mostrerà automaticamente le tabelline dei numeri compresi tra 1 e 7 (inclusi).

La tavola pitagorica

```
#include <stdio.h>

int main()
{
   /* riga e colonna sono le due variabili che ci permetteranno di
       creare la tavola pitagorica */
   int riga, colonna;

   /* messaggio di benvenuto */
   printf("\nLA TAVOLA PITAGORICA\n");

   /* il ciclo for più esterno scorrerà le righe della tavola
       pitagorica. La condizione di uscita è riga<8 (ossia le prime
      sette righe della tabella */
   for(riga=1; riga<8; riga++)
   {
       /* il secondo ciclo for scorre le colonne (in modo analogo
          a quanto visto per le righe). Viene inizializzato
          e ripetuto per ogni riga della tabella. */
       for(colonna=1; colonna<8; colonna++)
       {
          /* stampa a video il prodotto tra riga e colonna e
            inserisce un carattere di tabulazione per spaziare */
         printf("%d\t", riga*colonna);
       }

      /* Quando il programma arriva a questo punto, ha terminato
          la scrittura di una riga. Deve essere mandato a capo il
          cursore */

       printf("\n"); /*manda a capo il cursore */
    }


   /* la tavola pitagorica è stata stampata. L'utilizzo dei
       cicli for, come puoi vedere, ci permette di automatizzare
       i processi di inizializzazione e incremento delle due
       variabili riga e colonna. */


   getch();  /* attende la pressione di un tasto */
   return 1;
}
```