Unità F3

Iterazione

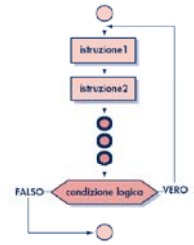
Obiettivi

- Conoscere le varie tipologie di strutture di controllo iterative
- Essere in grado di realizzare programmi con cicli e cicli annidati
- Essere in grado di realizzare programmi con iterazioni e selezioni
- Comprendere la differenza tra iterazioni a condizione iniziale e a condizione finale

Iterazione

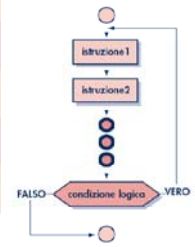
- L'iterazione è una **struttura** che consente di ripetere più volte l'esecuzione di un insieme di istruzioni.
- Due tipi di iterazione:
  - **iterazioni enumerative:** consentono di eseguire un blocco di istruzioni un numero predefinito di volte.
  - **iterazioni guidate da eventi:** non è noto a priori quante volte saranno eseguite.
    Tutto dipende dal fatto che un certo evento si verifichi o meno; se una condizione è vera (o è ancora vera) il blocco viene eseguito, altrimenti si passa all'istruzione successiva al termine del blocco.

Iterazioni postcondizionali



- L'**iterazione postcondizionale** è quella fondamentale, infatti tutte le altre possono essere definite a partire da questa.
- La logica è quella di eseguire, almeno una volta, il blocco delle istruzioni che fanno parte dell'iterazione, verificare alla fine una condizione e ripetere il blocco iterativo fintanto che la condizione risulti vera.
- Sono soddisfatti i requisiti della programmazione strutturata: un solo punto d'ingresso e un solo punto d'uscita.

Pseudolinguaggio: Ripeti ... FinQuando




```
Ripeti
  <istruzione1>
  <istruzione2>
  . . .
FinQuando
  (<condizione>)
```

- L'insieme di istruzioni (`<istruzione1>`, `<istruzione2>`…) è detto **corpo del ciclo**, viene eseguito e continua a essere eseguito fintanto che `<condizione>` non risulta vera.

Il ciclo do … while

- **Sintassi:**

```
do
  <istruzione>
while (<espressione>);
```

- L'istruzione continua a essere eseguita fintanto che **`<espressione>`** risulta vera.
- Nel caso, molto frequente, in cui il corpo del ciclo sia costituito da più istruzioni, si utilizza una coppia di parentesi graffe per racchiudere l'insieme delle istruzioni che fanno parte del ciclo.

```
do
{
  <istruzione1>;
  <istruzione2>;
  …
  <istruzioneN>;
} while (<espressione>);
```

Esercizio 1 – Iterazione guidata da eventi

- Scrivere un programma che richiede in input un numero intero e lo continua a richiedere finché non viene inserito un numero pari.

Esercizio 1 – Soluzione 1

```
/*
 * Richiede in input un valore intero pari
 * poi lo visualizza
*/
#include <iostream>

using namespace std;

int main()
{
    int valore;
    cout<<"Inserisci un numero pari ";
    do
        cin>>valore;
    while ((valore%2)!=0);
    cout<<endl<<valore<<" e' un numero pari"<<endl;
    system ("pause");
    return 0;
}
```

Esercizio 1 – Soluzione 2

```
#include <iostream>

using namespace std;

int main()
{
    int valore;
    do
    {
        cout<<"Inserisci un numero pari ";
        cin>>valore;
    }
    while ((valore%2)!=0);
    cout<<endl<<valore<<" e' un numero pari"<<endl;
    system ("pause");
    return 0;
}
```

Esercizio 1 – Soluzione 3

```
#include <iostream>
using namespace std;
int main()
{
    int valore;
    cout<<"Inserisci un numero pari ";
    do
    {
        cin>>valore;
        if ((valore%2)!=0)
            cout<<"Attenzione il numero inserito non e' pari. Inserirlo nuovamente ";
    }
    while ((valore%2)!=0);
    cout<<endl<<valore<<" e' un numero pari"<<endl;
    system ("pause");
    return 0;
}
```

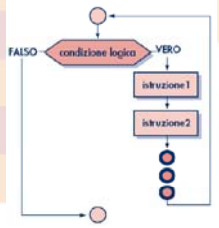
Esercizio 2 – Iterazione enumerativa

- Realizza un programma che determini la potenza intera di un numero utilizzando soltanto l'operatore prodotto.
  Il valore della potenza (l'esponente) e il numero (la base) sono immessi dall'utente.

Esercizio 2 – Soluzione

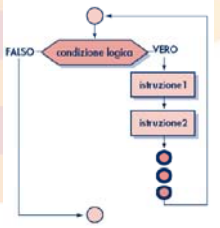
```
#include <iostream>
using namespace std;
int main()
{
    int base,esponente,potenza;
    int contatore;              // conta quante volte viene eseguito il ciclo
    cout<<"Inserisci la base ";    cin>>base;
    cout<<"Inserisci l' esponente ";    cin>>esponente;
    potenza = 1;        // inizializzazione della potenza
    contatore = 0;      // inizializzazione del contatore (il ciclo è stato eseguito 0 volte)
    do
    {
        potenza = potenza * base;
        contatore = contatore + 1;      // incremento del contatore (aumenta ogni volta
che viene eseguito il ciclo)
    }
    while (contatore<esponente);        // controllo di fine ciclo
    cout<<base<<" elevato a "<<esponente<<" = "<<potenza<<endl;
}
```

Iterazioni precondizionali



- Le iterazioni precondizionali hanno la condizione **in testa** al ciclo.
- La logica è quella di verificare una condizione e continuare a eseguire il blocco delle istruzioni che fanno parte dell'iterazione fintanto che questa risulti vera.
- Il blocco di istruzioni può, nel caso in cui la condizione risulti subito falsa, non essere mai eseguito.

Pseudolinguaggio: Finche'... Esegui... FineFinche'



- **Sintassi:**

```
Finche'
  (<condizione>)
  Esegui
    <istruzione1>
    <istruzione2>
    ...
FineFinche'
```


Il ciclo while

- Il linguaggio C mette a disposizione il ciclo while

```
while (<espressione>)
    <istruzione>;
```

- L'istruzione continua a essere eseguita fino a quando **`<espressione>`** risulta vera.
- Nel caso, molto frequente, in cui il corpo del ciclo è costituito da più istruzioni, si utilizza una coppia di parentesi graffe per racchiudere l'insieme delle istruzioni che fanno parte del ciclo.

Esempio while

```
/*
 * Richiede in input un valore intero pari
 * poi lo visualizza
*/
#include <iostream>

using namespace std;

int main()
{
    int valore;
    cout<<"Inserisci un numero pari ";
    cin>>valore;
    while ((valore%2)!=0)
        cin>>valore;
    cout<<endl<<valore<<" e' un numero pari"<<endl;
    system ("pause");
    return 0;
}
```

Esempio while - 2

```
/*
 * Richiede in input un valore intero pari
 * poi lo visualizza
*/
#include <iostream>

using namespace std;

int main()
{
    int valore;
    cout<<"Inserisci un numero pari ";
    cin>>valore;
    while ((valore%2)!=0)
    {
        cout<<"Il numero inserito non e' pari. Inserirlo nuovamente ";
        cin>>valore;
    }
    cout<<endl<<valore<<" e' un numero pari"<<endl;
    system ("pause");
    return 0;
}
```

```
/* Riceve come dato d'ingresso una sequenza di numeri terminante per 0, i numeri sono
    al massimo 100,
* non è conosciuta a priori la lunghezza della sequenza.
* Visualizza il valore del numero maggiore e di quello minore.*/
…
int main()
{
    int num;        // valore ricevuto in input
    int min;        // valore minimo
    int max;        // valore massimo
    int cont;       // conta il numero dei valori ricevuti in input
    // Inserimento del primo valore
    cout<<"Inserire un valore : ";
    cin>>num;
    // Inizializzazioni
    cont=1;         // un valore ricevuto
    min=num;        // il minimo è per il momento il primo valore
    max=num;        // il massimo è per il momento il primo valore
    while ((num!=0)&&(cont<100))
    {
        cout<<"Inserire un valore : ";
        cin>>num;
        cont++;             // ho ricevuto un altro valore
        if (num<min)
            min=num;        // ho trovato un nuovo minimo
        if (num>max)
            max=num;        // ho trovato un nuovo massimo
    }
    cout<<"Il valore minimo e'  "<<min<<endl;
    cout<<"Il valore massimo e' "<<max<<endl;
}
```

Esercizio

- Realizza un programma che determini la potenza intera di un numero utilizzando soltanto l'operatore prodotto. Il valore della potenza (l'esponente) e il numero (la base) sono immessi dall'utente.
  Utilizzare il ciclo while

Iterazioni enumerative

- In molti casi, è noto a priori quante volte il blocco di istruzioni interne al ciclo debba essere eseguito
- Le strutture iterative presentate possono essere utilizzate anche per risolvere questo tipo di problemi.
- Viene utilizzata una variabile (**contatore)** che ha la funzione di "contare" il numero di ripetizioni;
- Il contatore viene inizializzato prima del ciclo con un valore di partenza, e poi incrementato a ogni esecuzione;
- La condizione di ciclo controlla il raggiungimento del numero di iterazioni voluto.

Esempio

```
contatore <-1                          Inizializzazione
Finche' (contatore<=m) Esegui          Controllo terminazione
  prodotto <- prodotto*n               Corpo del ciclo
  contatore <- contatore+1             Incremento del contatore
FineFinche'
```


Problema

- **Problema**: *calcolare la somma, il prodotto e la media di* n *valori immessi dall'utente.*
- **Input:** la lunghezza *n* della sequenza dei valori in ingresso, gli *n* valori della sequenza.
- **Output:** somma, prodotto e media dei valori immessi.

Soluzione

Pseudocodifica

```
/* Somma, prodotto e media di n valori
   immessi dall'utente  */

Algoritmo OperazioniSuValoriImmessi
numero Di Tipo Reale   //valore immesso
somma Di Tipo Reale    //somma dei valori immessi
prodotto Di Tipo Reale //prodotto dei valori immessi
media Di Tipo Reale    //media dei valori immessi
i Di Tipo Intero
lunghezzaSequenza Di Tipo Intero  //numero di valori inseriti

Scrivi ("Calcola somma, prodotto e media di n valori ")
Scrivi ("Quanti valori vuoi immettere: ")
Leggi (lunghezzaSequenza)
somma <- 0
prodotto <- 1
Per i da 1 a lunghezzaSequenza Passo 1
  Scrivi ("Immetti un numero: ")
  Leggi (numero)
  somma <- somma + numero
  prodotto <- prodotto * numero
FinePer
media <- somma/lunghezzaSequenza
Scrivi ("Somma: ", somma)
Scrivi ("Prodotto: ", prodotto)
Scrivi ("Media: ", media)
FineAlgoritmo OperazioniSuValoriImmessi
```


Il ciclo for

- Ogni linguaggio presenta uno o più **costrutti di programmazione** per implementare le iterazioni enumerative
- Il C fornisce il ciclo **for** che, in realtà, è un costrutto molto potente e può essere utilizzato per qualsiasi tipo di iterazione.
- Per semplicità, lo utilizzeremo solo per implementare iterazioni con struttura analoga a quella del ciclo **Per** del nostro pseudolinguaggio.

For - sintassi

```
for (<istruzione1>;<espressione>;<istruzione2>)
  <istruzione3>;
```

<istruzione1> viene eseguita prima del ciclo e verrà utilizzata per inizializzare la variabile indice di ciclo;

<espressione> è la condizione logica che deve essere soddisfatta per eseguire il corpo del ciclo;

<istruzione2> viene eseguita a ogni iterazione e verrà utilizzata per l'incremento/decremento della variabile indice di ciclo;

<istruzione3> è il corpo del ciclo; nel caso sia composto da più istruzioni queste vanno inserite all'interno di un blocco delimitato da una coppia di parentesi graffe.

Esempi

```
/* Tabellina del 2 */
#include <stdio.h>

main ()
{
  int indice, prodotto;
  for(indice=1; indice<=10; indice=indice+1)
    {
     prodotto = 2 * indice;
     printf("2 per %d = %d\n",indice,prodotto);
    }
}
```

```
/* Tabellina del 2 in ordine inverso */
#include <stdio.h>

main ()
{
  int indice, prodotto;
  for(indice=10; indice>=1; indice=indice-1)
    {
     prodotto = 2 * indice;
     printf("2 per %d = %d\n",indice,prodotto);
    }
}
```


Esempi

```
/* Calcola la somma, prodotto e media di n valori immessi dall'utente */
#include <stdio.h>

main()
{
  float numero, somma, prodotto, media;
  int i, lunghezzaSequenza;
  printf("Calcola somma, prodotto e media di n valori\n\n");
  printf("Quanti valori vuoi immettere: ");
  scanf("%d", &lunghezzaSequenza);
  somma = 0;
  prodotto = 1;
  for(i=1; i<=lunghezzaSequenza; i=i+1)
  {
    printf("Immetti un numero: ");
    scanf("%f", &numero);
    somma = somma + numero;
    prodotto = prodotto * numero;
  }
  media = somma/lunghezzaSequenza;
  printf("Somma: %f\n", somma);
  printf("Prodotto: %f\n", prodotto);
  printf("Media: %f\n", media);
}
```


Cicli annidati

- Se all'interno del blocco iterativo si può inserire qualsiasi tipo di istruzione, questo significa che si può includere anche un'istruzione di selezione o un altro ciclo; in questo caso si parla di **annidamento** (o **nidificazione**) dei cicli.

Esempio

```
/* Cicli annidati per tavola pitagorica del 10 */
#include <stdio.h>

main()
{
 int r, c, prod;
 printf("Tavola pitagorica del 10\n\n");
 for(r=1; r<=10; ++r)
 {
   for(c=1; c<=10; ++c)
   {
     prod=r*c;
     printf(" %3d ",prod);  //tre spazi per ogni valore
   }
   printf("\n");  //"a capo" per la formattazione dell'output
 }
```


Il tipo di dato logico

- Il **tipo di dato logico** consente la definizione di variabili che assumono il valore vero o falso.
- Queste variabili vengono utilizzate all'interno degli algoritmi quando si intende rappresentare il verificarsi o meno di un determinato evento o condizione.
- **Pseudolinguaggio**
  `<nome_variabile> Di Tipo Logico`
- Nonostante l'implementazione risulti molto semplice, il tipo logico è presente solo in alcuni linguaggi di programmazione: in C viene sostituito dal **tipo intero** e dalla convenzione che associa vero al valore 1 e falso al valore 0 o in ogni caso a valori diversi da 1.

Esempio

```
/* Esempio di utilizzo del tipo di dato Logico */
Algoritmo Esempio_Tipo_Logico
X, Y Di Tipo Intero
L Di Tipo Logico

L ← vero  //Alla variabile L viene assegnato il valore vero
Se (L) Allora
  Scrivi ("Istruzione sempre eseguita")
Altrimenti
  Scrivi ("Istruzione mai eseguita")
FineSe
X ← 10
Y ← 5
L ← (X<Y)  //Alla variabile L viene assegnato il valore falso
L ← NOT(L) //ora il valore di L e' vero
FineAlgoritmo Esempio_Tipo_Logico
```