Istruzioni di iterazione

- Le istruzioni di iterazione forniscono strutture di controllo per esprimere la necessità di ripetere una certa istruzione durante il verificarsi di una certa condizione.
- Per il Teorema di Jacopini-Böhm, *una* struttura di controllo iterativa è sufficiente (insieme al blocco e a un' istruzione di scelta) per calcolare ogni funzione computabile.
- Tuttavia, per migliorare l' espressività del linguaggio, vengono rese disponibili vari tipi di istruzioni iterative:
  - `while( )`
  - `for( ; ; )`
  - `do … while( )`

Motivare "iterazioni" con un problema

- Dato un numero naturale *N*, calcolare la corrispondente somma dei primi *N* numeri naturali

Definiamo Somma(N) = 0 + 1 + 2 + … + *N*
allora

- Somma(0) = 0
- Somma(1) = 0 + 1 = Somma(0) +1
- Somma(2) = 0 + 1 + 2 = Somma(1)+2

…

- Somma(N) = Somma(N-1) + N

Soluzione del problema

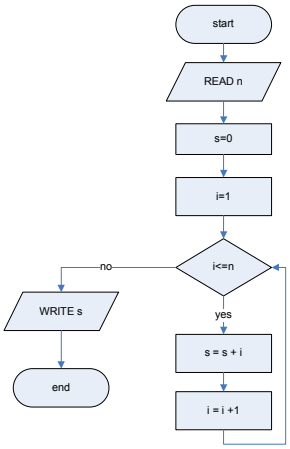
È quindi necessario

- **Inizializzare** la somma a 0: Somma(al passo 0)=0
- **Ripetere l'operazione** di somma per N volte: Somma(al passo i-esimo) = Somma(al passo i-1) + i

Rappresentazione dei dati

- La variabile ***n*** rappresenta il numero naturale *N* dato
- La variabile ***i*** è il **contatore**: rappresenta un ausilio per "scorrere" tutti i valori naturali da 1 fino a *N* ovvero conta il *numero di iterazioni*
- La variabile ***s*** rappresenta la somma calcolata fino al **passo *i-esimo***

Algoritmo e Programma

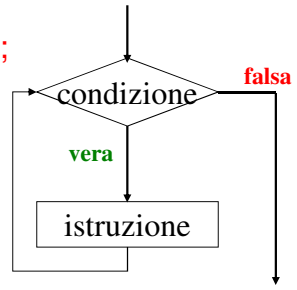



```
main ()
{
   int n, i, s;

   printf("Inserisci il numero: ");
   scanf("%d",&n);
   s = 0;
    i = 1;
   finché resta vero che (i<=n)
    { s = s+i;
      i++;
   }
    printf("La somma fino a %d è %d", n, s);
}
```

Istruzione iterativa "while"

<istruzione-while> ::=
while (*condizione*) < istruzione>;



- <istruzione> viene ripetuta per tutto il tempo in cui l'espressione *condizione* rimane vera
- **Se *condizione* è già inizialmente falsa, il ciclo non viene eseguito neppure una volta.**
- In generale, non è noto a priori *quante volte* <istruzione> verrà eseguita

Osservazioni

- Direttamente o indirettamente, **<istruzione> deve modificare prima o poi la *condizione***, altrimenti si ha un *ciclo infinito*
- Per questo motivo, **molto spesso <istruzione> è in realtà un blocco,** che contiene una ***istruzione di modifica*** di qualche variabile che compare nella *condizione*

Esempio (precedente)

```
main ()
 {
   int n, i, s;
   …
    finché resta vero che (i<=n)
     {s = s+i; i++; }
 }
```

Istruzione di modifica della condizione del while

```
main ()
 {
   int n, i, s;
   …
    while (i<=n)  {s = s+i; i++; }
  }
```


Esercizio

- Codificare l'algortimo per la ricerca del minimo di K numeri già descritto con i flowchart

<u>Rappresentazione dati</u>

- La variabile ***k*** rappresenta il numero di valori su cui deve essere calcolato il minimo
- La variabile ***x*** rappresenta il valore inserito dall' utente
- La variabile ***i*** è il **contatore**: conta il *numero di valori inseriti*
- La variabile ***min*** rappresenta il minimo fino all' *i-esimo valore inserito*

Codifica dell'algortimo di ricerca del minimo

```
#include <stdio.h>

main(){
   int k, i;
   int x, min;

   printf("Quanti valori vuoi inserire?");
   scanf("%d", &k);
   printf("Inserisci un valore?");
   scanf("%d", &x);
   min = x;
   i=1;
   while(i<k){
        printf("Inserisci un valore?");
        scanf("%d", &x);
        if (x<min)
            min = x;
        i=i+1;}
   printf("Il minimo valore inserito è %d", min);
}
```

Inizializzazione del minimo (caso base)

Confronto fra il numero di valori inseriti e quelli da inserire

Aggiornamento del minimo (caso induttivo)

Aggiornamento del numero di elementi inseriti

Esercizio

- Scrivere un programma che calcola il minimo tra un insieme di valori interi dati. *Il numero di valori inseriti non è noto a priori*

OSSERVAZIONE

- La definizione per induzione rimane invariata
- L'unica differenza è che non conosco il numero di valori che verranno inseriti e quindi *il numero di iterazioni non è più controllato da **k** e **i*** ma sarà l'utente a dire al programma quando non ha più valori da inserire

Rappresentazione dei dati

- La variabile ***min*** rappresenta il minimo tra i valori inseriti fino a quel momento dall'utente
- La variabile ***x*** rappresenta il valore inserito dall'utente
- La variabile ***continua*** è un **flag** per indicare la "fine della immissione dei dati"
  - ***continua*** può essere una variabile booleana (ovvero di tipo int) e assumere valore 0 (falso) o 1 (vero)
  - Oppure una variabile di tipo char e assumere valore 's' (si) oppure 'n' (no)

Codifica dell'algortimo di ricerca del minimo

```
#include <stdio.h>

main(){
   int x, min;
   int continua;

   printf("Inserisci un valore?");
   scanf("%d", &x);
   min = x;
   printf("Vuoi continuare (0-no /1-si)?");
   scanf("%d", &continua);
   while(continua){
        printf("Inserisci un valore?");
        scanf("%d", &x);
        if (x<min)
            min = x;
        printf("Vuoi continuare (0-no/1-su)?");
        scanf("%d", &continua);}
   printf("Il minimo valore inserito è %d", min);
}
```

Viene richiesto all'utente se desidera continuare

Condizione di continuazione

Viene richiesto all'utente se desidera continuare

Esercizi

- **Indicare l'output dei seguenti frammenti di codice:**

```
int x=2, t=-3;
while (x<=4 && fabs(t)%4<4) {t+=x*x; ++x; } printf ("%d %d\n",x,t);
```

_____

```
int numero=4, quanti=5, sum=10, min =2;
while (0<quanti)
{ numero=numero-1; if ((numero%2==0)&&(numero>min))
 {sum=sum-numero; quanti--;} };
printf("%d", sum);
```

_____

- Dato il seguente programma, indicare quante volte viene scritto il carattere 'A' al variare di **i**.

```
int i=…, j=1, k=3;
while(!(i%3+2*--j)||i%2 && k)
{printf("A"); i+=2;}
```

Se **i** vale 0: _____

Se **i** vale 4: _____

Esercizi

- **Indicare l'output dei seguenti frammenti di codice:**

```
int x=2, t=-3;
while (x<=4 && fabs(t)%4<4) {t+=x*x; ++x; } printf ("%d %d\n",x,t);
```

_____5 26_____

```
int numero=4, quanti=5, sum=10, min =2;
while (0<quanti)
{ numero=numero-1; if ((numero%2==0)&&(numero>min))
 {sum=sum-numero; quanti--;} };
printf("%d", sum);
```

____non esce dal ciclo_____

- **Dato il seguente programma, indicare quante volte viene scritto il carattere 'A' al variare di i:**

```
int i=…, j=1, k=3;
while(!(i%3+2*--j)||i%2 && k)
{printf("A"); i+=2;}
```

Se **i** vale 0: _____2_____

Se **i** vale 4: _____0_____

Istruzione iterativa "for"

- **È una evoluzione dell'istruzione *while***, rispetto a cui mira ad eliminare alcune frequenti sorgenti di errore:
  - mancanza delle necessarie ***inizializzazioni delle variabili***
  - mancanza dell' ***istruzione di modifica della condizione del ciclo***
    → **rischi di cicli infiniti**
- Per questo, **l'istruzione `for` comprende esplicitamente:**
  - una **espressione di inizializzazione**
  - una **espressione di modifica** del ciclo

Esempio

```
main ()
{  …
   i = 1;
   while (i<=n){
      s = s+i;
      i++; }
   …
}
```

```
main ()
{ …
      for (i=1; i<=n; i++)   s = s+i;
        …
 }
```

Espressione di inizializzazione del ciclo

Espressione di modifica della condizione del for

```
main()
{ …
  i=1;
  while(i<k){
     printf("Inserisci un valore?");
     scanf("%d", &x);
     if (x<min)
        min = x;
     i=i+1;}
    …
}
```

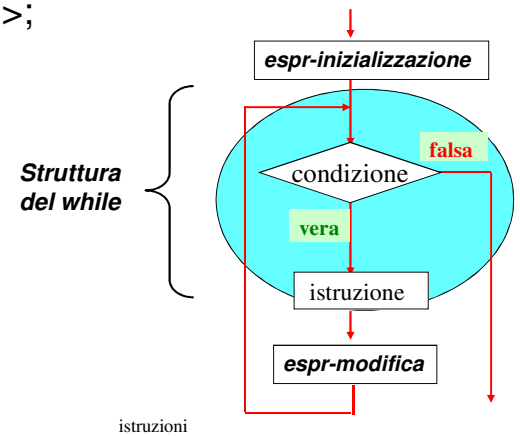
```
main()
{ …
  for(i=1; i<k; i++){
     printf("Inserisci un valore?");
     scanf("%d", &x);
     if (x<min)
        min = x;}
   …
}
```

Istruzione iterativa “for” (sintassi)

<istruzione-for> ::=

for (*espr-inizializz.* **;** *condizione* **;** *espr-modifica* **)**

<istruzione>;



Osservazioni

- Per inizializzare (o modificare) più variabili si può usare una *espressione composta* (operatore ,)
- L’espressione di modifica della condizione appare in modo più evidente (la sua mancanza “si nota”!)
- L’ <istruzione> che compare nel corpo del ciclo è *solo l’operazione vera e propria da ripetere*:
  - migliore leggibilità
- Ognuna delle espressioni può essere omessa, ma il separatore **;** deve rimanere
  Se manca *condizione*, la si assume *sempre vera*

Esercizio (*Specifica*)

- Codificare mediante i costrutti *while* e *for* l'algoritmo che esegue la moltiplicazione di due numeri letti in input mediante somma (già descritto con I flowchart)

Rappresentazione dei dati

- Servono 2 variabili (*int*) per rappresentare i valori da moltiplicare: **a**, **b**
- Per l'iterazione, servono due variabili ausiliarie (*int*), una come contatore e l'altra per contenere il prodotto calcolato ad ogni passo: **i**, **prod**

Esercizio (*Programma*)

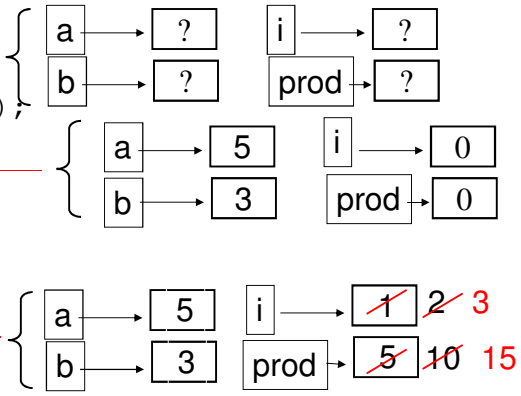
```
#include <stdio.h>
main()
 {
  int a, b, i, prod;
  printf("Inserisci i due valori da moltiplicare: ");
  scanf("%d", &a);
  scanf("%d", &b);
  prod=0; i=0;
  while (i<b)
   { prod=prod+a;
     i++;
     }
```

```
for (prod=0,i=0; i<b; i++)
      prod=prod+a;
```

```
  printf("Risultato del prodotto di %d e %d = %d\n",
  a, b, prod);
 }
```

Esercizio (*verifica per un caso*)

```
main()
 {
  int a, b, i, prod;
  printf("Inserisci …: ");
  scanf("%d", &a);
  scanf("%d", &b);
  prod=0; i=0;
  while (i<b)
  { prod=prod+a;
    i++;
     }
  printf("Risultato del prodotto di %d e %d =
  %d\n", a, b, prod);
 }
```



Output: Risultato del prodotto di 5 e 3 = 15

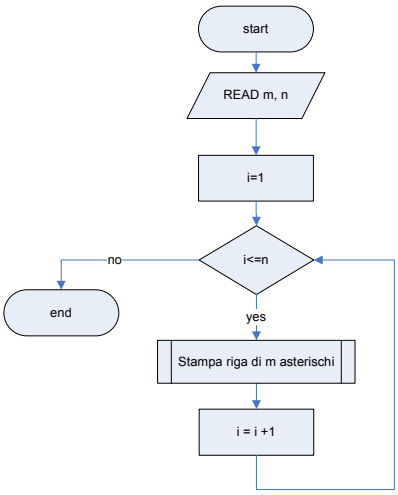
Esercizio (*Specifica*)

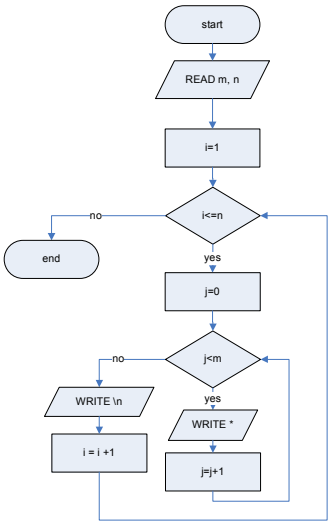
- Scrivere un programma che stampi un rettangolo costituito da nxm caratteri asterisco, dove i valori di m e n sono inseriti da tastiera.

Ad esempio: m=5 n=2 deve apparire il rettangolo

```
*****
*****
```

Esercizio (*Algoritmo*)





Esercizio (*Rappresentazione dei dati*)

- Servono 2 variabili (*int*) per rappresentare il numero delle righe e il numero delle colonne: **n**, **m**
- Servono, poi, due variabili ausiliarie (*int*) :
  - **i**, per contare il numero di righe stampate
  - **j**, per contare il numero di caratteri asterisco stampati sulla i-esima riga corrente

Esercizio (*Programma*)

```
#include <stdio.h>
main()
 {
  int n, m, i, j;
  printf("Inserisci il numero di righe e di colonne: ");
  scanf("%d", &n);
  scanf("%d", &m);
  for(i=1; i<=n;i++){
   for(j=1;j<=m;j++)
      printf("*");
  printf("\n");
      }
 }
```


Esercizio (*Quesito*)

- È possibile unificare le due inizializzazioni, i=1 e j=1, ovvero i seguenti frammenti di codice

```
for(i=1; i<=n;i++){
   for(j=1;j<=m;j++)
      printf("*");
   printf("\n");
      }
 }
```

```
for(i=1, j=1; i<=n;i++){
   for(;j<=m;j++)
      printf("*");
   printf("\n");
      }
 }
```

sono equivalenti?

NO, perchè j deve essere inizializzato a 1 per ogni valore di i da 1 a n

Esercizi (Varianti sul tema)

- Scrivere un programma che stampi un triangolo rettangolo costituito da 1, 2, …, n caratteri asterisco rispettivamente sulla 1° , 2° ,…,n° -esima riga, dove il valore di n è inserito da tastiera.

  Ad esempio: n=3 deve apparire il triangolo rettangolo

  ```
  *
  **
  ***
  ```

- Scrivere un programma che stampi un triangolo isoscele di base n inserito da tastiera
- Scrivere un programma che stampi un rettangolo nxm costituito da una cornice di caratteri cancelletto (#) e l'interno di caratteri asterisco (*), dove i valori di m e n sono inseriti da tastiera.

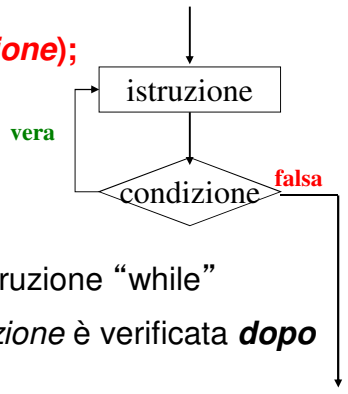
Ad esempio: m=5 n=3 deve apparire il rettangolo

```
#####
#***#
#####
```


Istruzione iterativa "do…while"

<istruzione-do-while> ::=

do < istruzione> **while (*condizione*);**



- È una "variante sul tema" dell' istruzione "while"
- A differenza del "while", la *condizione* è verificata ***dopo*** aver eseguito <istruzione>

  Quindi il ciclo *viene sempre eseguito almeno una volta*

Osservazioni

- Analogamente al "while", per evitare il ciclo infinito, **<istruzione> deve modificare prima o poi la *condizione***
- Si noti che, come nel caso del "while", si esce dal ciclo quando la condizione è falsa
- **È adatta** a quei casi in cui, per valutare *condizione*, è necessario aver già eseguito <istruzione>
  Esempio tipico: **Controllo valori di input**
- **Non è adatta** a quei casi in cui il ciclo può non dover essere *mai eseguito*.

Controllo valori di input

Esempio 1: *n deve essere positivo*

```
do
   scanf("%d", &n);
while (n<=0);
```

Esempio 2: *n deve essere un voto (ovvero compreso fra 0 e 30)*

```
do
    scanf("%d", &n);
while ((n<0) || (n>30));
```

Esempio 3: *n deve essere negativo o compreso fra 3 e 15*

```
do
    scanf("%d", &n);
while ((n>=0) && ((n<3) || (n>15)));
```

Esempio 4: risposta deve essere 's' o 'n'

```
do
    scanf("%c", &risposta);
while ((risposta != 's') && (risposta != 'n'));
```

Esercizio

- Modificare il programma del calcolo della data successiva affinché, in caso di data sbagliata, venga richiesto di inserire nuovamente la data.

Calcolo di funzioni matematiche

- Scrivere un programma per calcolare la funzione seno *sin(x)* definita dallo sviluppo della serie:

  $$\sin(x) = x/1! - x^3/3! + x^5/5! - x^7/7! + \ldots x^n/n!$$

  L'utente fornisce in input il valore di *x* e l'errore *e* ammissibile. Il calcolo deve continuare finchè il valore assoluto del termine n-esimo $|x^n/n!|$ è maggiore di *e*.
- Scrivere un programma per il calcolo della costante matematica *e* definita dalla sviluppo della serie:

  $$e=1/0!+1/1!+1/2!+1/3!+\ldots$$

  Il programma approssima il valore di *e* sommando *n* termini con *n* inserito da tastiera.

[Provare a sviluppare entrambi i programmi usando il costrutto for]

Esercizi

- Scrivere un programma che inverte l'ordine delle cifre che formano un intero positivo composto da un numero arbitrario di cifre. L'utente fornisce in input l'intero e il numero di cifre di cui è composto.
- Scrivere un programma che inverte l'ordine delle cifre che formano un intero positivo composto da un numero arbitrario di cifre. L'utente fornisce in input esclusivamente l'intero.
- Scrivere un programma che, forniti in input una data *d* nel formatto gg/mm/aa e un numero di giorni *n*, calcola la data dopo *n* giorni partendo da *d*.
  Ad esempio 3 giorni dopo il 27/11/2008 sarà il 30/11/2008, 4 giorni dopo il 27/11/2008 sarà il 1/12/2008, ecc. ecc.
- Scrivere un programma che dato un valore intero *n* stampa il fattoriale di tutti i numeri da 1 a *n*