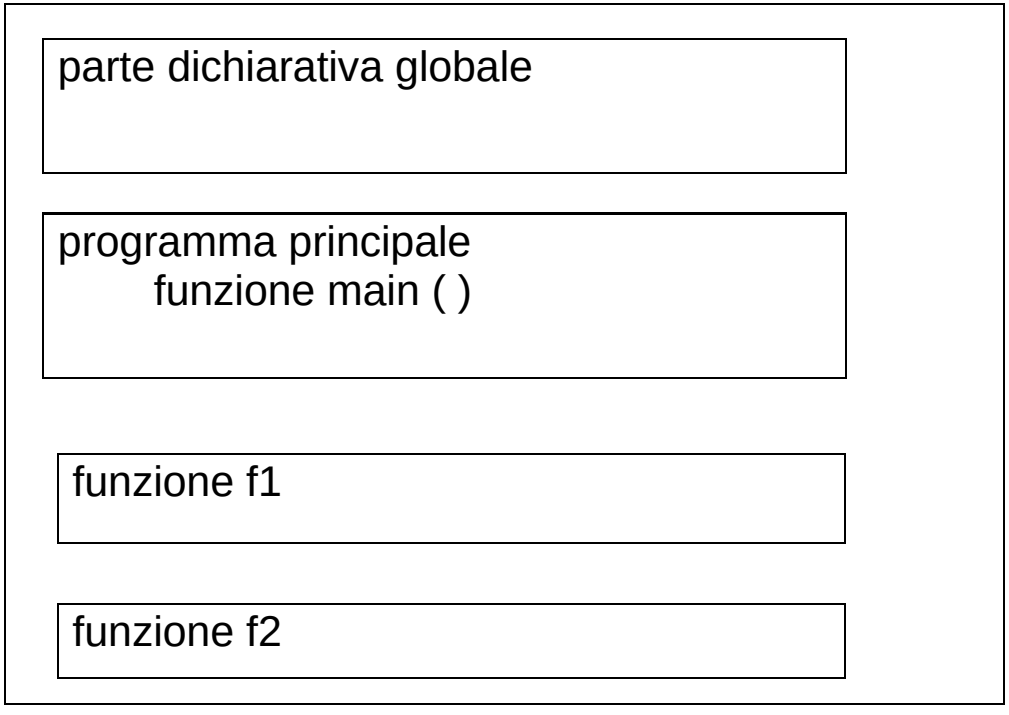
- 3° Modulo – parte a
  - Struttura di Programma
  -
  - Sequenza
  - Selezione
  - Iterazione
  - _____

http://www.elet.polimi.it/upload/martucci/index.html

STRUTTURA DI UN PROGRAMMA C - 1

programma C



Stile di scrittura di un programma ⇒ **leggibilità**

- scelta degli identificatori
- indentazione: «struttura grafica» che rispecchia la struttura logica

- uso di commenti

STRUTTURA DI UN PROGRAMMA C - 2

Parte dichiarativa globale:

- servizi (funzioni) importate da altri moduli (file), cioè definite e codificate in altri file

- «oggetti» (tipi di dati, variabili, costanti simboliche, prototipi di funzioni) visibili (utilizzabili) da tutto il programma, cioè da main e dalle altre funzioni.

Programma principale:

```
main ()

{

  parte dichiarativa
  locale



  parte esecutiva

}
```

parola riservata (identificatore di funzione)
appare una e una sola volta nel programma
definisce l'inizio dell'esecuzione
è (formalmente) una funzione

definisce l'insieme di «oggetti» usati dal programma principale per l'esecuzione. sono oggetti visibili (locali) a main.

insieme di istruzioni che costituiscono il programma principale

STRUTTURA DI UN PROGRAMMA C - 3

Parte dichiarativa locale

1. dichiarazione di costanti
2. definizione di «nuovi» tipi definiti dall'utente (ridenominazione)
3. dichiarazione di variabili
4. prototipi di funzioni

«Regole» sintattiche sulle dichiarazioni sia locali che globali:

- ogni identificatore usato deve essere prima definito
- ogni variabile usata deve essere prima dichiarata

Parte esecutiva: istruzioni (per tipologia)

- istruzioni di assegnamento
- istruzioni composte
- costrutti di (modifica del flusso di) controllo (costrutti condizionali, costrutti ciclici)
- «istruzioni» di ingresso e uscita
- chiamate di sottoprogrammi (funzioni)

ESEMPIO 1 - Dichiarazioni e istruzioni

```
#include <stdio.h>


main ( )

{
 int potenza, base;
 int esponente, i;         /* devono essere >=0*/

 printf("Inserisci il valore della base:");
 scanf("%d",&base);
 printf("Inserisci il valore dell'esponente:");
 scanf("%d", &esponente);

 potenza=1;
 i=0;

 while (i<esponente)
   {
   potenza=potenza*base;
   i=i+1;
    }

 printf("Potenza=%d \n", potenza);
}
```

ESEMPIO 2 - Dichiarazioni e istruzioni

```
#include <stdio.h>

main ( )

{
 int dato, sommatoria;

 sommatoria=0;
 printf("Inserisci il prossimo dato:");
 scanf("%d",&dato);

 while(dato!=0)
  /*ciclo di acquisizione e somma */
 {
  sommatoria=sommatoria+dato;
  printf("Inserisci il prossimo dato:");
  scanf("%d",&dato);
 }

 printf("Il valore della sommatoria e': %d",
sommatoria);
}
```

Programmazione strutturata (Dijkstra, 1969)

La programmazione strutturata nasce come proposta per regolamentare e standardizzare le metodologie di programmazione.

Obiettivo:

rendere piu' facile la lettura dei programmi (e quindi la loro modifica e manutenzione).

Idea di base:

La parte istruzioni (o parte esecutiva) di un programma viene vista come un comando (complesso, o *strutturato*) ottenuto componendo altri comandi elementari (assegnamento, chiamata di procedura) e non , mediante alcune regole di composizione (strutture di controllo).

Strutture di controllo:

- **concatenazione** (o composizione);
- **alternativa** (o istruzione condizionale)
- **ripetizione** (o iterazione)

eliminazione dei salti incondizionati

Programmazione Strutturata

☞ **Abolizione di salti incondizionati** (**goto**) nel flusso di controllo.

Vantaggi:

- **leggibilita`**

- supporto a metodologia di progetto **top-down:** soluzione di problemi complessi attraverso scomposizione in sotto-problemi, a loro volta scomponibili in sotto problemi, etc:
  La soluzione si ottiene componendo le soluzioni dei sottoproblemi attraverso concatenazione, alternativa e ripetizione.

- supporto a metodologia **bottom-up:** la soluzione di problemi avviene aggregando componenti gia` disponibili mediante concatenazione, alternativa e ripetizione (programmazione per componenti.

- facilita` di **verifica** e **manutenzione**

Teorema di Böhm e Jacopini

Le strutture di **concatenazione**, **iterazione** e **alternativa** costituiscono un insieme completo in grado di esprimere tutte le funzioni calcolabili.

☞ L'uso di queste sole strutture di controllo non limita il potere espressivo.

Un linguaggio composto dalle istruzioni:

lettura, scrittura (scanf, printf)

assegnamento

istruzione composta ({...})

istruzione condizionale(if...else...)

istruzione di iterazione (while...)

e` un ***linguaggio completo*** (in grado di esprimere tutte le funzioni calcolabili).

Istruzioni strutturate in C

- istruzione composta: (o blocco) {}
- alternativa: `if, switch`
- istruzioni di iterazione: `while, do, for`

Le istruzioni strutturate (o di controllo) sono alla base della **programmazione strutturata**.

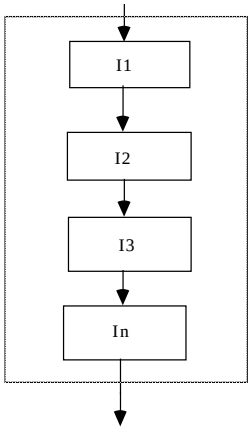
Istruzione composta { }

Determina l'esecuzione nell'ordine testuale delle istruzioni componenti.

Sintassi:

```
{
  <Dichiarazioni e Definizioni>
  <Sequenza di Istruzioni>
}
```

```
<sequenza-istruzioni> ::=
        <istruzione> {; <istruzione> }
```



Sintatticamente equivalente a una singola istruzione (strutturata).

Istruzione Composta

Dichiarazioni e definizioni:

E` possibile definire variabili che hanno visibilità e tempo di vita limitato al blocco stesso.

Ad esempio:

```
…
{ int A;
  A=100;
  printf("Valore di A: %d\n", A);
}
…
```

Formalmente, il corpo del main e` costituito da un'istruzione composta:

```
main()
{ int A;
  A=100;
  printf("Valore di A: %d\n", A);
}
```

Istruzione Composta

Esempio:

```
/*programma che letti due numeri a
  terminale ne stampa la somma*/

#include <stdio.h>

main()
{ int X,Y;

  scanf("%d%d",&X,&Y);
  printf("%d",X+Y);
}
```

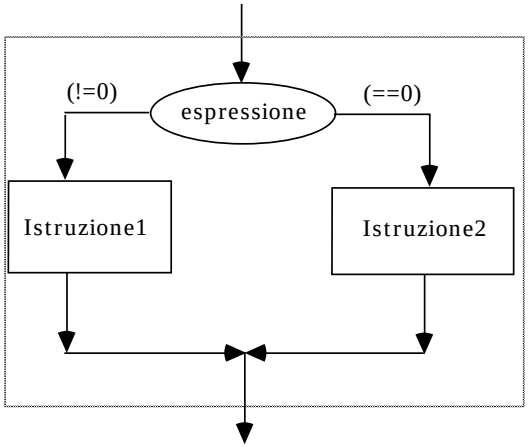

Istruzioni di alternativa

Istruzione if

seleziona l'esecuzione di una sola tra le istruzioni componenti in base al risultato di un'espressione logica (detta selettore).

```
if (<espressione>)
    <istruzione1>
[else
    <istruzione2>]
```

se il risultato di <espressione> è *vero* (cioe`, diverso da zero) viene eseguita <istruzione1>, altrimenti viene eseguita <istruzione2>.



Istruzione if

Esempio:

```
#include <stdio.h>
main()
{ int A, B;
scanf("%d%d", &A, &B);
if (B==0)
  printf ("B vale zero!\n");
else
  printf("Quoziente: %d\n", A/B);
}
```

La parte **else** dell'istruzione if è opzionale.

```
#include <stdio.h>
main()
{ int A, B;
scanf("%d%d", &A, &B);
if (B==0)
  return; /*termina l'esecuzione
            del main*/
printf("Quoziente: %d\n", A/B);
}
```

Indentazione:

l'"indent" delle linee del testo del programma rispetta l'annidamento delle varie istruzioni ➠ si aumenta la leggibilita` del programma (modifica piu` facile).

Istruzione if

if (<espressione>)
    <istruzione1>
else
    <istruzione2>;

☞ <istruzione1> ed <istruzione2> possono essere di tipo **qualunque**, semplice o strutturato (ad es. istruzione composta, o if).

Esempio:

```
   ...
int Eta;

if (Eta >=6)
   { if (Eta <=14)
        printf("%s","scolare");
   }
else
  {
     printf("%s","Non scolare");
     printf("%d",Eta);
   };
```

..

Oppure:

```
int Eta;

if ((Eta >=6) && (Eta <=14))
   printf("%s","scolare");
else {
      printf("%s","Non scolare");
      printf("%d",Eta);
   };
```


if: esempi

Esempio:

Programma che legge due numeri e determina qual e` il maggiore.

```
/* determina il maggiore tra due numeri
file c2.c */

#include <stdio.h>
main()
{
  int primo,secondo;

  scanf("%d%d",&primo,&secondo);
  if (primo>secondo)
       printf("%d",primo);
  else printf("%d",secondo);
}
```

Esempio

```
/* Programma che calcola le radici di
un'equazione di secondo grado*/

#include <stdio.h>
#include <math.h> /*lib. matematica*/
main()
{
  float a,b,c;
  float  d,x1,x2;

scanf("%f%f%f",&a,&b,&c);
if ((b*b) < (4*a*c))
 printf("%s","radici complesse");
else
 {
  d=sqrt(b*b-4*a*c);
  x1=(-b+d)/(2*a);
  x2=(-b-d)/(2*a);
  printf("%f%f",x1,x2);
 };
}
```


Esempio: if

Risolvere un sistema lineare di due equazioni in due incognite

$$a_1x + b_1y = c_1$$
$$a_2x + b_2y = c_2$$

$x = (c_1b_2-c_2b_1) / (a_1b_2- a_2b_1) = XN / D$
$y = (a_1c_2- a_2c_1) / (a_1b_2- a_2b_1) = YN / D$

Soluzione:

```
#include <stdio.h>
main()
{
  float   A1,B1,C1,A2,B2,C2,XN,YN,D;
  float  X,Y;

  scanf("%f%f%f%\n",&A1,&B1,&C1);
  scanf("%f%f%f%\n",&A2,&B2,&C2);
  XN = (C1*B2 - C2*B1);
  D =  (A1*B2 - A2*B1);
  YN = (A1*C2 - A2*C1);
  if (D == 0)
     {if (XN == 0)
        printf("sist. indeterm.\n");
      else
        printf("Nessuna soluz.\n");
     }
  else
     {X= XN/D;
      Y= YN/D;
      printf("%f%f\n",X,Y);
     }
}
```

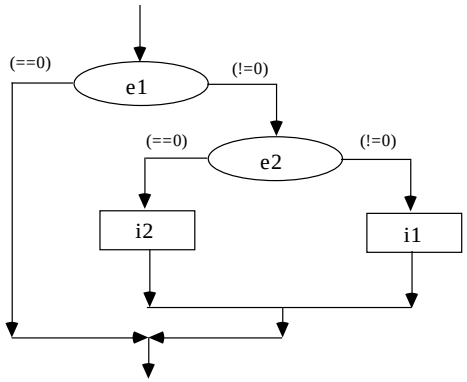
If annidati

if (<espressione>)
    <istruzione1>
else
    <istruzione2>;

- <istruzione1> ed <istruzione2> possono essere ancora istruzioni if.

☞ Si possono avere piu` **istruzioni if annidate**:

if (<e1>)
    if (<e2>)
        <i1>;
    else
        <i2>;



L'**else** si riferisce sempre all'**if** più "interno", se non specificato altrimenti.

```
if (n > 0)
 if (a > b)
     n = a;      /*n > 0 && a > b  */
 else   n = b; /*  n > 0 && a <= b */
```

☞ Se si vuole riferire l'**else** all'**if** piu` esterno, si usano le graffe:

```
if (n > 0)
 { if (a > b) n = a; }/*n>0 && a>b */
else
 n = b;          /*  n <= 0  */
```

If annidati

✏ **Esercizio:**

Dati tre numeri interi positivi che rappresentano i tre lati di un triangolo, si stampi il tipo del triangolo (equilatero, isoscele o scaleno).

Prima specifica:

```
main()
{
/*dichiarazione dati */
    /* leggi le lunghezze dei lati */
    /*se triangolo, stampane il tipo */
}
```


Codifica:

Come dati occorrono tre variabili intere per rappresentare le lunghezze dei segmenti.

```
/*dichiarazione dati */
unsigned  int   primo,secondo,terzo;

 /* leggi le lunghezze dei segmenti */
 scanf("%d%d%d",&primo,&secondo,&terzo);

 /*se triangolo, stampane il tipo */
if (primo + secondo>terzo)
  if (primo==secondo)/*det.il tipo */
     {if (secondo==terzo)
         printf("equilatero");
      else
         printf("isoscele");}
  else
     {if (secondo==terzo)
         printf("isoscele");
      else
         {if (primo==terzo)
             printf("isoscele");
          else
             printf("scaleno");}
     }
```

If-else

Single statement conditioned on the if():

form

```
if( expression1 )
    statement1;


if( expression2 )
    statement2;
else
    statement3;
```

example

```
if( hour <= 12 )
   time_of_day = am;


if( overdue )
    print_notice();
else
    print_receipt();
```

if-else

Multiple statements conditioned:

```
if( expression )
{
    statement1;
    statement2;
}
else
{
    statement3;
    statement4;
}
```

```
if( pymt < amt_due )
{
    post_acct_bal();
    print_notice();
}
else
{
    bal -= pymt;
    incr_credit_lim();
}
```

Nested if

```
if( expression1 )
   statement1;
else
   if( expression2 )
      statement2;
   else
      statement3;
```

```
if( expression1 )
   statement1;
else
{
   statement2;
   if( expression2 )
      statement3;
   else
      statement4;
}
```

extra statement causes braces to be required

ISTRUZIONI IF ANNIDATE

```
if (<condizione1>)
    if (<condizione2>)
        istr1;
        else
        istr2;
istr3;
```

`else` viene automaticamente associato all'ultimo `if` «aperto»

```
if (<condizione1>)
{
  if (<condizione2>)
      istr1;
}
else
  istr2;
istr3;
```

Nested if

What is the difference, if any?

```
if( expression1 )
   if( expression2 )
      statement1;
   else
      statement2;
```

```
if( expression1 )
   if( expression2 )
      statement1;
else
   statement2;
```

Nested if-else-if

This series of statements:

```
if( expression1 )
    statement1;
else
    if( expression2 )
        statement2;
    else
        if( expression3 )
            statement3;
        else
            statement4;
```

is commonly written:

```
if( expression1 )
    statement1;
else if( expression2 )
    statement2;
else if( expression3 )
    statement3;
else
    statement4;
```

if conditions

Using logical operators and expressions:

```
if( a > b )

if( a == b )

if( a )

if( !c )

if( a > b || c == d && d < a )          (what happens here?)

if( (a > b) || ((c == d) && (d < a)) )
```

Logical AND (&&) takes precedence over logical OR (||)

Precedence

- Use parentheses when in doubt or to improve readability:

Level	Operators
15L	-> . [] ()
14R	sizeof ++ -- ~ ! + - * & (cast) + - unari* indiretto &indirizzo
13L	* / %
12L	+ -
11L	<< >>
10L	< <= > >=
9L	== !=
8L	&
7L	^
6L	|
5L	&&
4L	||
3L	?:
2R	= *= /= %= += -= <<= >>= &= |= ^=
1L	,

Irwin Sheer

Superconducting Super Collider Laboratory

MS 2300, 2550 Beckleymeade Ave., Dallas, TX 75237

Tel: (214) 708-1050; Fax: (214) 708-6354

e-mail: Irwin_Sheer@ssc.gov

if with Null stmt

```
int x, y;

x = 3;
y = 5;

if( x == 3 )
    printf("%d\n", x);
else ;
    printf("%d\n", y);
```

What is the output?

Is the indentation proper?

Why or why not?

Istruzione Switch

Consente di selezionare l'esecuzione di una (o piu`) tra gli N blocchi di istruzioni componenti, in base al valore di una espressione.

Sintassi:

```
switch (<EspressioneIntegralType>)
{
case <costante1>: <blocco1>; [break;]
case <costante2>: <blocco2>; [break;]
...
case <costanteN>: <bloccoN>; [break;]
[default: <bloccoDiDefault>;]
}
```

- L'espressione e` detta **selettore**. Deve restituire un valore di tipo IntegralType (enumerabile).
- Ogni costante associata a una "etichetta" **case** deve essere dello stesso tipo del selettore.
- Un valore puo` comparire al piu` in un'etichetta.

Istruzione Switch

Significato:

Se l'espressione restituisce un valore uguale ad una delle costanti indicate (per esempio <costante1>), si esegue il <blocco1> e tutti i blocchi dei rami successivi.

- ☞ I blocchi non sono mutuamente esclusivi: possibilità di eseguire in sequenza più blocchi di istruzioni.

Pe ottenere la mutua esclusione tra i blocchi:

- ☞ Necessità di **forzare l'uscita** mediante l'istruzione **break** (che provoca l'uscita forzata dallo switch).

Esempio:

```
int X;
switch (X%2)
{
case 0: printf("X e` pari"); break;
case 1:printf("X e` dispari"); break;
}
```


Ramo di default:

E` possibile specificare un'etichetta **default**: essa viene eseguita per qualunque valore ritornato dal selettore.
In pratica, consente di eseguire un blocco nel caso in cui il valore dell'espressione non corrisponde ad alcuna etichetta.

Esempio: calcolo della durata di un mese

1ª soluzione:

```
#define   GENNAIO 1
#define   FEBBRAIO   2
#define   MARZO   3
...
#define   NOVEMBRE   11
#define   DICEMBRE   12
...
int mese, anno, giorni;
...
switch (mese)
{
case GENNAIO:  giorni = 31;  break;
case FEBBRAIO:
  if (<anno bisestile>) giorni = 29;
  else giorni = 28;
  break;
case MARZO:  giorni = 31;  break;
case APRILE:  giorni = 30;  break;
case MAGGIO:  giorni = 31;  break;
case GIUGNO:  giorni = 30;  break;
case LUGLIO:  giorni = 31;  break;
case AGOSTO:  giorni = 31;  break;
case SETTEMBRE:  giorni = 30; break;
case OTTOBRE:  giorni = 31;  break;
case NOVEMBRE:  giorni = 31;  break;
case DICEMBRE: giorni = 31;
}
...
```


2ª soluzione:

```
switch (mese)
{
case FEBBRAIO:
  if (<bisestile>) giorni = 29;
  else giorni = 28;
  break;
case APRILE:  giorni = 30;  break;
case GIUGNO:  giorni = 30;  break;
case SETTEMBRE:  giorni = 30;  break;
case NOVEMBRE:  giorni = 30;  break;
default:  giorni = 31;
}
```

3ª soluzione:

```
switch (mese)
{
case FEBBRAIO:
  if (<bisestile>) giorni = 29;
  else giorni = 28;
  break;
case APRILE:
case GIUGNO:
case SETTEMBRE:
case NOVEMBRE:  giorni = 30;  break;
default:  giorni = 31;
}
```

switch

- Like a special instance of if else-if else ...
- MUST have INTEGER condition for branching
- Evaluates integer expression then compares it to CONSTANT VALUES in each case
- Each constant “case” must be different like a special C label

switch

```
switch ( integer expression )
{
        case int-const1:   statement1;
                           statement2;
                           break;

        case int-const2:   statement3;
                           statement4;
                           break;
                 :
                 :
          default:         statement5;
                           statement6;
                           break;
}
```

switch

- Can have multiple statements per case
  - statements for a case need NOT be enclosed by braces
- Normally a case is concluded by a break
  - break causes immediate exit from switch
  - if NO break, ALL subsequent statements will be executed because switch is actually like a computed “goto”
  - break statement also used with iteration (looping) in C
- default is a “case” for all other conditions

```
char a, b, c, d, f, grade;
int actr, bctr, cctr, dctr, fctr, ictr;

printf("Enter letter grade: ");
scanf("%c", &grade);

switch (grade)
{
      case a:  ++actr;
               break;
      case b:  ++bctr;
               break;
      case c:  ++cctr;
               break;
      case d:  ++dctr;
               break;
      case f:  ++fctr;
               break;
      default: ++ictr;
               break;
}
```

/* given these declarations */

What's wrong with this section of code?

```
char a, b, c, d, f, grade;
int actr, bctr, cctr, dctr, fctr, ictr;

printf("Enter letter grade: ");
scanf("%c", &grade);

switch (grade)
{
      case 'a':  ++actr;
                 break;
      case 'b':  ++bctr;
                 break;
      case 'c':  ++cctr;
                 break;
      case 'd':  ++dctr;
                 break;
      case 'f':  ++fctr;
                 break;
      default: ++ictr;
                 break;
}
```

/* given these declarations */

I valori dei `case` devono essere costanti intere!!

Esempio: Una calcolatrice

Si vuole realizzare un programma che emuli il comportamento di una calcolatrice in grado di effettuare le operazioni aritmetiche su due operandi A e B.
Si supponga che l'utente immetta l'operazione nel formato:

A *op* B =

dove *op* puo` valere +, -, *, /, %, :.

```
#include <stdio.h>

main()
{
int A, B;
char op;

printf("Digita l'operazione che vuoi effettuare
  [formato da usare A op B=] :\n");
scanf("%d%c%d =", &A, &op, &B);
switch (op)
  {
  case '+': printf("%d\n", A+B);break;
  case '-': printf("%d\n", A-B);break;
  case '*': printf("%d\n", A*B);break;
  case '%': printf("%d\n", A%B);break;
  case '/': printf("%d\n", A/B);break;
  case ':': printf("%f\n",(float)A/(float)B);break; /*
divisione non intera*/
  default: printf("\n operazione non prevista\n");
  }
}
```

COSTRUTTI CICLICI

Costituiscono un modo per indicare che un blocco di istruzioni va rieseguito ciclicamente.

Elementi essenziali per un costrutto di iterazione:

- **inizializzazione**
  le variabili interessate, ed in particolare quelle usate nell'espressione della condizione, devono essere inizializzate prima della valutazione della condizione.
- **test**
  deve essere prevista una fase di valutazione della condizione (di permanenza nel ciclo) che determini la ripetizione o la terminazione del ciclo
- **modifica**
  almeno una delle variabili della condizione deve essere modificata all'interno del ciclo, in modo che prima o poi la condizione di ripetizione diventi falsa (terminazione).

Iteration

- 2 Basic types of repetition control

- Counter controlled

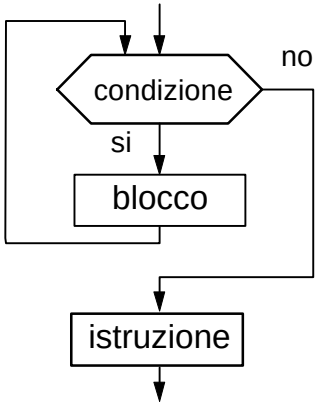
  loop is done until counter reaches a predetermined ending value

  needs: NAME of counter, INITIAL VALUE, INCREMENT amount, and FINAL VALUE

- Sentinel Controlled

  looping continues until some event occurs or some value is encountered

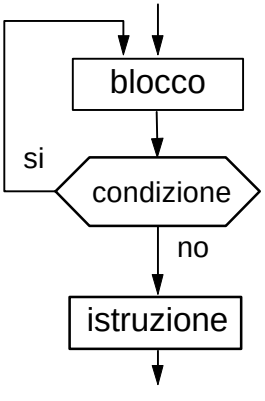
CICLO A CONDIZIONE INIZIALE (Sentinel Controlled)



Sintassi

```
while (<condizione>)
{
    <blocco>
}
<istruzione>
```

CICLO A CONDIZIONE FINALE



Sintassi

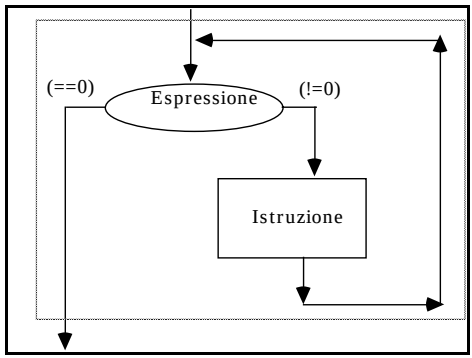
```
do
{
    blocco
} while (<condizione>);
<istruzione>
```

Istruzioni di iterazione: `while`

Consente la ripetizione dell'istruzione componente in modo controllato da una espressione.

Sintassi:

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



Significato

L'espressione (condizione di ripetizione) viene valutata all'**inizio di ogni ciclo**.

- L'<istruzione> viene eseguita finche` il valore dell'<espressione> rimane *vero* (diverso da zero).
- Si esce dal ciclo quando l'espressione restituisce un valore =0 (*falso* logico).
- Se inizialmente <espressione> ha valore zero, il corpo del ciclo non viene **mai** eseguito.

Esempio: somma dei primi dieci interi

```
#include <stdio.h>

main()
{
int somma, j;
somma=0;
j = 1;
while (j <= 10)
  {somma = somma + j;
   j++;
  }
printf("Risultato: %d\n", somma);
}
```

Esempio: scarto dei blank in lettura

```
char car=' ';
while (Car==' ')
  {
   scanf("%c",&Car);
  }
```


Esercizio:

Scrivere un programma che calcoli la media degli N voti riportati da uno studente.

```
/* Media di n voti */
#include <stdio.h>
main()
{
  int voto,N,i;
   float  media, sum;

   printf("Quanti sono i voti ?");
   scanf("%d",&N);
  sum=0;
/* ripeti ...*/
   printf("Dammi un voto:");
  scanf("%d",&voto);
  sum=sum+voto;
/* ... per N volte */
  media=sum/N;
  printf("Risultato: %f",media);
}
```

Esercizio (continua):

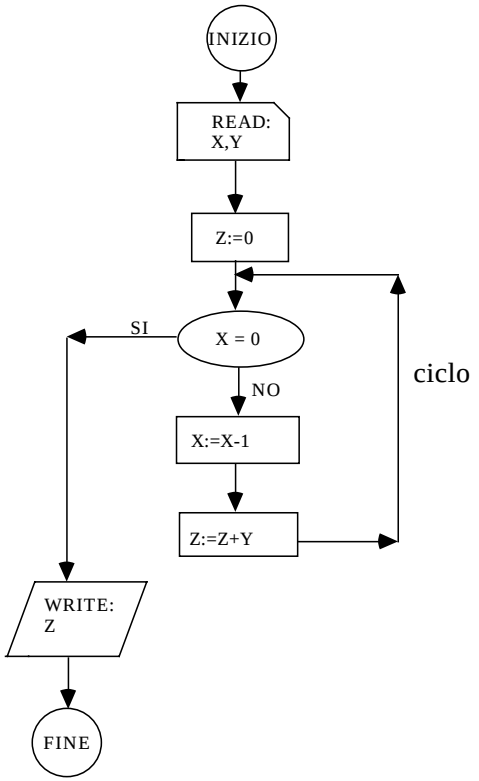
```
/* Media di n voti*/
#include <stdio.h>

main()
{
  int voto,N,i;
  float sum, media;

  printf("Quanti sono i voti ?");
  scanf("%d",&N);
  sum=0;
  i=1;
  while (i <= N)
    { /* corpo ciclo while */
    printf("Dammi il voto n.%d:",sum);
    scanf("%d",&voto);
    sum=sum+voto;
    i=i+1;
    }
  media=sum/N;
  printf("Risultato: %f",media);
}
```


✏ Esercizio:

Programma che calcola il prodotto X*Y come sequenza di somme (si suppone Y>0, X >=0).



Codifica:

```
/* moltiplicazione come sequenza di
   somme */
#include <stdio.h>
main()
{
 int  X,Y,Z;

 printf("Dammi i fattori:");
 scanf("%d%d",&X,&Y);
 Z=0;
 while (X!=0)
     {  /* corpo ciclo while */
          Z=Z+Y;
          X=X-1;
        }
 printf("%d",Z);
}
```


Cicli Innestati

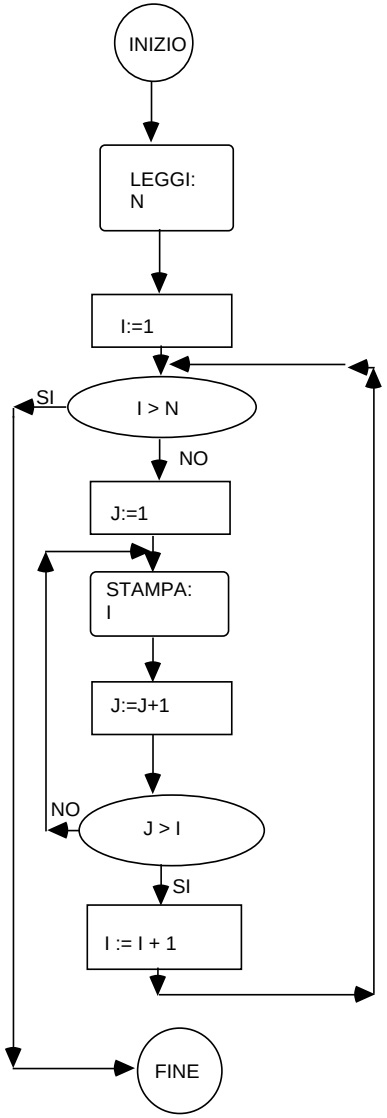
```
while (<espressione>)
        <istruzione>;
```

- <istruzione> puo` essere una qualunque singola istruzione (semplice o strutturata): eventualmente anche una istruzione while ➠ cicli **innestati**

✏ **Esercizio:**

Si legga un numero naturale N. Stampare una volta il numero 1, due volte il numero 2,..., N volte il numero N.

- Dominio di ingresso (0,1,2,...,N)
- Dominio di uscita ((), (1), (1,2,2),(1,2,2,3,3,3)...).



Codifica:

```
#include <stdio.h>
main()
{
  int    N,I,J;

  printf(“dammi N:”);
  scanf("%d",&N);
  I=1;
  while (I<=N)
   {  /* corpo ciclo esterno */
      printf(“Prossimo valore:”);
      printf("%d",I);
      J=1;
      while (J<I)
         {  /* corpo ciclo interno */
            printf("%d",I);
            J=J+1;
         }
      I=I+1;
   }
}
```

✏ **Esercizio:**
Stabilire se un numero naturale N e` primo.
(Un numero naturale N e` primo, se non e` divisibile per alcun numero intero minore di esso.)

Nota: non e` necessario eseguire tutte le divisioni di N per 2, 3, ..., (n-1), ma basta eseguire le divisioni per i naturali minori o uguali alla radice quadrata di N.

```
/* Numero primo */
#include <stdio.h>
#include <math.h>
main()
{
  typedef enum {false,true} boolean;
  int     N, I;
  float   N1;
  boolean primo;

  scanf("%d",&N);
  N1=sqrt(N);
  I=2;primo=true;
  while ((I<=N1) && (primo==true))
   {if (((N / I) * I) == N)
        {primo=false;}
    else  I=I+1; }
  if (primo==true)
      printf("numero primo");
  else printf("numero non primo");
}
```


✏ **Esercizio:**

Calcolo del **fattoriale** di un numero intero non negativo N:

- fattoriale(0) = 1
- fattoriale(N) = N * (N-1)*...*1=fattoriale(N-1)*N

```
/* Calcolo del fattoriale */
#include <stdio.h>
#include <math.h>
main()
{
  int     N, F, I;

  printf("Dammi N:");
  scanf("%d",&N);
  F=1;
  I=2;
  while (I <= N)
        { F=F*I;
          I=I+1; }
  printf("%s%d","Fattoriale: ",F);
}
```

Esempio:

Divisione tra numeri interi positivi attraverso somma e sottrazione.

```
/* divisione tra interi positivi */
#include <stdio.h>
main()
{
  unsigned int  X,Y,Quoz,Resto;

 scanf("%d%d",&X,&Y);
 Resto=X;
 Quoz=0;
 while (Resto >= Y)
       { Quoz=Quoz+1;
         Resto=Resto-Y; }
 printf("%d%s%d%s%d%s%d",
     X," diviso ", Y, " = ", Quoz,
     " resto ", Resto);
}
```


Istruzioni di iterazione: do..while

- Nell'istruzione **while**, la condizione di ripetizione viene verificata **all'inizio di ogni ciclo**

Istruzione do:

consente di ripetere l'istruzione eseguendo il controllo a fine iterazione.

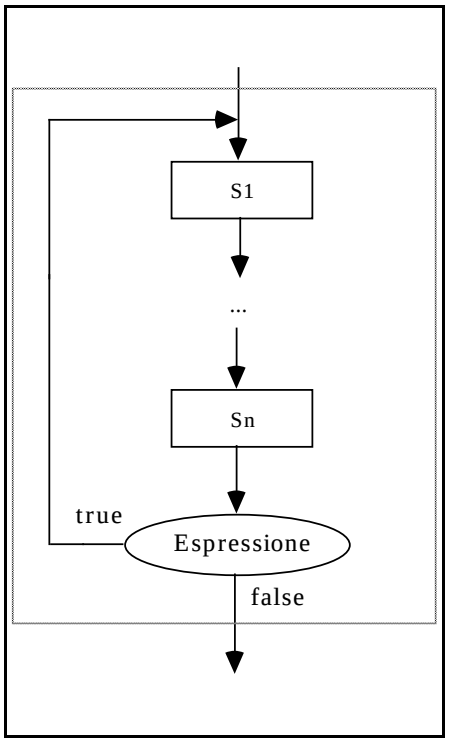
Sintassi:

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

- La condizione di ripetizione viene verificata **alla fine di ogni ciclo**

☞ • la prima ripetizione viene **sempre** eseguita.

Istruzione do:



Esempio:

```
do
  scanf("%c",&Car)
while (Car ==' ') /* salta spazi
                     bianchi */
```

Prima dell'esecuzione del ciclo, il valore di Car e` indeterminato.

Con il while:

Car=' ';
while (Car==' ') scanf("%c", &Car);

while() vs do while()

```
int x = 10, y = 1;

while ( x < 10 )
{
    if( x == 10 )
        y = x;
    ++x;
}

printf("x = %d\n", x);
printf("y = %d\n", y);
```

```
int x = 10, y = 1;

do
{
    if( x == 10 )
        y = x;
    ++x;
} while ( x < 10 );

printf("x = %d\n", x);
printf("y = %d\n", y);
```

Do these produce the same result?

Istruzioni Ripetitive: for

E` una istruzione di ripetizione particolarmente adatta per realizzare *cicli a contatore*.

Sintassi:

```
for(<espressione1>; <espressione2>;  <espressione3>)
  <istruzione>;
```

Significato:

- <espressione1> e` l'espressione di inizializzazione: viene eseguita una volta sola, prima di entrare nel ciclo
- <espressione2> rappresenta la condizione di permanenza nel ciclo (valutata all'inizio di ogni iterazione)
- <espressione3> e` l'espressione di passaggio al ciclo successivo (valutata alla fine di ogni iterazione).

for è equivalente a:

```
<espressione1>;          /* Inizializzazione  */
while (<espressione2>)   /* Condizione di
                                Ripetizione  */
{
      <istruzione>;
      <espressione3>;    /* InizioNuovoCiclo  */
}
```


Esempio while/for:

```
somma = 0;  /* while */
j = 1;
while (j <= n)
  {
   somma=somma+j;
    j++;
  }
```

con il for:

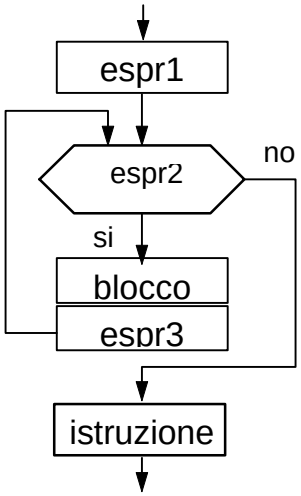
```
somma = 0; /* for */
for(j=1;j<=n;j++)
  somma = somma + j;
```

CICLO A CONTEGGIO: CICLO FOR

```
var_cont = val_iniz;
while (var_cont <= val_fin)
{
    <istruzioni del corpo del ciclo>
    var_cont = var_cont +1;
}
```

Il linguaggio C prevede anche un costrutto di ciclo che racchiude **inizializzazione**, **test** e **modifica** di una variabile.

```
for (espr1; espr2; espr3)
   {
    <istr del ciclo>
   }
   <istruzione>
```



espressione1: deve definire il valore iniziale della variabile di conteggio
espressione2: deve definire la condizione sul valore finale della variabile di conteggio
espressione3: deve definire la modifica della variabile di conteggio

```
for (var_cont=val_iniz; var_cont<=val_fin;var_cont++)
   {
    <istruzioni del corpo del ciclo>
   }
   <istruzione>
```

for()

```
for ( expr1 ; expr2 ; expr3 )
{
      statement1;
      statement2;
}
```

```
for ( ctr=1 ; ctr<10 ; ++ctr )
      sum += ctr;
```

expr1 - initializes the loop; done only once

expr2 - tests the condition; done at TOP of loop

expr3 - updates the loop; done AFTER statements in body of for()

Braces not needed if only one statement in body

Any or ALL expressions can be null statements

```
for ( ; ; );
```

Body may not be needed!

for()

Logic of for() is like that of while()

Compare the code:

```
int ctr;

ctr = 1;

while ( ctr < 10 )
{
      printf("Ctr = %d\n", ctr);
      ++ctr;
}
```

```
int ctr;

for( ctr = 1; ctr < 10; ++ctr )
      printf("Ctr = %d\n", ctr);
```

a for() loop

- initialize variables i, j, and k each to 1
- test to **end loop** when i > 10 **or** j > 3 * k
- increment i by 1, j by 2, and k by 2 * i
- print i, j, and k in the body of the loop

```
for ( i = 1, k = 1, j = 1;  i <= 10 && j <= 3 * k ; ++i, j += 2, k += 2 * i )
    printf("i = %d, j = %d, k = %d\n", i, j, k);
```

For:

```
for(<espressione1>; <espressione2>;  <espressione3>)
  <istruzione>;
```

- Ognuna delle espressioni **può essere omessa** (il punto e virgola deve rimanere)
- Se manca espressione2, si ha un ciclo infinito.

Cosa eseguono i seguenti **for** ?

```
for (i = 1;  i <= n;  i++)  printf("%d ",i);
for (;;)  { ... }
```


Esempio: fattoriale con istruzione for.

```
/* Calcolo del fattoriale */
#include <stdio.h>
#include <math.h>
main()
{
  int     N, F, I;

  printf("Dammi N:");
  scanf("%d",&N);
  F=1;
  I=2;
  for (I=2,F=1;I<=N;I++)
      F=F*I;
  printf("%s%d","Fattoriale: ",F);
}
```