Corso di Informatica A.A. 2009-2010

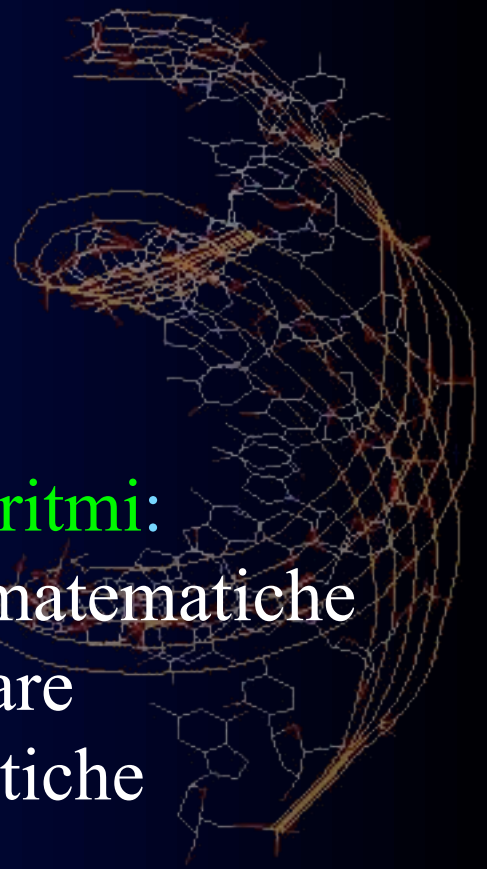
Lezione 6

Algoritmi e loro proprietà

- Proprietà formali degli Algoritmi

Cos'è l'informatica ?

L'informatica è la scienza della rappresentazione e dell'elaborazione dell'informazione



L'informatica è lo studio degli algoritmi:

- delle loro proprietà formali e matematiche
- delle loro realizzazioni hardware
- delle loro realizzazioni linguistiche
- delle loro applicazioni

Che cos'è un algoritmo ?

Un insieme ben *ordinato* e *finito*
di operazioni *non ambigue*
ed *effettivamente calcolabili* che,
applicate ad un insieme di condizioni iniziali,
produce un *risultato* e
termina in una *quantità di tempo finita*.

Un esempio di algoritmo

La somma dei primi 100 numeri interi (da 0 a 99!):

1. Poni *somma* = 0
2. Poni *indice* = 1
3. Finché *indice* è minore di 100 ripeti i passi 4. e 5.
    4. Aggiungi *indice* a *somma*
    5. Incrementa *indice* di 1
6. Stampa *somma*
7. Fermati

Proprietà formali e matematiche degli algoritmi

La prima proprietà è ovviamente la *correttezza*

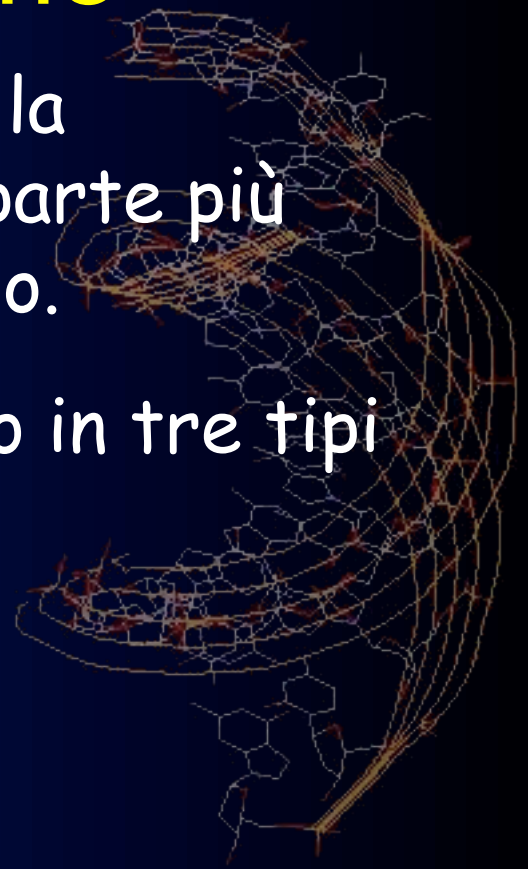
Ma ha una fondamentale importanza anche *l'efficienza*, che si misura rispetto alla risorsa *spazio* e rispetto alla risorsa *tempo*

Scrivere un algoritmo

La ricerca del giusto algoritmo per la soluzione di un dato problema è la parte più creativa del lavoro di un informatico.

Ogni algoritmo può essere scomposto in tre tipi fondamentali di operazioni:

- Operazioni sequenziali
- Operazioni condizionali
- Operazioni iterative



Operazioni *sequenziali*

- Istruzioni di elaborazione
  - Esempi:

    Poni *indice* = 1

    Aggiungi *indice* a *somma*

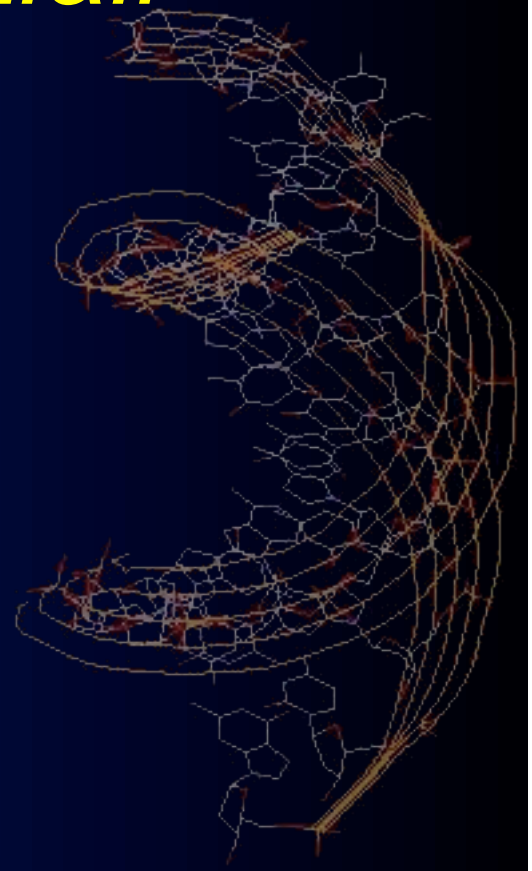
    Incrementa *indice* di 1

- Istruzioni di Input/Output (I/O)
  - Esempi:

    Acquisisci *misura*

    Acquisisci *errore assoluto*

    Stampa *errore relativo*



Operazioni *condizionali*

Esempio:

1. Se *misura* ≠ 0 allora
   2. Esegui algoritmo trova errore relativo
3. Altrimenti
   4. Stampa il messaggio "*errore relativo non definito*"
5. . . . . .

Operazioni *iterative*

Ciclo "do While":

Inizio ciclo: esegui

. . . . .

Fine del ciclo

Fino a che (condizione) rimane vera ripeti ciclo

eseguito almeno 1 volta

Ciclo While

Mentre (condizione) rimane vera ripeti:

. . . . . .

Fine del ciclo

eseguito 0 o più volte

Pseudocodice

Gli algoritmi presentati sono stati scritti secondo uno schema strutturato chiamato pseudocodice

Notate che:

- tutte le linee contengono un verbo specifico (azione)
- i passi sono numerati in modo tale che per ogni riga viene eseguita una sola azione
- se la linea richiede due passi viene suddivisa in linee successive indentate

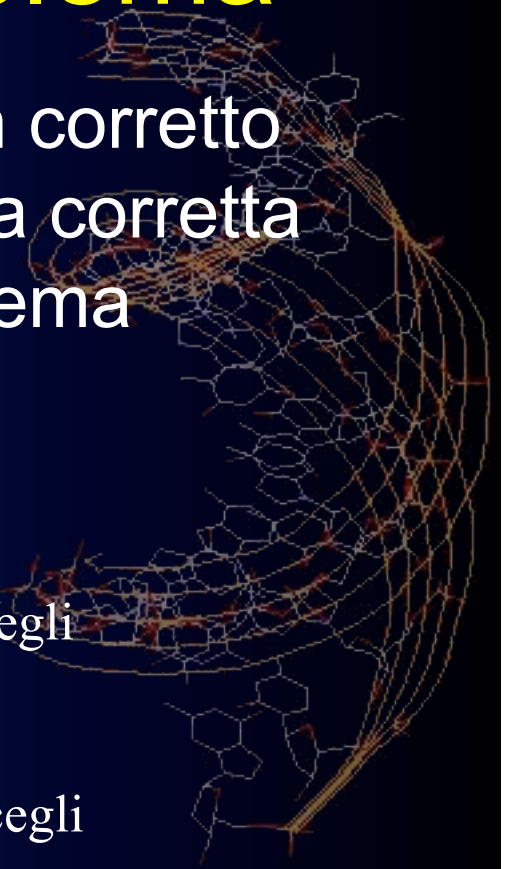
Correttezza di un algoritmo

Determinare la correttezza dell'algoritmo elaborato per un dato problema può essere arduo…le condizioni per essere ragionevolmente sicuri di averlo trovato sono:

- Comprensione effettiva del problema
- Validità della soluzione indipendentemente
  da condizioni particolari o
  dal valore dei dati in ingresso
- Livello di approssimazione del risultato sufficiente agli scopi di progetto

Comprensione del problema

La condizione necessaria per trovare un corretto algoritmo a un problema è ovviamente la corretta formulazione e comprensione del problema



Esempio: Qual è la via più breve fra le città A e B ?

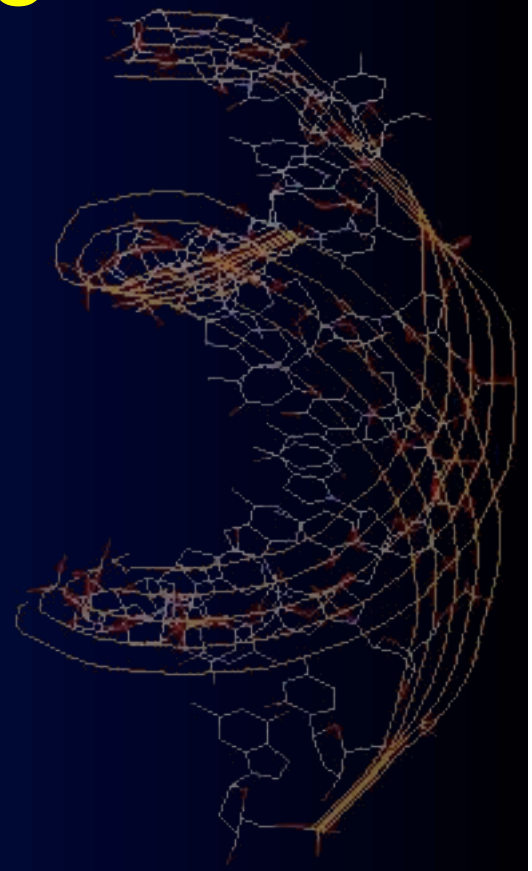
Algo 1: Esamina tutte le strade che congiungono A a B e scegli quella di lunghezza inferiore

Algo 2: Esamina tutte le strade che congiungono A a B e scegli quella il cui tempo di percorrenza medio è minimo

Dipendenza dai dati in ingresso

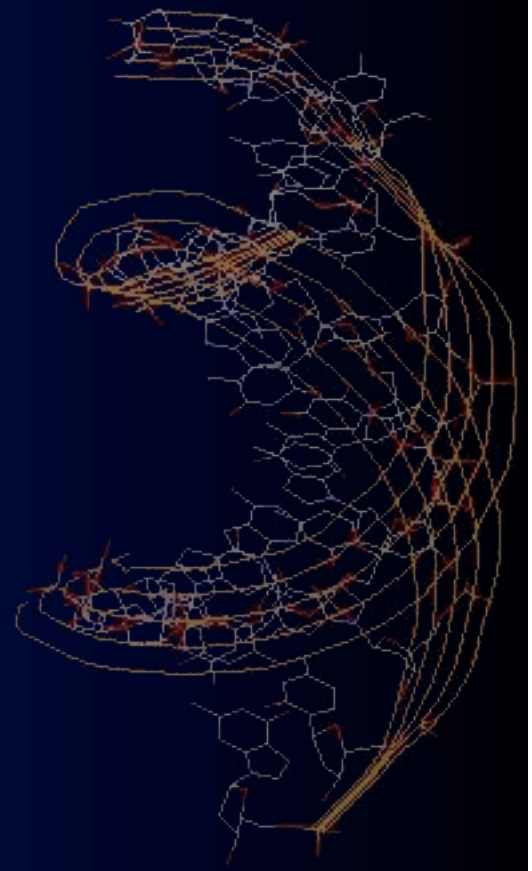
Algoritmo per risolvere equazioni algebriche di secondo grado:

1. Acquisisci *a, b, c*
2. Poni $x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$ $x_{21} = \frac{-b - \sqrt{b^2 - 4ac}}{2a}$
3. Stampa x_1 , x_2
4. Fermati



Dipendenza dalle condizioni del problema

Esempio: gittata di un proiettile:



1. Acquisisci v, θ
2. Poni $$G = \frac{2v^2 sin\theta \cos\theta}{g}$$
3. Stampa G
4. Fermati

…valido solo se il cannone è al suolo...

Livello di approssimazione richiesto

Spesso il risultato di un algoritmo è solo un'approssimazione, più o meno buona della risposta corretta.

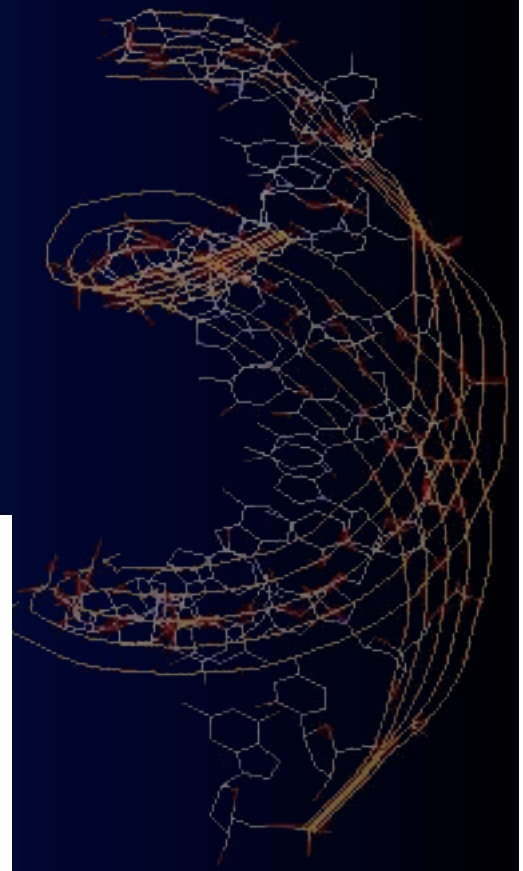
Ad esempio se il risultato è un numero reale, un calcolatore potrà fornire solo un numero finito di cifre significative: occorrerà stabilire in base all'uso che della soluzione si deve fare quale approssimazione è accettabile come corretta.

I costrutti del C

BUG BUSTER

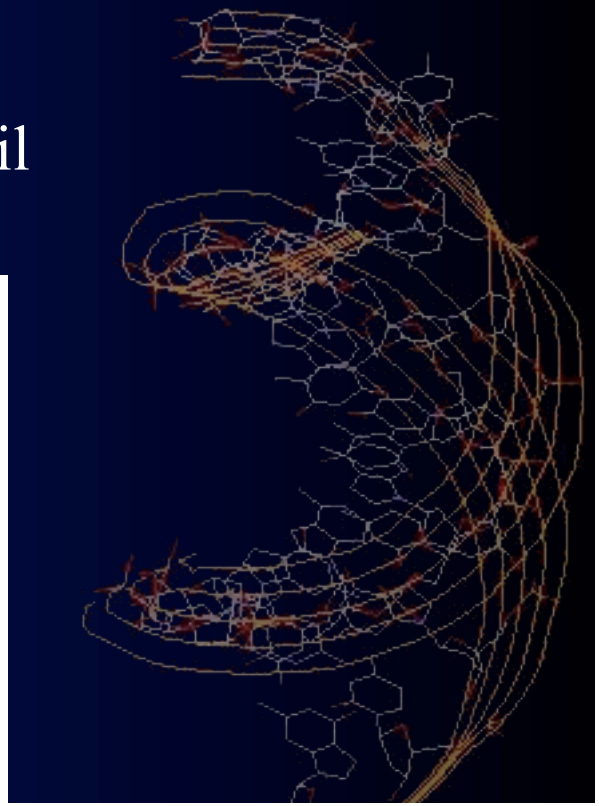
```
switch( answer )
{
    case 'Y': printf("You answered yes");
              break;
    case 'N': printf( "You answered no");
}
```

```
switch( choice )
{
    default:
        printf("You did not choose 1 or 2");
    case 1:
        printf("You answered 1");
        break;
    case 2:
        printf( "You answered 2");
        break;
}
```



Homework

Riscrivere le seguenti linee di codice usando il costrutto `if`

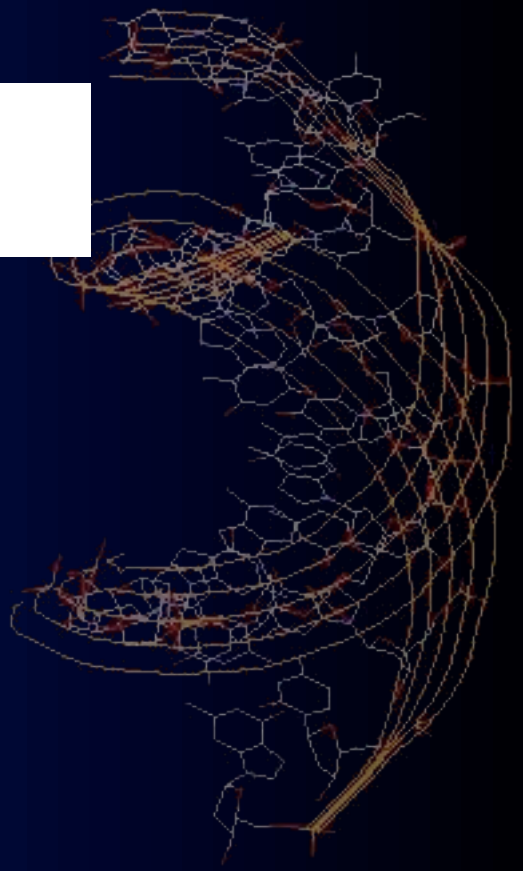


```
switch( choice )
{
    default:
        printf("You did not choose 1 or 2");
    case 1:
        printf("You answered 1");
        break;
    case 2:
        printf( "You answered 2");
        break;
}
```

```
if( choice == 1 )
    printf("You answered 1");
else if( choice == 2 )
        printf( "You answered 2");
    else
        printf( "You did not choose 1 or 2");
```

BUG BUSTER

```
for (counter = 1; counter < MAXVALUES; counter++);
    printf("\nCounter = %d", counter );
```



Homework

Quanto vale **x** al termine del seguente ciclo ?

```
for (x = 0; x < 100, x++) ;
```

Quanto vale **ctr** al termine del seguente ciclo ?

```
for (ctr = 2; ctr < 10; ctr += 3) ;
```

Scrivere un ciclo **for** per contare da 1 a 100 di 3 in 3

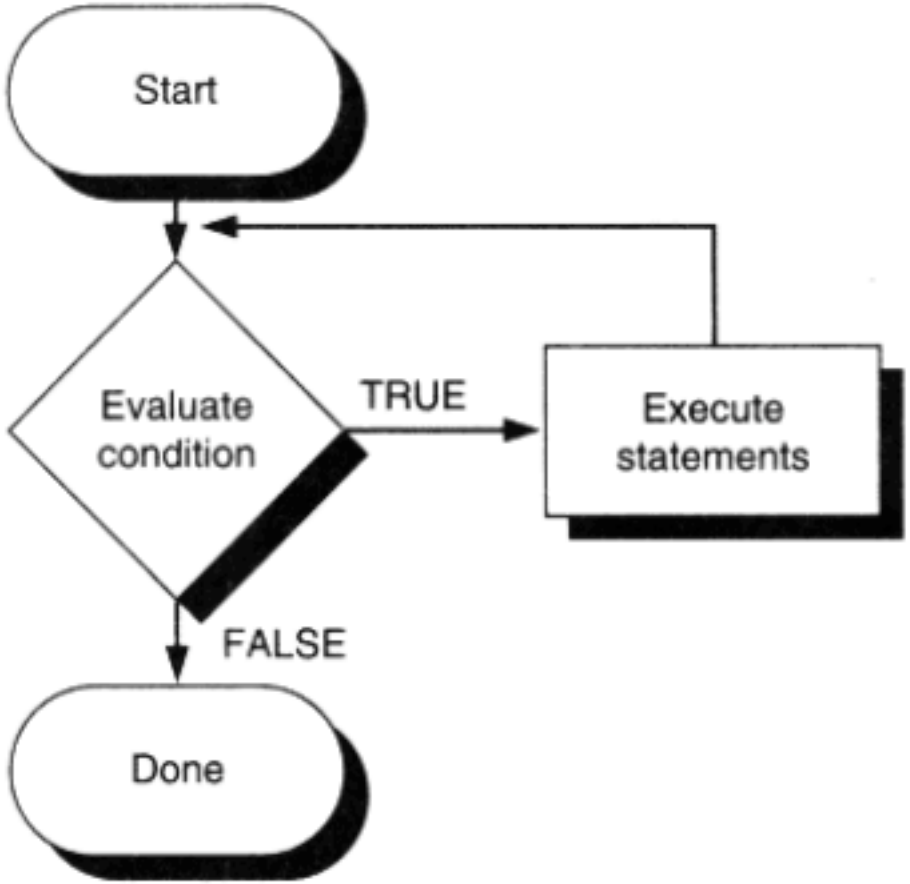
Ciclo `while`

L'istruzione `while` consente di eseguire ripetutamente un blocco di una o più istruzioni sino a quando è soddisfatta una condizione prefissata

L'istruzione `while` ha la seguente struttura

```
while (condition)
      statement;
```

```
while (condition)
      statements;
```



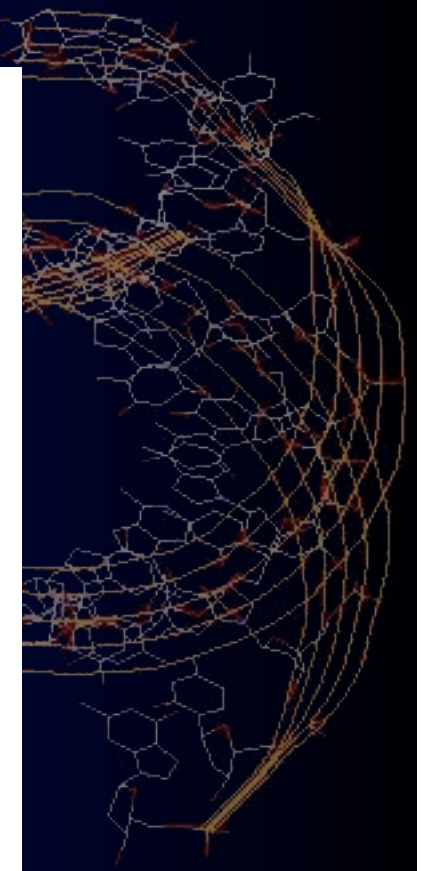
Ciclo while

La condizione di controllo viene verificata prima di eseguire qualunque istruzione: il ciclo potrebbe non venire mai eseguito.

```
somma = 0;
i = 0;
while (i<100)
{
somma = somma + i;
i++;
}
printf("Somma = %d",somma);
```

Esempio

```
/* Demonstrates a simple while statement */

#include <stdio.h>

int count;

int main()
{
    /* Print the numbers 1 through 20 */

    count = 1;

    while (count <= 20)
    {
        printf("%d/n", count);
        count++;
    }
return 0;
}
```



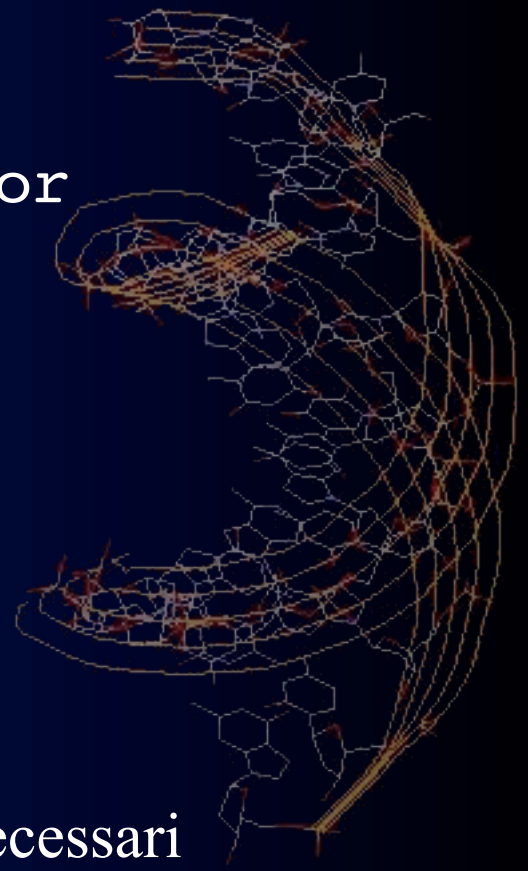
Ciclo `while`

L'istruzione `while` equivale ad un'istruzione `for` in cui mancano l'inizializzazione e l'incremento

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

equivale a

```
while (condition)
```



Quando l'inizializzazione e l'incremento sono necessari è sempre preferibile utilizzare l'istruzione `for`

Example 1

```
int x = 0;
while (x < 10)
{
    printf("\nThe value of x is %d", x );
    x++;
}
```

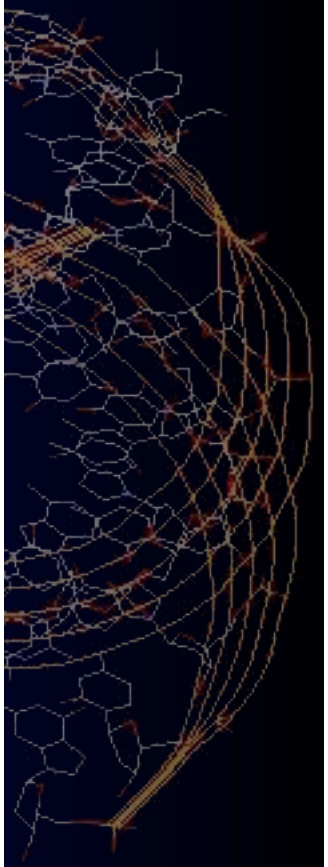
Example 2

```
/* get numbers until you get one greater than 99 */
int nbr=0;
while (nbr <= 99)
    scanf("%d", &nbr );
```

Example 3

```
/* Lets user enter up to 10 integer values        */
/* Values are stored in an array named value. If 99 is */
/* entered, the loop stops                                */
int value[10];
int ctr = 0;
int nbr;
while (ctr < 10 && nbr != 99)
{
    puts("Enter a number, 99 to quit ");
    scanf("%d", &nbr);
    value[ctr] = nbr;
    ctr++;
}
```

```c
/* Demonstrates nested while statements */

#include <stdio.h>

int array[5];

int main()
{
   int ctr = 0,
       nbr = 0;

    printf("This program prompts you to enter 5 numbers\n");
    printf("Each number should be from 1 to 10\n");

    while ( ctr < 5 )
    {
        nbr = 0;
        while (nbr < 1 || nbr > 10)
        {
            printf("\nEnter number %d of 5: ", ctr + 1 );
            scanf("%d", &nbr );
        }

        array[ctr] = nbr;
        ctr++;
    }

    for (ctr = 0; ctr < 5; ctr++)
        printf("Value %d is %d\n", ctr + 1, array[ctr] );

    return 0;
}
```



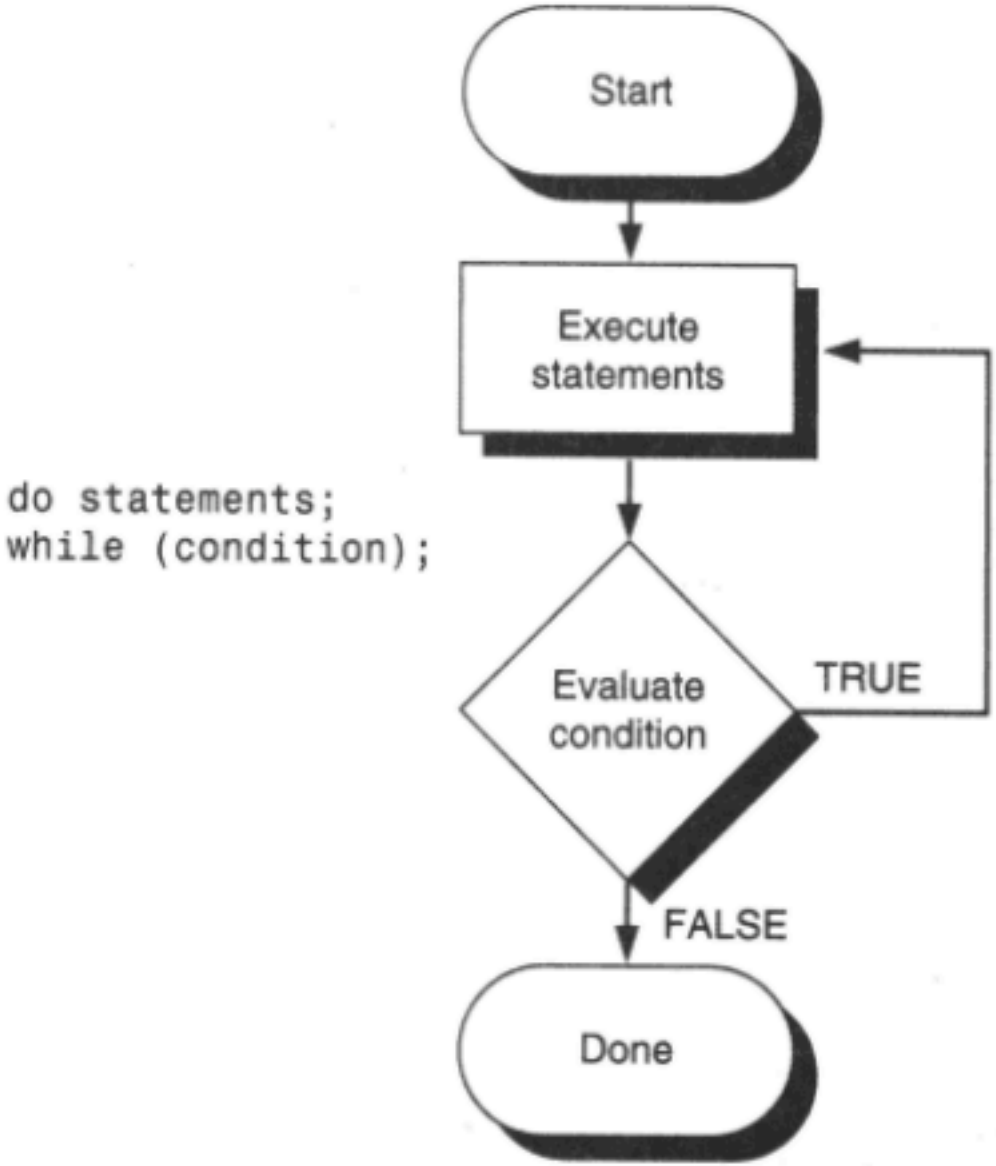
Ciclo do … while

L'istruzione `do … while` consente di eseguire ripetutamente un blocco di una o più istruzioni sino a quando è soddisfatta una condizione prefissata

La condizione è verificata al termine del ciclo

L'istruzione `do … while` ha la seguente struttura

```
do
      statement
while (condition);
```


Start
Execute statements
do statements;
while (condition);
Evaluate condition
TRUE
FALSE
Done

```c
/* Demonstrates a simple do . . . while statement */

#include <stdio.h>

int get_menu_choice( void );

int main()
{
int choice;

 choice = get_menu_choice();

 printf("You chose Menu Option %d\n", choice );

 return 0;
 }

 int get_menu_choice( void )
 {
 int selection = 0;

 do
 {
 printf("\n" );
 printf("\n1 - Add a Record" );
 printf("\n2 - Change a record");
 printf("\n3 - Delete a record");
 printf("\n4 - Quit");
 printf("\n" );
 printf("\nEnter a selection: " );

 scanf("%d", &selection );

 }while ( selection < 1 || selection > 4 );

 return selection;
}
```

INPUT/OUTPUT:

```
1 - Add a Record
2 - Change a record
3 - Delete a record
4 - Quit


Enter a selection: 8


1 - Add a Record
2 - Change a record
3 - Delete a record
4 - Quit


Enter a selection: 4
You chose Menu Option 4
```

Nested loop

Per *nested loop* (ciclo innestato) si intende un ciclo che è contenuto all'interno di un altro ciclo

Non ci sono limitazioni al numero di nested loops che si possono utilizzare, purché ogni ciclo *interno* sia completamente contenuto in quello *esterno*

Nested loop

```
for ( count = 1; count < 100; count++)
{
     do
     {
          /* the do . . . while loop */
} /* end of for loop */
     }while (x != 0);
```

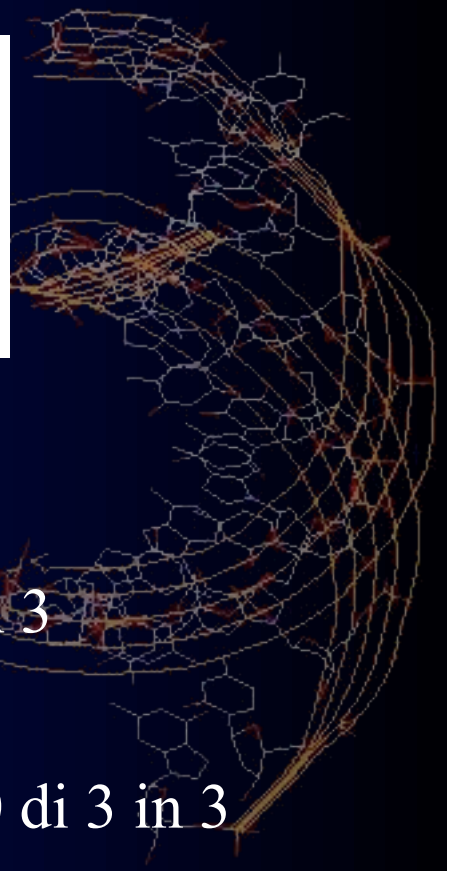
NO

```
for (count = 1; count < 100; count++)
{
     do
     {
          /* the do . . . while loop */
     }while (x != 0);
} /* end of for loop */
```

SI

BUG BUSTER

```
record = 0;
while (record < 100)
{
    printf( "\nRecord %d ", record );
    printf( "\nGetting next number . . . " );
}
```



Homework

Scrivere un ciclo `while` per contare da 1 a 100 di 3 in 3

Scrivere un ciclo `do ... while` per contare da 1 a 100 di 3 in 3