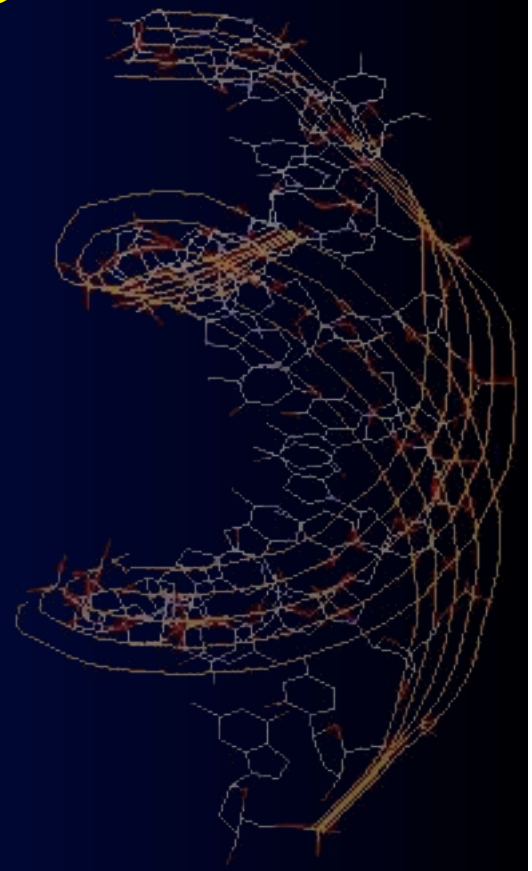
I costrutti del C

- Strutture condizionali
- Strutture iterative
- Introduzione alle funzioni

Le Strutture del C

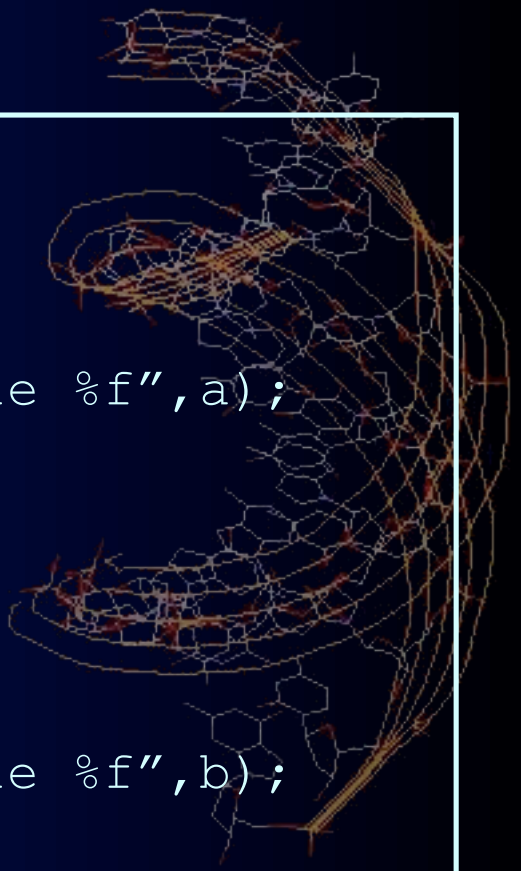
- Condizionali
  - `if-else` (`else if`)
  - `switch`
- Iterative
  - `while`
  - `for`
  - `do while`
  - `break` and `continue`



Il costrutto if –else (I)

```
if (a>b)
{
    max_ab = a;
    printf("Il massimo è a e vale %f",a);
}
else
{
    max_ab = b;
    printf("Il massimo è b e vale %f",b);
}
```

Indentazione



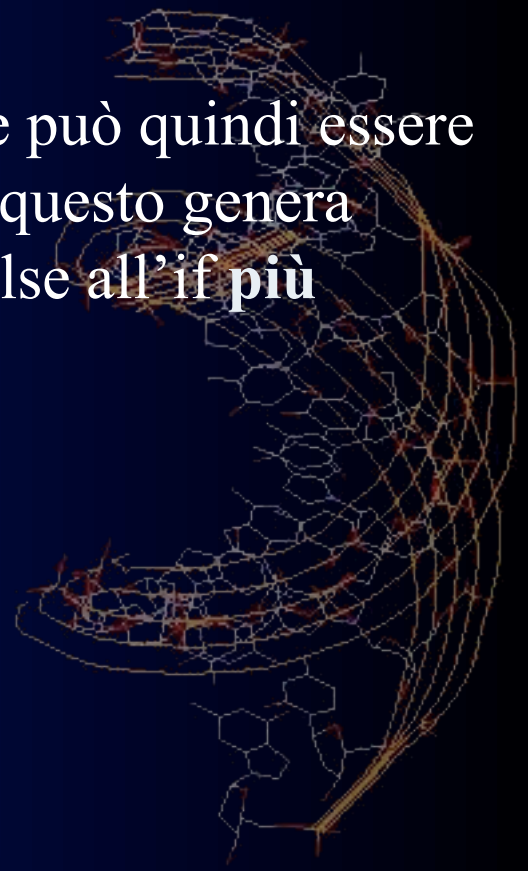
Il costrutto if –else (II)

```
if (a=b)
      printf("a e b sono uguali");
else
      printf("a e b sono diversi");
```

```
if (a==b)
      printf("a e b sono uguali");
else
      printf("a e b sono diversi");
```

Il costrutto if –else (III)

La parte else di un costrutto if-else è opzionale, e può quindi essere omessa. Nel caso di più costrutti if-else annidati questo genera un'ambiguità che viene risolta associando ogni else all'if **più interno** che ne è privo.



```
if (n>0)
      if(a>b)
            z=a;
      else
            z=b;
```

Il costrutto if –else (IV)

Quanto visto può talvolta essere fonte di errori:

```
if (n>0)
        if(a>b)
                z=a;
else
        printf("n non è positivo");
```

Il messaggio "n non è positivo" in questo caso viene stampato quando è verificato il primo if (n>0) e b è maggiore o uguale ad a !!!

Il costrutto if-else (V)

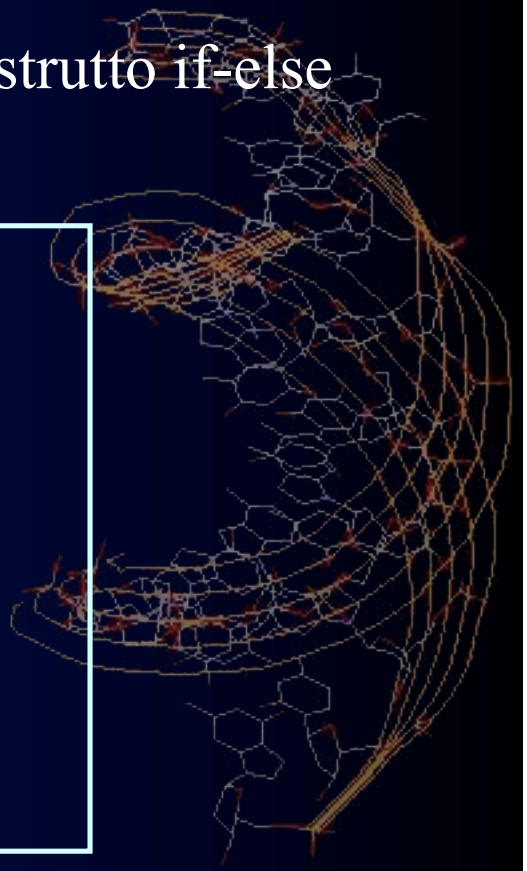
```
if (n>0)
{
        if(a>b)
                z=a;
}
else
        printf("n non è positivo");
```

Insomma USATE LE PARENTESI !!!!!

Il costrutto else if

Quando la scelta è fra più di due alternative il costrutto if-else può essere esteso utilizzando il costrutto else if :

```
if (n>0)
      printf("n è positivo");

else if (n==0)
      printf("n è nullo");
else
      printf("n è negativo");
```

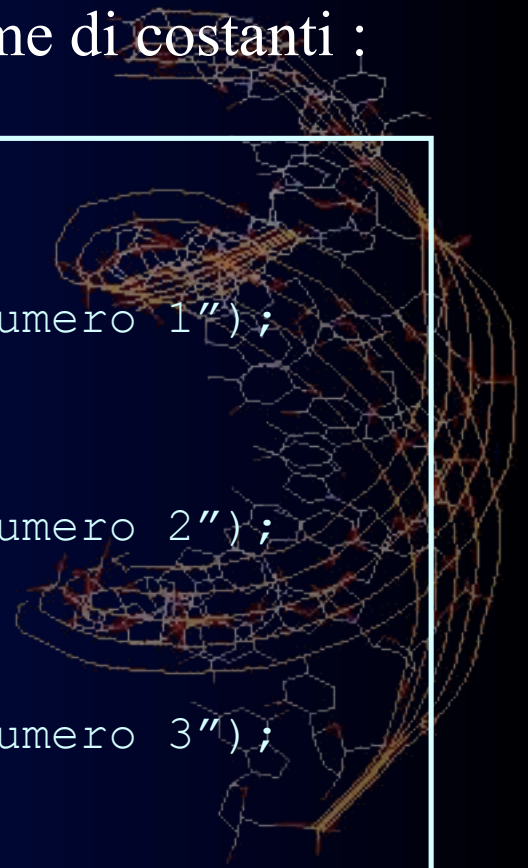


Le alternative vengono esaminate in sequenza, e l'ultimo else (opzionale) corrisponde alla scelta "nessuna delle precedenti".

Il costrutto switch (I)

Viene usato per una scelta multipla fra un insieme di costanti :

```
switch (nBusta){
        case 1:
                printf("Hai scelto la busta numero 1");
                break;
        case 2:
                printf("Hai scelto la busta numero 2");
                break;
        case 3:
                printf("Hai scelto la busta numero 3");
                break;
        default:
                printf("Non hai scelto una busta valida");
                break;
}
```



Il costrutto switch (II)

```
scanf("%c",&carattere_letto);
switch (carattere_letto){
      case 'a': case 'e' : case 'i' : case 'o':
      case 'u':
            printf("Hai scritto una vocale");
            break;
      default:
            printf("Hai scritto una consonante");
            break;
}
```

(cosa succede se inserisco il carattere '!' o il carattere 'A' ?)

Ciclo while

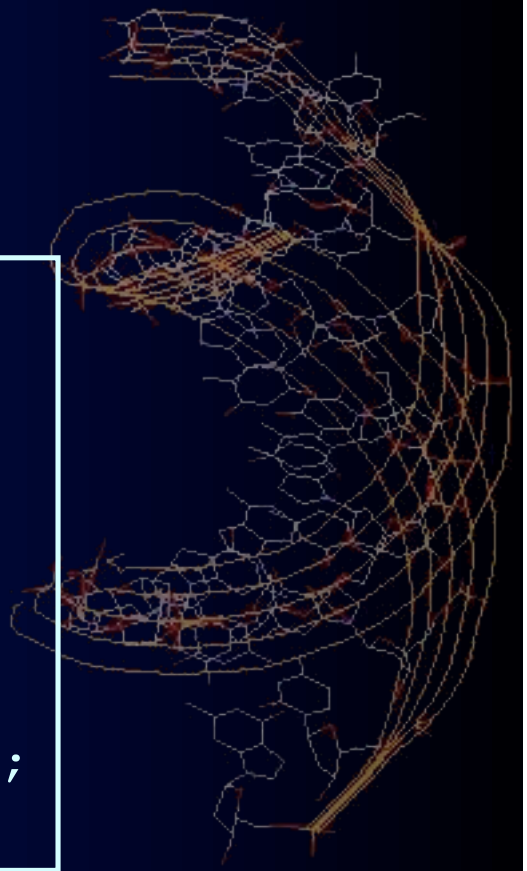
La condizione di controllo viene verificata prima di eseguire qualunque istruzione: il ciclo potrebbe non venire mai eseguito.

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

Ciclo while

Una versione più compatta…

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



Esempio di *getchar* e *putchar*

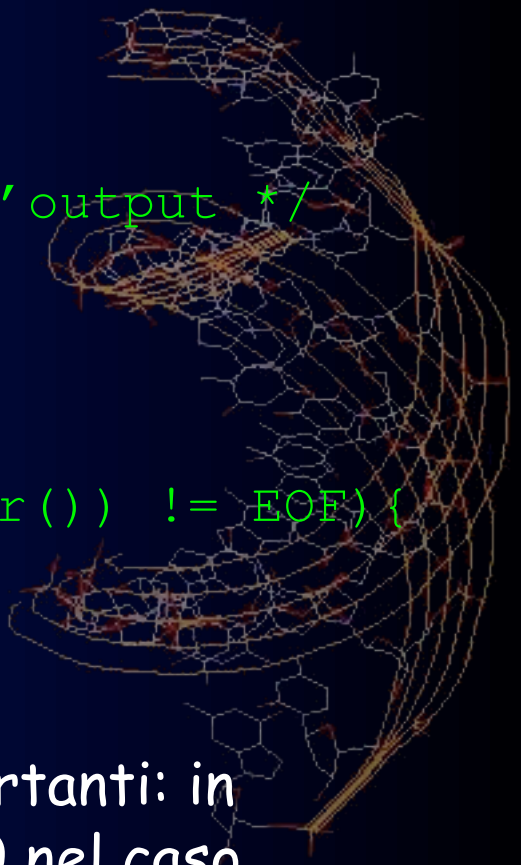
```
/* copia input nell'output */
#include<stdio.h>
main()
{
   int c;

   c = getchar();
   while(c != EOF){
        putchar(c);
        c = getchar();
   }
}
```

Il valore della costante EOF (definita in stdio.h) permettere di distinguere tra un input valido e la fine dell'input.

Esempio di *getchar* e *putchar*

Nota: in C, ogni assegnazione e' una espressione. Ogni espressione ha un valore che corrisponde a quello della variabile a sinistra dell'espressione dopo l'assegnazione.

```
/* copia input nell'output */
#include<stdio.h>
main()
{
   int c;
   while((c = getchar()) != EOF){
       putchar(c);
   }
}
```

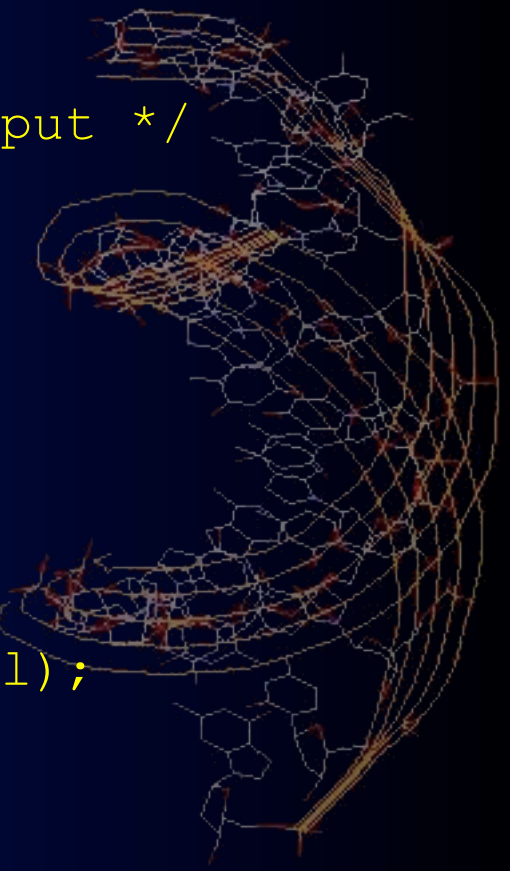


Le parentesi intorno all'assegnazione sono importanti: in assenza di parentesi, `c` assumerebbe il valore 0 nel caso dell' EOF, e 1 altrimenti:

```
c = getchar()!= EOF        c = (getchar()!= EOF)
```

Line Counting

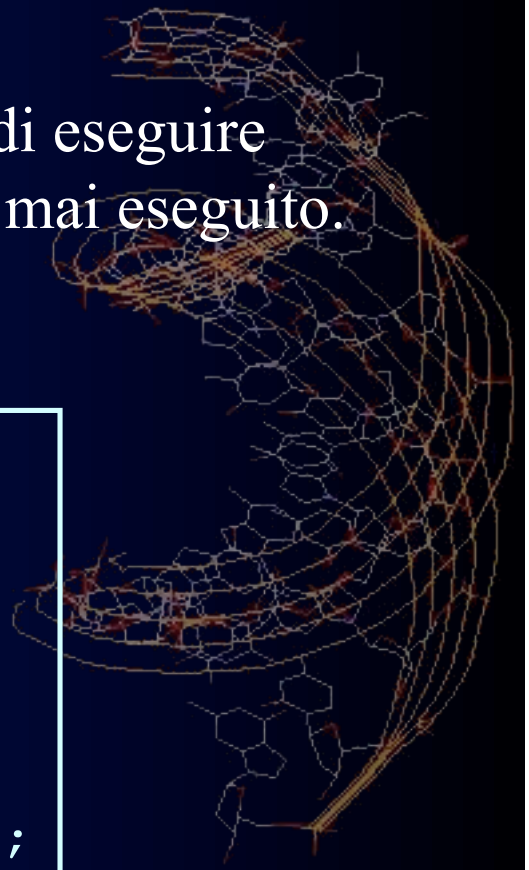
```
/* count the number of lines in input */
#include<stdio.h>
main()
{
  int c, nl = 0;

  while((c = getchar()) != EOF){
        if(c == '\n') nl++;
  }
  printf("Number of Lines = %d", nl);
}
```



E' possibile con questo programma contare il numero di linee di un file di testo ?

Ciclo for

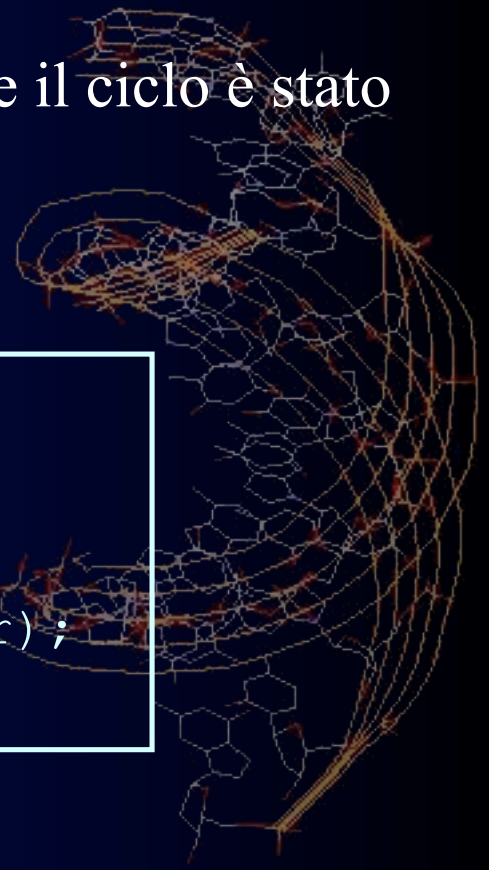
La condizione di controllo viene verificata prima di eseguire qualunque istruzione: il ciclo potrebbe non venire mai eseguito.



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

Ciclo do while

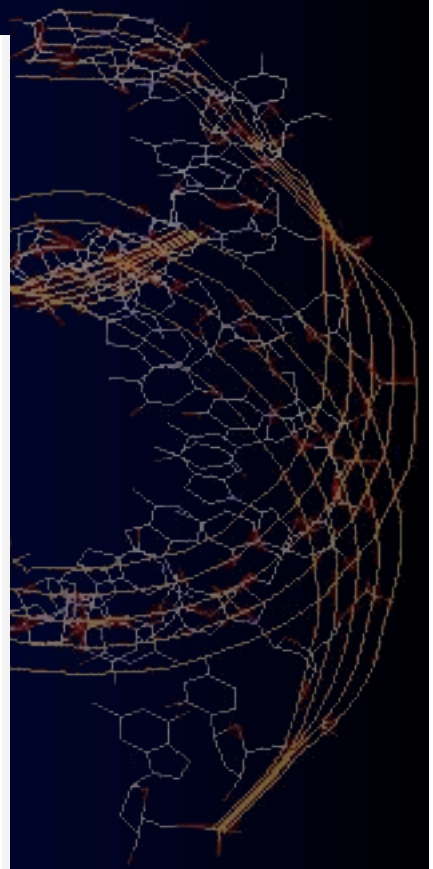
La condizione di controllo viene verificata dopo che il ciclo è stato eseguito almeno una volta.



```
do
{
scanf("%c",&inputchar);
printf("Hai digitato %c",inputchar);
} while(inputchar !='q')
```

Esempi

```c
#include <stdio.h>

main()
{
    long z;             /*numero introdotto <2>*/
    long f;             /*fattoriale calcolato*/
    long i;             /*temporanea per i calcoli*/

    const long zmax = 16;   /*limite perch\202 il fattoriale stia in un long*/

    printf("\n Calcolo del fattoriale\n\n");        /*titolo*/

    do {                                            /*ciclo di input <3>*/
        z = 0L;                                     /*default <4>*/
        printf("Numero (0..%ld): ",zmax);           /*prompt*/
        scanf("%ld",&z);                            /*input numero*/
    } while (z < 0L || z > zmax);                   /*ripete se non ok <5>*/

    f = 1L;                                         /*calcolo fattoriale <6>*/
    for (i = z; i > 1; i--) {                       /*con moltipliche <7>*/
        f *= i;
    }

    printf("\n%ld! = %ld\n",z,f);                   /*stampa fattoriale*/
}
```



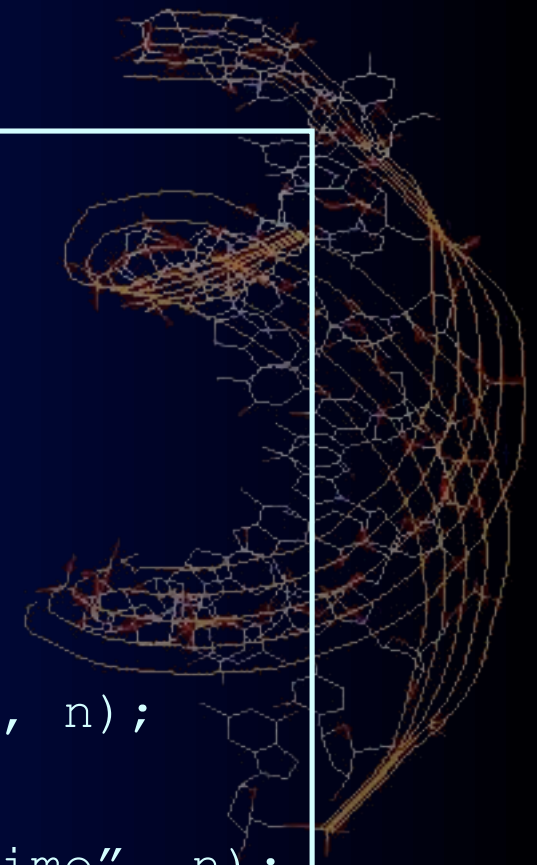
break e continue

Talvolta può essere utile uscire da un ciclo senza controllare, all'inizio o alla fine dell'iterazione, la condizione di terminazione. L'istruzione `break` provoca l'uscita incondizionata da un for, un while oppure un do, così come consente di uscire da uno switch.

L'istruzione `continue` è simile al break ma forza l'inizio della successiva iterazione di un for, un while oppure un do: essa cioè provoca l'immediata esecuzione della parte di controllo del ciclo.

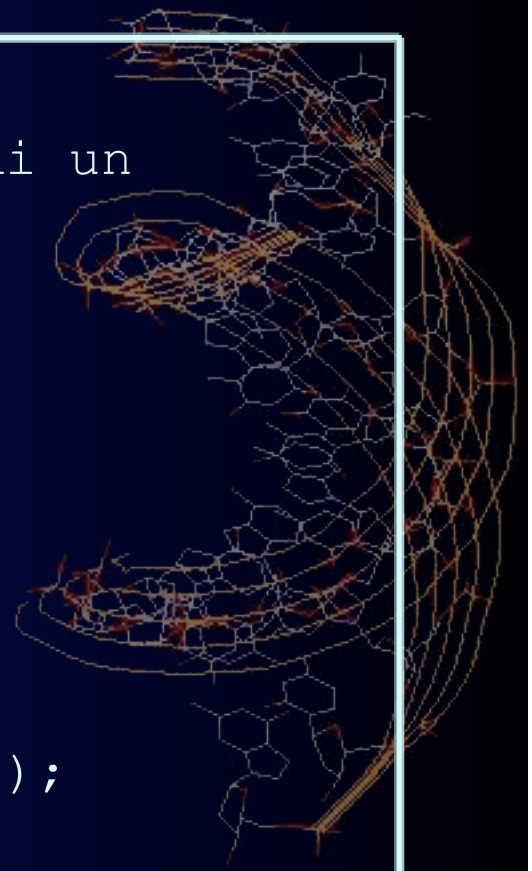
Esempio

```
for (i=n-1; i>=2; i--)
{
resto = n%i;
if(resto == 0)
      break;
}
if(i==1)
       printf("%d è un numero primo", n);
else
       printf("%d non è un numero primo", n);
```



Esempio: area di un cerchio

```
/*
 programma per il calcolo dell'area di un
 cerchio
*/
#include<stdio.h>
main()
{
   float raggio, area;

   printf("\n Dammi il raggio [cm]: ");
   scanf("%f",&raggio);
   area = 3.14159 * raggio * raggio;
   printf("\n Area = %f ", area);
}
```

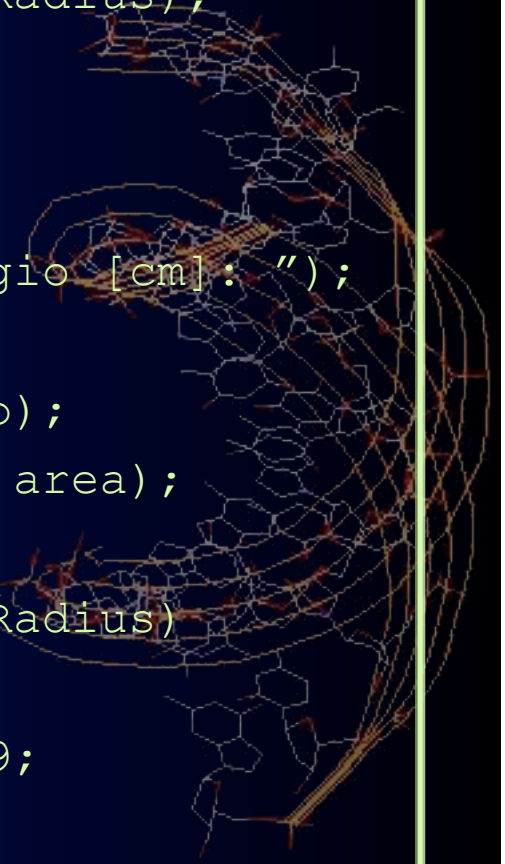


. . . continua

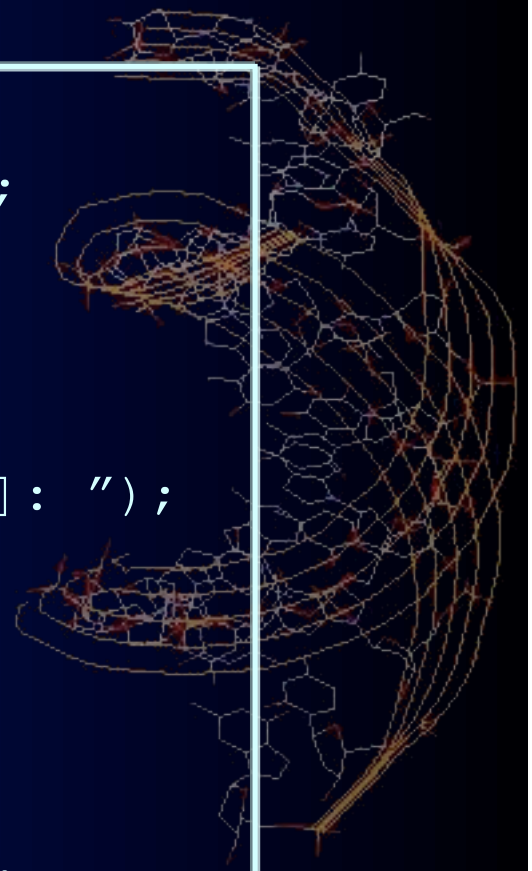
function prototype o function declaration

function definition

```
#include<stdio.h>
float CircleArea(float Radius);
main()
{
float raggio, area;
printf("\n Dammi il raggio [cm]: ");
scanf("%f", &raggio);
area = CircleArea(raggio);
printf("\n Area = %f ", area);
}
float CircleArea(float Radius)
{
const float PI = 3.14159;
float a;
a = PI * Radius * Radius;
return (a);
}
```



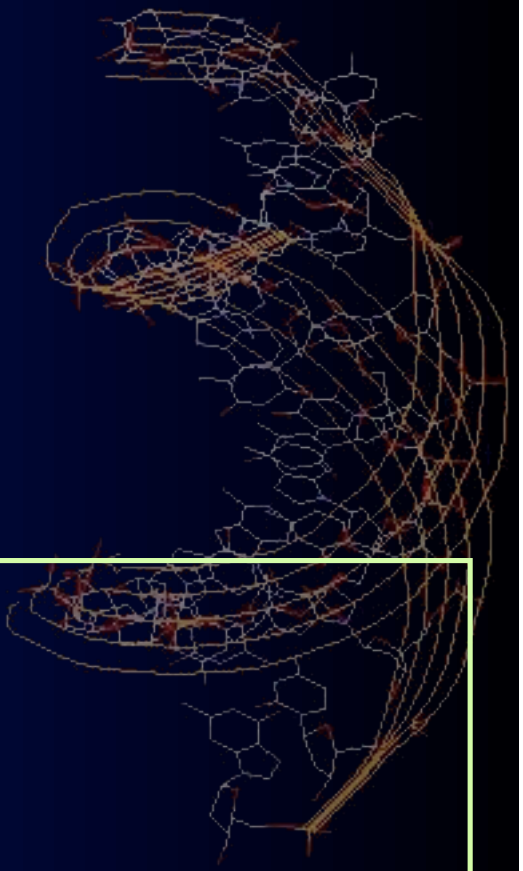
. . . Error Checking nel main



```
#include<stdio.h>
float CircleArea(float Radius);
main()
{
float raggio, area;
printf("\n Dammi il raggio [cm]: ");
scanf("%f", &raggio);
if(raggio < 0)
     area = 0.;
else
     area = CircleArea(raggio);
printf("\n Area = %f ", area);
}
```

. . .Error Checking nella funzione

```
float CircleArea(float Radius)
{
const float PI = 3.14159;
float a;

if( Radius < 0) return 0.;
a = PI * Radius * Radius;
return (a);
}
```



```
float CircleArea(float Radius)
{
const float PI = 3.14159;
float a;

a = ( Radius < 0. )? 0.: PI*Radius*Radius;
return (a);
}
```

Operatore Condizionale: ? :

espressione1 **?** *espressione2* **:** *espressione 3***;**

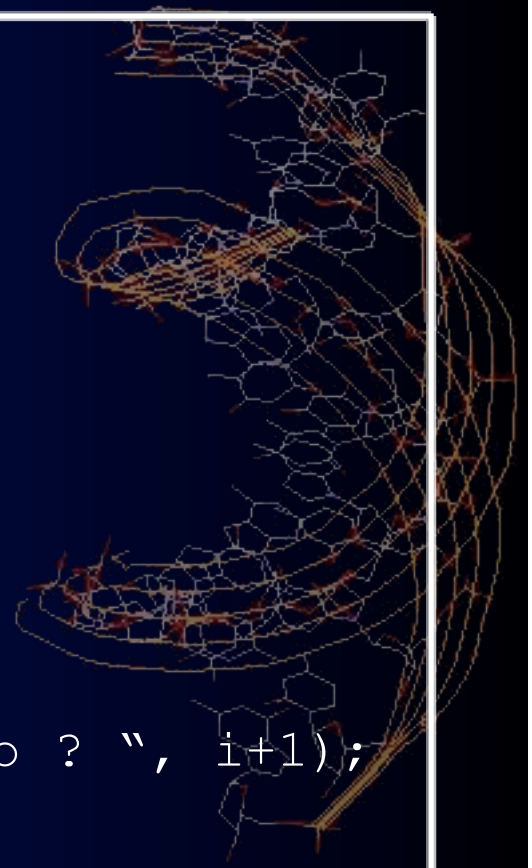
Viene prima valutata l'espressione1; se risulta *vera*, viene valutata l'espressione2 il cui valore diviene il valore della espressione condizionale. Se invece l'espressione1 risulta falsa, l'espressione3 viene valutata ed il suo valore diviene il valore dell'espressione condizionale.

```
max = (a > b)? a: b;
```

```
if(a > b)
    max = a;
else
    max = b;
```

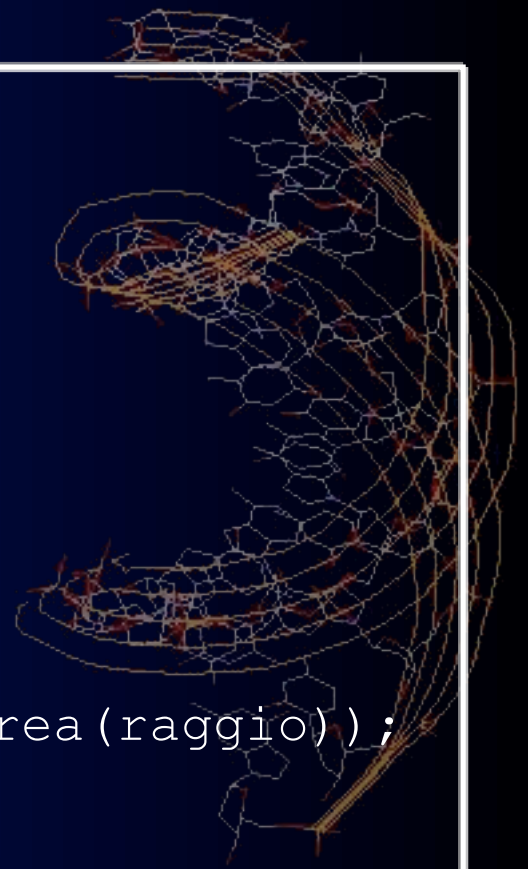
Area di diversi cerchi

```
#include<stdio.h>
float CircleArea(float Radius);
main()
{
  int i, n;
  float raggio;
  printf("\n Quanti cerchi? ");
  scanf("%d",&n);
  for(i = 0 ; i < n ; i++){
      printf("\n Cerchio no. %d: Raggio ? ", i+1);
      scanf("%f",&raggio);
      printf("\n Area = %f ", CircleArea(raggio));
  }
}
```



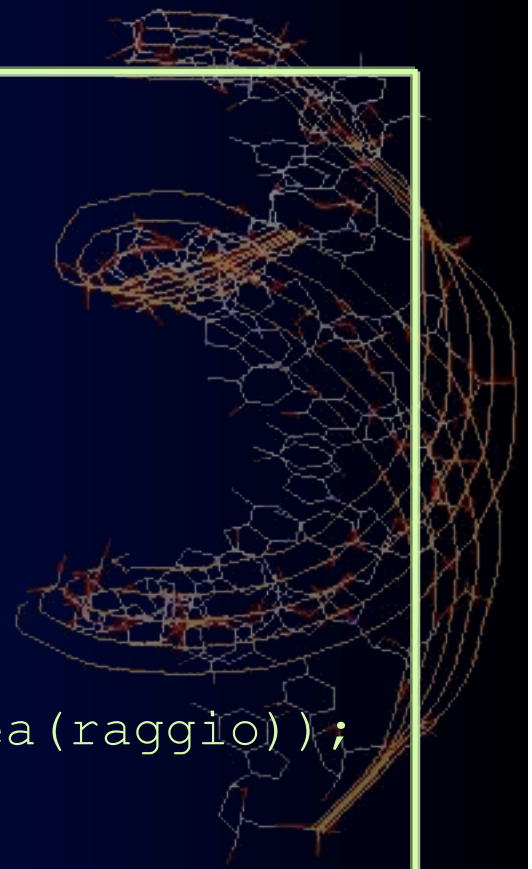
Area di un numero imprecisato di cerchi (I)

```
main()
{
   int i;
   float raggio;
   printf("\n Enter 0 to QUIT.");
   printf("\n Radius ? ");
   scanf("%f", &raggio);
   for(i = 0 ; raggio != 0. ; i++){
        printf("\n Area = %f ", CircleArea(raggio));
        printf("\n Radius ? ");
        scanf("%f",&raggio);
   }
}
```



Area di un numero imprecisato di cerchi (II)

```
main()
{
   float raggio;

   printf("\n Enter 0 to QUIT.");
   printf("\n Radius ? ");
   scanf("%f", &raggio);
   while(raggio != 0){
        printf("\n Area = %f ", CircleArea(raggio));
        printf("\n Radius ? ");
        scanf("%f",&raggio);
   }
}
```

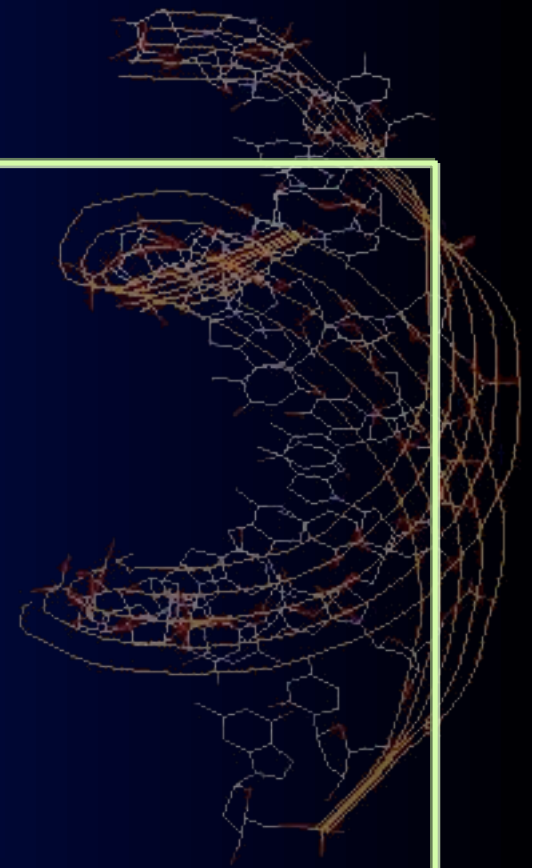


. . . nota che

Nel C, qualunque numero diverso da zero e' interpretato come valore logico *vero*, e lo zero come valore logico *falso*. Le due espressioni seguenti sono pertanto equivalenti:

```
while(raggio != 0)

while(raggio)
```

Area di un numero imprecisato di cerchi (III)

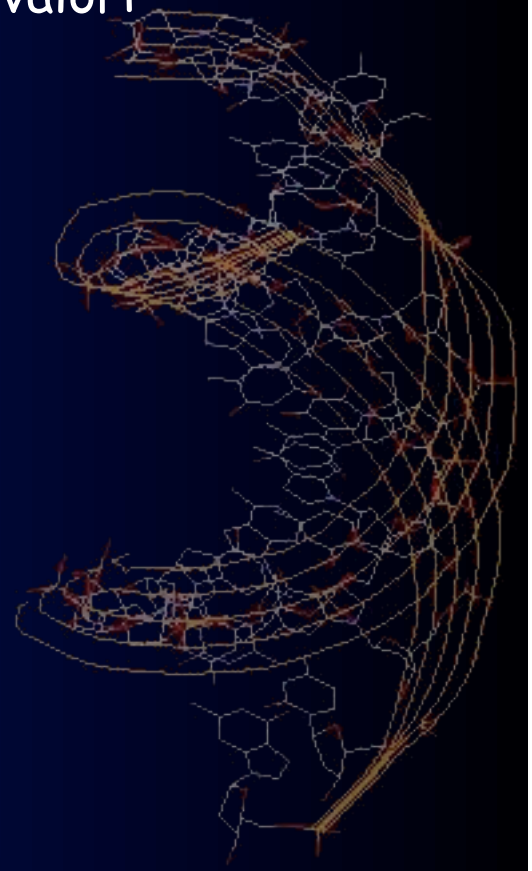


```
main()
{
  int i;
  float raggio;

  printf("\n Enter 0 to QUIT.");
  do{
      printf("\n Radius ? ");
      scanf("%f", &raggio);
      printf("\n Area = %f ", CircleArea(raggio));
  } while(raggio != 0)
}
```

. . . . un’altra variante

In alcuni casi e’ necessario conservare un insieme di valori “omogenei”calcolati in sequenza.

Il C consente l’uso degli *array* (vettori):



Array (I)

Un array e' un insieme contiguo di celle di memoria tutte contenenti dati dello stesso tipo. Queste celle, che costituiscono in pratica un gruppo di variabili, hanno tutte lo stesso nome ed sono accessibili attraverso un indice.

```
int a[10], b[100];
float f[30];
char c[200];
struct Coord
A[10];
```

La dichiarazione di un array (statico) è analoga a quella di una variabile, ad eccezione del fatto che il nome dell'array è seguito dal numero di elementi che lo compongono, racchiuso tra parentesi quadre.

Array Statici (I)

Dichiarazione di array statici:

```
const int Dim, Dim1,. . DimN;
TipoDato nome_array[Dim][Dim1]..[DimN];
```

Esempi di dichiarazioni

```
const int Dim1 = 5, Dim2 = 20;
int a[Dim1], mat[10][20];
unsigned int Coeff[3][10];
float f[Dim2];
struct Coord A[10];
int B[] = {10, 20, 2, 8};
```

Gli array unidimensionali sono anche detti vettori; gli array bidimensionali sono detti matrici.

Array (II)

In C sono possibili due categorie di array:

Array Statici: sono quelli la cui dimensione viene definita all'interno del programma nella dichiarazione.

```
int a[10], mat[10][20];
float f[30];
struct Coord A[10];
```

Array Dinamici: sono quelli la cui dimensione viene determinata, ed eventualmente anche modificata, solo durante l'esecuzione del programma (in *run-time*).

Array Statici (II)

Reminder: se int `a[10]` e' un array, il primo elemento e' `a[0]` e l'ultimo e' `a[9]`.

Cosa accade se si tenta di referenziare un elemento che non fa parte dell'array, ad esempio `a[10]` ?

Il compilatore non puo' rivelare un tale errore: la memoria relativa ad a[10] può essere sempre letta e scritta, ma non fa parte dell'array!!
In caso di lettura, il valore letto non ha alcun significato logico ai fini del programma, mentre in caso di scrittura si rischia di sovrascrivere qualche altro dato importante che risiede nella locazione di memoria subito dopo le locazioni assegnate all'array. Per il compilatore, a[10] è semplicemente la word che si trova a 320 byte dall'indirizzo al quale l'array è memorizzato.

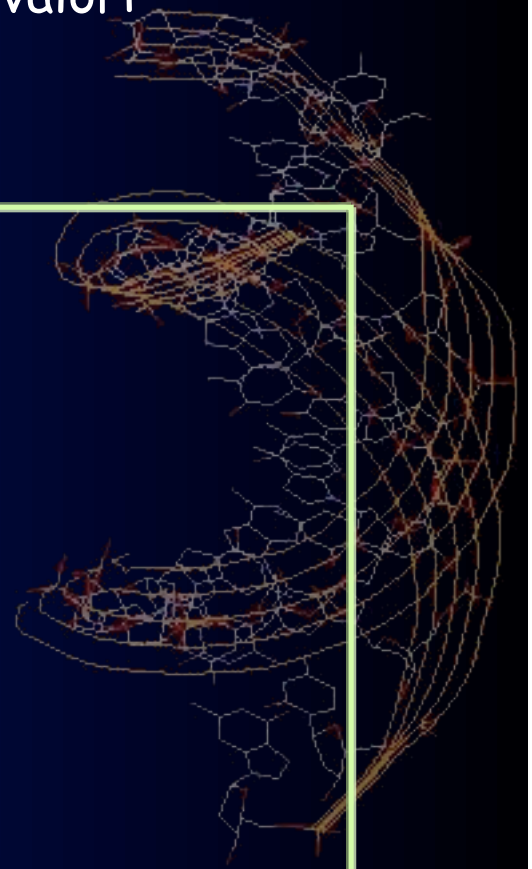
Se la locazione di memoria referenziata non appartiene al programma, il programma stesso termina con un errore (segmentation fault).

. . . . un'altra variante

In alcuni casi e' necessario conservare un insieme di valori "omogenei"calcolati in sequenza.

Il C consente l'uso degli *array* (vettori):

```
main()
{
   int i = -1,n=0;
   float raggio[100], area[100];
   printf("\n Enter 0 to QUIT.");
   do{
        i++;
        printf("\n Radius ? ");
        scanf("%f", &raggio[i]);
        area[i] = CircleArea(raggio[i]);
   } while(raggio[i] != 0)
. . . .
```



. . . .continua

```
. . . . .
  n = --i;
  for(i = 0; i < n; i++){
      printf("\n Radius =%f, Area = %f",
                          raggio[i], area[i]);
  }
}
```

Reminder: se un array e' composto di n elementi, l'indice degli elementi dell'array corre da 0 a n-1.

Attenzione: che succede se i > 99? E' necessario includere anche un controllo sul numero massimo di cerchi: i non puo' superare il valore 99!!