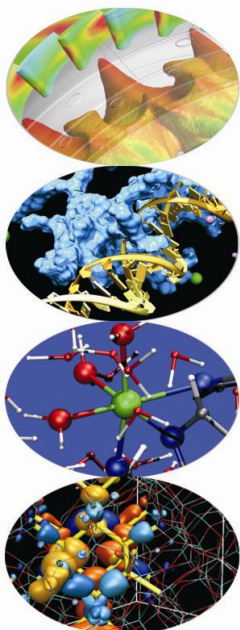
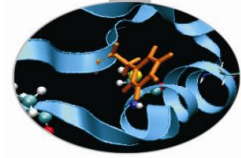


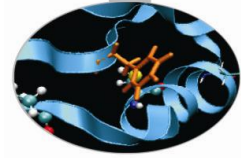
# Librerie Standard



# Indice



- **stdlib.h**
- **math.h**
- **time.h**
- **stdio.h**
- **string.h**



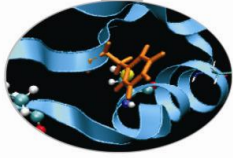
# C++ standard library

La libreria standard del C++ è una collezione di funzioni, costanti e strutture dati che estendono il linguaggio fornendo diverse funzionalità molto comuni che possono essere suddivise in tre grandi gruppi:

- **C standard library** insieme di funzioni costanti e algoritmi ereditati dal C. Suddivisione per header file: [stdio.h](#), [stdlib.h](#), [string.h](#), [time.h](#), [math.h](#) (includibili anche con i nomi [cstdio](#), [cstdlib](#), [cstring](#), [ctime](#), [cmath](#), con la sola differenza che questi sono dichiarati all'interno del namespace std);
- **IOstream library** questa è una libreria orientata agli oggetti per l'I/O in C++;
- **STL (Standard Template Library)** insieme di template di classe; comprende anche vettori, code, liste, stringhe...

Nel seguito verranno prese in considerazione alcune delle librerie più comuni all'interno della C standard library.

Per gli altri gruppi di librerie si rimanda al modulo II del corso.

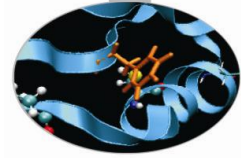


# stdlib.h

La libreria **stdlib.h** è un insieme di funzioni che svolgono operazioni molto diverse tra loro, dal calcolo del valore assoluto di una variabile numerica all'allocazione dinamica della memoria. All'interno della libreria è possibile individuare cinque grandi sottoinsiemi:

1. funzioni di conversione di tipo: `atof`, `atoi`, `atol`, `ectv`, `fcvt`, `itoa`, `ltoa`, `strtod`, `strtol`, `strtoul`, `ultoa`;
2. funzioni di allocazione/deallocazione dinamica della memoria: `calloc`, `free`, `malloc`, `realloc`;
3. funzioni di controllo di processi e di variabili d'ambiente: `abort`, `atexit`, `exit`, `getenv`, `putenv`, `system`;
4. funzioni di ordinamento e di ricerca: `bsearch`, `lfind`, `lsearch`, `qsort`, `swap`;
5. funzioni matematiche: `abs`, `div`, `labs`, `ldiv`.

Abbiamo già incontrato alcune di queste funzioni. Vedremo ancora alcuni esempi sull'utilizzo di qualcun'altra fra le più comuni di queste funzioni.



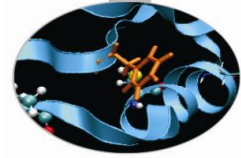
# stdlib.h : la funzione system

```
int system(const* char cmd)
```

consente di richiamare l'interprete dei comandi che eseguirà il comando cmd da shell.  
Terminata l'operazione, il programma continua con le istruzioni successive. La funzione restituisce 0 se il comando ha avuto successo, -1 in caso contrario.

## Esempio:

```
/* file sys.c */  
#include<stdio.h>  
#include<stdlib.h>  
int main(){  
    int resp;  
    resp = system("echo ciao > file.dat");  
    if(resp==0)  
        printf("File creato. \n");  
    else  
        printf("Errore. \n");  
    return 0;}
```



# stdlib.h : la funzione system

## Output:

```
$ ll
```

```
total 12
```

```
-rwxr-xr-x    1 marchisi interni    11672 Apr 19 17:11  
  sys.x
```

```
$ ./sys.x
```

```
File creato.
```

```
$ ll
```

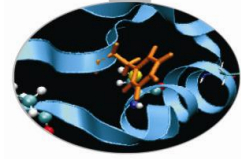
```
total 16
```

```
-rw-r--r--    1 marchisi interni         5 Apr 19 17:16  
  file.dat
```

```
-rwxr-xr-x    1 marchisi interni    11672 Apr 19 17:11  
  sys.x
```

```
$ more file.dat
```

```
ciao
```



# stdlib.h: la funzione qsort

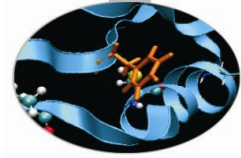
```
void qsort (void* arr, size_t num_el, size_t  
            dim_el, int (*fun_compare))
```

implementa l'algoritmo *quicksort* per riordinare num\_el elementi contenuti nell'array arr. Viene specificata la dimensione in byte (width) di ciascun elemento e la funzione, fun\_compare che realizza il confronto fra due elementi. Tale funzione deve avere il seguente prototipo:

```
int fun_compare(const void* e1, const void* e2)
```

e può essere realizzata in modo tale da restituire un intero minore di zero ( $e1 < e2$ ), uguale a zero ( $e1 = e2$ ) o maggiore di zero ( $e1 > e2$ ).

# Esempio



```
#include<stdio.h>
#include<stdlib.h>

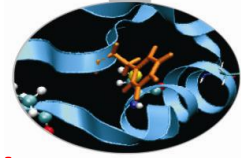
int fun_compare (const void * e11, const void * e12){
    int diff;
    diff = ( *(double*)e11 - *(double*)e12 );
    return diff;
}

int main (){
    int i;
    const int dim=5;
    double array[5]={42.31, 12.23, 52.342, 91, 3.22};
    qsort(array, dim, sizeof(double), fun_compare);
    for (i=0; i<dim; i++){
        printf ("%3.3f ",array[i]);
    }
    printf("\n");
    return 0;
}
```

## Output

```
3.220 12.230 42.310 52.342 91.000
```



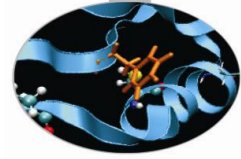


# stdlib.h: le funzioni matematiche

Per motivi storici alcune funzioni matematiche sono contenute nella libreria **stdlib.h**. Qui di seguito verranno elencate e commentate queste funzioni.

- `abs(val)`, `labs(val)`: restituiscono il valore assoluto di `val` quando esso è un valore `integer`, oppure un `long integer` rispettivamente;
- `int rand(void)`: genera un numero intero pseudo-casuale compreso tra 0 e `RAND_MAX`. Quest'ultima è una costante, definita anch'essa in `stdlib.h`, ed ha un valore costante che dipende dall'ambiente; spesso vale `2147483647`;
- `void srand(unsigned int seed)`: viene usata insieme con `rand()` ed utilizza il parametro `seed` per far generare a `rand()` una sequenza diversa di numeri pseudo-casuali;
- `div_t div(int numeratore, int denominatore)`: ritorna una struct `div_t` (definita nella `stdlib.h`) i cui dati membro interi `quot` e `rem` contengono rispettivamente il valore del quoziente e del resto della divisione tra numeratore e denominatore. `ldiv(long, long)` è analoga per i `long int`.

# Esempio: uso di rand()



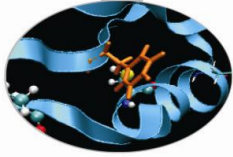
```
#include<stdio.h>
#include<stdlib.h>
int main (){
    int pseudo;
    int i;
    printf("Serie di cinque numeri tra 1 e 10 \n");
    for(i=0; i<5; i++){
        pseudo = rand()%10+1;
        printf("%d ",pseudo);
    }
    printf("\n");
    return 0;}
```

## output:

```
./casuale.x
Serie di cinque numeri tra 1 e 10
4 7 8 6 4
```

```
./casuale.x
Serie di cinque numeri tra 1 e 10
4 7 8 6 4
```

# Esempio: uso di rand() e srand()



```
/* file s_casuale.c */
```

```
#include <stdio.h>
```

```
#include <stdlib.h>
```

```
int main (){
```

```
    int i, pseudo;
```

```
    srand(20);
```

```
    printf ("Serie di 5 numeri tra 1 e 10: \n");
```

```
    for(i=0; i<5; i++){
```

```
        pseudo=rand() %10+1;
```

```
        printf("%d ",pseudo);
```

```
    }
```

```
    srand(10);
```

```
    printf ("\nSerie di 5 numeri tra 1 e 10: \n");
```

```
    for(i=0; i<5; i++){
```

```
        pseudo=rand() %10+1;
```

```
        printf("%d ",pseudo);
```

```
    }    printf ("\n");
```

```
    return 0;
```

```
}
```

- output:

```
./s.x
```

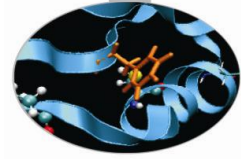
```
Serie di 5 numeri tra 1 e 10:
```

```
2 9 8 10 7
```

```
Serie di 5 numeri tra 1 e 10:
```

```
6 9 9 6 9
```

# Esempio: uso di div()



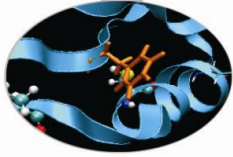
```
#include<stdio.h>
#include<stdlib.h>

int main() {
    div_t res;
    int x,y;
    x=9;
    y=7;
    res = div( x, y );
    printf("%d diviso %d fa %d con resto di %d \n", x, y,
        res.quot, res.rem);
    return 0;
}
```

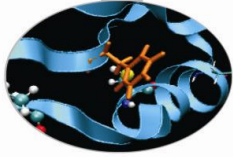
Output:

9 diviso 7 fa 1 con resto di 2

# Le funzioni matematiche in C e C++

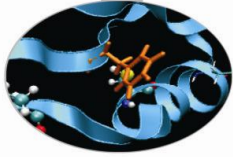


- C e C++ hanno un insieme di funzioni matematiche predefinite; originariamente supportavano entrambi le stesse funzioni matematiche.
- C++ è poi maturato producendo come effetto un disallineamento tra i due linguaggi.
- Il gruppo fondamentale di funzioni matematiche rimane comunque supportato da entrambi i linguaggi.



# math.h: le funzioni matematiche C

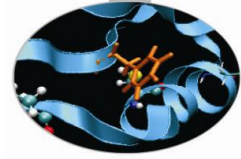
- Le funzioni che il linguaggio C e C++ hanno in comune sono analizzate di seguito:
- `acos(val)`, `asin(val)`, `atan(val)`: calcolano l'inverso delle funzioni trigonometriche. Restituiscono un `double` che rappresenta il valore in radianti.
- `ceil(val)`, `floor(val)`: restituiscono il valore del più piccolo/grande intero non più piccolo/grande di `val`.
- `cos(val)`, `sin(val)`, `tan(val)`, `cosh(val)`, `sinh(val)`, `tanh(val)`: funzioni trigonometriche di base, argomenti in radianti.
- `atof(const char* s)`: converte la stringa `s` in un `double`, è definita anche all'interno di `stdlib.h`.



# math.h: le funzioni matematiche C

- `exp(val)`, `pow(base, val)`: restituiscono rispettivamente un `double` pari al valore dell'elevamento di 'e'/base alla potenza val.
- `log(val)`, `log10(val)`: restituiscono rispettivamente un `double` pari al valore del logaritmo naturale/base10 di val.
- `frexp(double num, int* exp)`, `ldexp(double num, int exp)`, `modf(double num, double *i)`: funzioni che restituiscono un `double` pari rispettivamente a:
  - - mantissa, ove:  $num = mantissa * 2^{exp}$
  - -  $num * (2^{exp})$
  - - la parte frazionale di un numero dopo averne memorizzato parte intera in i.
- `fmod(val1, val2)`: restituisce un `double` pari al resto della divisione tra val1 e val2.
- `sqrt(val)`: ritorna un `double` pari al valore della radice quadrata di val.
- **N.B.:** il compilatore C richiede l'opzione **-lm** per linkare le librerie matematiche ad un programma.

# Esempio



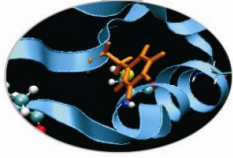
```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* file mat.c */
```

```
int main(int argc, char* argv[]){
    int ee;
    double mantissa;
    double parte_int;
    double parte_dec;
    if(argc < 2){
        printf("Use the format mat.x <double>\n");
        exit (1);
    }
    double nd = atof(argv[1]);
    printf("%f e' compreso tra %f e %f \n",nd,floor(nd),ceil(nd));
    mantissa = frexp(nd, &ee);
    printf("%f = %f * 2^%d \n", nd, mantissa, ee);
    parte_dec = modf(nd,&parte_int);
    printf("%f = %f + %f \n", nd, parte_int, parte_dec);
    printf("Il cubo di %f e' %f \n",nd,pow(nd,3));
    return 0;}
```

- output:

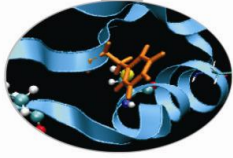
```
gcc mat.c -o mat.x -lm
./mat.x 3.54
3.540000 e' compreso tra 3.000000 e
4.000000
3.540000 = 0.885000 * 2^2
3.540000 = 3.000000 + 0.540000
```





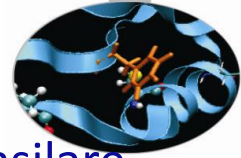
# Libreria matematica C++

- Le stesse funzioni supportate dalla libreria matematica del C per variabili di tipo *double* sono ancora supportate in C++ anche per variabili di tipo *float* e *long double* (overloading)
- Le operazioni eseguite dalle funzioni rimangono invariate
- Tutti gli angoli sono in radianti.



# time.h: date and time in C/C++

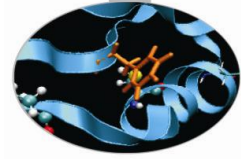
- Molto spesso può essere utile/necessario manipolare all'interno di un programma grandezze legate al tempo.
- Un tipico esempio è quello di voler valutare la performance di un codice o di una sua parte.
- Le funzioni legate al tempo cadono in tre grandi categorie:
  - funzioni legate alla misura del tempo di CPU;
  - funzioni per la misura assoluta di date, tempi, calendari;
  - funzioni per la sincronizzazione di allarmi e timer.
- Qui presenteremo solo aspetti riguardanti le prime due categorie in quanto queste funzioni ricadono all'interno dell'header **time.h**



# Tempi del processore

- Cercando di ottimizzare o di valutare la bontà di un codice C/C++ è basilare essere in grado di estrarre informazioni circa le performance in termini di tempi di calcolo: processor time o CPU time.
- L'informazione che si cerca è diversa da quella che si otterrebbe interrogando il clock della macchina; nel CPU time non vengono computati i tempi di attesa per l'I/O o per il fatto che altri programmi stanno girando ma solo l'effettivo tempo di utilizzo della macchina da parte del codice.
- Il CPU time viene rappresentato tramite il tipo di dato `clock_t` (generalmente coincide con un `long int` e rappresenta il numero di cicli di clock relativi ad una unità di misura arbitraria).
- La funzione usata per estrarre questa informazione è:
  - `clock_t clock()` ;
- Per ottenere il valore in secondi bisogna utilizzare la costante macro:
  - `CLOCKS_PER_SEC` (esiste anche `CLK_TCK`, arcaico)

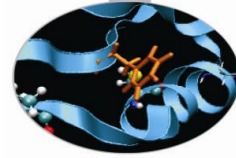
# Esempio: uso di clock()



```
#include<stdio.h>
#include<time.h>
int main(){
    clock_t start, end;
    double tp;
    int i;
    long cyc=1000000000;
    start = clock();
    printf("Start: %d \n",start);
    for(i=0; i<cyc; i++){
    end = clock();
    printf("End: %d \n",end);
    tp=(end-start);
    printf("CLOCKS_PER_SECOND: %d \n",CLOCKS_PER_SEC);
    printf("Tempo trascorso: %f sec. \n",tp/CLOCKS_PER_SEC);
    return 0;
}
```

- **output:**

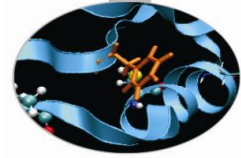
```
Start: 0
End: 2280000
CLOCKS_PER_SECOND: 1000000
Tempo trascorso: 2.280000 sec.
```



# Tempi, date, calendari

In accordo con il calendario Gregoriano si ha:

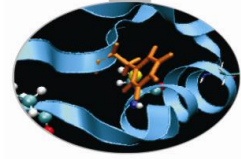
- Tipo di dato tempo del calendario: **time\_t**, può essere visto come un riferimento temporale assoluto che misura il numero di secondi trascorsi dalle 00:00:00 del 1 gennaio 1970 (di fatto è un int o long int);
- Struttura dati del tempo locale *broken-down time*: **struct tm**;
- Funzioni:
  - double **difftime**(time\_t t1, time\_t t0): misura il numero di secondi trascorsi tra i due istanti di tempo t1 e t2;
  - time\_t **time**(time\_t\* ptr): ritorna il valore corrente del tempo come variabile time\_t;
  - struct tm\* **localtime**(const time\_t \*time): converte il tempo di calendario in *broken-down time* espresso relativamente alla *time-zone* dell'utilizzatore.



# Tempi, date, calendari

- `struct tm* gmtime(const time_t *time)`: analoga alla precedente ma usa come riferimento le Coordinate Temporalì Universali (UTC) cioè quelle di Greenwich (GMT);
  - `time_t mktime(struct tm *bt)`: converte il *broken-down time* *bt* in un dato di tipo calendario
- Funzioni per la formattazione dei dati di calendario:
  - `char* asctime(const struct tm *brokentime)`: scrive il *broken-down time* in una stringa del tipo:  
"Giorno\_della\_settimana Mese Giorno\_del\_mese ore:min:sec anno\n"
  - `char* ctime(const time_t *time)`: simile alla precedente salvo per il fatto che scrive in formato calendario anziché in *local time*.

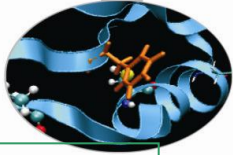
# struct tm



- Il tempo è misurato in secondi ed è comodo per i calcoli ma non per essere visualizzato;
- l'ora e la data rappresentata tramite il così detto *broken-down* time dipende ovviamente dalla zona;
- la struttura dati contiene i valori del *broken-down time* ed è costituita dai seguenti dati membro:

```
struct tm{  
    int tm_sec;  
    int tm_min;  
    int t_hour;  
    int tm_mday;           // giorno del mese  
    int tm_mon;           // numero del mese  
    int tm_year;  
    int tm_wday;          // giorno della settimana  
    int tm_yday;          // giorno dell'anno  
    int tm_isdst;         // setting di daylight saving  
    int tm_gmtoff;        // timezone usata  
    const char *tm_zone; /* abbreviazione di 3  
                           lettere con il nome della  
                           timezone */};
```

# Esempio: uso di srand() e time()



```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
```

```
int main () {
```

```
    int i, pseudo;
```

```
    srand(time(NULL));
```

```
    printf ("Serie di 5 numeri tra 1 e 10: \n");
```

```
    for(i=0; i<5; i++){
```

```
        pseudo=rand()%10+1;
```

```
        printf("%d ",pseudo);
```

```
    }
```

```
    printf ("\n");
```

```
    return 0;
```

```
}
```

- output

```
./tc.x
```

```
Serie di 5 numeri tra 1 e 10:
```

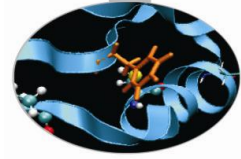
```
1 3 10 3 3
```

```
./tc.x
```

```
Serie di 5 numeri tra 1 e 10:
```

```
2 7 10 4 8
```





# Esempio

Uso di `time()`, `localtime()`, `asctime()` e `gmtime()`.

```
#include<stdio.h>
```

```
#include<time.h>
```

```
int main(){
```

```
    time_t tp;
```

```
    struct tm* stm;
```

```
    time (&tp);
```

```
    printf("Value returned by time(): %d \n",tp);
```

```
    stm = localtime (&tp);
```

```
    printf("Local time: %s", asctime (stm));
```

```
    stm=gmtime (&tp);
```

```
    printf("GMT time: %s", asctime (stm));
```

```
    return 0;
```

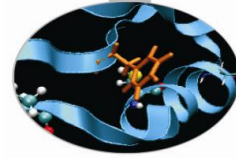
```
}
```

- **output:**

Value returned by `time()`: 1146058353

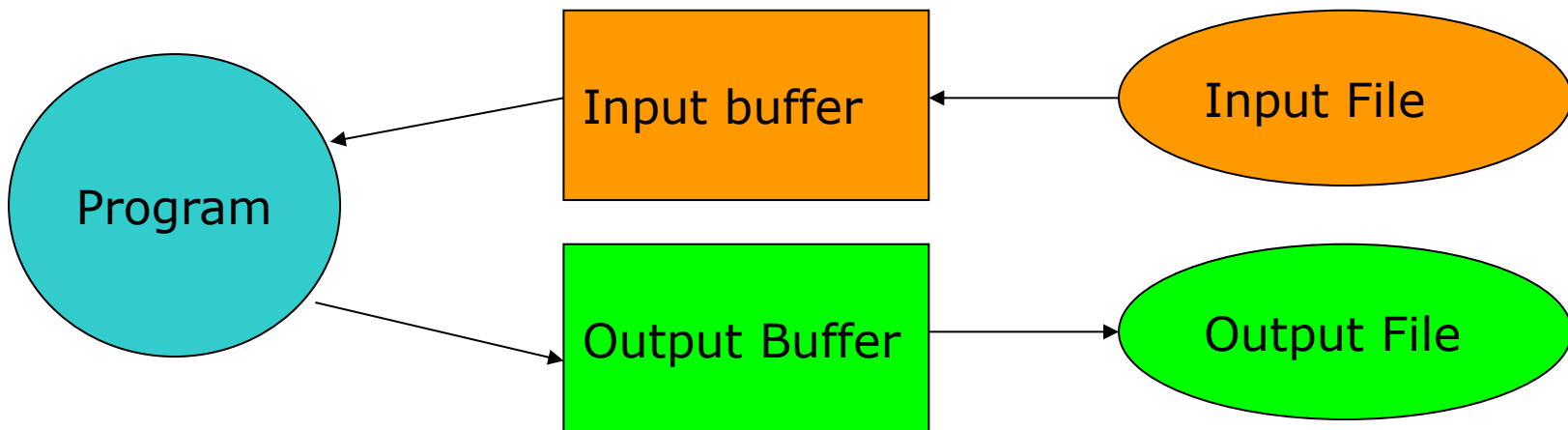
Local time: Wed Apr 26 15:32:33 2006

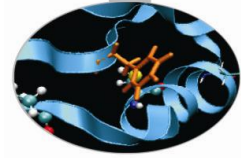
GMT time: Wed Apr 26 13:32:33 2006



# stdio.h

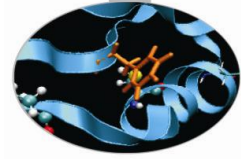
- All'interno della *Standard C Library*, la libreria **stdio.h** rappresenta l'insieme di funzioni C preposte alle operazioni di *I/O bufferizzato*.
- L' I/O bufferizzato viene utilizzato per evitare un sovraccarico nella comunicazione fra un programma ed una qualsiasi device di I/O, come i file ed il video.
- I dati di input vengono letti, anziché uno per uno, in gruppi che vanno ad occupare momentaneamente il buffer, prima di raggiungere il programma. Un analogo procedimento avviene anche in fase di output.





# stdio.h: funzioni per l'I/O

- **int printf**(const char \* <formato [ , argomenti , ... ]>): scrive dati formattati su standard output;
- **int scanf**(const char \* <formato [ , argomenti , ... ]>): legge dati formattati da stdin;
- **int getchar**(void): ritorna il primo carattere che viene passato allo stdin. Il carattere viene letto come un carattere e ritornato come intero;
- **int putchar**(int): analogamente pone il carattere nello stdout;
- **char\* gets**(char\*): legge un insieme di caratteri dallo stdin e li pone nel buffer fino a che non incontra il carattere di new line o EOF (invio o spazio);
- **int puts**(const char\*): scrive una stringa su stdout e la completa con un carattere di new line.

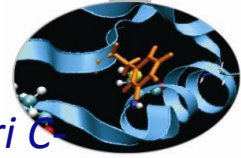


```
/* file program.c */  
#include <stdio.h>  
  
int main(){  
    char c;  
    puts("Enter a text. Dot ('.') to exit:");  
    do{  
        c=getchar();  
        putchar(c);  
    }while (c != '.');  
    printf("\n");  
    return 0;  
}
```

## Output:

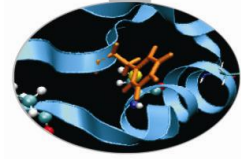
```
./program.x  
Enter a text. Dot ('.') to exit:  
Hello, it's just an example, bye.  
Hello, it's just an example, bye.
```

# string.h: libreria per le stringhe letterali



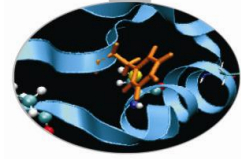
La maggior parte delle funzioni atte a manipolare e gestire le *stringhe di caratteri C like* è racchiusa nella libreria standard del C **string.h**. Fra le più importanti ricordiamo:

- **char\* strcat (char \* s1, const char \* s2)**: aggiunge la stringa s2 al termine della stringa s1. La stringa risultante termina con il carattere nullo ('\0');
- **char\* strchr (const char\* s, int c)**: restituisce la prima occorrenza del carattere c all'interno della stringa s. Poiché restituisce un char\*, il risultato va interpretato attraverso l'aritmetica dei puntatori;
- **int strcmp (const char \* s1, const char \* s2)**: mette a confronto le stringhe s1 ed s2 char per char, restituendo 1 se esse risultano uguali, 0 in caso contrario;
- **char\* strcpy (char \* s1, const char \* s2)**: copia la stringa s2 dentro la stringa s1;
- **long strlen (const char\* s)**: restituisce la lunghezza della stringa s.



```
#include <stdio.h>
#include <string.h>
int main () {
    char stg[30] = "Questo e' un corso di C e C++";
    char* ptr_c;
    char ch='C';
    int diff;
    printf ("Ricerca di 'C' all'interno di: \"%s\" \n",stg);
    ptr_c=strchr(stg,ch);
    printf("ptr_c: %X %c \n",ptr_c, *ptr_c);
    printf("stg: %X \n", stg);
    diff=ptr_c-stg;
    printf("diff: %d \n",diff);
    while (ptr_c!=NULL) {
        printf ("Trovata una C in posizione %d \n",ptr_c-stg+1);
        ptr_c=strchr(ptr_c+1,ch);
    }
    return 0;}

```



# Esempio

## Output:

```
Ricerca di 'C' all'interno di: "Questo e' un  
corso di C e C++"
```

```
ptr_c: BFFFD77C C
```

```
stg: BFFFD766
```

```
diff: 22
```

```
Trovata una C in posizione 23
```

```
Trovata una C in posizione 27
```