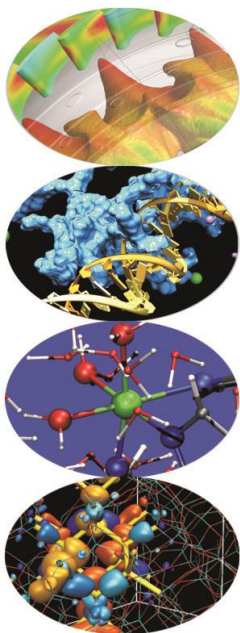
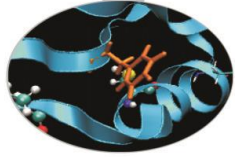


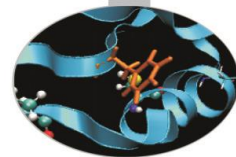
Librerie Standard



Indice



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

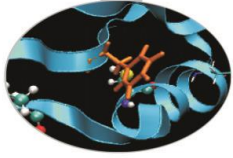


C standard library

La **C standard library** è un insieme di funzioni, costanti e algoritmi.

È suddivisa per header file: [stdio.h](#), [stdlib.h](#), [string.h](#), [time.h](#), [math.h](#)

Nel seguito verranno citate alcune delle funzioni più comuni della C standard library.

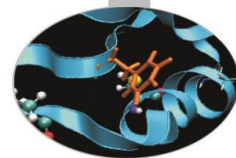


stdlib.h

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
2. funzioni di allocazione/deallocazione dinamica della memoria
3. funzioni di controllo di processi e di variabili d'ambiente
4. funzioni di ordinamento e di ricerca
5. funzioni matematiche

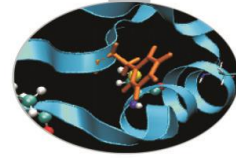
stdlib.h



Alcune funzioni:

1. funzioni di conversione di tipo: `atof`, `atoi`, `fcvt`, `itoa`, `strtod`, `strtol`;
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: `qsort`, `swap`;
5. funzioni matematiche: `abs`, `div`.

Vedremo ora alcuni esempi sull'utilizzo di qualcuna fra le più comuni di queste funzioni.

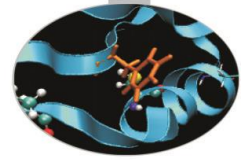


stdlib.h : la funzione system

```
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.

```
/* e-system.c */  
#include<stdio.h>  
#include<stdlib.h>  
  
int main() {  
    int resp;  
    printf("Cartella corrente: \n");  
    resp = system("echo $PWD");  
    if(resp != 0)  
        printf(" !! system() error. \n");  
    return 0; }
```



stdlib.h: la funzione qsort

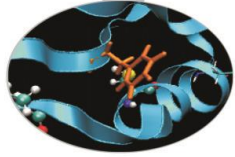
```
void qsort (void* arr, size_t num_el, size_t  
           dim_el, int (*fun_compare))
```

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 ($e1 > e2$).

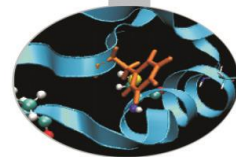
Esempio e-sorting.c



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

int fun_compare (const void * el1, const void * el2){
    int diff;
    diff = ( *(double*)el1 - *(double*)el2 );
    return diff;
}

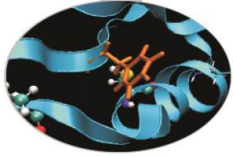
int main (){
    int i;
    const int dim=5;
    double array[5]={42.31, 12.23, 52.342, 91, 3.22};
    qsort(array, dim, sizeof(double), fun_compare);
    for (i=0; i<dim; i++){
        printf ("%3.3f ",array[i]);
    }
    printf("\n");
    return 0;
}
```

stdlib.h: alcune funzioni matematiche

Per motivi storici alcune funzioni matematiche sono contenute nella libreria **stdlib.h**.

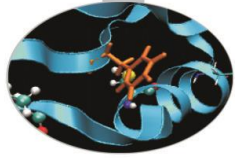
- `int abs(int val)` : valore assoluto per `val` integer
- `long int labs(long int val)` : valore assoluto per `val` di tipo long integer
- `div_t div(int num, int denom)` : ritorna una `struct div_t` i cui dati membro interi `quot` e `rem` contengono rispettivamente il valore del quoziente e del resto della divisione tra numeratore e denominatore .
- `ldiv_t ldiv(long int numer, long int denom)` : analoga alla precedente per i long integer



stdlib.h: alcune funzioni matematiche

- `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 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;

Esempio: uso di rand()



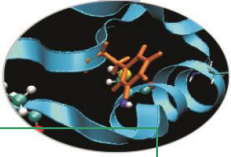
```
#include<stdio.h>
#include<stdlib.h>
int main (){
    int pseudo;
    int i;
    printf("Serie di cinque numeri tra 1 e 10 \n");
    for(i=0; i<5; i++){
        pseudo = rand()%10+1;
        printf("%d ",pseudo);
    }
    printf("\n");
    return 0;}
```

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()



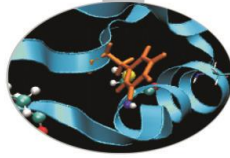
```
#include <stdio.h>
#include <stdlib.h>

int main (){    /* file e_casuale.c */
    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
```

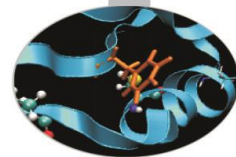


Esempio: uso di div()

```
/* e-divisione.c */  
#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:

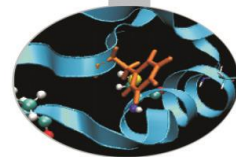
9 diviso 7 fa 1 con resto di 2



math.h: funzioni matematiche

Di seguito alcune funzioni matematiche di uso comune

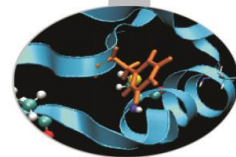
- `cos(val)`, `sin(val)`, `tan(val)`, `cosh(val)`, `sinh(val)`, `tanh(val)`: funzioni trigonometriche di base, argomenti in radianti.
- `acos(val)`, `asin(val)`, `atan(val)`: funzioni trigonometriche inverse; restituiscono un `double` che rappresenta il valore in radianti.
- `ceil(val)`, `floor(val)`: restituiscono il valore del più piccolo/grande intero non più piccolo/grande di `val`.
- `atof(const char* s)`, `atoi(const char* s)`: convertono la stringa `s` in un `double` e un `int` rispettivamente; sono definite anche in `stdlib.h`.



math.h: funzioni matematiche

- `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.
- `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.

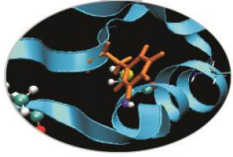


math.h: funzioni matematiche

- `double frexp(double num, int* exp):` ritorna mantissa (compresa fra 0.5 ed 1) ed `exp` tali che $\text{num} = \text{mantissa} * 2^{\text{exp}}$
- `double ldexp(double num, int exp):` ritorna $\text{num} * 2^{\text{exp}}$
- `double modf(double num, double *intr):` ritorna `modf` e `intr` tali che `modf` è la parte frazionaria di `num`, `intr` la parte intera

N.B.: il compilatore C richiede l'opzione **-lm** per linkare le librerie matematiche ad un programma.

Esempio

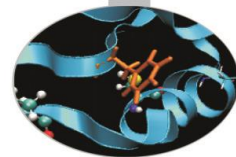


```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
/* file e-mat.c */
```

```
int main(int argc, char* argv[]){
    int ee;
    double mantissa, nd;
    double parte_int;
    double parte_dec;
    if(argc < 2){
        printf("Use the format mat.x <double>\n");
        exit (1);
    } 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;}
```

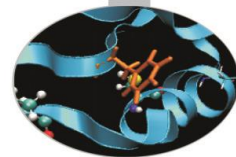
- output:

```
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
```



Alcune note C

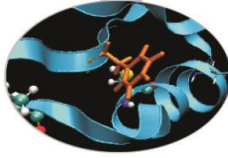
- Buona parte delle funzioni supportate dalla libreria matematica del C per variabili di tipo *double* nello standard c89 sono state estese nello standard c99 anche per variabili di tipo *float* e *long double*
- Tutti gli angoli sono in radianti.



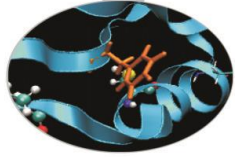
time.h: date and time

- Spesso è utile/necessario manipolare grandezze legate al tempo.
- Un tipico esempio è quello di voler valutare la performance di un codice o di una sua parte.
- In tal senso il linguaggio nell'header **time.h** mette a disposizione:
 - funzioni per la misura del tempo di CPU;
 - funzioni per la misura assoluta di date, tempi, calendari;

Tempi del processore



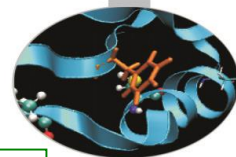
- Cercando di ottimizzare o di valutare la bontà di un codice è utile poter estrarre informazioni circa le performance in termini sia di tempi di calcolo sia di durata di esecuzione.
- Nel CPU time non vengono computati i tempi di attesa per l'I/O o per il fatto che altri programmi stanno girando, ma solo l'effettivo tempo di utilizzo della macchina da parte del codice.
- Viceversa il tempo solare di esecuzione è influenzato anche dalle operazioni di accesso al disco e alle periferiche, oltre che da eventuali esecuzioni concorrenti.



Tempi del processore

- 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(){ /* e-clock.c */

    clock_t start, end;
    double tp;

    int i;
    long cyc=1000000000;
    start = clock();
    for(i=0; i<cyc; i++){
    end = clock();
    printf("Start, End: %d, %d \n",start,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; }
```

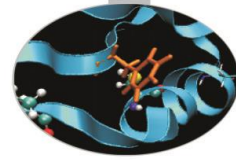
• output:

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

Esperimento:

Compilare l'esempio con flag di ottimizzazione crescenti.

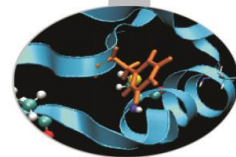
Cosa si osserva?



Tempi, date, calendari

In accordo con il calendario Gregoriano si ha:

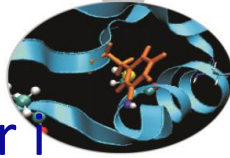
- Tipo di dato tempo del calendario: **time_t**, può essere visto come un riferimento temporale assoluto che misura il numero di secondi trascorsi dalle **00:00:00 del 1 gennaio 1970** (di fatto è un `int` o `long int`);
 - `time_t time(time_t* ptr)` : ritorna il valore corrente del tempo come variabile `time_t`;
 - `double difftime(time_t t1, time_t t0)`: misura il numero di secondi trascorsi tra i due istanti di tempo `t0` e `t1`;



Tempi, date, calendari

- Struttura dati del tempo locale *broken-down time*: **struct tm**
 - `struct tm* localtime(const time_t *time)` : converte il tempo di calendario in *broken-down time* espresso relativamente alla *time-zone* dell'utilizzatore
 - `struct tm* gmtime(const time_t *time)` : analoga alla precedente ma usa come riferimento le Coordinate Temporalì Universali (UTC) cioè quelle di Greenwich (GMT)

struct tm

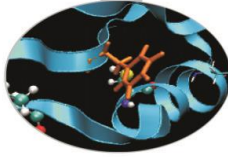


Il tempo di calendario è misurato in secondi ed è comodo per i calcoli ma non per essere visualizzato;

l'ora e la data rappresentati tramite il così detto *broken-down time* dipende ovviamente dalla zona;

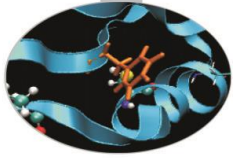
la struttura dati *broken-down time* è costituita dai seguenti dati membro:

```
struct tm{  
    int tm_sec;   int tm_min;   int tm_hour;  
    int tm_mday;           // giorno del mese  
    int tm_mon;           // numero del mese  
    int tm_year;          // anno dal 1900  
    int tm_wday;          // giorno della settimana  
    int tm_yday;          // giorno dell'anno  
    int tm_isdst;         // setting di daylight saving  
    int tm_gmtoff;        /* numero di secondi per passare da  
                           UTC a local time */  
  
    const char *tm_zone; /* abbreviazione di 3  
                           lettere con il nome della  
                           timezone */  
};
```



Tempi, date, calendari

- `time_t mktime(struct tm *bt)`: converte il *broken-down time* *bt* in un dato di tipo calendario nell'ora locale
- `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 riceve in input un valore
`time_t`.



Tempi, date, calendari

Riassumendo:

time_t - calendar time (secondi) - *time()*

struct tm - broken-down time - *localtime()*, *gmtime()*

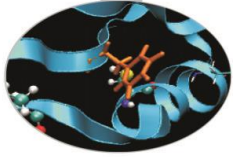
mktime() : broken-down time -> calendar time

difftime() : time interval (secondi)

asctime() : broken-down time -> char string

ctime() : calendar time -> char string

Esempio: uso di srand() e time()



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

int main () { /* e-casuale_time.c */

    int i, pseudo;
    srand(time(NULL));

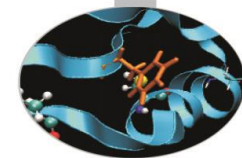
    printf ("Serie di 5 numeri tra 1 e 10: \n");
    for(i=0; i<5; i++){
        pseudo=rand()%10+1;
        printf("%d ",pseudo);
    }

    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(){ /* e-time.c */
```

```
    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));
```

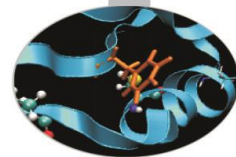
```
    return 0; }
```

- **output:**

Value returned by `time()`: 1146058353

Local time: Sun Jan 22 15:45:26 2017

GMT time: Sun Jan 22 14:45:26 2017

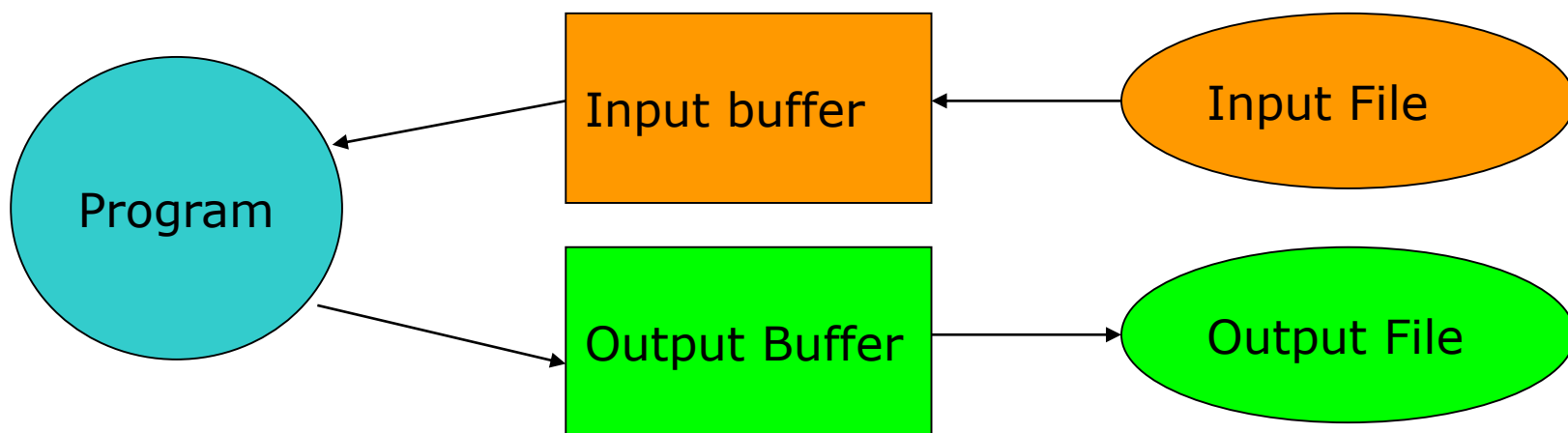


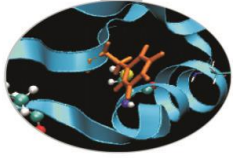
stdio.h

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 un qualsiasi device di I/O, come i file ed il video.

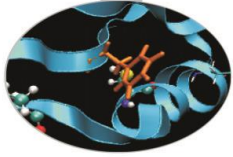
I dati di input vengono letti, anziché uno per uno, in gruppi che vanno ad occupare momentaneamente il buffer, prima di raggiungere il programma. Un analogo procedimento avviene anche in fase di output.





stdio.h: funzioni per l'I/O

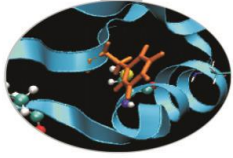
- **int scanf**(const char * <formato [, argomenti , ...]>) : legge dati formattati da stdin;
- **int getchar**(void) : ritorna il primo carattere che viene passato allo stdin. Il carattere viene letto come un carattere e ritornato come intero;
- **char* gets**(char*) : legge un insieme di caratteri dallo stdin e li pone nel buffer fino a che non incontra il carattere di new line o EOF (invio o spazio);



stdio.h: funzioni per l'I/O

- `int printf(const char * <formato [, argomenti , ...]>):` scrive dati formattati su standard output;
- `int putchar(int) :` scrive un carattere nello stdout;
- `int puts(const char*) :` scrive una stringa in stdout e la completa con un carattere di new line.

Esempio: uso di typewriter

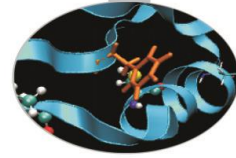


```
#include <stdio.h>

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

Output:

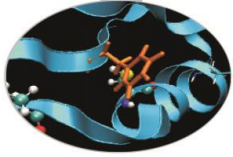
```
./e-program.x
Enter a text. Dot ('.') to exit:
Hello, it's just an example, bye.
Hello, it's just an example, bye.
```



string.h: funzioni per 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 citiamo:

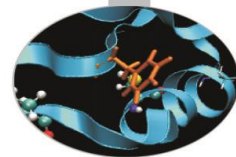
- **char* strcat** (**char *** destination, **const char *** source) : aggiunge la stringa source al termine della stringa destination. Ritorna destination.
- **char* strchr** (**const char*** s, **int** c) : restituisce la prima occorrenza del carattere c all'interno della stringa s. Poiché restituisce un char*, il risultato va interpretato attraverso l'aritmetica dei puntatori;



```
#include <stdio.h>

#include <string.h>

int main () { /* e-stringhe.c */
    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: %p %c \n",ptr_c, *ptr_c);
    printf("stg: %p \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: uso di strchr

Output:

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

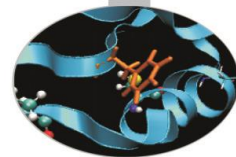
```
ptr_c: 0x7ffc0e4a5b26 C
```

```
stg: 0x7ffc0e4a5b10
```

```
diff: 22
```

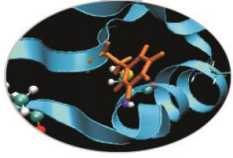
```
Trovata una C in posizione 23
```

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



string.h: funzioni per stringhe letterali

- **int strcmp** (const char * s1, const char * s2) : mette a confronto le stringhe s1 ed s2 carattere per carattere, restituendo 0 se coincidono;
- **char* strcpy** (char * s1, const char * s2) : copia la stringa s2 nella stringa s1;
- **long strlen**(const char* s) : restituisce la lunghezza della stringa s.



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

```
#include <string.h>
```

```
int main () {      /* file e-compara.c */  
    char frase[80], compara[80] = "campanile";  
    int diff;  
  
    strcpy(frase, "bampanile");  
    printf("Si comparano le stringhe di caratteri:\n  
    %s\n    %s\n", frase, compara);  
    diff=strcmp(frase, compara);  
    printf(" diff = %d\n", diff);  
  
    return 0; }
```