37. Le stringhe
Una stringa in C è un *array* di *char*, il cui ultimo elemento è il carattere nullo ('**\0**'),
Esempio: *char stringa[]= "Ciao";*
equivalente a *char stringa[]= {'C','i','a','o','\0'};*
Contiene 5 elementi: i 4 caratteri della costante stringa e il carattere nullo che viene inserito automaticamente.

Importante: la dimensione dell'array deve essere maggiore di almeno 1 rispetto al numero di caratteri che si vuole rappresentare. Infatti deve essere contenuto anche l'elemento nullo **\0**.

Dato che negli array non è definito l'assegnamento, l'uso di una costante stringa (ovvero una cosa del tipo "Ciao") per specificare il valore di una stringa è permesso solo nell'inizializzazione.

È possibile assegnare a una stringa una dimensione *DIM* e poi associarle la costante stringa di una dimensione minore.
char s[100]="Ciao"; //può contenere stringhe di massimo 99 caratteri (c'è '/0')
char s[]="Ciao"; //può contenere stringhe di massimo 4 caratteri

37.1 Input e output di stringhe
Gli operatori di I/O >>, << operano direttamente su stringhe (a differenza dei normali array).

L'operatore di ingresso >>
1. Legge caratteri da cin
2. i memorizza in sequenza finché non incontra una spaziatura (**che non viene letta**)
3. memorizza '**\0**' nella stringa dopo l'ultimo carattere letto
4. termina l'operazione

! Viene letto tutto ciò che c'è fino alla spaziatura (esclusa) **!**

L' operatore di uscita <<
Scrive in sequenza su cout i caratteri della stringa, fino al primo '**\0**' (che non viene scritto).

Esempio:
Leggere una stringa di massimo 255 caratteri e stamparla (ricordiamo che l'ultimo carattere è riservato al carattere nullo **\0** che segna la fine della stringa).

```
char buffer[256];
cin >> buffer;
cout << buffer;
cout << buffer[0]; //stampa la prima lettera della stringa

while(cin>>buffer){          //si inserisce e si stampa una stringa
cout << buffer << endl;      //solitamente non si fa con l'utente, ma con un file dato in input
}
```

Dando un file in input al programma e usando questo ciclo, verranno stampate tutte le parole all'interno del file.

37.1.1 Fornire un file in input al programma
Al momento dell'esecuzione dopo *./a.out* scriviamo: <*directory/nomefile* dove *directory* è la cartella in cui si trova il nostro file e *nomefile* e il nome del file dove ci saranno scritte le cose che il programma inserirà non appena troverà il comando *cin >> … ;*

Nell'esempio sopra riportato:

```
while(cin>>buffer){
cout << buffer << endl;
}
```

```
./a.out <inputs/file1.txt
```

Il ciclo continua finché non si raggiunge l'*end of file*, ovvero finché non è stato letto tutto ciò che si trova nel file passato in input al programma.

N.B.! Le parole dovranno avere tanti caratteri al massimo quanto la dimensione dell'array buffer meno 1 (c'è '\0').

Così invece di scrivere noi l'input al nostro programma, scriviamo ciò che avremmo scritto su un file e successivamente fornito al programma il file.

37.1.2 Alcune funzioni della libreria *iostream* in ottica stringhe

cin.eof(): ritorna un valore diverso da 0 se lo stream ha raggiunto la fine.
! Spazio dopo ultimo elemento
cin.fail(): ritorna un valore diverso da 0 se lo stream è in errore o è in *end of file*
cin.clear(): ripristina lo stato normale dallo stato di errore
cin.ignore(): ignora tutti gli elementi rimasti nello stream dopo l'errore
(entrambi vanno usati dopo che è stato rilevato un errore)

Leggendo input dai file è possibile che si scrivano delle cose che non vanno bene (numeri al posto di stringhe o viceversa). Dopo ogni operazione di lettura da file è opportuno:

- verificare, con *cin.fail()* se c'è un errore in lettura (con un *if*)
  - se non ci sono errori, procedere con le operazioni da fare
  - se ci sono errori, lanciare i comandi *cin.clear()* e *cin.ignore()* per eliminare lo stato di errore

È possibile effettuare un *cin.ignore()* "fai da te" eliminando i caratteri di troppo, mettendoli in una stringa spazzatura:

```
char stringa[100];
cin >> stringa;
```

<u>Per leggere input da un file</u>
In questo modo si continua finché non si ha raggiunto l'eof (efficiente per le stringhe); si possono anche controllare gli errori in seguito.

```
while(cin>>buffer){
… ;
}
```

In questo caso la condizione del ciclo è *false* solo quando siamo alla fine del file (quando si usa *cin.eof()* ricordarsi lo spazio). Oppure con un *do-while.*

```
cin >> value;
while(!cin.eof()){
… ;
cin >> value;
}
```

```
do{
cin>>value;
controlli; istr;
} while(!cin.eof());
```

```
cin >> value;
while(!cin.fail()){
controlli; istr;
cin>> value;
}
```

! Le istruzioni devono essere eseguite a seconda dei controlli. Se grazie ai controlli si vede che un valore non è accettabile si termina il ciclo e si legge un altro valore.

```
while(cin...){  if(controllo){cin.clear();cin.ignore()} else {istr;}  }
```

Altre funzioni (*s* è una stringa, *c* è un carattere e *n* è un intero)
cin.getline(s,n): legge da cin una riga in s fino a capo linea, per un massimo di n-1 caratteri (lo '\n' non viene letto). Restituisce (un oggetto equivalente a) 0 se incontra eof.
Stampa su *s* una riga fino al capolinea.
Può essere anche messo come condizione di un *while* (stile *cin loop*) e in questo modo si legge un file intero fino alla fine (soffermandosi su ogni riga).

cin.get(c): legge da cin in *c* un singolo carattere (spaziature comprese) e restituisce *c* ('/0' se arriva all'eof). Qundi su *c* si salvano tutti i caratteri di un testo. Può essere usato come condizione di un *cin loop* e in questo caso si riescono a leggere tutti i caratteri di un file. Nel *while* giocare con i caratteri precedenti ponendo alla fine *pre=ch* (dove *ch* è il carattere appena letto), così alla chiamata successiva avremo il carattere precedente di quello letto. Assegnare come valore iniziale a *pre* '\0'.

cout.put(c): scrive su cout il singolo carattere c. Stampa il carattere c. *c* può magari essere ottenuto dal *cin.get(c)*.

Importante: il codice ASCII per l'andare a capo è 10, mentre per lo spazio singolo è 32.

37.1.3 Dirigere l'output
Analogamente a come si fornisce un file in input al programma, si può dire al programma di stampare i suoi risultati (i vari output) non a video, ma in un file.
./a.out > directory/file
In questo modo l'output del programma viene stampato sul file di nome *file*. Se non esiste, viene creato.

Si può sia fornire un input che dirigere un output: *./a.out <directory/file1 >directory/file2*.

37.1.3 Funzioni della libreria *cstring*

Nelle funzioni che seguono *s* e *t* sono stringhe e *c* è un carattere:
strlen(s): restituisce la lunghezza di *s* escluso '\0';
strchr(s,c): restituisce un puntatore alla prima occorrenza di *c* in *s*, oppure NULL se c non si trova in *s;*
strrchr(s,c): come sopra ma per l'ultima occorrenza di *c* in *s*;
strstr(s,t): restituisce un puntatore alla prima occorrenza della sottostringa *t* in *s*, oppure NULL se *t* non si trova in *s*;
strcpy(s,t): copia *t* in *s* e restituisce *s;* Dopo questa funzione *s* viene modificata;
strncpy(s,t,n): copia n caratteri di *t* in *s* e restituisce *s*.
Se non c'è lo '\0' negli n caratteri la stringa s non è ben formata!
strcat(s,t): concatena *t* al termine di *s* e restituisce *s;*
strncat(s,t,n): concatena *n* caratteri di *t* al termine di *s* e restituisce *s*.
La stringa di destinazione contiene sempre '\0', vengono copiati n caratteri e sempre aggiunto '0').
***strcmp(s,t)*: restituisce un valore negativo, nullo o positivo se s è alfabeticamente minore, uguale o maggiore di *t*.**
isalnum(c): dato un carattere *c* restituisce TRUE se *c* è alfanumerico, FALSE se non lo è.
Bisogna usare questa funzione per confrontare stringhe, facendo *s==t* si confronta l'l-value dei due array

! Prestare molta attenzione nelle funzioni sopra allo '\0' e che dimensioni delle stringhe di destinazioni siano in grado di contenere risultato!

Note sulle funzioni
strlen(s) restituisce la lunghezza della stringa *s* anche se questa viene inserita dall'utente (in questo caso restituisce solo il numero di caratteri scritti: se inizializziamo una stringa di massimo e la diamo all'utente che ne scrive 8, la funzione *strlen* restituirà 8).
strchr, strrchr e *strstr* restuiscono un indirizzo di memoria (che memorizziamo in un *char* * chiamato *p*). Quindi per ottenere la cella dell'array *s* in cui si trova ciò che cerchiamo facciamo *p-s* (*s* è comunque un array e un puntatore alla prima cella).
strcpy(s,t) sovrascrive completamente *s*.
strncpy(s,t,n) sovrascrive i primi *n* caratteri di *s* con i primi *n* caratteri di *t*. Se *t* presenta meno caratteri di *n* sovrascrive tutta *t* in *s* (infatti sovrascrive il carattere '/0' e termina la stringa).
strcat(s,t) assicurarsi che *s* sia sufficientemente grande da contenere *t*.
strcmp(s,t): va usato per vedere se due stringhe sono uguali. Non si può fare s==c.
isalnum(c): è utile perché può essere usato come co-condizione di un while: si leggono i caratteri della parola fino a che non si legge un carattere diverso da un numero o da una lettera (utile perché nella stringa "ciao, come va?", vengono considerate come "parole" *ciao, come* e *va?*. Quindi con *salnum* si riesce a considerare solamente *ciao come* e *va*).

37.2 Array di stringhe

argv come si vedrà in seguito è una matrice di char, ovvero un array di stringhe.
char tante_parole[MAX_PAROLE][MAX_CHAR_PER_PAROLA+1];

tante_parole = {{Ciao},{Ciao, come va?}, {Ciao}, … };
tante_parole[2][3]='o';
*cout << tante_parole[1]; //*Stampa "Ciao"

Magari contenendo tutte le parole di un file, si può fare con un semplice algoritmo, l'eliminazione dei doppioni. **!** Usarlo " "un array a una dimensione" ": infatti, essendo un array di stringhe (e ipotizzando le stringhe come elementi uniti) stampando con un *for* tante_parole[i] con *i* che va da 0 a *DIM-1* si stampano tutte le parole.

38. Argomenti da linea di comando

In C++ è possibile passare ai programmi argomenti (es. valori numerici, nomi di file,...) direttamente da linea di comando.

./a.out 1000 22.5 miofile //ho passato il valore *1000*, il valore *22.5* e il nome del file *miofile*

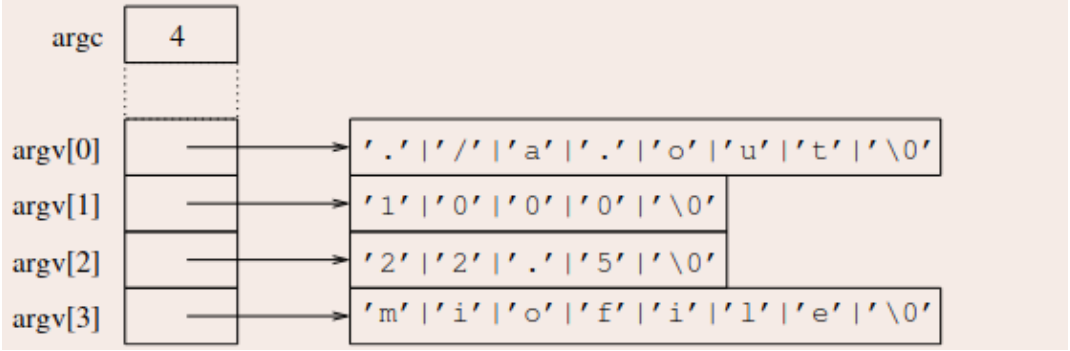
Ciò è possibile tramite due parametri formali predefiniti della funzione main: **argc** e **argv**.

int main (int argc, char * argv[])

- Nellintero **argc**, in cui viene automaticamente copiato il numero delle parole della riga di comando ("./a.out" o analogo inclusa, quindi nell'esempio sopra *argc=4*);
- Nell'array di puntatori a caratteri (stringhe) **argv** vengono automaticamente copiate le parole della linea di comando e ha dimensione **argv**.

Importante! Gli argomenti sono stringhe: se rappresentano numeri, devono essere convertiti tramite le funzioni *atoi* o *atof* della libreria *<cstdlib>*.

```
int main (int argc, char * argv[])
{...}
>./a.out 1000 22.5 miofile
```



Se volessi usare *1000* e *22.5* come due valori, il primo di tipo *int* e il secondo di tipo *float*, avrei dovuto fare così:

int val1=atoi(argv[0]);
int val2=atof(argv[1]);

Se voglio che due parole separate formino la stessa stringa le metto tra "".

Per stampare a video ciò che è stato passato a linea di comando

```
for (int i = 0;i<argc;i++)
   cout << argv[i] << endl;
```

38.1 Controllo dell'inserimento

È importante controllare che l'utente abbia inserito il numero corretto di parametri.

Dopo il *main* è opportuno verificarlo, verificano *argc*: se doveva inserirne 4, verifico che se ho argc!=5 (contando l'*a.out*), devo mandare un messaggio di errore *cout << "Errore" << endl;* e comunicare il corretto utilizzo della linea di comando *cout << "Usage: ./a.out <a> <b> <c> <d>" << endl;* dove *a*, *b*, *c* e *d* sono i quattro argomenti che vengono inseriti.

In caso di errore nell'inserimento è opportuno chiudere il programma con il costrutto ***exit(1)*** (che blocca immediatamente l'esecuzione del programma). L'1 tra parentesi indica che c'è stato un errore.

Importante: facendo *argv[n][m]* prendo l'emmesimo carattere della ennesima+1 parola inserita.

Ad esempio: *./a.out ciao 12 → argv[1][3] = 'o' argv[2[0]='1'*

Pattern importante per vedere quanti caratteri ha *argv[x]*
Ricordando che *argv[]* è un array di stringhe ovvero una matrice *argv[m][n]* dove *m* sono le stringhe e *n* il massimo dei caratteri di ciascuna stringa, possiamo ricavare il numero di caratteri di una generica stringa *argv[m]*.

```
int i=0;
while(argv[1][i]!='\0'){
i++;
}

int dimARGV=i;
```

dimARGV è la dimensione della stringa, senza il carattere nullo '/0'.
Se *dimARGV[2]*, per ho *argv[1][0]="char"* e *argv[1][1]="char"*. Ho chiaramente anche *argv[1][2]='/0'*.

39. Stream e I/O su File di Testo
In C++ sono possibili operazioni di I/O direttamente da file di testo (senza usare <,> e fornire manualmente un file in input al programma).
È possibile tramite la libreria ***<fstream>***. È possibile definire uno *stream*, a cui associare (i nomi di) file di testo.

Lo stream viene aperto e associato al nome di un file tramite il comando *open*, in tre possibili modalità: lettura da file (input, ovvero si legge ciò che c'è sul file come valore di input), scrittura su file (output, ovvero si scrive il risultato del programma sul file) e scrittura a fine file (append).
Lo stream può essere utilizzato per tutte le operazione di letture [resp. scrittura] a seconda della modalità di apertura. Si vedrà che non è detto essere un file, ma può essere anche un device (sia per l'input che per l'output), … .
! Uno stream, quando è stato utilizzato, deve essere chiuso mediante la funzione *close*.

N.B.! *fstream* "eredita" da *iostream* sostanzialmente tutti i suoi operatori e funzioni di lettura e scrittura, nelle rispettive modalità (Es: <<, >>, get, put, getline, eof, fail, clear, ignore, …).

39.1 Sintassi delle operazioni su stream.
Definizione di uno stream:
Sintassi: *fstream nomestream;*
Esempio: *fstream myin,myout,myapp;*

Apertura di uno stream:
Sintassi: *nomestream.open(nomefile,modo);*
Se non si specifica il modo è specificato di default sia su scritura che su lettura (ma è meglio sempre specificare).
! *nomefile* identifica il file su cui verranno fatte le operazioni di lettura e scrittura **!**
Esempio:
- Lettura: *myin.open("ingresso.txt",ios::in);*
- Scrittura: *myout.open("uscita.txt",ios::out);*
- Append: *myapp.open("uscita2.txt",ios::out|ios::app);*

Utilizzo di uno stream:
Sintassi: analoga a quella di *cin* e *cout*
Esempio: *myin>> a; myout << b; myin.get(c); myapp.put(c); myin.fail();* …

Chiusura di uno stream:
Sintassi: *nomestream.close();*
Esempio: *myin.close(); myout.close();*

39.1.1 Apertura di un file
Apertura in modalità lettura (*ios::in*)
Il file associato deve già essere presente. Il puntatore si sposta all'inizio dello stream.

Apertura in modalità scrittura *(ios::out)*
Il file associato se non presente viene creato (se presente si cancella tutto ciò che aveva sopra). Il puntatore si posizione all'inizio dello stream (sovrascrivendo il file).

Apertura in modalità append *(ios::out|ios::app)*
Il file associato se non è presente viene creato. Il puntatore si posiziona alla fine dello stream.

39.1.2 Chiusura di uno stream

Alla fine del programma tutti gli stream aperti vengono automaticamente chiusi
Una volta chiuso, uno stream può essere riaperto in qualunque modalità e associato a qualunque file. Uno stream per essere utilizzato deve essere riaperto (in qualunque modalità e associato a qualunque file).
Importante: È buona prassi di programmazione chiudere ogni stream aperto quando non più necessario (il sistema operativo ha un limite sul numero di file che possono essere aperti per un programma, vedi *ulimit -a* su bash).

39.1.3 Passaggio di stream a funzioni

Uno stream può essere dichiarato globalmente, ma per essere passato ad una funzione la cosa migliore è fare il passaggio per parametri PER RIFERIMENTO dello stream alla funzione.
void funz(fstream &);

39.2 Verificare e gestire eventuali errori

Quando si effettua I/O su files è ricorrente che accadano degli errori. I più comuni e quelli che devono essere gestiti con degli *if* sono:

- file in input non esistente,
- file in output non scrivibile,
- errore nell'apertura degli stream (da verificare con *nomestream.fail()*).

Un file può essere reso non scrivibile in questo modo: *chmod -w nomefile*. Ora l'output su questo programma non è disponibile (come non è disponibile aprire il file e scriverci sopra). Col comando *chmod 777 nomefile* si ritorna a poter fare tutto sul file.

39.3 Uso di *fstream* con *argc* e *argv.*

Potrebbe essere desiderabile passare i nomi dei file ai programmi. In questo caso si usano i due parametri formali del *main* per passargli informazioni.
./a.out file1.txt file2

I nomi dei file vengono passati tramite *argc* e *argv.*

```
int main (int argc, char * argv[]){
fstream myin,myout;
myin.open(argv[1],ios::in);
myout.open(argv[2],ios::out);
}
```

Controllo errori

Ipotizziamo un programma dove a command line vengono dati due argomenti: rispettivamente un file di lettura e un file di scrittura.

L'utente non mette i giusti argomenti

```
if (argc!=3) {
cerr << "Usage: ./a.out <source> <target>\n";
exit(1);          //per terminare il programma
}
```

```
fstream myin, myout; //apriamo i vari stream
myin.open(argv[1], ios::in);
myout.open(argv[2], ios::out);
```

Il file input non esiste
```
if (myin.fail()) {
cerr << "Il file " << argv[1] << " non esiste\n";
myin.close(); myout.close();          //ricordarsi di chiudere gli stream
exit(1);
}
```
Errori col file di output (ad esempio non è scrivibile)
```
if (myout.fail()) {
cerr << "Errore con " << argv[2] << "\n";
myin.close(); myout.close();          //ricordarsi di chiudere gli stream
exit(1);
}
```
Di solito se il file non è scrivibile non viene segnalato errore, ma non viene poi successivamente scritto nulla in esso.

Si può stampare il messaggio di errore sia con *cerr* che con *cout*.

Comando *grep* va a cercare in un file l'occorrenza di una stringa.

Estrarre solo gli interi da un file con numeri e caratteri
! Ricordarsi di mettere un separatore dopo l'ultimo elemento lettoc
```
map >> val;
 while(!map.eof()){
  if(map.fail()){
   map.clear();
   map.ignore();
  } else {
   mappa[m]=val;
   m++;
  }
  map >> val;
 }
```

39.4 Comando imperativo sulla shell
Se sul termine lanciamo il comando *file <nomefile>*, ci usciranno a video una serie di informazioni sul file.
! Importante ! È utile con i file *.txt* per vedere se terminano con con un separatore (vediamo a video il messaggio *ASCII text*) o senza (*ASCII text, with no line terminators*).

39.5 Alcuni pattern importanti

Salvataggio di tutte le parole di un file in un array di stringhe, così definito
char parole[MAX_PAROLE][MAX_CARATTERIPERPAROLA + 1];

```
while(input >> buffer){
  strcpy(parole[i],buffer);
  i++;
}
```

Dove *i* (inizializzata prima a zero) sarà la dimensione virtuale del nostro array.

Salvatagio di tutti i caratteri (spazi inclusi) in un array di caratteri, così definito *char caratteri[MAX_CARATTERI +1];*

```
while(input.get(c)){
 parole[i]=c;
  i++;
}
```

Copia di un file su un altro file

```
while (myin.get(c)) {
  myout.put(c);
}
```

Si possono fare varianti, copiando solo determinati caratteri, ecc.

Stampa di tutte le parole di un file

```
for(int c=0;c<i;c++)
  cout << parole[c] << endl;
```

Rimozione di parole uguali in un array di stringhe

```
for(int tot=0;tot<i;tot++){
  for(int check=0;check<i;check++){
    if(!strcmp(parole[tot],parole[check])){
        if(check!=tot){
          for(int m=check;m<i;m++){
            strcpy(parole[m],parole[m+1]);
          }
          i--;
        }
    }
  }
}
```

In questo modo si aggiorna anche la dimensione virtuale della stringa, togliendo i doppioni e mantenendo le parole singole.

Con l’allocazione dinamica di memoria si può definire un array della dimensione giusta.