LE STRINGHE (manualetto salvavita)

Una stringa è un insieme di caratteri.

*(Libro di terza pagg. 257-275).*

Una stringa può essere implementata con un array di char l’ultimo carattere della stringa deve essere seguito dal terminatore di stringa \0;

Dovendo contenere il terminatore occorre ricordare che.. a dimensione dell’array deve essere di un carattere più lunga della stringa massima .

char FELINI[10]; // dichiarazione di una stringa in C.

char FELINI[10]=”4GATTI”; //dichiarazione con inizializzazione contestuale.

char FELINI[10]={ ‘4’, ‘G’ , ’A’ , ’T’ , ’T’ , ’I’ ,’\0’}; // ……………….

char FELINI[]=”4GATTI”;  //dichiarazione con inizializzazione dimensionamento contestuale.

char FELINI[]={‘G’ , ’A’ , ’T’ , ’T’ , ’I’ , ‘\0’}; // ……………….

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4 | G | A | T | T | I | \0 | … | … | … |

Si può accedere al contenuto di ogni posizione

cout<<FELINI[2]; // visualizza A

FELINI=”tigre” //nun se po’ fa’

…. Questo si può fare:

FELINI[0]=’t’;

FELINI[1]=’i’;

…

FELINI[4]=’e’;

FELINI[5]=’\0’;

Nel file header del C <string.h> sono contenuti i prototipi di alcune funzioni che facilitano l’uso delle stringhe.

int strlen(char[]) ; // funzione che restituisce la lunghezza della stringa passata come parametro.

Esempio: cout<<strlen(FELINI); //visualizza 6

… ed altre sulle stringhe ( <https://www.codingcreativo.it/libreria-string-h-in-c/>)

**strcmp()**, **strcpy()**, **strcat(), strncmp(), strncpy()**, **strncat().**

… completa tu

… ed altre sui caratteri: (ctype.h) (<http://web.tiscali.it/kirishima/c_libreria.htm>)

(tolower, toupper, isgraph, islower, isprint, ispunct, isspace,isupper, isdigit….)

**Lettura stringhe da tastiera:**

cin legge solo fino al primo carattere spazio.

cin>> a; //se leggo “ciao mondo”

cout <<a; //scrive ciao

gets(a); //se leggo “ciao mondo”

anche ….. getline (cin,a)

cout <<a; //*scrive* ciao mondo

la lettura ripetuta di stringhe può dare problemi perché il carattere di return rimane nel buffer di input e nella successiva lettura viene letto come se fosse un nuovo input.

Tra una lettura e l’altra svuotare il buffer con fflush(stdin);

|  |  |  |
| --- | --- | --- |
| **Stringhe di Caratteri.** |  |  |
|  | | |

Nei programmi visti finora abbiamo usato soltanto variabili numeriche i cui valori possono essere unicamente dei numeri (ricordiamo che anche i valori di variabili di tipo **char** sono interpretati come numeri interi). Oltre alle variabili numeriche si possono anche dichiarare variabili i cui valori sono stringhe di caratteri. Tali variabili ci permettono di elaborare successioni di caratteri quali parole, frasi, nomi, testi, eccetera. Finora abbiamo usato le stringhe di caratteri come costanti ma non abbiamo ancora visto variabili che possano contenerle.

In C/C++ non vi è un tipo di variabile  *predefinito* in grado di memorizzare delle stringhe di caratteri. Dobbiamo usare degli array di caratteri.

In C++ sono comunque predefinite alcune operazioni e funzioni sugli array di caratteri (ereditate dal C) che facilitano il loro uso come variabili di tipo *stringa di caratteri*.

Vediamo  ora il trattamento standard (stile C) delle stringhe come array di caratteri.

Il seguente array:

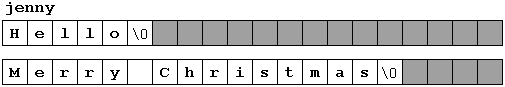
**char jenny [20];**

può memorizzare una stringa di al più 20 caratteri. Possiamo rappresentarlo come segue:



Naturalmente non è necessario usare tutti e 20 i caratteri dell'array. L'array **jenny** si può usare per memorizzare sia la stringa di 5 caratteri **"Hello"** sia la stringa di 15 caratteri **"Merry Christmas"**. Siccome l'array può contenere stringhe più corte della sua dimensione occorre prevedere una indicazione del punto in cui termina la stringa. Per convenzione tale punto viene indicato dal carattere nullo che si può scrivere sia **0** sia **'\0'**.

Ad esempio:



Osserviamo che dopo il contenuto effettivo della stringa viene aggiunto un carattere nullo (**'\0'**) per indicare la fine della stringa. Pertanto l'array jenny può contenere al più stringhe di 19 caratteri.

## Inizializzazione delle stringhe

Per inizializzare una stringa di caratteri si può usare la stessa notazione usata per gli array:

**char mystring[] = { 'H', 'e', 'l', 'l', 'o', '\0' };**

Abbiamo così inizializzato una stringa (array) di 6 valori di tipo **char**: la parola **Hello** più il carattere nullo **'\0'**.

Possiamo anche inizializzare un array di caratteri usando una stringa costante. Abbiamo già visto delle stringhe costanti quali ad esempio:

"Risultato: "

e le abbiamo usate in istruzioni di stampa. Esse si rappresentano racchiudendo la sequenza di caratteri tra doppi apici (**"** ). Alle stringhe costanti viene sempre aggiunto implicitamente un carattere nullo finale **'\0'**.

Possiamo quindi inizializzare indifferentemente la variabile **mystring**  in uno dei modi seguenti:

**char mystring [] = { 'H', 'e', 'l', 'l', 'o', '\0' };  
char mystring [] = "Hello";**

In entrambi i casi la dimensione dell'array **mystring**   è di 6 elementi di tipo **char**: i 5 caratteri di **Hello** e il carattere nullo finale ( '\0' ).

Attenzione: una stringa costante può essere usata per dare un valore iniziale ad una variabile di tipo array di **char**soltanto al momento della dichiarazione dell'array, ossia in fase di inizializzazione. Assegnazione quali:

**mystring = "Hello";  
mystring[] = "Hello";**

non sono permesse, come non è permesso:

**mystring = { 'H', 'e', 'l', 'l', 'o', '\0' };**

## Assegnazione di valori alle stringhe

Siccome in una assegnazione l'*lvalore* può essere soltanto un elemento di un array e non l'intero array, per assegnare una stringa di caratteri ad un array di **char**dobbiamo scrivere:

**mystring[0] = 'H';**  
**mystring[1] = 'e';**  
**mystring[2] = 'l';**  
**mystring[3] = 'l';**  
**mystring[4] = 'o';**  
**mystring[5] = '\0';**

Siccome questo non è molto pratico la libreria standard **cstring** (che si può includere con **#include <string.h>** ) contiene la definizione di un certo numero di funzioni quali **strcpy** (**str** ing **c**o**py**) che si può richiamare nel seguente modo:

**strcpy (***string1***,***string2***);**

L'effetto è copiare il contenuto di *string2*   in *string1*.  *string2* può essere sia un array sia una stringa costante, il che ci permette di assegnare la stringa costante **"Hello"** al'array di caratteri **mystring** usando la seguente notazione:

**strcpy (mystring, "Hello");**

Vediamo un esempio:

|  |  |
| --- | --- |
| ***// assegnazione a stringhe* #include <iostream.h> #include <string.h>  int main () {   char stMyName [20];   strcpy (stMyName,"J. Soulie");   cout << stMyName;   return 0; }** | **J. Soulie** |

Possiamo anche definire noi una semplice funzione **setstring** che si comporta esattamente come **strcpy** della libreria standard:

|  |  |
| --- | --- |
| ***// assegnazione a stringhe* #include <iostream.h>  void setstring (char stOut [], char stIn []) {   int n=0;   do {     stOut[n] = stIn[n];   } while (stIn[n++] != '\0'); }  int main () {   char stMyName [20];   setstring (stMyName,"J. Soulie");   cout << stMyName;   return 0; }** | **J. Soulie** |

Un altro modo per assegnare un valore ad un array di caratteri è quello di usare direttamente il flusso di input **cin**. Nella libreria iostream è infatti definita una funzione **getline**il cui prototipo è:

**cin.getline ( char***buffer***[], int***length***, char***delimiter***= ' \n');**

  dove ***buffer*** è l'array di caratteri in cui memorizzare l'input, ***length*** è la dimensione dell'array stesso e ***delimiter***  è il carattere usato per indicare la fine dell'input e per il quale è previsto il carattere *nuova linea* (**'\n'** ) come valore di default.

Ecco un esempio di uso di **cin.getline** :

|  |  |
| --- | --- |
| ***// uso di cin.getline* #include <iostream.h>  int main () {   char buff [100];   cout << "Come ti chiami? ";   cin.getline (buff,100);   cout << "Salve " << buff << ".\n";   cout << "La tua squadra preferita? ";   cin.getline (buff,100);   cout << "L'" << buff << " piace anche a me.\n";   return 0; }** | **Come ti chiami? Juan Salve Juan. La tua squadra preferita? Inter L'Inter piace anche a me** |

Si può anche usare l'operatore di estrazione  (**>>**)  per leggere delle stringhe da **cin** :

**cin >> buff;**

che funziona ma con le seguenti limitazioni che **cin.getline** non ha:

* si possono leggere soltanto parole e non intere frasi in quanto l'operatore di estrazione usa come delimitatore qualsiasi occorrenza di un carattere invisibile (spazio, tabulazione, nuova linea, ritorno carrello).
* non si può specificare la dimensione dell'array il che rende instabile il programma nel caso in cui l'input sia una parola più lunga della dimensione dell'array.

Le seguenti sono alcune altre funzioni che operano su stringhe e che sono definite nella libreria **cstring** (**string.h** ):

**strcat:**

**char\* strcat (char\****dest***, const char\****src***);**

Aggiunge (appende) la stringa *src* alla fine della stringa *dest*. Ritorna *dest*.

**strcmp:**

**int strcmp (const char\****str1***, const char\****str2***);**

Confronta le stringhe *str1* ed *str2* . Ritorna **0** se sono uguali.

**strcpy:**

**char\* strcpy (char\****dest***, const char\****src***);**

Copia il contenuto di *src* in *dest*. Ritorna *dest*.

**strlen:**

**size\_t strlen (const char\****str***);**

Ritorna la lunghezza di *str*.

NOTA: **char\*** ha lo stesso significato di **char[]**