LE STRINGHE

Manuale linguaggio C

Le stringhe

Non esiste un tipo specifico per le stringhe; esse si rappresentano con un vettore di caratteri. Es. "Hello World" è:

Nella rappresentazione interna il vettore stringa è terminato dal carattere nullo \0 (NULL terminator) che ne rappresenta il delimitatore finale in modo che il programma possa trovarne la fine. Es.:

```
char stringa[100];
```

Un vettore di 100 caratteri, cioè potenzialmente una stringa di 99 caratteri più \0.

Per usare un vettore di caratteri come una stringa serve assicurarsi che la lunghezza del vettore sia maggiore di quella dell'inizializzazione. In caso contrario il compilatore ometterà il carattere terminatore rendendo il vettore non usufruibile come stringa.

Dichiarazione e inizializzazione

Una stringa di caratteri si può inizializzare, come ogni altro array, elencando le singole componenti:

```
char s[4] = {'a', 'p', 'e', '\0'};
```

oppure anche, più brevemente, con la forma compatta seguente:

```
char s[4] = "ape" ;
```

oppure

```
char nome[ ] = “ape” ;
```

in questo caso la stringa viene dimensionata automaticamente a 4

Il carattere di terminazione ‘\0’ è automaticamente incluso in fondo. Attenzione alla lunghezza!

Dichiarazione e inizializzazione

Il compilatore segnala un errore se si dichiara la lunghezza della stringa n, e si inizializza con una stringa costante di lunghezza >n

```
char str[3]="quattro"; /* SCORRETTO */
char str1[3]="tre";      /* CORRETTO */
```

I compilatori ANSI, generalmente, consentono di specificare una dimensione di array che non includa il carattere terminatore . Esempio:

```
char str[7]="quattro"; /* CORRETTO */
```

Non c'è alcuno spazio per il carattere terminatore e quindi il compilatore non tenta di metterne uno.

Se la stringa con cui si inizializza un vettore è più corta della dimensione del vettore, il compilatore aggiunge caratteri terminatori, così come succede con il valore zero quando viene inizializzato un vettore di interi. Esempio:

```
char str[10]="quattro";
```

str conterrà i caratteri: q u a t t r o \0 \0 \0

Vettori di caratteri e ptr a caratteri

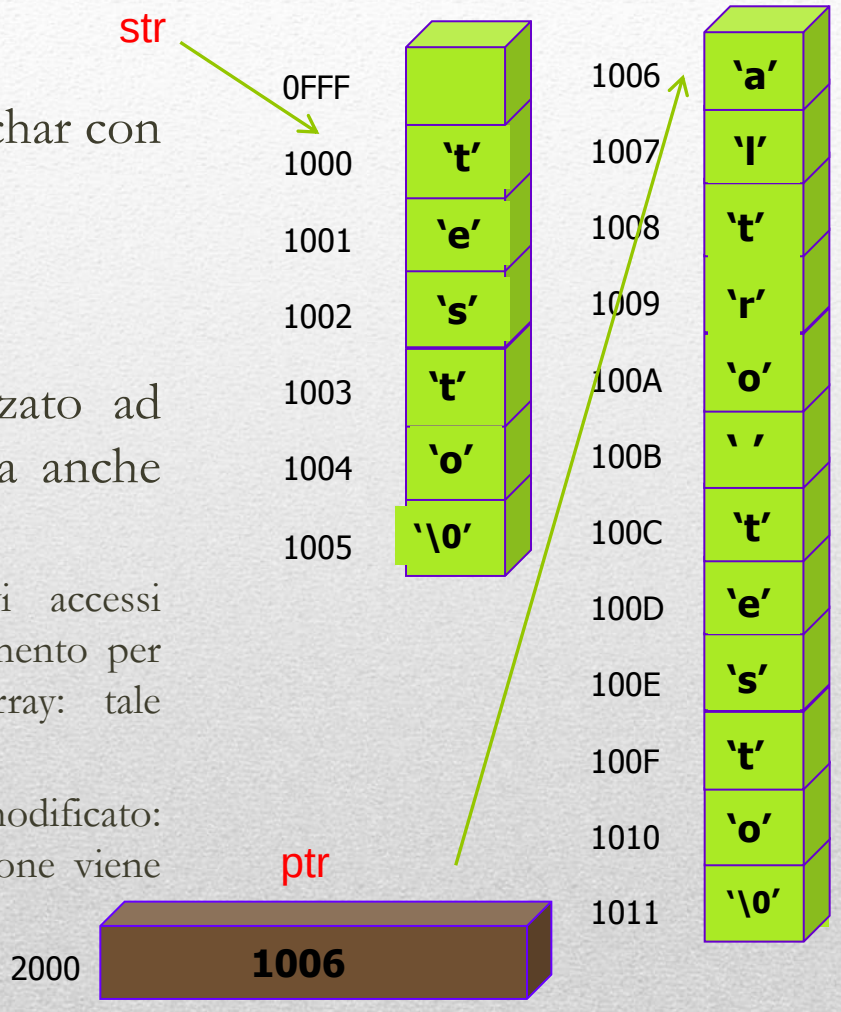
char str[]="testo";

È possibile inizializzare un puntatore a char con una stringa costante:

char *ptr = "altro testo";

si crea un array di caratteri, inizializzato ad "altro testo", riservando però memoria anche per il puntatore

- Nel caso dell'array, tutti i successivi accessi utilizzano il nome dell'array come riferimento per l'indirizzo dell'elemento iniziale dell'array: tale indirizzo non può essere modificato
- Il puntatore è una variabile e può essere modificato: l'indirizzo relativo alla prima inizializzazione viene però perso



Vettori di caratteri e ptr a caratteri

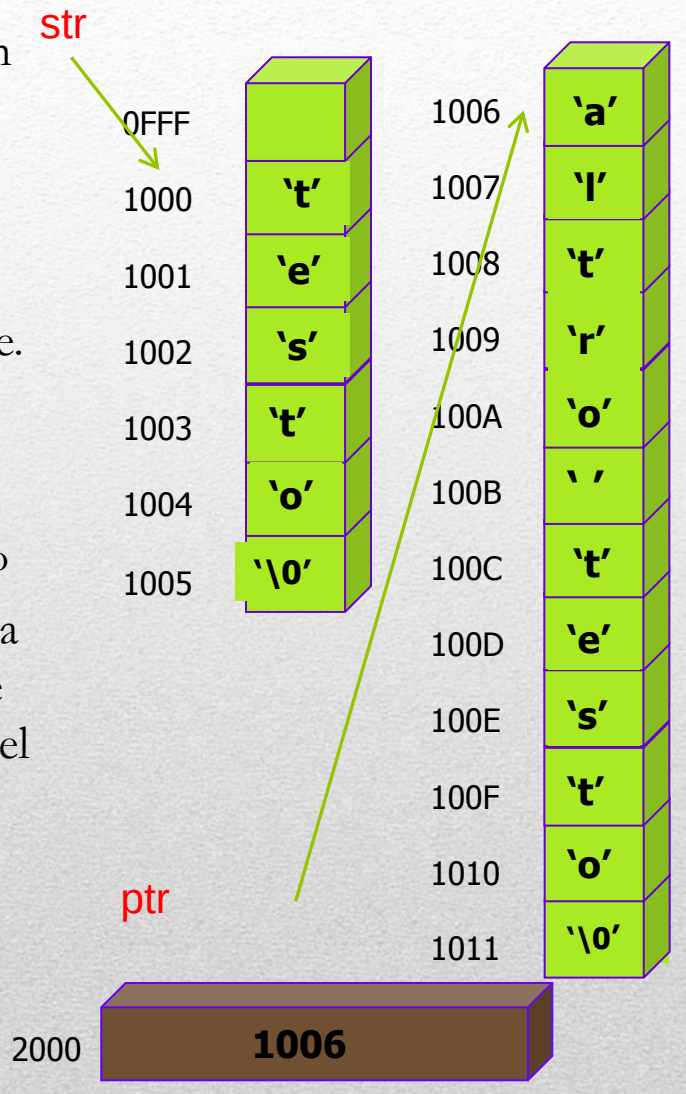
Qualsiasi funzione che si aspetti che le venga passato un vettore di caratteri o un puntatore a carattere, accetterà come argomento entrambe le versioni. Ma:

- Nella versione vettore, i caratteri presenti possono essere modificati come tutti gli elementi di un vettore. Nella versione puntatore, il puntatore punta a una stringa letterale che non deve essere modificata:

  *ptr='a'; // SBAGLIATO , comportamento indefinito

- Nella versione vettore, str è il nome del vettore. Nella versione puntatore ptr è una variabile che può essere fatta puntare ad altre stringhe durante l'esecuzione del programma.

Se abbiamo bisogno di una stringa che possa essere modificata, è nostra responsabilità creare un vettore di caratteri nel quale memorizzarla



Gli assegnamenti a stringhe

Una stringa costante viene quindi interpretata come un puntatore al primo carattere della stringa. Utilizzare un puntatore non inizializzato è un errore molto grave

```
#include <stdlib.h>
int main()
{
        char array[10];
        char *ptr1 = "10 spazi";
        char *ptr2;

        array = "not OK"; /* SBAGLIATO, non è possibile assegnare un indirizzo */
        array[5] = 'A';     /* OK */
        *ptr1='a';          /* SBAGIATO, non si modifica una stringa letterale */
        *ptr2 = "not OK";/* SBAGLIATO, tipi non compatibili*/
        ptr2[0]='a';        /* SBAGLIATO, ptr2 non inizializzato */
        return 0;
}
```

Stringhe e caratteri

Occorre notare la differenza fra stringhe costanti e costanti di tipo carattere:

```
char ch = 'a';  /* Per 'a' è riservato un byte */
```

```
/* Vengono riservati due byte per "a", oltre allo
 * spazio necessario alla memorizzazione di ps
 */
char *ps = "a";
```

È possibile assegnare una costante carattere al contenuto di un puntatore a char; è invece scorretto effettuare la stessa operazione relativamente ad una stringa

```
char *p1;
*p1 = 'a';  /* OK */
*p1 = "a"; /* not OK */
```

```
char *p2;
p2 = 'a'; /* not OK */
p2 = "a";/* OK */
```

Le stringhe sono interpretate come puntatori a carattere

Stringhe e caratteri

Le inizializzazioni e gli assegnamenti non sono simmetrici; è infatti possibile scrivere

```
char *p = "string";
```

Puntatore a carattere

ma non…

```
*p = "string";
```

Carattere

Nota: vale per inizializzazioni ed assegnamenti di tutti i tipi di dati

```
float f;
float *pf = &f;  //OK
*pf = &f;  // SBAGLIATO
```

Puntatore a float

Float

Puntatori e stringhe

Due modi per dichiarare ed inizializzare una stringa:

```
char str_array[] = "Hello World!";
char *str_ptr    = "Hello World!";
```

Per quanto visto prima le seguenti espressioni:

```
str_array[6]
*(str_array+6)
*(str_ptr+6)
str_ptr[6]
```

restituiscono tutte il carattere 'W'.

Scrivere le stringhe

Per stampare stringhe tramite **printf()** usare la specifica %s, anche specificando il numero di caratteri da stampare:

```
char str[] = "stringa di prova";
printf("%s\n", str);
printf("%.6s\n", str);
```

L'argomento della funzione printf() deve essere un puntatore ad un array di caratteri terminato dal carattere nullo (che non viene stampato)

Oppure, usare la funzione puts() che accetta come unico argomento la stringa che deve essere stampata e che avanza alla riga di output successiva dopo la stampa .

```
puts(str);
```

Leggere le stringhe

Per leggere le stringhe tramite **scanf()** usare la specifica %s, essa salterà gli spazi bianchi e poi leggerà tutti i caratteri fino a un carattere che rappresenta uno spazio bianco memorizzandoli nel vettore. La scanf() mette sempre il carattere terminatore *null* alla fine della stringa.

```
char str[20];
scanf("%s", str);
```

Una stringa letta tramite la **scanf()** non conterrà mai degli spazi bianchi; quindi generalmente **non** viene usata per leggere **un'intera riga dell'input.**

Per leggere un'intera riga di input serve usare la funzione **gets()**: legge i caratteri di input, li memorizza nel vettore e alla fine aggiunge il terminatore. Ma:

- la gets() non salta gli spazi bianchi che precedono l'inizio della stringa
- la gets() legge fino a quando trova new-line (la scanf() si ferma al primo carattere che rappresenta uno spazio bianco). La gets() scarta il carattere new-line e al suo posto memorizza il terminatore di stringa.

Leggere le stringhe: esempio

```
char sentence[SENT_LEN+1];


printf("Enter a sentence:\n");
scanf("%s", sentence);
```

Attenzione alla **dimensione del vettore**; sia la scanf() che la gets() non hanno modo di stabilire quando sia pieno, rischiando di salvare dei caratteri oltre la fine del vettore e provocando un **comportamento indefinito**!! La scanf() può essere usata specificando il massimo numero di caratteri da memorizzare.

Se l'utente inserirà la riga:

Prova di inserimento di una riga di input

la scanf() memorizzerà solo la stringa «Prova» nella variabile *sentence*. Un eventuale chiamata successiva alla scanf() riprenderà la lettura della riga dallo spazio successivo alla parola «Prova».

Se invece leggiamo la stringa usando la funzione gets():

```
printf("Enter a sentence:\n");
gets(sentence);
```

La gets() è intrinsecamente non sicura, la funzione **fgets()** è un'alternativa decisamente migliore.

A fronte dello stesso input, la gets() salverà all'interno del vettore **tutta** la riga di input

Morris worm

Nel 1988 Robert Morris svolgeva il primo anno del dottorato in informatica preso la Coronell University, dopo essersi laureato ad Harvard; grazie a questo aveva ottenuto un account per accedere ad Internet tramite il computer dell'università. Nel tentativo di determinare le dimensioni di Internet, Morris lancia un programma C auto-replicante, il primo worm, che apriva una connessione con altre macchine in remoto per poi installarvicisi.

Per un errore di programmazione il "Morris Worm" arrivó ad intasare circa il 10% dell'allora Rete: quasi 6.000 computer, tra cui sistemi universitari e governativi che rimasero offline per due giorni creando, di fatto, un nuovo livello di problemi di sicurezza informatica.

Morris fu la prima persona condannata per violazione del **Computer Fraud and Abuse Act**, una legge approvata appena due anni prima: la sua pena consistette in tre anni di libertà condizionata, 400 ore di servizi socialmente utili e 10.050 dollari di multa.

Un **worm** é simile ad un virus, ma a differenza di quest'ultimo non necessita di un file ospite a cui agganciarsi per agire e diffondersi sebbene provveda a modificare il computer che infetta, in modo da essere eseguito ad ogni avvio rimanendo costantemente attivo.

Morris worm

WORM E' un processo che sfrutta il meccanismo di generazione per minare le prestazioni del sistema; esso genera continuamente copie di sé stesso logorando le risorse, talvolta fino a rendere il sistema inutilizzabile da tutti gli altri processi.
I worm sono potenti sulle reti di trasmissione perché hanno la possibilità di riprodursi sui diversi sistemi a esse collegati e quindi di far cadere l'intera rete.

Il suo worm sfrutta, fra gli altri, il bug caratteristico di molte funzioni di I/O presenti nelle librerie C che non controllano la dimensione dei buffer in gioco, permettendo di uscire dallo stack. I problemi di buffer overflow si verificano quando i dati scritti in un buffer corrompono anche i valori dei dati in indirizzi di memoria adiacenti a causa di insufficiente controllo dei limiti..

I **Buffer overflow** devono quindi essere evitati mantenendo un elevato grado di correttezza nel codice che esegue la gestione del buffer, ad esempio evitando funzioni della libreria standard che non operano controlli sui limiti, come gets(), scanf() e strcpy() o usando strncpy() invece di strcpy() per limitare i bordi del buffer.

In particolare, Morris sovrascrisse buffer di 512 character con 536 caratteri; i 24 extra contenevano il programma eseguibile.

Morris worm

Esempio:

```
int main()
{
        int buffer[10];
….
        buffer[20]=5;
}
```

La zona di memoria *buffer* viene creata nello stack ed è temporanea per la funzione main

Non viene effettuato alcun tipo di controllo di dimensione

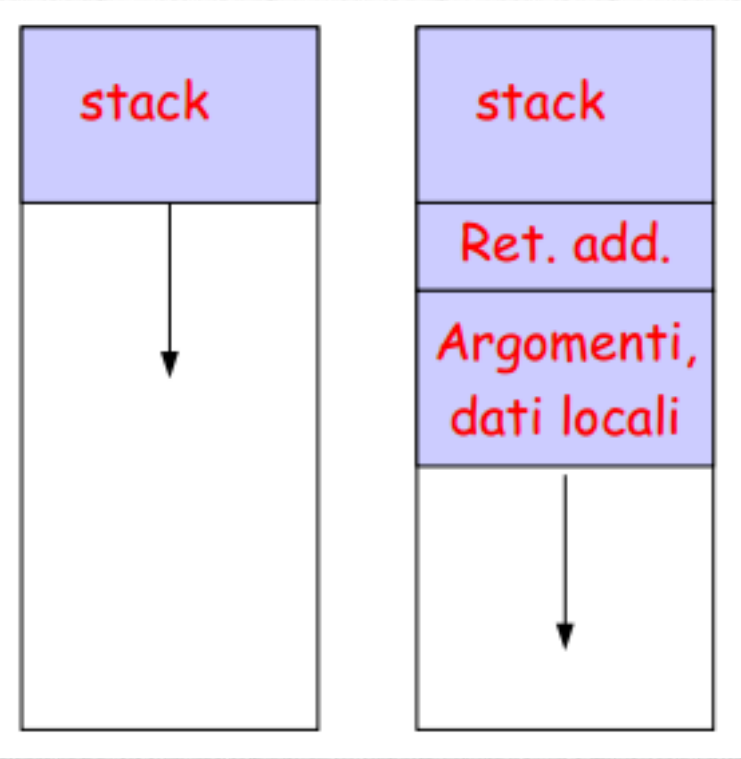
L'assegnamento del valore 5 sovrascrive eventuali altri dati già presenti nello stack.

Morris worm

Ad ogni chiamata di funzione, nello stack vengono collocati:

- Indirizzo di ritorno
- Eventuali argomenti della funzione
- Eventuali dati locali

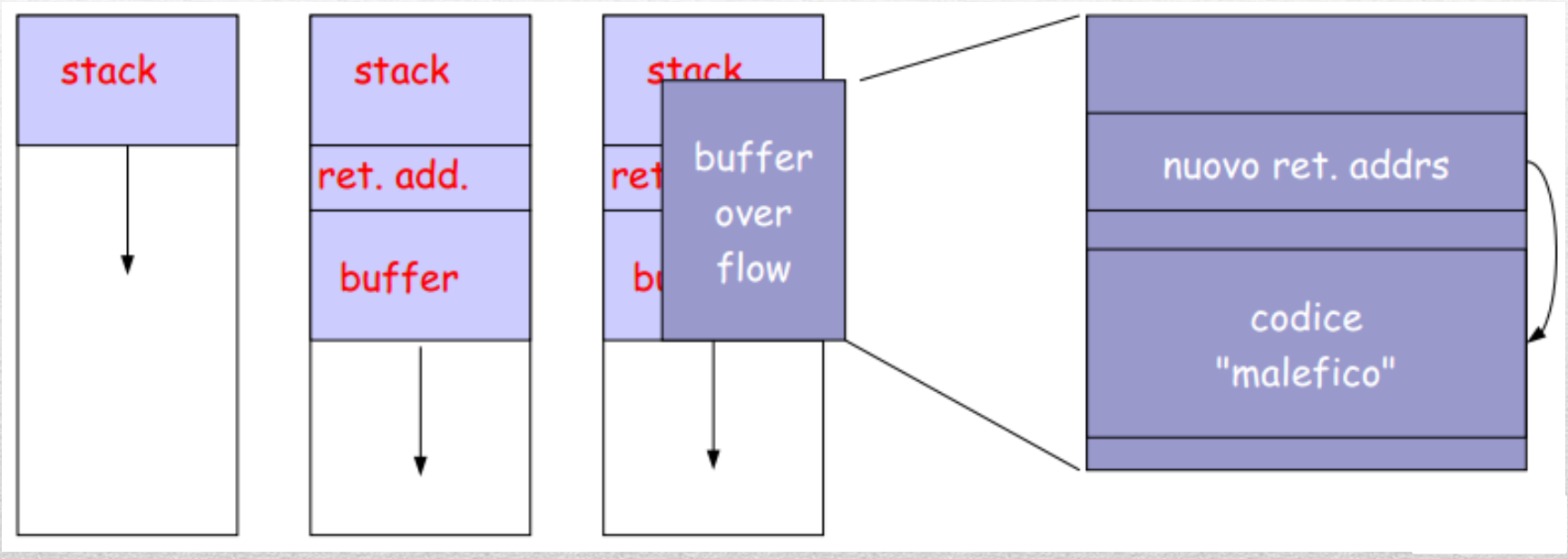
L'uso di funzioni di libreria deboli che non controllano la dimensione dei loto argomenti può provocare un **buffer overflow.**



Morris worm

L'idea generale è quella di riempire il buffer:

- con codice **maligno**
- Un nuovo indirizzo di ritorno che punti a tale codice maligno



Lettura e scrittura di stringhe: esempio

Scrivere un programma che legga una stringa dalla periferica d'ingresso di default e la stampi cento volte

```
#include <stdio.h>
#include <stdlib.h>
#define MAX_CHAR 80

int main()
{
        char str[MAX_CHAR];
        int i;

        printf("Introdurre una stringa:");
        scanf("%s", str);
        for (i=0; i<100; i++)
            printf("%s\n", str);
        return 0;
}
```

Leggere le stringhe

Scrivendo programmi per leggere le stringhe un carattere alla volta, possiamo eliminare la possibile fonte di errori della scanf() e della gets() garantendo un maggior controllo rispetto alle funzioni standard. **Esempio**:

Data una stringa di caratteri, copiarla in un altro array di caratteri (di lunghezza non inferiore).

Ipotesi:

La stringa è "ben formata", ossia correttamente terminata dal carattere '\0'.

Algoritmo:

- scandire la stringa elemento per elemento, fino a trovare il terminatore '\0' (che esiste certamente)
- nel fare ciò, copiare l'elemento nella posizione corrispondente dell'altro array

Esempio

```
int main() {
  char s[] = "Nel mezzo del cammin di";
  char s2[40];
  int i;

  for (i=0; s[i]!='\0'; i++)
          s2[i] = s[i];
  s2[i] = '\0';
  return 0;
}
```

La dimensione deve essere tale da garantire che la stringa non «debordi»

Al termine, occorre garantire che anche la nuova stringa sia “ben formata”, inserendo esplicitamente il terminatore

Attenzione

Avremmo potuto fare diversamente? Perché non copiarla "tutta in un colpo"?

Perché non fare così?

```
int main() {
        char s[] = "Nel mezzo del cammin di";
        char s2[40];
        s2 = s;
        return 0;
}
```

ERRORE DI COMPILAZIONE:
incompatible types in assignment

Perché non dovrebbe funzionare???

Esempio

Problema:

Data una stringa di caratteri, copiarla in un altro array di caratteri, con eventuale troncamento se la stringa è più lunga dell'array dato.

Algoritmo:

- scandire la stringa elemento per elemento, o fino a trovare il terminatore '\0' (che esiste certamente), o fino alla lunghezza dell'array di destinazione
- nel fare ciò, copiare l'elemento nella posizione corrispondente dell'altro array.

Esempio

```
#define N 20

int main() {

  char s[] = "Nel mezzo del cammin di";

  char s2[N];

  int i;

  for (i=0; s[i]!='\0' && i<N-1; i++)

            s2[i]=s[i];

  s2[i] = '\0';

  return 0;
}
```

Si prosegue solo se non si è incontrato il terminatore e inoltre c'è spazio nell'array destinazione

deve rimanere uno spazio per '\0'

Esempio

Problema:

Date due stringhe di caratteri, decidere quale precede l'altra in ordine alfabetico.

Rappresentazione dell'informazione:

poiché vi possono essere tre risultati usiamo un intero (negativo, zero, positivo) per distinguerli.

Algoritmo:

- scandire uno ad uno gli elementi di uguale posizione delle due stringhe, o fino alla fine delle stringhe, o fino a che se ne trovano due diversi
- nel primo caso, le stringhe sono uguali
- nel secondo, sono diverse
    - confrontare i due caratteri così trovati, e determinare qual è il minore
    - la stringa a cui appartiene tale carattere precede l'altra

Esempio

```
main() {

  char s1[] = "Sempre caro mi fu quell'ermo colle";
  char s2[] = " Sempre caro mi fu quell'ermo colle , e questa siepe";
  int i, stato;


  for (i=0; s1[i]!='\0' && s2[i]!='\0' && s1[i]==s2[i] ; i++);
      stato = s1[i]-s2[i];
}
```

negativo	⇔	s1 precede s2
positivo	⇔	s2 precede s1
zero	⇔	s1 è uguale a s2

string.h

Funzioni della libreria <string.h>

Nome	Parametri	Restituisce	Descrizione	Esempi
strlen	char s[N]	int	Lunghezza della stringa	lun = strlen(s) ;
strcpy	char dst[N], char src[M]		Copia il contenuto di src all'interno di dst	strcpy(s1, s2) ; strcpy(s, "") ; strcpy(s1, "ciao") ;
strncpy	char dst[N], char src[M], int nc		Copia il contenuto di src (max nc caratteri) all'interno di dst	strncpy(s1, s2, 20) ; strncpy(s1, s2, MAX) ;
strcat	char dst[N], char src[N]		Accoda il contenuto di src alla fine di dst	strcat(s1, s2) ; strcat(s1, "␣") ;
strncat	char dst[N], char src[M], int nc		Accoda il contenuto di src (max nc caratteri) alla fine di dst	strncat(s1, s2, 50) ;
strcmp	char s1[N], char s2[M]	int	Risultato <0 se s1 precede s2, ==0 se s1 è uguale a s2, >0 se s1 segue s2	if(strcmp(s, r)==0) while(strcmp(r,"*")!=0)
strncmp	char s1[N], char s2[M], int n	int	Come strcmp, ma confronta solo i primi n caratteri	if(strncmp(r, "buon", 4)==0)

string.h

`strchr`	**char** `s[N],`
	char `ch`
		`==NULL`
		o
		`!=NULL`
			Risultato `!=NULL` se il carattere `ch` compare nella stringa, `==NULL` se non compare.	**if**`(strchr(s, '.')!=NULL)`
				if`(strchr(s, ch)==NULL)`

`strstr`	**char** `s[N],`
	char `r[N]`
		`==NULL`
		o
		`!=NULL`
			Risultato `!=NULL` se la sotto-stringa `r` compare nella stringa `s`, `==NULL` se non compare.	**if**`(strstr(s, "xy")!=NULL)`
				if`(strstr(s, s1)==NULL)`

`strspn`	**char** `s[N],` **char** `r[N]`	**int**	Restituisce la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri presenti in `r` (in qualsiasi ordine).	`lun = strspn(s, "␣") ;` `lun = strspn(s, ".,") ;`
`strcspn`	**char** `s[N],` **char** `r[N]`	**int**	Restituisce la lunghezza della parte iniziale di `s` che è composta esclusivamente dei caratteri *non* presenti in `r`.	`lun = strspn(s, "␣") ;` `lun = strspn(s, ".,") ;`

Calcolo dell'età

```
/*  Esempio di conversione da stringa ad intero */

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

int main()
{
        char anno_nascita[5], anno_corrente[5];
        int anni;

        printf("Inserire l'anno di nascita: ");
        scanf("%s", anno_nascita);
        printf("Inserire l'anno corrente: ");
        scanf("%s", anno_corrente);

        /* atoi() converte una stringa in un intero */
        anni = atoi(anno_corrente) - atoi(anno_nascita);
        printf("Età: %d\n", anni);
        return 0;
}
```

Parole nella stringa

```
/*  Conta il numero di parole in una stringa */
#include <stdio.h>
#include <ctype.h>
int main()
{
        char *s;
         s="     Stringa di prova";
        int count=0;
        while (*s !=`\0´)
        {
          while (isspace(*s))      /* salta la spaziatura */
             ++s;
          if (*s !=`\0´)           /* trovata una parola */
           {
             ++count;
             while (!isspace(*s) && *s !=`\0´)
                ++s;               /* salta la parola */
          }
        }
        printf(«Il numero di parole di cui è composta la frase e': %d\n",count);
        return 0;
}
```

Parole palindrome

```
/*  Letta in input una stringa, verifica se è palindroma */
#include <stdio.h>
#include <string.h>

int main()
{
        char parola[32], i=0, n;
        printf("Inserisci una parola (lunga al max 31 caratteri): ");
        scanf("%s", parola);
        n = strlen(parola);
        while ((i <= n/2) && (parola[i] == parola[n-1-i]))
            i++;
        if  (i > n/2)
            printf("La parola %s è palindroma.\n", parola);
        else
            printf("La parola %s non è palindroma.\n", parola);
        return 0;
}
```

Da minuscole a maiuscole

```
/*  Letta in input una stringa alfabetica, la riscrive utilizzando solo lettere maiuscole*/
#include <stdio.h>
#include <string.h>

int main()
{
        char s[100], t[100];
        int i;
        printf("Inserisci una stringa: ");
        scanf("%s", s);
        for (i=0; i<strlen(s); i++)
        {
            if (s[i] >= 97 && s[i] <= 122)
               t[i] = s[i] – 32;
            else
               t[i] = s[i];
        }
        t[i]='\0';
        printf("Stringa maiuscola: %s\n", t);
        retrun 0;
}
```

```
// in alternativa
for (i=0; i<strlen(s); i++ )
  t[i]=toupper(s[i]);

t[i]='\0';
printf("Stringa maiuscola: %s\n", t);
```

Progetto di programmazione – leggi riga

Scrivere un programma C che legga una riga di input saltando tutti gli spazi bianchi prima di iniziare a salvare i caratteri di input.

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

int main()
{
  int ch, n=100, i = 0;
  char str[100];

  while ((ch = getchar()) != '\n')
    if (i == 0 && isspace(ch))
      ;   /* ignore */
    else if (i < n)
      str[i++] = ch;
  str[i] = '\0';
  return i;
}
```

Progetto di programmazione – leggi riga

Scrivere un programma C che legga una riga di input interrompendosi al primo carattere bianco.

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

int main()
{
  int ch, n=100, i = 0;
  char str[100];

   while (!isspace(ch = getchar()))
    if (i < n)
      str[i++] = ch;
  str[i] = '\0';
  return i;
}
```

Progetto di programmazione – leggi riga

Scrivere un programma C che legga una riga di input interrompendosi non appena si incontra un carattere new-line e memorizzandolo nella stringa.

```
#include <stdio.h>

int main()
{
  int ch, n=100, i = 0;
  char str[100];

do {
    ch = getchar();
    if (i < n)
      str[i++] = ch;
  } while (ch != '\n');
  str[i] = '\0';
  return i;
}
```

Progetto di programmazione – leggi riga

Scrivere un programma C che legga una riga di input in modo che i caratteri per i quali non c'è spazio vengano lasciati al loro posto.

```
#include <stdio.h>

int main()
{
  int ch, n=100, i = 0;
  char str[100];

for (i = 0; i < n; i++) {
    ch = getchar();
    if (ch == '\n')
      break;
    str[i] = ch;
  }
  str[i] = '\0';
  return i;
}
```

Progetto di programmazione – leggi/sostituisci

Scrivere un programma C che modifichi una stringa rimpiazzando ogni occorrenza di «new» con «old».

```
#include <stdio.h>

int main()
{
  int, i;
  char str[100];

  for (i = 0; str[i] != '\0'; i++)
    if (str[i] == ‘n' && str[i+1] == ‘e' && str[i+2] ==‘w') {
      str[i] = ‘o’;
      str[i+1] = ‘l’;
      str[i+2] = ‘d';
    }
}
```

Progetto di programmazione – promemoria

Scrivere un programma C che stampi l'elenco dei promemoria giornalieri di un mese. L'utente immette una serie di note, ognuna associata a un giorno del mese; quando l'utente immette uno 0 invece di un giorno valido, il programma stampa la lista di tutti i promemoria immessi, ordinati per giorno. Esempio:

input: Enter day and remind: 24 compleanno Ele
Enter day and remind: 5 18.00 cena con Luca
Enter day and remind: 12 15.00 appuntamento dentista
Enter day and remind: 9 10.00 verifica progetto
Enter day and remind: 0

output: Day reminder:
5 18.00 cena con Luca
9 10.00 verifica progetto
12 15.00 appuntamento dentista
24 compleanno Ele

Progetto di programmazione – promemoria

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

#define MAX_REMIND 50   /* maximum number of reminders */
#define MSG_LEN 60      /* max length of reminder message */

int main(void)
{
  char reminders[MAX_REMIND][MSG_LEN+3];
  char day_str[3], msg_str[MSG_LEN+1];
  int ch, day, i, j, num_remind = 0;

  for (;;) {
    if (num_remind == MAX_REMIND) {
      printf("-- No space left --\n");
      break;
    }

// CONTINUA
```

Progetto di programmazione – promemoria

```
// CONTINUA

   printf("Enter day and reminder: ");
   scanf("%2d", &day);
   if (day == 0)
     break;
   sprintf(day_str, "%2d", day);

   i = 0;
   while ((ch = getchar()) != '\n')
     if (i < MSG_LEN)
       msg_str [i++] = ch;
   msg_str [i] = '\0';

   for (i = 0; i < num_remind; i++)
     if (strcmp(day_str, reminders[i]) < 0)
       break;
   for (j = num_remind; j > i; j--)
     strcpy(reminders[j], reminders[j-1]);

// CONTINUA
```

Progetto di programmazione – promemoria

```
// CONTINUA

    strcpy(reminders[i], day_str);
    strcat(reminders[i], msg_str);

    num_remind++;
  }

  printf("\nDay Reminder\n");
    for (i = 0; i < num_remind; i++)
      printf(" %s\n", reminders[i]);

  return 0;
}
```

Sostituisci carattere

Scrivere un programma in linguaggio C che legga una frase introdotta da tastiera. La frase è terminata dall'introduzione del carattere di invio e contiene complessivamente al più 100 caratteri. Il programma deve svolgere le seguenti operazioni:

- visualizzare la frase inserita
- costruire una nuova frase in cui tutte le occorrenze del carattere `'.'` sono sostituite con il carattere di ritorno di linea `'\n'`. Il programma deve memorizzare la nuova frase in una opportuna variabile
- visualizzare la nuova frase.

Sostituisci carattere

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

int main(void)
{
    const int MAXDIM = 100 ;            /* dimensione max stringa di caratteri */

    char frase[MAXDIM + 1] ;            /* stringa di caratteri inserita */
    char frasemodificata[MAXDIM + 1] ; /* nuova stringa modificata */
    int lung_stringa ;                  /* lunghezza della stringa inserita */
    int i ;                             /* indice dei cicli */

    /* LEGGI LA FRASE INSERITA DA TASTIERA */
    printf ("Inserisci una frase di al massimo %d caratteri: ", MAXDIM) ;
    gets(frase) ;

    /* CALCOLA LA LUNGHEZZA DELLA FRASE */
    lung_stringa = strlen(frase) ;

    /* STAMPA LA FRASE INSERITA */
    printf("La frase inserita e': ") ;
    puts(frase) ;
    printf("La frase contiene %d caratteri (inclusi gli spazi)\n", lung_stringa) ;
```

Sostituisci carattere

```
30
        /* ANALIZZA LA FRASE INSERITA CARATTERE PER CARATTERE. RICOPIA LA FRASE
        NELLA STRINGA "frase modificata". SE LA STRINGA INSERITA CONTIENE IL
        CARATTERE ".", SOSTITUISCILO CON IL CARATTERE DI RITORNO DI LINEA "\n" */
        for ( i=0; i<lung_stringa; i=i+1 )
35      {
            if ( frase[i] == '.' )
                frasemodificata[i] = '\n' ;
            else
                frasemodificata[i] = frase[i] ;
40      }
        frasemodificata[lung_stringa] = '\0' ;

        /* STAMPA LA FRASE MODIFICATA */
        printf("La frase modificata e': \n") ;
45      puts(frasemodificata) ;
        exit(0) ;
    }
```

Parola palindroma

Scrivere un programma in linguaggio C che riceve in ingresso una parola inserita da tastiera. Si consideri che la parola può contenere sia caratteri maiuscoli che caratteri minuscoli, e complessivamente al massimo 30 caratteri. Il programma deve svolgere le seguenti operazioni:

- visualizzare la parola inserita
- aggiornare la parola in modo che tutti i caratteri siano minuscoli. Il programma deve visualizzare la parola ottenuta
- verificare se la parola è palindroma. Una parola è palindroma se può essere letta indifferentemente da sinistra verso destra e da destra verso sinistra. Ad esempio, le seguenti parole sono palindrome: `otto`, `madam`.

Parola palindroma

```
5
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>
10
int main(void)
{
    const int MAXDIM = 30 ;      /* dimensione massima stringa di caratteri */

15  char parola[MAXDIM+1] ;      /* stringa di caratteri inserita */
    int numcaratteri ;           /* numero di caratteri della stringa inserita */
    int palindroma ;             /* flag per la ricerca */
    int i, j ;                   /* indici dei cicli */
```

Parola palindroma

```
20
        /* LEGGI LA STRINGA DI CARATTERI INSERITA DA TASTIERA */
        printf("Inserisci una parola di al massimo %d caratteri: ", MAXDIM) ;
        scanf("%s", parola) ;

25      /* VISUALIZZA LA STRINGA DI CARATTERI INSERITA */
        printf("La parola inserita e': %s \n", parola) ;

        /* LEGGI IL NUMERO DI CARATTERI DELLA STRINGA */
        numcaratteri = strlen(parola) ;
30      printf("La parola contiene %d caratteri\n", numcaratteri) ;

        /* CONVERTI TUTTI I CARATTERI DELLA STRINGA IN CARATTERI MINUSCOLI */
        for ( i=0; i < numcaratteri ; i++ )
            parola[i] = tolower(parola[i]) ;
35
        /* VISUALIZZA LA STRINGA DI CARATTERI DOPO LA CONVERSIONE */
        printf("La parola inserita scritta solo con caratteri in minuscolo e': %s\n",
               parola) ;

40      /* VERIFICA SE LA STRINGA "parola" E' PALINDROMA */

        /* INIZIALIZZA IL FLAG "palindroma". IL FLAG ASSUME I VALORI
        -- "palindroma" E' UGUALE A 1 SE "parola"  E' PALINDROMA
        -- "palindroma" E' UGUALE A 0 SE "parola"  NON E' PALINDROMA
45      */
        palindroma = 1 ;
```

Parola palindroma

```
        /* IL CICLO FOR SCANDISCE LA STRINGA DI CARATTERI "parola" E VERIFICA
        SE E' PALINDROMA L'INDICE "i" SCORRE LA PRIMA META' DI "parola". L'INDICE
50      "j" SCORRE LA SECONDA META' DI "parola" PARTENDO DALL'ULTIMO CARATTERE.
        LA RICERCA TERMINA QUANDO SI TROVA SI VERIFICA CHE LA STRINGA "parola"
        NON E' PALINDROMA O QUANDO SONO STATI CONSIDERATI TUTTI I CARATTERI
        DI "parola" */

55      for ( i=0, j=numcaratteri - 1 ;
                i< numcaratteri/2 && palindroma==1;
                i++, j-- )
        {
            if ( parola[i] != parola[j] )
60              palindroma = 0 ;
        }

        /* STAMPA DEL RISULTATO */
        if ( palindroma == 1 )
65          printf("La␣parola␣e'␣palindroma\n") ;
        else
            printf("La␣parola␣non␣e'␣palindroma\n") ;

        exit(0) ;
70  }
```

Parola palindroma

```
        /* IL CICLO FOR SCANDISCE LA STRINGA DI CARATTERI "parola" E VERIFICA
        SE E' PALINDROMA L'INDICE "i" SCORRE LA PRIMA META' DI "parola". L'INDICE
50      "j" SCORRE LA SECONDA META' DI "parola" PARTENDO DALL'ULTIMO CARATTERE.
        LA RICERCA TERMINA QUANDO SI TROVA SI VERIFICA CHE LA STRINGA "parola"
        NON E' PALINDROMA O QUANDO SONO STATI CONSIDERATI TUTTI I CARATTERI
        DI "parola" */

55      for ( i=0, j=numcaratteri - 1 ;
                i< numcaratteri/2 && palindroma==1;
                i++, j-- )
        {
            if ( parola[i] != parola[j] )
60              palindroma = 0 ;
        }

        /* STAMPA DEL RISULTATO */
        if ( palindroma == 1 )
65          printf("La␣parola␣e'␣palindroma\n") ;
        else
            printf("La␣parola␣non␣e'␣palindroma\n") ;

        exit(0) ;
70  }
```

Rubrica telefonica

Si realizzi un programma in linguaggio C in grado di gestire una rubrica di nomi e numeri telefonici. La rubrica deve contenere fino a 100 voci diverse. Ciascuna voce è composta da un nome (max 40 caratteri) e da un numero di telefono (max 20 caratteri).

Il programma deve fornire all'utente un menù di scelta, con le seguenti voci:

```
1) Aggiungi nuova voce in rubrica
2) Ricerca esatta per nome
3) Ricerca approssimata per nome
4) Stampa completa rubrica
0) Esci dal programma
```

Una volta che l'utente ha scelto l'operazione desiderata (1-4), il programma acquisirà i dati necessari dall'utente ed eseguirà il comando. Nota: nella rubrica non possono esistere due voci con lo stesso nome.

Rubri

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

int main(void)
{
    const int MAX = 100 ; /* numero max di voci */
    const int LUNN = 40 ; /* lunghezza del nome */
    const int LUNT = 20 ; /* lunghezza n. telefono */

    char nome[MAX][LUNN+1] ;
    char tel[MAX][LUNT+1] ;

    int N ; /* numero di voci memorizzate */

    int comando ; /* comando dell'utente 0-4 */

    char riga[200] ;
    char sn[LUNN+1] ;
    char st[LUNT+1] ;
    int i, duplicato, trovato, pos ;

    /* INIZIALIZZAZIONI */

    N = 0 ;

    do
    {
        /* STAMPA DEL MENU */
        puts("1) Aggiungi nuova voce in rubrica") ;
        puts("2) Ricerca esatta per nome") ;
        puts("3) Ricerca approssimata per nome") ;
        puts("4) Stampa completa rubrica") ;
        puts("0) Esci dal programma") ;

        /* LETTURA DEL COMANDO */
        printf("Inserisci il comando: ") ;
        gets(riga) ;
        comando = atoi( riga ) ;
```

R

```
        /* ESECUZIONE DEL COMANDO */
        switch ( comando )
        {
        case 1:
            /* Acquisisci i dati */
            printf("Inserisci␣il␣nome␣da␣aggiungere:␣") ;
            gets(sn) ;
            printf("Inserisci␣il␣numero␣di␣telefono␣corrispondente:␣") ;
            gets(st) ;

            /* Verifica se i dati sono validi */
            if ( N == MAX )
            {
                puts("ERRORE:␣rubrica␣piena") ;
                break ;
            }

            duplicato = 0 ;
            for ( i = 0 ; i < N ; i++ )
                if ( strcmp(sn, nome[i]) == 0 )
                    duplicato = 1 ;

            if ( duplicato == 1 )
            {
                puts("ERRORE:␣nome␣duplicato") ;
                break ;
            }

            /* Aggiungi il nome in rubrica */
            strcpy( nome[N], sn ) ;
            strcpy( tel[N], st ) ;
            N++ ;

            break ;
```

Ru

```
            case 2: /* ricerca esatta */
                printf("Inserisci il nome da ricercare: ") ;
                gets(sn) ;

85              trovato = 0 ;
                for ( i = 0 ; i < N && trovato == 0 ; i++ )
                {
                    if ( strcmp( sn, nome[i] ) == 0 )
                    {
90                      trovato = 1 ;
                        pos = i ;
                    }
                }

95              if ( trovato == 1 )
                {
                    printf("Il telefono di %s e': %s\n",
                           sn, tel[pos] ) ;
                }
100             else
                {
                    printf("Nessun %s e' presente in rubrica\n", sn) ;
                }

105             break ;

            case 3: /* ricerca approssimata */
                printf("Inserisci una parte del nome da ricercare: ") ;
                gets(sn) ;
110
                trovato = 0 ;
                for ( i = 0 ; i < N ; i++ )
                {
                    if ( strstr( nome[i], sn ) != NULL )
115                 {
```

Rubrica telefonica

```
                printf("%s: %s\n", nome[i], tel[i]) ;
                trovato = 1 ;
            }
        }

        if (trovato==0)
            printf("Non trovato...\n") ;
        break ;

    case 4:
        printf("CONTENUTO DELLA RUBRICA (%d VOCI)\n", N) ;

        for ( i = 0 ; i < N ; i++ )
            printf("%s: %s\n", nome[i], tel[i] ) ;
        break ;

    case 0:
        puts("Arrivederci") ;
        break ;

    default:
        printf("ERRORE NEL PROGRAMMA (comando=%d)\n", comando) ;
    }

}
while ( comando != 0 ) ;

exit(0) ;
}
```

Vettori di stringhe

Come memorizzare un vettore di stringhe? Potremmo creare un vettore bidimensionale di caratteri e poi memorizzare le stringhe all'interno del vettore, una per riga. Esempio:

```
char planets[][8]= {"Mercury", "Venus", "Earth",
                    "Mars", "Jupiter", "Saturn",
                    "Uranus", "Neptune", "Pluto" }
```

Siccome non tutte le stringhe sono lunghe a sufficienza per occupare un'intera riga del vettore, il C le riempe con caratteri null, sprecando così spazio (la stessa cosa avviene nel programma per la stampa dei promemoria).

Quello di cui abbiamo bisogno è un vettore *frastagliato*, in cui le righe possano avere lunghezza diversa fra di loro. Possiamo crearli utilizzando vettori i cui **elementi siano puntatori a stringhe.**

Vettori di stringhe

Vettore di puntatori a stringa. Esempio:

```
char *planets[] = {"Mercury", "Venus", "Earth",
                   "Mars", "Jupiter", "Saturn",
                   "Uranus", "Neptune", "Pluto" }
```

L'effetto sul modo in cui viene memorizzato il vettore è sostanziale. Ogni elemento di *planets* è un puntatore a una stringa terminante con null. Nelle stringhe non ci sono più sprechi di memoria, sebbene ora serva spazio per allocare anche i puntatori.

Per accedere a uno dei nomi dei pianeti, basterà indicizzare il vettore *planets*. Grazie alla relazione fra vettori e puntatori, si accederà ai caratteri nello stesso modo nel quale si accede a un elemento di un vettore bidimensionale. Esempio:

```
for (i=0;i<9;i++)
  if (planets[i][0]==‘M’)
    printf((“%s begins with M\n”, planets[i]);
```