Caratteri e stringhe

Caratteri

- I caratteri sono i blocchi costituenti dei programmi
- Ogni programma è composto da una sequenza di caratteri che è interpretata dal computer come una serie di istruzioni
- Un programma può contenere **costanti carattere:**
  - Valore `int` rappresentato come un carattere tra virgolette singole
- Il valore di una costante carattere è il valore intero del carattere nell'insieme di caratteri della macchina
- `'z'` rappresenta il valore intero della lettera z
- `'\n'` rappresenta il valore intero del carattere newline

Stringhe

- Una **stringa** è una serie di caratteri trattati come unità singola
- Una stringa può contenere
  - Lettere
  - Cifre
  - Caratteri speciali come +, -, *, /, e $
- Le **stringhe letterali** o **stringhe costanti** sono scritte tra virgolette doppie

`"John Q. Doe"` (un nome)
`"99999 Main Street"` (l'indirizzo di una via)
`"Waltham, Massachusetts"` (una città e uno stato)
`"(201) 555-1212"` (un numero di telefono)

Stringhe

- Tutte le stringhe devono finire con il **carattere nullo o di terminazione `'\0'`**
- Stampare una stringa che non termina con il carattere nullo è un errore logico:
  - Su alcuni sistemi la stampa continuerà dopo la fine della stringa finché non viene letto un carattere nullo
  - Su alcuni sistemi il programma si arresterà segnalando un "errore di segmentazione" o "violazione di accesso"

Stringhe e puntatori

- Si accede a una stringa mediante un puntatore al su primo carattere
- Il valore di una stringa è l'indirizzo del primo carattere
- Una stringa è un puntatore al primo carattere della stringa
- Le stringhe sono semplicemente **array di caratteri**

Inizializzazione di array di tipo char

- È possibile inizializzare con una stringa:
  - Un array di caratteri
  - Una variabile di tipo `char *`

```
char color[] = "blue";
```

- Crea un array color di 5 elementi che contengono i caratteri **modificabili** 'b', 'l', 'u', 'e' e '\0'

```
const char *colorPtr = "blue";
```

- Crea la variabile puntatore colorPtr che punta alla lettera ‘b’ della stringa **non modificabile** «blue»

Inizializzazione di array di tipo `char`

- Le definizioni precedenti determinano autonomaticamente la dimensione dell'array in base al numero di inizializzatori (5)
- L'array deve abbastanza grande da contenere la stringa e il carattere nullo di terminazione
- Non allocare spazio sufficiente per memorizzare il carattere nullo è un errore
- Se la stringa è più lunga dell'array in cui viene memorizzata, i caratteri oltre la fine dell'array potrebbero sovrascrivere dati in memoria
- Nel C standard una stringa letterale è **immutabile**
  - Per modificare una stringa letterale, deve essere memorizzata in un array di caratteri

Leggere una stringa

- La funzione scanf può leggere una stringa e memorizzarla in un array di caratteri
- Assumiamo di avere un array word di caratteri contenente 20 elementi
- Leggere una stringa e memorizzarla nell'array `word`:

```
scanf("%19s", word);
```

- Il nome dell'array è un puntatore al primo elemento
- Non è necessaria la `&` usata normalmente con gli argomenti di `scanf`

Leggere una stringa

- La funzione `scanf` legge i caratteri finché non legge:
  - Uno spazio
  - Una tabulazione
  - Un indicatore di nuova riga
  - L'indicatore di end-of-file
- 19 nell'istruzione precedente garantisce di leggere massimo 19 caratteri
  - Conserva l'ultimo carattere per il carattere nullo di terminazione della stringa
  - Evita di scrivere in memoria caratteri oltre l'ultimo elemento dell'array

Libreria ctype.h

- La libreria `<ctype.h>` include diverse funzioni che eseguono test e manipolazioni di dati di tipo carattere
- Ogni funzione riceve come argomento un `unsigned char` (rappresentato come un `int`) o EOF
- I caratteri sono spesso manipolati come interi
  - Un carattere in C è un intero di un **Byte**
  - EOF normalmente ha il valore -1

Libreria ctype.h

Prototipo	Descrizione della funzione
`int isblank(int c);`	Restituisce un valore vero se `c` è un carattere vuoto che separa le parole in una riga di testo e 0 (falso) altrimenti
`int isdigit(int c);`	Restituisce un valore vero se `c` è una cifra e 0 (falso) altrimenti.
`int isalpha(int c);`	Restituisce un valore vero se `c` è una lettera e 0 (falso) altrimenti.
`int isalnum(int c);`	Restituisce un valore vero se `c` è una cifra o una lettera e 0 (falso) altrimenti.
`int isxdigit(int c);`	Restituisce un valore vero se `c` è una cifra esadecimale e 0 (falso) altrimenti. (Si veda l'Appendice E, disponibile sulla piattaforma MyLab, per una descrizione dettagliata di numeri binari, numeri ottali, numeri decimali e numeri esadecimali.)
`int islower(int c);`	Restituisce un valore vero se `c` è una lettera minuscola e 0 (falso) altrimenti.
`int isupper(int c);`	Restituisce un valore vero se `c` è una lettera maiuscola e 0 (falso) altrimenti.

Fonte: Deitel & Deitel

Libreria ctype.h

`int tolower(int c);`	Se `c` è una lettera maiuscola, `tolower` restituisce `c` come lettera minuscola; altrimenti, restituisce l'argomento inalterato.
`int toupper(int c);`	Se `c` è una lettera minuscola, `toupper` restituisce `c` come lettera maiuscola; altrimenti, restituisce l'argomento inalterato.
`int isspace(int c);`	Restituisce un valore vero se `c` è un carattere di spaziatura – newline (`'\n'`), spazio (`' '`), avanzamento pagina (`'\f'`), ritorno a capo (`'\r'`), tab orizzontale (`'\t'`) o tab verticale (`'\v'`) – altrimenti, restituisce 0 (falso).
`int iscntrl(int c);`	Restituisce un valore vero se `c` è un carattere di controllo – tab orizzontale (`'\t'`), tab verticale (`'\v'`), avanzamento pagina (`'\f'`), avviso (`'\a'`), backspace (`'\b'`), ritorno a capo (`'\r'`), newline (`'\n'`) e altri – altrimenti, restituisce 0 (falso).
`int ispunct(int c);`	Restituisce un valore vero se `c` è un carattere stampabile diverso da uno spazio, da una cifra o da una lettera – come `$`, `#`, `(`, `)`, `[`, `]`, `{`, `}`, `;`, `:` o `%` – altrimenti, restituisce 0 (falso).
`int isprint(int c);`	Restituisce un valore vero se `c` è un carattere stampabile (cioè un carattere visibile sullo schermo) incluso uno spazio; altrimenti, restituisce 0 (falso).
`int isgraph(int c);`	Restituisce un valore vero se `c` è un carattere stampabile diverso da uno spazio; altrimenti, restituisce 0 (falso).

Fonte: Deitel & Deitel

Esempi di applicazioni

```c
// fig08_01.c
// Uso delle funzioni isdigit, isalpha, isalnum e isxdigit
#include <ctype.h>
#include <stdio.h>

int main(void) {
   printf("%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
      isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
      isdigit( '#' ) ? "# is a " : "# is not a ", "digit");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalpha:",
      isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
      isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
      isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
      isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter");

   printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isalnum:",
      isalnum( 'A' ) ? "A is a " : "A is not a ", "digit or a letter",
      isalnum( '8' ) ? "8 is a " : "8 is not a ", "digit or a letter",
      isalnum( '#' ) ? "# is a " : "# is not a ", "digit or a letter");

   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n", "According to isxdigit:",
      isxdigit( 'F' ) ? "F is a " : "F is not a ", "hexadecimal digit",
      isxdigit( 'J' ) ? "J is a " : "J is not a ", "hexadecimal digit",
      isxdigit( '7' ) ? "7 is a " : "7 is not a ", "hexadecimal digit",
      isxdigit( '$' ) ? "$ is a " : "$ is not a ", "hexadecimal digit",
      isxdigit( 'f' ) ? "f is a " : "f is not a ", "hexadecimal digit");
}
```

Fonte: Deitel & Deitel

Esempi di applicazioni

Output:

```
According to isdigit:
8 is a digit
# is not a digit
According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter
According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter
According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```

Fonte: Deitel & Deitel

Esempi di applicazioni

```c
// fig08_02.c
// Uso delle funzioni islower, isupper, tolower e toupper
#include <ctype.h>
#include <stdio.h>

int main(void) {
   printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to islower:",
      islower( 'p' ) ? "p is a " : "p is not a ", "lowercase letter",
      islower( 'P' ) ? "P is a " : "P is not a ", "lowercase letter",
      islower( '5' ) ? "5 is a " : "5 is not a ", "lowercase letter",
       islower( '!' ) ? "! is a " : "! is not a ", "lowercase letter");

    printf("%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n", "According to isupper:",
      isupper( 'D' ) ? "D is an " : "D is not an ", "uppercase letter",
      isupper( 'd' ) ? "d is an " : "d is not an ", "uppercase letter",
      isupper( '8' ) ? "8 is an " : "8 is not an ", "uppercase letter",
      isupper( '$' ) ? "$ is an " : "$ is not an ", "uppercase letter");

    printf("%s%c\n%s%c\n%s%c\n%s%c\n",
      "u converted to uppercase is ", toupper( 'u' ),
      "7 converted to uppercase is ", toupper( '7' ),
      "$ converted to uppercase is ", toupper( '$' ),
      "L converted to lowercase is ", tolower( 'L' ));
 }
```

Fonte: Deitel & Deitel

Esempi di applicazioni

Output:

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter
According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter
u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

Fonte: Deitel & Deitel

Esempi di applicazioni

```c
// fig08_03.c
// Uso delle funzioni isspace, iscntrl, ispunct, isprint e isgraph
#include <ctype.h>
#include <stdio.h>

int main(void) {
   printf("%s\n%s%s%s\n%s%s%s\n%s%s\n\n", "According to isspace:",
       "Newline", isspace ( '\n' ) ? " is a " : " is not a ",
       "whitespace character",
       "Horizontal tab", isspace ( '\t' ) ? " is a " : " is not a ",
        "whitespace character",
        isspace ( '%' ) ? "% is a " : "% is not a ", "whitespace character");

    printf("%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
        "Newline", iscntrl ( '\n' ) ? " is a " : " is not a ",
        "control character",
        iscntrl( '$' ) ? "$ is a " : "$ is not a ", "control character");

    printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to ispunct:",
        ispunct( ';' ) ? "; is a " : "; is not a ", "punctuation character",
        ispunct( 'Y' ) ? "Y is a " : "Y is not a ", "punctuation character",
        ispunct( '#' ) ? "# is a " : "# is not a ", "punctuation character");

    printf("%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
        isprint( '$' ) ? "$ is a " : "$ is not a ", "printing character",
        "Alert", isprint( '\a' ) ? " is a " : " is not a ",
        "printing character");

    printf("%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
        isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
        "printing character other than a space",
        "Space", isgraph( ' ' ) ? " is a " : " is not a ",
        "printing character other than a space");
 }
```

Fonte: Deitel & Deitel

Esempi di applicazioni

Output:

```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character
According to iscntrl:
Newline is a control character
$ is not a control character
According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character
According to isprint:
$ is a printing character
Alert is not a printing character
According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```

Fonte: Deitel & Deitel

Intervallo

Fonte: chatGPT

Conversione di stringhe

- La libreria `<stdlib.h>` include funzioni per la conversione di stringhe

Prototipo della funzione	Descrizione della funzione
`double strtod(const char *nPtr, char **endPtr);`	
	Converte la stringa `nPtr` in un `double`.
`long strtol(const char *nPtr, char **endPtr, int base);`	
	Converte la stringa `nPtr` in un `long`.
`unsigned long strtoul(const char *nPtr, char **endPtr, int base);`	
	Converte la stringa `nPtr` in un `unsigned long`.

Fonte: Deitel & Deitel

Funzione `strtod`

- `strtod` converte una sequenza di caratteri che rappresentano un valore in virgola mobile in double
- Restituisce 0 se non è in grado di convertire una porzione del suo primo argomento in double
- Due argomenti:
  - Una stringa (`char *`)
  - Un puntatore a stringa (`char **`)
- L'argomento stringa contiene la sequenza di caratteri da convertire in un double
- I caratteri di spaziatura all'inizio della stringa vengono ignorati

Funzione `strtod`

- La funzione usa l'argomento `char**` per far sì che un oggetto `char` nella funzione chiamante punti al primo carattere dopo la porzione convertita della stringa
- Se nessuna parte può essere convertita, la funzione indirizza il puntatore all'inizio della stringa

Funzione strtod

```c
// fig08_04.c
// Uso della funzione strtod
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   const char  *string =  "51.2% are admitted" ;
   char *stringPtr = NULL;

   double  d = strtod(string, &stringPtr);

    printf("The string \"%s\" is converted to the\n", string);
    printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
 }
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

Fonte: Deitel & Deitel

Funzione `strtol`

- `strtol` converte in un valore `long int` una sequenza di caratteri che rappresentano un intero
- La funzione restituisce 0 se non è in grado di convertire una porzione del suo primo argomento in un `long int`
- I tre argomenti della funzione sono:
  - Una stringa (char *)
  - Un puntatore a una stringa
  - Un intero
- Funzionamento simile a `strtod`
  - Il terzo elemento specifica la base del valore da convertire

Funzione strtol

```c
// fig08_05.c
// Uso della funzione strtol
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   const char  *string =  "-1234567abc" ;
   char *remainderPtr = NULL;

   long  x = strtol(string, &remainderPtr,  0 );

    printf("%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
       "The original string is ", string,
       "The converted value is ", x,
       "The remainder of the original string is ", remainderPtr,
       "The converted value plus 567 is ", x + 567);
 }
```

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

Fonte: Deitel & Deitel

Funzione `strtol`

- L'uso di `NULL` per il secondo argomento fa sì che il resto della stringa sia ignorato
- Il terzo argomento 0, indica che il valore da convertire può essere nel formato
  - Ottale (base 8)
  - Decimale (base 10)
  - O esadecimale (base 16)
- La base viene determinata dalla struttura della stringa

Funzione `strtoul`

- `strtoul` converte in un valore `unsigned long int` una sequenza di caratteri
- La funzione opera come `strtol`

Funzione strtoul

```c
// fig08_06.c
// Uso della funzione strtoul
#include <stdio.h>
#include <stdlib.h>

int main(void) {
   const char  *string =  "1234567abc" ;
   char *remainderPtr = NULL;

   unsigned long int  x = strtoul(string, &remainderPtr,  0 );

   printf("%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
       "The original string is ", string,
       "The converted value is ", x,
       "The remainder of the original string is ", remainderPtr,
       "The converted value minus 567 is ", x - 567);
 }
```

```
The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000
```

Fonte: Deitel & Deitel

Funzioni input/output

- Funzioni della libreria standard di input/output (`<stdio.h>`) per la manipolazione di caratteri e stringhe

Prototipo della funzione	Descrizione della funzione
`int getchar(void);`	Restituisce il carattere successivo dello standard input come un intero.
`char *fgets(char *s, int n, FILE *stream);`	Legge caratteri dallo stream specificato e li memorizza nell'array `s` finché non si incontra un carattere newline o end-of-file, oppure finché non vengono letti `n - 1` byte. In questo capitolo usiamo lo stream `stdin` – lo stream standard input – per leggere caratteri dalla tastiera. Un carattere nullo di terminazione è aggiunto in coda all'array. Restituisce la stringa che è stata letta in `s`. Se si incontra un newline, esso è incluso nella stringa memorizzata.
`int putchar(int c);`	Stampa il carattere memorizzato in `c` e lo restituisce come intero.
`int puts(const char *s);`	Stampa la stringa `s` seguita da un carattere newline. Restituisce un intero diverso da zero se ha successo o `EOF` se si verifica un errore.
`int sprintf(char *s, const char *format, ...);`	Equivalente a `printf`, ma l'output è memorizzato nell'array `s` anziché stampato sullo schermo. Restituisce il numero di caratteri scritti in `s` se ha successo o `EOF` se si verifica un errore.
`int sscanf(char *s, const char *format, ...);`	Equivalente a `scanf`, ma l'input è letto dall'array `s` invece che dalla tastiera. Restituisce il numero di elementi letti con successo dalla funzione o `EOF` se si verifica un errore.

Fonte: Deitel & Deitel

Funzioni `fgets` e `putchar`

```c
// fig08_07.c
// Uso delle funzioni fgets e putchar
#include <stdio.h>
#define SIZE 80

void  reverse( const char  *  const  sPtr);

int main(void) {
   char sentence[SIZE] = "";

   puts("Enter a line of text:");
    fgets(sentence,  SIZE , stdin); // leggi una riga di testo

    printf("\n%s", "The line printed backward is:");
    reverse(sentence);
    puts("");
 }

 // stampa ricorsivamente i caratteri della stringa in ordine inverso
 void reverse(const char * const sPtr) {
    // se si raggiunge la fine della stringa
    if ('\0' == sPtr[0]) { // caso di base
       return;
    }
    else { // se non e' la fine della stringa
        reverse(&sPtr[ 1 ]); // passo di ricorsione
        putchar(sPtr[ 0 ]); // usa putchar per stampare il carattere
    }
 }
```

```
Enter a line of text:
Characters and Strings
The line printed backward is:
sgnirtS dna sretcarahC
```

Fonte: Deitel & Deitel

Funzione `reverse`

- Il programma chiama la funzione ricorsiva reverse per stampare la riga di testo all'indietro
- Se il primo carattere è `'\0'`, reverse termina
- Altrimenti chiama ricorsivamente se stessa con l'indirizzo del sottoarray che inizia all'elemento `sPtr[1]`
- Riga 27 invia in uscita il carattere all'elemento `sPtr[0]` quando la chiamata ricorsiva è completata
- L'ordine delle istruzioni nelle righe 26 e 27 fa sì che reverse proceda verso il carattere nullo di terminazione prima di stampare un carattere

Funzione `getchar`

- Legge un carattere dallo standard input
- Restituisce il carattere come intero

```c
// fig08_08.c
// Uso della funzione getchar
#include <stdio.h>
#define SIZE 80

int main(void) {
   int c = 0; // variabile che contiene il carattere inserito dall'utente
   char sentence[SIZE] = "";
   int i = 0;

   puts( "Enter a line of text:" );

   // usa getchar per leggere ogni carattere
   while ((i < SIZE - 1 ) && (c = getchar()) != '\n' ) {
      sentence[i++] = c;
   }

   sentence[i] = '\0' ; // termina la stringa

   puts("\nThe line entered was:");
   puts(sentence); // stampa la stringa
}
```

```
Enter a line of text:
This is a test.
The line entered was:
This is a test.
```

Fonte: Deitel & Deitel

Funzione sprintf

```c
// fig08_09.c
// Uso della funzione sprintf
#include <stdio.h>
#define SIZE 80

int main(void) {
   int x = 0;
   double y = 0.0;

   puts("Enter an integer and a double:");
   scanf("%d%lf", &x, &y);

    char s[SIZE] = {'\0'}; // crea un array di char
    sprintf(s,  "integer:%6d\ndouble:%7.2f" , x, y);

    printf("The formatted output stored in array s is:\n%s\n", s);
 }
```

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:  87.38
```

Fonte: Deitel & Deitel

Funzione sscanf

```c
// fig08_10.c
// Uso della funzione sscanf
#include <stdio.h>

int main(void) {
   char s[] = "31298 87.375";
   int x = 0;
   double y = 0;

   sscanf(s,  "%d%lf" , &x, &y);
   puts("The values stored in character array s are:");
    printf("integer:%6d\ndouble:%8.3f\n", x, y);
 }
```

```
The values stored in character array s are:
integer: 31298
double:  87.375
```

Fonte: Deitel & Deitel

Manipolazione di stringhe

- La libreria per il trattamento delle stringhe (`<string.h>`) fornisce funzioni per:
  - Manipolare stringhe (copiare e concatenare)
  - Confrontare stringhe
  - Ricercare caratteri e stringhe all'interno di altre stringhe
  - Suddividere le stringhe
  - Determinare la lunghezza delle stringhe

Manipolazione di stringhe

Prototipo della funzione	Descrizione della funzione
`char *strcpy(char *s1, const char *s2)`	
	Copia la stringa `s2` nell'array `s1` e restituisce `s1`.
`char *strncpy(char *s1, const char *s2, size_t n)`	
	Copia al massimo `n` caratteri della stringa `s2` nell'array `s1` e restituisce `s1`.
`char *strcat(char *s1, const char *s2)`	
	Aggiunge la stringa `s2` in coda all'array `s1` e restituisce `s1`. Il primo carattere della stringa `s2` sovrascrive il carattere nullo di terminazione di `s1`.
`char *strncat(char *s1, const char *s2, size_t n)`	
	Aggiunge al massimo `n` caratteri della stringa `s2` in coda all'array `s1` e restituisce `s1`. Il primo carattere della stringa `s2` sovrascrive il carattere nullo di terminazione di `s1`.

Fonte: Deitel & Deitel

strcpy e strncpy

```c
// fig08_11.c
// Uso delle funzioni strcpy e strncpy
#include <stdio.h>
#include <string.h>
#define SIZE1 25
#define SIZE2 15

int main(void) {
   char x[] = "Happy Birthday to You"; // inizializza l'array di caratteri x
   char y[SIZE1] = ""; // crea un array di caratteri y
    char z[SIZE2] = ""; // crea un array di caratteri z

    // copia il contenuto di x in y
    printf("%s%s\n%s%s\n",
       "The string in array x is: ", x,
       "The string in array y is: ", strcpy(y, x));

    strncpy(z, x, SIZE2 - 1 ); // copia i primi 14 caratteri di x in z
    z[ SIZE2 - 1 ] = '\0' ; // termina la stringa in z, perche' '\0' non copiato
    printf("The string in array z is: %s\n", z);
 }
```

```
The string in array x is: Happy Birthday to You
The string in array y is: Happy Birthday to You
The string in array z is: Happy Birthday
```

Fonte: Deitel & Deitel

strcat e strncat

```c
// fig08_12.c
// Uso delle funzioni strcat e strncat
#include <stdio.h>
#include <string.h>

int main(void) {
   char s1[20] = "Happy "; // inizializza l'array di caratteri s1
   char s2[] = "New Year "; // inizializza l'array di caratteri s2
   char s3[40] = ""; // inizializza l'array di caratteri s3 come vuoto

    printf("s1 = %s\ns2 = %s\n", s1, s2);

    // aggiungi s2 in coda a s1
    printf("strcat(s1, s2) = %s\n", strcat(s1, s2));

    // aggiungi i primi 6 caratteri di s1 in coda a s3
    printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));

    // aggiungi s1 in coda a s3
    printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
 }
```

```
s1 = Happy
s2 = New Year
strcat(s1, s2) = Happy New Year
strncat(s3, s1, 6) = Happy
strcat(s3, s1) = Happy Happy New Year
```

Fonte: Deitel & Deitel

strcmp e strncmp

```c
// fig08_13.c
// Uso delle funzioni strcmp e strncmp
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *s1 = "Happy New Year"; // inizializza un puntatore a char
   const char *s2 = "Happy New Year"; // inizializza un puntatore a char
   const char *s3 = "Happy Holidays"; // inizializza un puntatore a char

    printf("s1 = %s\ns2 = %s\ns3 = %s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
       s1, s2, s3,
       "strcmp(s1, s2) = ", strcmp(s1, s2),
       "strcmp(s1, s3) = ", strcmp(s1, s3),
       "strcmp(s3, s1) = ", strcmp(s3, s1));

    printf("%s%2d\n%s%2d\n%s%2d\n",
       "strncmp(s1, s3, 6) = ", strncmp(s1, s3,  6 ),
       "strncmp(s1, s3, 7) = ", strncmp(s1, s3,  7 ),
       "strncmp(s3, s1, 7) = ", strncmp(s3, s1,  7 ));
}
```

```
s1 = Happy New Year
s2 = Happy New Year
s3 = Happy Holidays
strcmp(s1, s2) =  0
strcmp(s1, s3) =  1
strcmp(s3, s1) = -1
strncmp(s1, s3, 6) =  0
strncmp(s1, s3, 7) =  1
strncmp(s3, s1, 7) = -1
```

Fonte: Deitel & Deitel

Confronto di stringhe

- Tutti i caratteri sono rappresentati nel computer come codici numeri in un insieme di caratteri come ASCII e Unicode
- Per confrontare stringhe, le funzioni confrontano i codici numerici dei caratteri
  - Confronto **lessicografico**

Confronto di stringhe

- I valori negativi e positivi restituiti dalle funzioni `strcmp` e `strncmp` sono dipendenti dall'implementazione
- Per alcune implementazioni, questi valori sono -1 o 1
- Per altre, i valori restituiti sono la differenza tra i codici numerici dei primi caratteri diversi in ogni stringa

Funzioni per la ricerca

Prototipi e descrizioni delle funzioni

char `*strchr(const char *s, int c);`

Localizza la prima occorrenza del carattere `c` nella stringa `s`. Se `c` viene trovato, `strchr` restituisce un puntatore a `c` in `s`. Altrimenti, viene restituito un puntatore `NULL`.

`size_t strcspn(const char *s1, const char *s2);`

Determina e restituisce la lunghezza del segmento iniziale della stringa `s1` costituito da caratteri non contenuti nella stringa `s2`.

`size_t strspn(const char *s1, const char *s2);`

Determina e restituisce la lunghezza del segmento iniziale della stringa `s1` costituito solo da caratteri contenuti nella stringa `s2`.

char `*strpbrk(const char *s1, const char *s2);`

Localizza la prima occorrenza di un carattere della stringa `s2` nella stringa `s1`. Se viene trovato un carattere della stringa `s2`, `strpbrk` restituisce un puntatore a quel carattere in `s1`. Altrimenti, restituisce `NULL`.

char `*strrchr(const char *s, int c);`

Localizza l'ultima occorrenza di `c` nella stringa `s`. Se `c` viene trovato, `strrchr` restituisce un puntatore a `c` nella stringa `s`. Altrimenti, restituisce `NULL`.

char `*strstr(const char *s1, const char *s2);`

Localizza la prima occorrenza nella stringa `s1` della stringa `s2`. Se la stringa viene trovata, `strstr` restituisce un puntatore alla stringa in `s1`. Altrimenti, restituisce `NULL`.

char `*strtok(char *s1, const char *s2);`

Una sequenza di chiamate a `strtok` suddivide la stringa `s1` in token separati da caratteri contenuti nella stringa `s2`. I token sono unità logiche, come le parole in una riga di testo. La prima chiamata utilizza `s1` come primo argomento, mentre le successive chiamate per continuare a suddividere in token la stessa stringa richiedono `NULL` come primo argomento. A ogni chiamata viene restituito un puntatore al token corrente. Se non vi sono più token, `strtok` restituisce `NULL`.

Fonte: Deitel & Deitel

Funzioni per la ricerca

```c
// fig08_14.c
// Uso della funzione strchr
#include <stdio.h>
#include <string.h>

int main(void) {
   const char *string = "This is a test"; // inizializza un puntatore a char
   char character1 = 'a';
   char character2 = 'z';

    // se character1 e' stato trovato in string
    if (strchr(string, character1) != NULL) { // puoi rimuovere "!= NULL"
       printf("\'%c\' was found in \"%s\".\n", character1, string);
    }
    else { // se character1 non e' stato trovato
       printf("\'%c\' was not found in \"%s\".\n", character1, string);
    }

    // se character2 e' stato trovato in string
    if (strchr(string, character2) != NULL) { // puoi rimuovere "!= NULL"
       printf("\'%c\' was found in \"%s\".\n", character2, string);
    }
    else { // se character2 non e' stato trovato
       printf("\'%c\' was not found in \"%s\".\n", character2, string);
    }
 }
```

```
'a' was found in "This is a test".
'z' was not found in "This is a test".
```

Fonte: Deitel & Deitel

Funzione `strlen`

```
size_t strlen( const char *s);
```

- Riceve una stringa come argomento e ritorna il numero di caratteri
  - Il carattere nullo di terminazione non è incluso nella lunghezza

Programmazione sicura

- È importante convalidare i dati inseriti in un programma
- Possibili problemi durante l'inserimento di un `int` nell'intervallo 1-100:
  - L'utente potrebbe inserire un `int` fuori dall'intervallo richiesto (es. 102)
  - L'utente potrebbe inserire un `int` fuori dall'intervallo consentito per gli `int` su quel computer (es. 8.000.000.000 su una macchina con `int` a 32 bit).
  - L'utente potrebbe inserire un valore numerico non intero (es. 27,43)
  - L'utente potrebbe inserire un valore non numerico (es. "FOVR")
- Funzioni utili per la convalida dell'input:
  - Usare `fgets` per leggere l'input come riga di testo
  - Convertire la stringa in un numero con `strtol` e verificare il successo della conversione
  - Verificare che il valore sia compreso nell'intervallo richiesto

Recap

- Stringa = sequenza di caratteri
- A una stringa si accede tramite un puntatore al primo carattere
- `scanf` per memorizzare una stringa
- Esistono molte funzioni che permettono di analizzare, modificare, confrontare e cercare nelle stringhe