

# **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 suo 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 automaticamente 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
<code>int isblank(int c);</code>	Restituisce un valore vero se <code>c</code> è un carattere vuoto che separa le parole in una riga di testo e 0 (falso) altrimenti.
<code>int isdigit(int c);</code>	Restituisce un valore vero se <code>c</code> è una cifra e 0 (falso) altrimenti.
<code>int isalpha(int c);</code>	Restituisce un valore vero se <code>c</code> è una lettera e 0 (falso) altrimenti.
<code>int isalnum(int c);</code>	Restituisce un valore vero se <code>c</code> è una cifra o una lettera e 0 (falso) altrimenti.
<code>int isxdigit(int c);</code>	Restituisce un valore vero se <code>c</code> 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.)
<code>int islower(int c);</code>	Restituisce un valore vero se <code>c</code> è una lettera minuscola e 0 (falso) altrimenti.
<code>int isupper(int c);</code>	Restituisce un valore vero se <code>c</code> è una lettera maiuscola e 0 (falso) altrimenti.

# 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).

---

# Esempi di applicazioni

```
1 // fig08_01.c
2 // Uso delle funzioni isdigit, isalpha, isalnum e isxdigit
3 #include <ctype.h>
4 #include <stdio.h>
5
6 int main(void) {
7     printf("%s\n%s\n%s\n\n", "According to isdigit: ",
8         isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
9         isdigit( '#' ) ? "# is a " : "# is not a ", "digit");
10
11     printf("%s\n%s\n%s\n%s\n%s\n\n", "According to isalpha:",
12         isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
13         isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
14         isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
15         isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter");
16
17     printf("%s\n%s\n%s\n%s\n\n", "According to isalnum:",
18         isalnum( 'A' ) ? "A is a " : "A is not a ", "digit or a letter",
19         isalnum( '8' ) ? "8 is a " : "8 is not a ", "digit or a letter",
20         isalnum( '#' ) ? "# is a " : "# is not a ", "digit or a letter");
21
22     printf("%s\n%s\n%s\n%s\n%s\n\n", "According to isxdigit:",
23         isxdigit( 'F' ) ? "F is a " : "F is not a ", "hexadecimal digit",
24         isxdigit( 'J' ) ? "J is a " : "J is not a ", "hexadecimal digit",
25         isxdigit( '7' ) ? "7 is a " : "7 is not a ", "hexadecimal digit",
26         isxdigit( '$' ) ? "$ is a " : "$ is not a ", "hexadecimal digit",
27         isxdigit( 'f' ) ? "f is a " : "f is not a ", "hexadecimal digit");
28 }
```

# 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
```

# Esempi di applicazioni

```
1 // fig08_02.c
2 // Uso delle funzioni islower, isupper, tolower e toupper
3 #include <ctype.h>
4 #include <stdio.h>
4
5 int main(void) {
6     printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to islower:",
7         islower( 'p' ) ? "p is a " : "p is not a ", "lowercase letter",
8         islower( 'P' ) ? "P is a " : "P is not a ", "lowercase letter",
9         islower( '5' ) ? "5 is a " : "5 is not a ", "lowercase letter",
10        islower( '!' ) ? "! is a " : "! is not a ", "lowercase letter");
10
11    printf("%s\n%s%s\n%s%s\n%s%s\n\n", "According to isupper:",
12        isupper( 'D' ) ? "D is an " : "D is not an ", "uppercase letter",
13        isupper( 'd' ) ? "d is an " : "d is not an ", "uppercase letter",
14        isupper( '8' ) ? "8 is an " : "8 is not an ", "uppercase letter",
15        isupper( '$' ) ? "$ is an " : "$ is not an ", "uppercase letter");
15
16    printf("%s%c\n%s%c\n%s%c\n%s%c\n",
17        "u converted to uppercase is ", toupper( 'u' ),
18        "7 converted to uppercase is ", toupper( '7' ),
19        "$ converted to uppercase is ", toupper( '$' ),
20        "L converted to lowercase is ", tolower( 'L' ));
21 }
```

# 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



# Esempi di applicazioni

```
1 // fig08_03.c
2 // Uso delle funzioni isspace, iscntrl, ispunct, isprint e isgraph
3 #include <ctype.h>
4 #include <stdio.h>
5
6 int main(void) {
7     printf("%s\n%s%s\n%s%s\n%s\n\n", "According to isspace:",
8         "Newline", isspace( '\n' ) ? " is a " : " is not a ",
9         "whitespace character",
10        "Horizontal tab", isspace( '\t' ) ? " is a " : " is not a ",
11        "whitespace character",
12        isspace( '%' ) ? "% is a " : "% is not a ", "whitespace character");
13
14     printf("%s\n%s%s\n%s%s\n\n", "According to iscntrl:",
15        "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
16        "control character",
17        iscntrl( '$' ) ? "$ is a " : "$ is not a ", "control character");
18
19     printf("%s\n%s%s\n%s%s\n\n", "According to ispunct:",
20        ispunct( ';' ) ? "; is a " : "; is not a ", "punctuation character",
21        ispunct( 'Y' ) ? "Y is a " : "Y is not a ", "punctuation character",
22        ispunct( '#' ) ? "# is a " : "# is not a ", "punctuation character");
23
24     printf("%s\n%s%s\n%s%s\n\n", "According to isprint:",
25        isprint( '$' ) ? "$ is a " : "$ is not a ", "printing character",
26        "Alert", isprint( '\a' ) ? " is a " : " is not a ",
27        "printing character");
28
29     printf("%s\n%s%s\n%s%s\n\n", "According to isgraph:",
30        isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
31        "printing character other than a space",
32        "Space", isgraph( ' ' ) ? " is a " : " is not a ",
33        "printing character other than a space");
34 }
```

# 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
```

# Conversione di stringhe

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

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

# 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

```
1 // fig08_04.c
2 // Uso della funzione strtod
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7     const char *string = "51.2% are admitted" ;
8     char *stringPtr = NULL;
9
10    double d = strtod(string, &stringPtr);
11
12    printf("The string \"%s\" is converted to the\n", string);
13    printf("double value %.2f and the string \"%s\"\n", d, stringPtr);
14 }
```

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

# 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

```
1 // fig08_05.c
2 // Uso della funzione strtol
3 #include <stdio.h>
4 #include <stdlib.h>
5
6 int main(void) {
7     const char *string = "-1234567abc" ;
8     char *remainderPtr = NULL;
9
10    long x = strtol(string, &remainderPtr, 0 );
11
12    printf("%s\\\"%s\\\"\\n%s\\\"%ld\\n%s\\\"%s\\\"\\n%s\\\"%ld\\n\\n",
13        "The original string is ", string,
14        "The converted value is ", x,
15        "The remainder of the original string is ", remainderPtr,
16        "The converted value plus 567 is ", x + 567);
17 }
```

```
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
```



# 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

```
1 // fig08_06.c
2 // Uso della funzione strtoul
3 #include <stdio.h>
4 #include <stdlib.h>
4
5 int main(void) {
6     const char *string = "1234567abc" ;
7     char *remainderPtr = NULL;
7
8     unsigned long int x = strtoul(string, &remainderPtr, 0 );
8
9     printf("%s\n%s\n\n%s%lu\n%s\n%s\n\n%s%lu\n",
10         "The original string is ", string,
11         "The converted value is ", x,
12         "The remainder of the original string is ", remainderPtr,
13         "The converted value minus 567 is ", x - 567);
14 }
```

```
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
```

# Intervallo



Fonte: chatGPT

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

# Funzioni fgets e putchar

```
1 // fig08_07.c
2 // Uso delle funzioni fgets e putchar
3 #include <stdio.h>
4 #define SIZE 80
5
6 void reverse( const char * const sPtr);
7
8 int main(void) {
9     char sentence[SIZE] = "";
10
11     puts("Enter a line of text:");
12     fgets(sentence, SIZE, stdin); // leggi una riga di testo
13
14     printf("\n%s", "The line printed backward is:");
15     reverse(sentence);
16     puts("");
17 }
18
19 // stampa ricorsivamente i caratteri della stringa in ordine inverso
20 void reverse(const char * const sPtr) {
21     // se si raggiunge la fine della stringa
22     if ('\0' == sPtr[0]) { // caso di base
23         return;
24     }
25     else { // se non e' la fine della stringa
26         reverse(&sPtr[ 1 ]); // passo di ricorsione
27         putchar(sPtr[ 0 ]); // usa putchar per stampare il carattere
28     }
29 }
```

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

# 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 23 invia in uscita il carattere all'elemento `sPtr[0]` quando la chiamata ricorsiva è completata
- L'ordine delle istruzioni nelle righe 22 e 23 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

```
1 // fig08_08.c
2 // Uso della funzione getchar
3 #include <stdio.h>
4 #define SIZE 80
5
6 int main(void) {
7     int c = 0; // variabile che contiene il carattere inserito dall'utente
8     char sentence[SIZE] = "";
9     int i = 0;
10
11     puts( "Enter a line of text:" );
12
13     // usa getchar per leggere ogni carattere
14     while ((i < SIZE - 1) && (c = getchar()) != '\n') {
15         sentence[i++] = c;
16     }
17
18     sentence[i] = '\0'; // termina la stringa
19
20     puts("\nThe line entered was:");
21     puts(sentence); // stampa la stringa
22 }
```

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



# Funzione sprintf

```
1 // fig08_09.c
2 // Uso della funzione sprintf
3 #include <stdio.h>
4 #define SIZE 80
5
6 int main(void) {
7     int x = 0;
8     double y = 0.0;
9
10    puts("Enter an integer and a double:");
11    scanf("%d%lf", &x, &y);
12
13    char s[SIZE] = {'\0'}; // crea un array di char
14    sprintf(s, "integer:%6d\ndouble:%7.2f", x, y);
15
16    printf("The formatted output stored in array s is:\n%s\n", s);
17 }
```

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

# Funzione sscanf

```
1 // fig08_10.c
2 // Uso della funzione sscanf
3 #include <stdio.h>
4
5 int main(void) {
6     char s[] = "31298 87.375";
7     int x = 0;
8     double y = 0;
9
10    sscanf(s, "%d%lf" , &x, &y);
11    puts("The values stored in character array s are:");
12    printf("integer:%6d\ndouble:%8.3f\n", x, y);
13 }
```

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

# Manipolazione di stringhe

- La libreria per il trattamento delle stringhe (`<string.h>`) fornisce funzioni per:
  - Manipolare stringhe (copiare e concatenare)
  - Confrontare stringhe
  - Cercare 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
<code>char *strcpy(char *s1, const char *s2)</code>	Copia la stringa <code>s2</code> nell'array <code>s1</code> e restituisce <code>s1</code> .
<code>char *strncpy(char *s1, const char *s2, size_t n)</code>	Copia al massimo <code>n</code> caratteri della stringa <code>s2</code> nell'array <code>s1</code> e restituisce <code>s1</code> .
<code>char *strcat(char *s1, const char *s2)</code>	Aggiunge la stringa <code>s2</code> in coda all'array <code>s1</code> e restituisce <code>s1</code> . Il primo carattere della stringa <code>s2</code> sovrascrive il carattere nullo di terminazione di <code>s1</code> .
<code>char *strncat(char *s1, const char *s2, size_t n)</code>	Aggiunge al massimo <code>n</code> caratteri della stringa <code>s2</code> in coda all'array <code>s1</code> e restituisce <code>s1</code> . Il primo carattere della stringa <code>s2</code> sovrascrive il carattere nullo di terminazione di <code>s1</code> .

# strcpy e strncpy

```
1  // fig08_11.c
2  // Uso delle funzioni strcpy e strncpy
3  #include <stdio.h>
4  #include <string.h>
5  #define SIZE1 25
6  #define SIZE2 15
7
8  int main(void) {
9      char x[] = "Happy Birthday to You"; // inizializza l'array di caratteri x
10     char y[SIZE1] = ""; // crea un array di caratteri y
11     char z[SIZE2] = ""; // crea un array di caratteri z
12
13     // copia il contenuto di x in y
14     printf("%s%s\n%s%s\n",
15           "The string in array x is: ", x,
16           "The string in array y is: ", strcpy(y, x));
17
18     strncpy(z, x, SIZE2 - 1); // copia i primi 14 caratteri di x in z
19     z[SIZE2 - 1] = '\0'; // termina la stringa in z, perche' '\0' non copiato
20     printf("The string in array z is: %s\n", z);
21 }
```

```
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
```

# strcat e strncat

```
1  // fig08_12.c
2  // Uso delle funzioni strcat e strncat
3  #include <stdio.h>
4  #include <string.h>
5
6  int main(void) {
7      char s1[20] = "Happy "; // inizializza l'array di caratteri s1
8      char s2[] = "New Year "; // inizializza l'array di caratteri s2
9      char s3[40] = ""; // inizializza l'array di caratteri s3 come vuoto
10
11     printf("s1 = %s\ns2 = %s\n", s1, s2);
12
13     // aggiungi s2 in coda a s1
14     printf("strcat(s1, s2) = %s\n", strcat(s1, s2));
15
16     // aggiungi i primi 6 caratteri di s1 in coda a s3
17     printf("strncat(s3, s1, 6) = %s\n", strncat(s3, s1, 6));
18
19     // aggiungi s1 in coda a s3
20     printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
21 }
```

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

# strcmp e strncmp

```
1 // fig08_13.c
2 // Uso delle funzioni strcmp e strncmp
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *s1 = "Happy New Year"; // inizializza un puntatore a char
8     const char *s2 = "Happy New Year"; // inizializza un puntatore a char
9     const char *s3 = "Happy Holidays"; // inizializza un puntatore a char
10
11     printf("s1 = %s\ns2 = %s\ns3 = %s\n\n%s%2d\n%s%2d\n%s%2d\n\n",
12           s1, s2, s3,
13           "strcmp(s1, s2) = ", strcmp(s1, s2),
14           "strcmp(s1, s3) = ", strcmp(s1, s3),
15           "strcmp(s3, s1) = ", strcmp(s3, s1));
16
17     printf("%s%2d\n%s%2d\n%s%2d\n",
18           "strncmp(s1, s3, 6) = ", strncmp(s1, s3, 6),
19           "strncmp(s1, s3, 7) = ", strncmp(s1, s3, 7),
20           "strncmp(s3, s1, 7) = ", strncmp(s3, s1, 7));
21 }
```

```
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
```

# Confronto di stringhe

- Tutti i caratteri sono rappresentati nel computer come codici numerici 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**.

---

# Funzioni per la ricerca

```
1 // fig08_14.c
2 // Uso della funzione strchr
3 #include <stdio.h>
4 #include <string.h>
5
6 int main(void) {
7     const char *string = "This is a test"; // inizializza un puntatore a char
8     char character1 = 'a';
9     char character2 = 'z';
10
11     // se character1 e' stato trovato in string
12     if (strchr(string, character1) != NULL) { // puoi rimuovere "!= NULL"
13         printf("\'%c\' was found in \"%s\".\n", character1, string);
14     }
15     else { // se character1 non e' stato trovato
16         printf("\'%c\' was not found in \"%s\".\n", character1, string);
17     }
18
19     // se character2 e' stato trovato in string
20     if (strchr(string, character2) != NULL) { // puoi rimuovere "!= NULL"
21         printf("\'%c\' was found in \"%s\".\n", character2, string);
22     }
23     else { // se character2 non e' stato trovato
24         printf("\'%c\' was not found in \"%s\".\n", character2, string);
25     }
26 }
```

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

# 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