Linguaggio C: Stringhe

Generalità

- Stringa
  - Definita come array di char oppure puntatore a char:

```
char color[] = "blue";
char color[] = {'b','l','u','e','\0'};
char *colorPtr = "blue";
```

  - Quando definite come un array ha il carattere terminatore '\0' in più, quindi `color[]` ha 5 elementi
- Acquisizione
  - Usando `scanf("%s", color)` non c'è bisogno di & because perchè color punta al primo indirizzo di un array.

Generalità

Esempio:

```c
#include <stdio.h>

int main() {
  char string[10]; //inizializzo un array di 10 caratteri

  printf("Inserisci una stringa: ");  // inserisco un valore in input
  scanf("%9s", string); // leggo ESATTAMENTE 9 caratteri tra tutti quelli letti in input
  printf("Stringa letta: %s", string); // stampo
}
```

- se si digitano più di 9 caratteri, sono i primi 9 sono letti
- la acquisizione di una stringa termina quando trova una spaziatura

Librerie disponibili

- Elaborazione di caratteri
- Conversione di stringhe
- I/O di stringhe/caratteri da tastiera
- Copia di stringhe e caratteri
- Ricerca nelle stringhe

Elaborazione di caratteri

- Funzioni per manipolare stringhe attraverso i caratteri disponibili nella libreria `<ctype.h>`
- Ogni funzione ha un carattere (rappresentato come `int`) oppure la costante `EOF`

Elaborazione di caratteri

Prototype	Description
`int isdigit( int c );`	Returns `true` if `c` is a digit and `false` otherwise.
`int isalpha( int c );`	Returns `true` if `c` is a letter and `false` otherwise.
`int isalnum( int c );`	Returns `true` if `c` is a digit or a letter and `false` otherwise.
`int isxdigit( int c );`	Returns `true` if `c` is a hexadecimal digit character and `false` otherwise.
`int islower( int c );`	Returns `true` if `c` is a lowercase letter and `false` otherwise.
`int isupper( int c );`	Returns `true` if `c` is an uppercase letter; `false` otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns `true` if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`), or vertical tab (`'\v'`)—and `false` otherwise
`int iscntrl( int c );`	Returns `true` if `c` is a control character and `false` otherwise.
`int ispunct( int c );`	Returns `true` if `c` is a printing character other than a space, a digit, or a letter and `false` otherwise.
`int isprint( int c );`	Returns `true` value if `c` is a printing character including space (`' '`) and `false` otherwise.
`int isgraph( int c );`	Returns `true` if `c` is a printing character other than space (`' '`) and `false` otherwise.

Elaborazione di caratteri

```c
/* Fig. 8.2: fig08_02.c
   Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main()
{
   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" );

```

Elaborazione di caratteri

```
28     printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29         "According to isxdigit:",
30         isxdigit( 'F' ) ? "F is a " : "F is not a ",
31         "hexadecimal digit",
32         isxdigit( 'J' ) ? "J is a " : "J is not a ",
33         "hexadecimal digit",
34         isxdigit( '7' ) ? "7 is a " : "7 is not a ",
35         "hexadecimal digit",
36         isxdigit( '$' ) ? "$ is a " : "$ is not a ",
37         "hexadecimal digit",
38         isxdigit( 'f' ) ? "f is a " : "f is not a ",
39         "hexadecimal digit" );
40
41     return 0; /* indicates successful termination */
42
43 } /* end main */
```

Elaborazione di caratteri

```
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
```

Elaborazione di caratteri

```c
/* Fig. 8.3: fig08_03.c
   Using functions islower, isupper, tolower, toupper */
#include <stdio.h>
#include <ctype.h>

int main()
{
   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" );

```

Elaborazione di caratteri

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

```
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
```

Elaborazione di caratteri

```
1  /* Fig. 8.4: fig08_04.c
2     Using functions isspace, iscntrl, ispunct, isprint, isgraph */
3  #include <stdio.h>
4  #include <ctype.h>
5
6  int main()
7  {
8     printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
9         "According to isspace:",
10        "Newline", isspace( '\n' ) ? " is a " : " is not a ",
11        "whitespace character", "Horizontal tab",
12        isspace( '\t' ) ? " is a " : " is not a ",
13        "whitespace character",
14        isspace( '%' ) ? "% is a " : "% is not a ",
15        "whitespace character" );
16
17    printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
18        "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
19        "control character", iscntrl( '$' ) ? "$ is a " :
20        "$ is not a ", "control character" );
21
```

Elaborazione di caratteri

```
22    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
23        "According to ispunct:",
24        ispunct( ';' ) ? "; is a " : "; is not a ",
25        "punctuation character",
26        ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
27        "punctuation character",
28        ispunct( '#' ) ? "# is a " : "# is not a ",
29        "punctuation character" );
30
31    printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
32        isprint( '$' ) ? "$ is a " : "$ is not a ",
33        "printing character",
34        "Alert", isprint( '\a' ) ? " is a " : " is not a ",
35        "printing character" );
36
37    printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
38        isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
39        "printing character other than a space",
40        "Space", isgraph( ' ' ) ? " is a " : " is not a ",
41        "printing character other than a space" );
42
43    return 0; /* indicates successful termination */
44
45 } /* end main */
```

Elaborazione di caratteri

```
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

- Funzioni di conversion di stringhe in numeri interi e floating disponibili in `<stdlib.h>`

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

Conversione di stringhe

```c
/* Fig. 8.6: fig08_06.c
   Using atof */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   double d; /* variable to hold converted string */

   d = atof( "99.0" );

   printf( "%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Conversione di stringhe

```c
/* Fig. 8.7: fig08_07.c
   Using atoi */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int i; /* variable to hold converted string */

   i = atoi( "2593" );

   printf( "%s%d\n%s%d\n",
           "The string \"2593\" converted to int is ", i,
           "The converted value minus 593 is ", i - 593 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

Conversione di stringhe

```
1  /* Fig. 8.8: fig08_08.c
2     Using atol */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     long l; /* variable to hold converted string */
9
10    l = atol( "1000000" );
11
12    printf( "%s%ld\n%s%ld\n",
13            "The string \"1000000\" converted to long int is ", l,
14            "The converted value divided by 2 is ", l / 2 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

```
The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
```

Conversione di stringhe

```
1  /* Fig. 8.9: fig08_09.c
2     Using strtod */
3  #include <stdio.h>
4  #include <stdlib.h>
5
6  int main()
7  {
8     /* initialize string pointer */
9     const char *string = "51.2% are admitted";
10
11    double d;        /* variable to hold converted sequence */
12    char *stringPtr; /* create char pointer */
13
14    d = strtod( string, &stringPtr );
15
16    printf( "The string \"%s\" is converted to the\n", string );
17    printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

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

Conversione di stringhe

```c
/* Fig. 8.10: fig08_10.c
   Using strtol */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   const char *string = "-1234567abc"; /* initialize string pointer */

   char *remainderPtr; /* create char pointer */
   long x;             /* variable to hold converted sequence */

   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 );

   return 0; /* indicates successful termination */

} /* end main */
```

```
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
```

Conversione di stringhe

```c
/* Fig. 8.11: fig08_11.c
   Using strtoul */
#include <stdio.h>
#include <stdlib.h>

int main()
{
   const char *string = "1234567abc"; /* initialize string pointer */
   unsigned long x;    /* variable to hold converted sequence */
    char *remainderPtr; /* create char pointer */

    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 );

    return 0; /* indicates successful termination */

} /* end main */
```

```
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
```

Input/output di stringhe e caratteri

- Acquisizione ed immissione di stringhe e caratteri da dispositivo standard (tastiera) disponibili in `<stdio.h>`

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array `s` until a newline or end-of-file character is encountered. A terminating null character is appended to the array.
`int putchar( int c );`	Prints the character stored in `c`.
`int puts( const char *s );`	Prints the string `s` followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` instead of reading it from the keyboard.

Input/output di stringhe e caratteri

```
1  /* Fig. 8.13: fig08_13.c
2     Using gets and putchar */
3  #include <stdio.h>
4
5  int main()
6  {
7     char sentence[ 80 ]; /* create char array */
8
9     void reverse( const char * const sPtr ); /* prototype */
10
11    printf( "Enter a line of text:\n" );
12
13    /* use gets to read line of text */
14    gets( sentence );
15
16    printf( "\nThe line printed backwards is:\n" );
17    reverse( sentence );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
22
```

Input/output di stringhe e caratteri

```
23 /* recursively outputs characters in string in reverse order */
24 void reverse( const char * const sPtr )
25 {
26    /* if end of the string */
27    if ( sPtr[ 0 ] == '\0' ) {
28       return;
29    } /* end if */
30    else { /* if not end of the string */
31       reverse( &sPtr[ 1 ] );
32
33       putchar( sPtr[ 0 ] ); /* use putchar to display character */
34    } /* end else */
35
36 } /* end function reverse */
```

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC

Enter a line of text:
able was I ere I saw elba

The line printed backwards is:
able was I ere I saw elba
```

Input/output di stringhe e caratteri

```c
/* Fig. 8.14: fig08_14.c
   Using getchar and puts */
#include <stdio.h>

int main()
{
   char c;              /* variable to hold character input by user */
   char sentence[ 80 ]; /* create char array */
   int i = 0;           /* initialize counter i */

   /* prompt user to enter line of text */
   puts( "Enter a line of text:" );

   /* use getchar to read each character */
   while ( ( c = getchar() ) != '\n') {
      sentence[ i++ ] = c;
   } /* end while */

   sentence[ i ] = '\0';

   /* use puts to display sentence */
   puts( "\nThe line entered was:" );
   puts( sentence );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Input/output di stringhe e caratteri

```c
/* Fig. 8.15: fig08_15.c
   Using sprintf */
#include <stdio.h>

int main()
{
   char s[ 80 ]; /* create char array */
   int x;        /* define x */
   double y;     /* define y */

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

   sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

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

   return 0; /* indicates successful termination */

} /* end main */
```

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

Input/output di stringhe e caratteri

```c
/* Fig. 8.16: fig08_16.c
   Using sscanf */
#include <stdio.h>

int main()
{
   char s[] = "31298 87.375"; /* initialize array s */
   int x;                     /* define x */
   double y;                  /* define y */

   sscanf( s, "%d%lf", &x, &y );

   printf( "%s\n%s%6d\n%s%8.3f\n",
           "The values stored in character array s are:",
           "integer:", x, "double:", y );

   return 0; /* indicates successful termination */

} /* end main */
```

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

Copia di stringhe e caratteri

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string `s2` into array `s1`. The value of `s1` is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most `n` characters of string `s2` into array `s1`. The value of `s1` is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most `n` characters of string `s2` to array `s1`. The first character of `s2` overwrites the terminating null character of `s1`. The value of `s1` is returned.

disponibili in `<string.h>`

Copia di stringhe e caratteri

```c
/* Fig. 8.18: fig08_18.c
   Using strcpy and strncpy */
#include <stdio.h>
#include <string.h>

int main()
{
   char x[] = "Happy Birthday to You"; /* initialize char array x */
   char y[ 25 ];                       /* create char array y */
    char z[ 15 ];                       /* create char array z */

   /* copy contents of x into y */
   printf( "%s%s\n%s%s\n",
           "The string in array x is: ", x,
           "The string in array y is: ", strcpy( y, x ) );

   /* copy first 14 characters of x into z. Does not copy null
      character */
   strncpy( z, x, 14 );

   z[ 14 ] = '\0'; /* append '\0' to z's contents */
   printf( "The string in array z is: %s\n", z );

   return 0; /* indicates successful termination */
```

```
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
```

Confronto tra stringhe e caratteri

- Si confrontano le rappresentazione in codice ASCII delle stringhe:

```
int strcmp( const char *s1, const char *s2 );
```

  - Confronta i caratteri di s1 con quelli di s2
  - Restituisce <0 se `s1 < s2`, =0 if `s1 == s2` , >0 se `s1 > s2`

```
int strncmp( const char *s1, const char *s2,
  size_t n );
```

  - Confronta *n* caratteri di s1 con s2
  - Restituisce <0 se `s1 < s2`, =0 if `s1 == s2` , >0 se `s1 > s2`

- disponibili in `<string.h>`

Confronto tra stringhe e caratteri

```c
/* Fig. 8.21: fig08_21.c
   Using strcmp and strncmp */
#include <stdio.h>
#include <string.h>

int main()
{
   const char *s1 = "Happy New Year"; /* initialize char pointer */
   const char *s2 = "Happy New Year"; /* initialize char pointer */
   const char *s3 = "Happy Holidays"; /* initialize char pointer */

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

   return 0; /* indicates successful termination */

} /* end main */
```

Confronto tra stringhe e caratteri

```
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
```

Ricerca nelle stringhe

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into "tokens"—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

disponibili in `<string.h>`

Ricerca nelle stringhe

```
1  /* Fig. 8.23: fig08_23.c
2     Using strchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string = "This is a test"; /* initialize char pointer */
9     char character1 = 'a';                 /* initialize character1 */
10    char character2 = 'z';                 /* initialize character2 */
11
12    /* if character1 was found in string */
13    if ( strchr( string, character1 ) != NULL ) {
14       printf( "\'%c\' was found in \"%s\".\n",
15               character1, string );
16    } /* end if */
17    else { /* if character1 was not found */
18       printf( "\'%c\' was not found in \"%s\".\n",
19               character1, string );
20    } /* end else */
21
```

Ricerca nelle stringhe

```
22    /* if character2 was found in string */
23    if ( strchr( string, character2 ) != NULL ) {
24       printf( "\'%c\' was found in \"%s\".\n",
25               character2, string );
26    } /* end if */
27    else { /* if character2 was not found */
28       printf( "\'%c\' was not found in \"%s\".\n",
29              character2, string );
30    } /* end else */
31
32    return 0; /* indicates successful termination */
33
34 } /* end main */
```

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

Ricerca nelle stringhe

```
1  /* Fig. 8.24: fig08_24.c
2     Using strcspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "1234567890";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u",
13            "string1 = ", string1, "string2 = ", string2,
14            "The length of the initial segment of string1",
15            "containing no characters from string2 = ",
16            strcspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

```
string1 = The value is 3.14159
string2 = 1234567890

The length of the initial segment of string1
containing no characters from string2 = 13
```

Ricerca nelle stringhe

```
1  /* Fig. 8.25: fig08_25.c
2     Using strpbrk */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string1 = "This is a test"; /* initialize char pointer */
9     const char *string2 = "beware";         /* initialize char pointer */
10
11    printf( "%s\"%s\"\n'%c'%s\n\"%s\"\n",
12            "Of the characters in ", string2,
13            *strpbrk( string1, string2 ),
14            " is the first character to appear in ", string1 );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

```
Of the characters in "beware"
'a' is the first character to appear in
"This is a test"
```

Ricerca nelle stringhe

```
1  /* Fig. 8.26: fig08_26.c
2     Using strrchr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize char pointer */
9     const char *string1 = "A zoo has many animals "
10                          "including zebras";
11    int c = 'z'; /* initialize c */
12
13    printf( "%s\n%s'%c'%s\"%s\"\n",
14            "The remainder of string1 beginning with the",
15            "last occurrence of character ", c,
16            " is: ", strrchr( string1, c ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

```
The remainder of string1 beginning with the
last occurrence of character 'z' is: "zebras"
```

Ricerca nelle stringhe

```
1  /* Fig. 8.27: fig08_27.c
2     Using strspn */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     /* initialize two char pointers */
9     const char *string1 = "The value is 3.14159";
10    const char *string2 = "aehi lsTuv";
11
12    printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
13            "string1 = ", string1, "string2 = ", string2,
14            "The length of the initial segment of string1",
15            "containing only characters from string2 = ",
16            strspn( string1, string2 ) );
17
18    return 0; /* indicates successful termination */
19
20 } /* end main */
```

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Ricerca nelle stringhe

```
1  /* Fig. 8.28: fig08_28.c
2     Using strstr */
3  #include <stdio.h>
4  #include <string.h>
5
6  int main()
7  {
8     const char *string1 = "abcdefabcdef"; /* initialize char pointer */
9     const char *string2 = "def";          /* initialize char pointer */
10
11    printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
12            "string1 = ", string1, "string2 = ", string2,
13            "The remainder of string1 beginning with the",
14            "first occurrence of string2 is: ",
15            strstr( string1, string2 ) );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Ricerca nelle stringhe

```c
/* Fig. 8.29: fig08_29.c
   Using strtok */
#include <stdio.h>
#include <string.h>

int main()
{
   /* initialize array string */
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr; /* create char pointer */

   printf( "%s\n%s\n\n%s\n",
           "The string to be tokenized is:", string,
           "The tokens are:" );

   tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */

   /* continue tokenizing sentence until tokenPtr becomes NULL */
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); /* get next token */
   } /* end while */

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```

Esercizio

Scrivere un programma he permette all'utente di inserire un'intera frase. Il programma include una funzione che calcola il numero di parole che compone la frase. (Si supponga che una frase sia costituita di parole, dove una parola è una sequenza di caratteri diversi dallo spazio delimitata da uno o più spazi a destra o a sinistra).

Esercizio

Scrivere un programma che i) acquisisca in input la password inserita da un utente, ii) verifichi se la password contiene almeno una lettera maiuscola ed almeno un numero, iii) stampi in output un messaggio di conferma se la password è corretta, in alternativa, stampare un messaggio di errore.

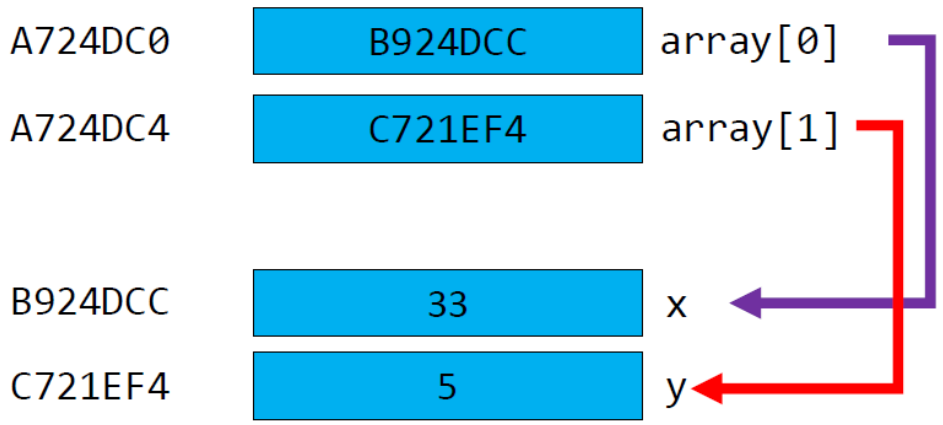
Esercizio

1. Scrivere un programma in C che legge un vettore di stringhe (matrice di caratteri). Si progetti per tale programma:
   a) la procedura SCAMBIO che avendo come argomenti il vettore di stringhe M e due indici I e J, effettui lo scambio tra la stringa memorizzata in posizione M[i] e la stringa memorizzata in posizione M[j]
   b) La funzione CONTAPAROLE che avendo come argomenti il vettore di stringhe M e una parola p conti le occorrenze di p all'interno di M
   c) La procedura RIMPIAZZA he avendo come argomenti il vettore di stringhe M, una parola p1 e una parola p2 rimpiazzi in M tutte le occorrenze di p1 con p2.

Array di puntatori

- In quanto trattati come normali variabili, i puntatori possono essere collezionati in un array.
- Ad esempio, possiamo inizializzare un array di puntatori:

```
int x=33;
y=5;
int *array[2]={&x, &y}
```



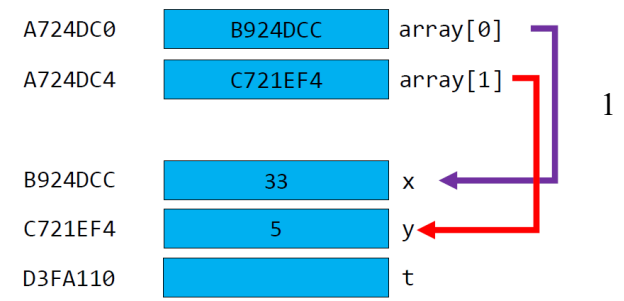
Array di puntatori

- Come la funzione *scambia()* potrebbe essere estesa per scambiare elementi consecutivi in un array di puntatori?

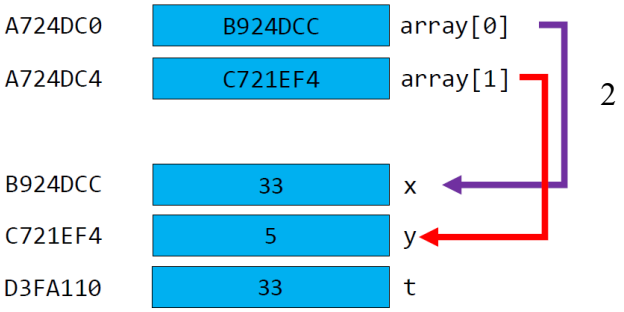
```
void scambia(int* a, int* b) {
   int t; // variabile locale di appoggio
   t = *a; // scambio dei valori
   *a = *b;
   *b = t;
}

int main() {
   int x = 33, y = 5;
   scambia(&x, &y);
   printf("x = %d, y = %d\n", x, y);
}
```

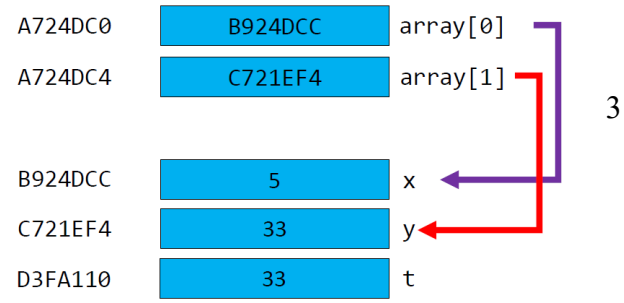
Array di puntatori



1



2



3

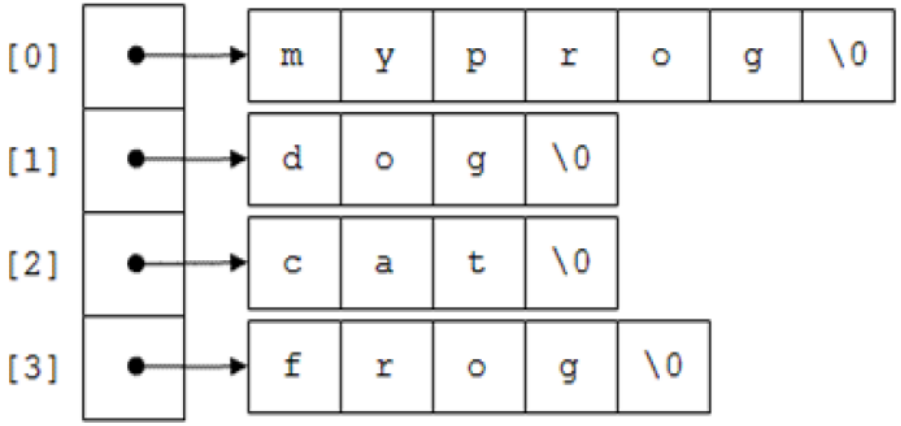
```
void scambia(int* a[]) {
   int t; // variabile locale di appoggio
   t = *(*a);    // scambio dei valori
   *(*a) = *(*(a+1));
   *(*(a+1)) = t;}

int main() {
   int x = 33, y = 5;
   int* array[2] = {&x, &y}
   scambia(array);
   printf("x = %d, y = %d\n", x, y);
}
```

Ricordiamo che *(a+i) accede all'i-esimo elemento, che è un indirizzo in questo caso, quindi *(*(a+i)) corrisponderà al valore puntato.

Array di puntatori: Stringhe

- Un particolare array di puntatori è un array di stringhe.
- Infatti, un array di char è comunemente associato ad una stringa, mentre il nome di un array denota il primo indirizzo di un array.
- Ne consegue che il nome di una stringa (cioè, array di char) punta al primo carattere dell'array. Pertanto, un array di stringhe è un array di puntatori ai primi caratteri delle stringhe.
- Le stringhe possono avere differente lunghezza.




```
char* array[4] = {"myprog", "dog", "cat" , "frog"};
```

Array di puntatori: Stringhe

- Esempio: scambiare il nome di due persone

```
int main() {
  char* persona_A[2] = {"Mario", "Rossi"};
  char* persona_B[2] = {"Luigi", "Bianchi"};

  // stampa, scambia i valori e stampa di nuovo
  printf("\nBefore\n----\n%s \t %s \n%s \t %s",
         persona_A[0], persona_A[1], persona_B[0], persona_B[1]);

  scambia(persona_A, persona_B);

  printf("\n\nAfter\n----\n%s \t %s \n%s \t %s",
         persona_A[0], persona_A[1], persona_B[0], persona_B[1]);
}
```

```
#include <stdio.h>

// equivalente a: scambia(char* a[], char* b[])
void scambia(char** a, char** b) {
  char* s;
  s = *a;
  *a = *b;
  *b = s;
}
```

Array di puntatori: Stringhe

- Esempio: scambiare anche il cognome di due persone

```c
int main() {
  char* persona_A[2] = {"Mario", "Rossi"};
  char* persona_B[2] = {"Luigi", "Bianchi"};

  // stampa, scambia i valori e stampa di nuovo
  printf("\nBefore\n----\n%s \t %s \n%s \t %s",
         persona_A[0], persona_A[1], persona_B[0], persona_B[1]);

  scambia(persona_A, persona_B);

  printf("\n\nAfter\n----\n%s \t %s \n%s \t %s",
         persona_A[0], persona_A[1], persona_B[0], persona_B[1]);
}
```

```c
// equivalente a: scambia(char* a[], char* b[])
void scambia(char** a, char** b) {
  char* s;
  s = *(a+1);
  *(a+1) = *(b+1);
  *(b+1) = s;
}
```

Usi di puntatori: restituzione di stringhe

- Una stringa può essere restituita da una funzione attraverso un puntatore.
- Più generalmente, un array può essere restituito da una funzione usando una variabile puntatore.
- Tuttavia, un compilatore C in modo standard alloca memoria per le variabili attraverso una gestione **statica**, cioè è necessario sapere a tempo di compilazione la dimensione delle variabili, mentre la dimensione (lunghezza) di una stringa potrebbe non essere nota a compilazione.

Usi di puntatori: restituzione di stringhe

- Si ricorre alla Allocazione **dinamica**, impiegando **operatori** per la gestione della memoria

  `malloc(), calloc()`

  ```
  void *calloc(size_t nitems, size_t size)
  ```

- che ’’mettono da parte ‘’ memoria che il programma potrà usare

Usi di puntatori: restituzione di stringhe

- Esempio:

```
char* hidePassword(char* string) {

      char* s = (char*)  calloc(10, sizeof(char));
...

return s

}
```

Esercizio

Scrivere un programma che acquisisca una stringa (password) e converta le vocali in ‘$’ e le consonanti in ‘*’, restituendo la stringa finale.

Esercizio

Scrivere un programma che acquisisca una stringa (password) e converta le vocali in ‘$’ e le consonanti in ‘*’, restituendo la stringa finale.

Esercizio

Scrivere un programma C che legge un vettore di parole e stampa a video le parole distinte memorizzate in tale vettore.

Esercizio

Scrivere un programma C che legge un vettore di parole e lo ordina in maniera crescente