First Year Engineer in computer science

CHAPTER 11:

Strings

2023-2024

Sid Ahmed BERRABAH

Definition

- A **string** is a sequence of characters enclosed in double quotes **""**.

Example: "this is a string".

- A variable of string type in the C language is an array of characters (char) whose last character is the null character (**\0**).

Initializing a string

- A string can be initialized in the traditional way with a list of constants:

```
char message[6] = {'h','e','l','l','o','\0'};
```

- C also offers the ability to initialize a string using a literal string directly:

```
char message[6] = "hello";
```

The compiler automatically adds the null character at the end of the string. The array must have at least one element more than the number of characters in the literal string.

Initializing a string

- The declared size for the array can be larger than the size of the string (it's the null character that specifies the end of the string to the program).

```
char message[100] = "hello";
```

Only the first 6 characters of the message variable are used.

- It is also possible to not specify the size of the array. In this case, the compiler will automatically allocate the correct size for the array, including the null character.

```
char message[] = "coucou"; //allocates 7 bytes
```

- You can also use a pointer to a literal string

```
char *message = "this is a string";
```

Strings: Pointers vs Arrays

Example :

```
char message[100];
message = "hello"; // error: not allowed
```

At runtime, message is converted to a constant char pointer that contains the address of the first element of the array. The instruction message = "hello"; tries to assign the address of the beginning of the string to message, which is impossible (since it is constant).

- On the other hand, the following example is perfectly legal:

```
char *message;
message = "hello";
```

This is because the char pointer message receives the address of the beginning of the string "hello" that the compiler placed somewhere in memory when it was created.

Strings: Pointers vs Arrays

- **a string literal is constant and cannot be modified :**

```
char *s1, s2[] = "message";
s1 = "hello";
s1[0] = 'H';  // ERROR not allowed
s2[0] = 'M';  // Correct
```

Read and write strings

- **scanf**

```
char str[10];
scanf("%s", str);
```

In the above code, the function **scanf()** reads a sequence of characters from the keyboard and stores them in the array str and automatically adding a null character at the end of the string. The **%s** format specifier tells scanf() to read a string.

The reading stops when a delimiter (space or newline) is encountered.

Therefore, scanf cannot read a string starting with a space or containing a space.

scanf only works with an array (or a pointer initialized by malloc). The memory space must be reserved to accommodate the string.

Read and write strings

Notes:

- The **scanf()** function can cause a buffer overflow if the user enters more characters than the array can hold.

In the example

```
char str[10];
scanf("%s", str);
```

the array str is declared with a size of 10 characters. If the user enters a string of 11 characters, the excess character will be placed after the end of the array. This could overwrite other data in memory.

To prevent buffer overflows, it is important to generously reserve the space needed to store the string. In the example above, the array str could be declared with a size of 255 characters. This would ensure that there is enough space to store any string that the user might enter.

Read and write strings

- **printf :**

To display a string with printf, you can use the %s format

Example :

```
char s[100] = "good morning!";
printf("s = %s", s);
```

will output:

```
s = good morning!
```

It should be noted that printf cannot know the length of the string. It only knows the start address of the string s (which is equal to &s[0]) and displays all the characters in the string until it encounters a null character (\0)..

Read and write strings

- **printf :**

It is possible to use a string variable instead of the literal string of the printf as in this example:

```
char s1[100] = "an example";
char s2[100] = "s1 = %s";
printf(s2, s1);
```

Displays:

s1 = an example

Read and write strings

- What do the following codes display?

1)

```
int main()
{
   char *s;
   s = "Good morning";
   printf("%s\n", s);
   s = "Sir !";
   printf("%s\n", s);
}
```

Read and write strings

- What do the following codes display?

1)

```
int main()
{
   char *s;
   s = "Good morning";
   printf("%s\n", s);
   s = "Sir !";
   printf("%s\n", s);
}
```

Good morning
Sir !

Read and write strings

- What do the following codes display?

1)

```
int main()
{
   char *s;
   s = "Good morning";
   printf("%s\n", s);
   s = "Sir !";
   printf("%s\n", s);
}
```

Good morning
Sir !

2)

```
#define NB_CHAR 100
int main()
{
   char *s;
   s1 = malloc(NB_CHAR * sizeof(char));
   scanf("%s", s);
   printf("%s\n", s);
}
```

enter : hello world !

Read and write strings

- What do the following codes display?

1)

```
int main()
{
   char *s;
   s = "Good morning";
   printf("%s\n", s);
   s = "Sir !";
   printf("%s\n", s);
}
```

Good morning
Sir !

2)

```
#define NB_CHAR 100
int main()
{
   char *s;
   s1 = malloc(NB_CHAR * sizeof(char));
   scanf("%s", s);
   printf("%s\n", s);
}
```

enter : hello world !

hello

Read and write strings

- What do the following codes display?

3)

```
int main()  {
        char *s = "hello world !";
        int i;
        for (i = 0; i < 4; i++)
                putchar(s[i]);      //printf("%c",s[i]);
        printf("\n");
        i = 0;
        while(s[i] != '\0')
              putchar(s[i++]);
        printf("\n");
        i = 0;
        while(s[i])
              putchar(s[i++]);
}
```

Read and write strings

- What do the following codes display?

3)

```
int main()  {
        char *s = "hello world !";
        int i;
        for (i = 0; i < 4; i++)
                putchar(s[i]);      //printf("%c",s[i]);
        printf("\n");
        i = 0;
        while(s[i] != '\0')
              putchar(s[i++]);
        printf("\n");
        i = 0;
        while(s[i])
              putchar(s[i++]);
}
```

hell
Hello world !
Hello world !

Read and write strings

- **Exercise :** modify the previous program using only pointers.

Read and write strings

- **Exercise :** modify the previous program using only pointers.

3)

```
int main()  {
        char *s = "hello world !";
        char *p;
        for (p=s; p < s+4; p++)
                putchar(*p);      //printf("%c",*p);
        printf("\n");
        p = s;
        while(*p != '\0')
              putchar(*p++);
        printf("\n");
        p = s;
        while(*p)
              putchar(*p++);
}
```

Read and write strings

- C also offers the **gets** function to read a string and **puts** to display a string.

  They only handle strings.

Example :

```
char s[100];
gets(s1);
puts(s1);
```

The execution gives::

```
^^one^^two^^three↵
^^one^^two^^three
```

The **gets(s)** statement will read a sequence of characters and store it in s1, terminated by a null character.

The **puts(s)** statement displays the characters found from &s[0] until reaching the null character, then performs a line feed (this is the only difference with printf).

Read and write strings

It should be noted that, unlike **scanf()**, when using **gets()**:

- No delimiter is skipped before reading.
- Spaces are read like other characters.
- Reading stops only when a newline character is encountered, which is not copied into the string.
- Since there is no control over the number of characters to be read, an overflow is quite possible if the input string is too long. This is a major security flaw in the C language.

Read and write strings

Exercise : Write a C program with the following behavior using the functions gets, puts, scanf, and printf.

What is your name and first name: **Berrabah Ahmed** ↵

Where do you live? **Tlemcen** ↵

Hello Mr. Berrabah Ahmed from Tlemcen

Read and write strings

- **sscanf**

sscanf is a function that reads formatted data from a string. It is similar to the scanf function, but instead of reading data from the keyboard, it reads data from a string.

Example :

```
char s1[100] = "1 2";
int x, y;
sscanf(s1, "%d%d", &x, &y);
printf("x = %d, y = %d", x, y);
```

This code will output:

```
x = 1, y = 2
```

Read and write strings

- **sprintf**

sprintf is similar to printf, but instead of printing to the screen, it prints to a string.

Example :

```
char s1[100] ;
int x = 1, y = 2;
sprintf(s1, "x = %d, y = %d\n", x, y);
printf(s1);
```

This code will output:

```
x = 1, y = 2
```

Determining the Length of a String

The ***strlen()*** (STRing LENgth) function in string.h allows you to determine the length of a string (between the start address of the string and the null character).

It takes a string as input, which it cannot modify (this is the role of the const type qualifier), and returns the length of the string.

The null character is not counted.

Determining the Length of a String

Example:

```
int main()
{
    char s1[100] = "hello";
    int l;

    l = strlen(s1);
    printf("the length of the string %s is %d", s1, l);
    return 0;
}
```

This code will output:

the length of the string hello is 5

Copying a string to another string

The ***strcpy()*** (STRing CoPY) function in string.h allows you to copy a string to another string.

Its prototype:

```
char * strcpy(char *destination, const char *source);
```

It copies the source string (which it cannot modify) into the destination string, including the null character. It returns the address of the destination string.

Note: When you use **strcpy()**, the size of the destination string should be large enough to store the copied string. Otherwise, it may result in undefined behavior.

Copying a string to another string

Example

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

main()
{
      char s1[100], s2[100], *s3;

      s3 = strcpy(s1, "bonjour"); /* s3 → s1 */
      s3 = strcpy(s2, "bonjour"); /* s3 → s2 */

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

Copying a string to another string

Example

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

main()
{
      char s1[100], s2[100], *s3;

      s3 = strcpy(s1, "bonjour"); /* s3 → s1 */
      s3 = strcpy(s2, "bonjour"); /* s3 → s2 */

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

Output :
s1 = bonjour
s2 = bonjour

Copying a string to another string

The **strncpy** function is similar to the **strcpy** function, but it allows you to specify the number of characters to be copied. This can be useful for avoiding buffer overflows.

Example : the following code will copy the first 10 characters from the source string to the destination string:

```
char source[] = "This is a test string.";
char destination[20];
strncpy(destination, source, 10);
printf("%s\n", destination);
```

Copying a string to another string

The **strncpy** function is similar to the **strcpy** function, but it allows you to specify the number of characters to be copied. This can be useful for avoiding buffer overflows.

Example : the following code will copy the first 10 characters from the source string to the destination string:

```
char source[] = "This is a test string.";
char destination[20];
strncpy(destination, source, 10);
printf("%s\n", destination);
```

This code will output:

This is a

Compare two strings

The strcmp (STRing CoMPare) function in string.h allows you to compare two strings character by character:

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

It compares the two strings from the first character until the characters are different or one string ends.

It returns an integer:

- < 0 if chaine1 is less than chaine2 (in the sense of ASCII codes),
- = 0 if the two strings are identical,
- > 0 if chaine1 is greater than chaine2.

Letter	ASCII Code	Binary	Letter	ASCII Code	Binary
a	097	01100001	A	065	01000001
b	098	01100010	B	066	01000010
c	099	01100011	C	067	01000011
d	100	01100100	D	068	01000100
e	101	01100101	E	069	01000101
f	102	01100110	F	070	01000110
g	103	01100111	G	071	01000111
h	104	01101000	H	072	01001000
i	105	01101001	I	073	01001001
j	106	01101010	J	074	01001010
k	107	01101011	K	075	01001011
l	108	01101100	L	076	01001100
m	109	01101101	M	077	01001101
n	110	01101110	N	078	01001110
o	111	01101111	O	079	01001111
p	112	01110000	P	080	01010000
q	113	01110001	Q	081	01010001
r	114	01110010	R	082	01010010
s	115	01110011	S	083	01010011
t	116	01110100	T	084	01010100
u	117	01110101	U	085	01010101
v	118	01110110	V	086	01010110
w	119	01110111	W	087	01010111
x	120	01111000	X	088	01011000
y	121	01111001	Y	089	01011001
z	122	01111010	Z	090	01011010

Compare two strings

Example :

```
int main()
{
    char s1[] = "abcdef", s2[] = "abcdEf";
    int cmp;

    cmp = strcmp(s1,s2);
    if(cmp == 0)
       printf("s1 is equal to s2\n");
    else if(cmp < 0)
       printf("s1 is lower then s2\n");
    else
       printf("s1 is higher then s2\n");

   return 0;
}
```

Compare two strings

Example :

```
int main()
{
    char s1[] = "abcdef", s2[] = "abcdEf";
    int cmp;

    cmp = strcmp(s1,s2);
    if(cmp == 0)
       printf("s1 is equal to s2\n");
    else if(cmp < 0)
       printf("s1 is lower then s2\n");
    else
       printf("s1 is higher then s2\n");

   return 0;
}
```

Output :

s1 is higher s2

Concatenate two strings

The **strcat** (STRing conCATenation) function in string.h allows you to concatenate two strings, i.e. it places one string at the end of another.

```
char * strcat(char *destination, const char *source);
```

It adds the source string (which it cannot modify) to the end of the destination string. It returns the address of the destination string.

Concatenate two strings

Example :

```
int main()
{
    char s1[100] = "Hello", s2[] = " World!", *s3;

    s3 = strcat(s1,s2);
    printf("s3 = %s", s3);

    return 0;
}
```

Concatenate two strings

Example :

```
int main()
{
    char s1[100] = "Hello", s2[] = " World!", *s3;

    s3 = strcat(s1,s2);
    printf("s3 = %s", s3);

    return 0;
}
```

Output:

s3 = Hello World!

Concatenate two strings

The strncat function is similar to strcat, but it only concatenates the first n characters of the source string.

For example:

```
strncat(s1, s2, 10);
```

This will only add the first 10 characters of s2 to the end of s1.

Searching for a character in a string

The **strchr** (STRing CHaRacter) function in string.h searches for a character in a string.

```
char * strchr(const char *str, char c);
```

It searches the string chaine for the first occurrence of the character c and returns a pointer to that character or a NULL pointer if the character does not exist.

Searching for a character in a string

Example :

```c
int main()
{
   char str[] = "Hello, world!";
   char c = 'o';

   char *p = strchr(str, c);

   if (p == NULL)
       printf("The character '%c' does not exist in the string '%s'.\n", c, str);
   else
       printf("The character '%c' is at position %d in the string '%s'.\n", c, p - str, str);

   return 0;
}
```

Searching for a character in a string

Example :

```c
int main()
{
   char str[] = "Hello, world!";
   char c = 'o';

   char *p = strchr(str, c);

   if (p == NULL)
       printf("The character '%c' does not exist in the string '%s'.\n", c, str);
   else
       printf("The character '%c' is at position %d in the string '%s'.\n", c, p - str, str);

   return 0;
}
```

Output:

The character 'o' is at position 4 in the string 'Hello, world!'.

Searching for a string within another string

The **strstr()** (STRing STRing) function in string.h is used to search for a string within another string.

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

It searches the string s1 for the first occurrence of the string s2 and returns a pointer to the character in s1 where s2 begins, or a null pointer if s2 does not exist in s1.

Searching for a string within another string

Example :

```
int main()
{
   char *s1 = "Hello, world!";
   char *s2 = "world";

   char *p = strstr(s1, s2);

   if (p == NULL)
       printf("The string s2 does not exist in the string s1.\n");
   else
       printf("The string s2 is found at index %d in the string s1.\n", p - s1);

       return 0;
}
```

Searching for a string within another string

Example :

```
int main()
{
   char *s1 = "Hello, world!";
   char *s2 = "world";

   char *p = strstr(s1, s2);

   if (p == NULL)
       printf("The string s2 does not exist in the string s1.\n");
   else
       printf("The string s2 is found at index %d in the string s1.\n", p - s1);

       return 0;
}
```

Output

The string s2 is found at index 7 in the string s1.

Conversion Functions

- **Converting a String to a Numeric Value**

There are three functions in stdlib.h that can be used to convert a string to a numeric value of type int, long, or double. These functions ignore any spaces at the beginning of the string and use the following characters to construct a numeric value (digits 0..9, the decimal point (.), the exponent (e or E), the plus (+) and minus (-) signs)

The first invalid character stops the scan. If no characters are usable, these functions return a null result.

- **atoi (str)** return an int
- **atol (str)** return a long
- **atof (str)** return a double

Note that these functions do the same job as sscanf applied to a single variable, with the appropriate format code.

Conversion Functions

Example :

```c
int main()
{
    int i, j;
    long l;
    double d;

    i = atoi("123");
    sscanf("45", "%d", &j);
    l = atol("1234567890");
    d = atof("3.14159");

    printf("i = %d\n", i);
    printf("i = %d\n", j);
    printf("l = %ld\n", l);
    printf("d = %f\n", d);

    return 0;
}
```

Conversion Functions

Example :

```c
int main()
{
    int i, j;
    long l;
    double d;

    i = atoi("123");
    sscanf("45", "%d", &j);
    l = atol("1234567890");
    d = atof("3.14159");

    printf("i = %d\n", i);
    printf("i = %d\n", j);
    printf("l = %ld\n", l);
    printf("d = %f\n", d);

    return 0;
}
```

Output:

i = 123

i = 45

l = 1234567890

d = 3.141590

Arrays of strings

Arrays of strings are arrays where each elements are array od characters. Arrays of strings are 2D arrays.

Example :

```
char tab[5][10] = {"zero", "one", "two", "three", "four"};
```

The memory storage can be represented as follows:

z	e	r	o	\0					
o	n	e	\0						
t	w	o	\0						
t	h	r	e	e	\0				
f	o	u	r	\0					

Arrays of strings

Example :

```
char tab[5][10] = {"zero", "one", "two", "three", "four"};
```

To read the fourth line (string) of the this array, use:

```
scanf("%s", &tab[3][0]);
```

scanf reads characters from the input and stores them in the array starting at the address specified by &tab[3][0].

If the number of characters read is greater than 10, the characters will wrap around to the next line.

Arrays of strings

Example :

```
char tab[5][10] = {"zero", "one", "two", "three", "four"};

strcpy(&tab[1][0], "This is a test string");
```

The array in the memory will be:

z	e	r	o	\0					
T	h	i	s	^	i	s	^	a	^
t	e	s	t	^	s	t	r	i	n
g	\0								
f	o	u	r	\0					

Arrays of strings

This method of declaring arrays of strings is not practical. In addition, a large amount of memory can be wasted if the strings are of different lengths.

There is a much more efficient way to solve this problem, which is to use an array of pointers to char:

```
Char * tab[5] = {"zero", "one", "two", "three", "four"};
```

In the array tab, we now have 5 pointers (4 bytes per pointer).

Each pointer in the array stores the address of the corresponding string's start location in memory.

This method minimizes memory wastage by only allocating space for the actual characters in each string.

It also provides flexibility in handling strings of varying lengths.

Arrays of strings

Warning: If you only declare

```
char *tab[5];
```

no memory reservation is made to store strings. You have only created an array of 5 pointers that point to nothing.

You must explicitly allocate memory for each string before using it.

Arrays of strings

Example:

```c
int main()
{
   char * day[7] = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday"};

   int i;

   printf("enter an integer between 1 and 7 : ");
   scanf("%d", &i);

   printf("the day number %d in the week is %s", i, day[i-1]);

   return 0;
}
```

Output :

```
enter an integer between 1 and 7 : 3
the day number 3 in the week is Tuesday
```