CHARACTER STRINGS

1. Overview

The learning objective of this lab session is to:

- Understand the internal representation of character strings
- Acquire skills in manipulating character strings with standard string handling functions.

2. Brief theory reminder

2.1. Internal memory representation of a character string

A character string is stored as one-dimensional array of char type. Each character is represented on a byte by its ASCII code. The last character in the string is the null character ('\0').
The name of the array that stores the string is a constant pointer of the character string.

We have the following relations:

string[i], where **i** $\in$ **[0,16]** represents the ASCII code of the ith character of the string;

string + i, where **i** $\in$ **[0,16]** is the address of the ith character of the string;

***(string + i)** has the same effect as **string[i]**.

A string may be declared also as follows:

char *const string="CHARACTER STRING";

An array of strings may be declared as:

char *tab[]={string_0, string _1,..., string _n};

In this case, **tab[i]**, for **i** $\in$ **[0, n]**, is a pointer to the string "**string_i**".
The statement:

printf("%s\n", tab[i]);

will display the text **string_i .**

2.2. Standard string handling functions

The standard input/output functions for strings are:

- **gets/puts;**
- **scanf/printf;**
- **sscanf/sprintf**

They were presented in Lab. 1.

Next we present string handling functions with prototype in **string.h.**

2.2.1. String length

The length of a string (without '\0') is returned by the function **strlen**, having the prototype:

unsigned strlen (const char *s);

Example:

```
/* Program L9Ex1.c */

/* The usage of the function  strlen */
#include <stdio.h>
#include <string.h>
#define prompt "Press a key!"

int main(void)
{
   char string1[]="STRING OF CHARACTERS";
   char  * string2="STRING OF CHARACTERS";
   int n1, n2, n3;

   n1=strlen(string1);
   n2=strlen(string2);
   n3=strlen("STRING OF CHARACTERS");
   /* The values of n1, n2 and n3 are the same, i.e. 20 */
   printf("\n n1=%d n2=%d n3=%d\n", n1, n2, n3);
   printf("%s\n", prompt);
   getchar();
   return 0;
}
```

2.2.2. String copy

To copy a string, from a source memory area (of address **source**) into another memory area (of address **dest**) you can use the function **strcpy**, having the prototype:

char *strcpy (char *dest, const char *source);

Notes:
- The copy includes the ASCII null character.
- The function returns the address of the destination.

To copy of no more than ***n*** characters of a string from a source memory area (having the address **source**) into another memory area (having the address **dest**) you can use the function **strncpy**, having the prototype:

char *strncpy (char *dest, const char *source, unsigned n);

After the last character that is transferred, you must append a null ASCII character ('\0').

Obviously, if ***n*** is greater than the length of the source string, the entire source string is copied.

Example:

```
/* Program L9Ex2.c */
/* Usage of the function strcpy */
#include <stdio.h>
#include <string.h>
#define prompt "\nPress a key!"

int main(void)
{
   char string1[]="STRING OF CHARACTERS";
   char *string2="STRING OF CHARACTERS";
   char string3[100], string4[100], string5[100];

   strcpy(string3, string1);
   printf("\nstring3: %s\n", string3);
   strcpy(string4, "Standard string handling functions");
   printf("\nstring4: %s\n", string4);
   strncpy(string5, string2, 9);/* string5 contains "STRING OF" and whatever was there
before */
   printf("\nunterminated string5: %s\n", string5);
   string5[6]='\0'; // terminate string 5 at position 6
   printf("\nstring5 (terminated at position 6): %s\n", string5);
   puts(prompt);
   getchar();
   return 0;
}
```

2.2.3. String concatenation

Appending a source character string, located in a memory area of address **source**, after the last character before '\0' of another string located in a memory area of **dest,** is done using the function **strcat**, having the prototype:

char *strcat(char *dest, const char *source);

It is important to put the ASCII character null ('\0') at the end of the resulting string. The function returns the address of the destination.

You can take only ***n*** characters from the source string, using the function **strncat,** having the prototype:

char *strncat (char *dest, const char *source, unsigned n);

The null ASCII character is automatically appended to the result string. If ***n*** is greater than the length of the source string, **strncat** has the effect of **strcat.**
Example:

```
/* Program L9Ex3.c */

/* Usage of the function strcat */
#include <stdio.h>
#include <string.h>
#define prompt "\nPress a key!"

int main(void)
{
   char string1[]="STRING1 OF CHARACTERS";
   char *string2="STRING2 OF CHARACTERS";
   char string3[100];
   int i;
   printf("string1: %s", string1);
   strcpy(string3, string1);
   printf("string3(=string1): %s", string3);
   strcat(string3, string2);
   printf("\nstring3(=string3+string2): %s\n", string1);
   strncat(string3, string2, 5);
   /* After the last character of the string string3, '\0' is placed  by default */
   printf("Contents of string 3 in hex:");
   for (i=0; i <= strlen(string3)+1; ++i)
      printf("%x", string3[i]);
   printf("\nstring3(as text): %s\n", string3);
   puts(prompt);
   getchar();
   return 0;
}
```

2.2.4. String comparison

The comparison of two string is done by taking sequentially the pairs of characters located on the *i* th position in the compared strings, based on their ASCII codes, until:

- the *i* th character in the first string is reached, and it's different from the corresponding *i* th character from the second string;
- the end of one of the compared strings, or the end of both strings is reached.

The comparison is done using the function **strcmp** having the prototype:

int strcmp(const char *string1, const char * string2);

This function returns:

- a negative value if the string having the address **string1** is less than the string having the address **string2**;
- zero if the two strings are equal;
- a positive value if the string having the address **string1** is greater than the string having the address **string2**;

If only the first *n* characters from the two strings are to be compared, use the function **strncmp** having the prototype:

int strncmp (const char * string1, const char * string2, unsigned n);

If the lowercase and uppercase letters are considered identical, then use the corresponding functions **(stricmp=strcasecmp, strnicmp= strcasecmp)**:

int stricmp (const char *sir1, const char *sir2);
int strnicmp (const char *sir1, const char *sir2, unsigned n);

Example:

```
/* Program L9Ex4.c */

/* The usage of strcmp function*/
#include <stdio.h>
#include <string.h>
#define prompt "\nPress a key!"

int main(void)
{
   char string1[100]="STRING OF CHARACTERS";
   char *string2="STRING of characters";
   int i;

   i=strcmp(string1, string2); /* i<0 , then  string1< string2 */
   printf("\ni=%d (%s%c%s)\n", i, string1, ((i<0)?'<':(i==0)?'=':'>'), string2);
   i=strncasecmp(string1, string2, 3); /* i=0 , then the first 3 characters from string1 and
string2 are equal */
   printf("\ni=%d (%s%c%s)\n", i, string1, ((i<0)?'<':(i==0)?'=':'>'), string2);
   i=strcasecmp(string1, string2); /* i=0, the two strings are equal */
   printf("\ni=%d (%s%c%s)\n", i, string1, ((i<0)?'<':(i==0)?'=':'>'), string2);
   i=strncasecmp(string1, "STRING of 22 characters", 6); /*i=0 */
   printf("\ni=%d (%s%c%s)\n", i, string1, ((i<0)?'<':(i==0)?'=':'>'), string2);
   printf(prompt);
   getchar();
   return 0;
}
```

3. Lab Tasks

3.1. Analyze and execute the examples provided above.

3.2. Write a function to extract, from a source string, a substring identified by the position in the source string and by the length expressed as a number of characters.

3.3. Write a function to insert a source character string in a destination character string, at a given position.

3.4. Write a function to delete a substring from a given character string, specifying the beginning position and the length of the substring.

3.5. Write a function to verify if a given string is a substring of another character string. If it is, specify the beginning position of the substring.

3.6. Write two functions, the first to convert an integer or real number into a string of characters, and the second to perform the inverse operation.

3.7. Write a program to read ***n*** strings of characters and display both the longest string and the biggest one as seen as an alphanumeric sequence.

3.8. Read from the standard input the author, the title and the publication year for a number of ***n*** books. Display the following:

a) the names of the authors in alphabetic order;
b) the names of the authors and the titles of their books in order of the publication year.

3.9. Read from the standard input the names of ***n*** kings and the corresponding year limits of their state leading periods. Display:

- the list of kings as they were read
- a list of kings in alphabetic order, completed with the number of years they ruled.
- A list kings in the ascending order of their ruling time.

3.10. Read from the keyboard strings of at most 80 characters, representing integer or real non-exponential numbers, separated by spaces. Compute the sum of the real numbers and of the integer numbers of each string, except the incorrect values.

An incorrect value is a character string delimited by spaces, containing non-numeric characters, or having the length greater than 5.