**ICS 103 – Computer Programming in C**

**Lab 11: String functions**

**Objectives:** Practice how to use strings by studying:

* String literals
* String declarations and Initializations
* Passing strings to and returning strings from functions.
* String console I/O and string file I/O
* String standard functions
* Looping over a string

In C, a string is a character array terminated by a null character: **'\0'**. Each character in a string takes only 1 byte.

**1. String literals**

A string literal is a sequence of characters in double quotes "". The C compiler automatically add a null character at the end of a string literal.

Example of a string literal:  **"Hello!"**

You can break a long string literal into multiple string literals that may span multiple lines:

Example: The following three string literals are identical strings:

**"hello, dear"**

**"hello, "**

**"dear"**

**"hello, " "d" "ear"**

**2. String Declaration and Initialization**

Here are some examples of declaring C strings as arrays of char:

char s1[20]; // Character array - can hold a C string, but is not yet a valid C string

char s2[20] = { 'h', 'e', 'l', 'l', 'o', '\0' }; // Array initialization, must place ‘\0’ explicitly

char s3[20] = "hello"; // Shortcut array initialization, ‘\0’ is automatically inserted

char s4[20] = ""; // Empty or null C string of length 0, contains only ‘\0’

Strings can also be initialized without specifying the size as in:

char str[ ]="I like C";

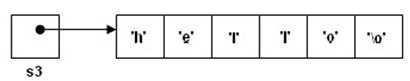
In this case, sufficient storage is allocated including that for **NULL**

**Note:** If the string is shorter than the specified array size, the remaining elements of the array are initialized to '\0'.

It is also possible to declare a C string as a pointer to a char:

char\* s3 = "hello";

This creates an unnamed character array just large enough to hold the string (including the null character) and places the address of the first element of the array in the char pointer **s3**



**String assignment**

A character array (including a C string) cannot have a new value assigned to it after it is declared.

char s1[20] = "This is a string";

char s2[20];

s1 = "Another string"; // error: invalid array assignment

s2 = s1; // error: invalid array assignment

The reason is that the name of an array is a constant pointer. However, if a pointer to char is used to point to a string, that pointer can be assigned to another string:

char\* s3 = "hello";

s3 = "student";

**3. Passing strings to and returning strings from functions**

Regardless of how a C string is declared, when you pass the string to a function or return it from a function, the data type of the string can be specified as either **char[ ]** (array of char) or **char\*** (pointer to char). In both cases, the string is passed or returned by address (i.e, call by reference). It is not necessary to pass the length of a string to a function; because that length can be calculated by the string **strlen** function.

**4. String I/O**

The prototypes of string I/O functions are in the **stdio.h** header file.

In the following table, assume the following declarations:

char str[128];

FILE inptr = fopen("input.txt", "r");

FILE outptr = fopen("output.txt", "w");

|  |  |
| --- | --- |
| **String Console I/O function** | **String File I/O function** |
| **scanf("%s", str);**  scanf that uses %s skips over leading white space. It stops reading when it encounters embedded or ending white space. The string is then null terminated. No ending white space is read.    If scanf encounters '\n', the new line character IS NOT INCLUDED in the string and it remains in the input buffer **stdin**.  **Note:** The name of a string is a pointer; don’t include **&** when reading strings. The following is invalid:  **scanf("%s", &str);** | **fscanf(inptr, "%s", str);**  fscanf behaves like scanf. It skips over leading white space (with respect to where it starts to read) and stops reading when it encounters embedded or ending white space  fscanf returns **EOF** on end of file |
| **scanf("%7s", str);**  reads a maximum of 7 characters in **str**. |  |
| **printf("%20s", str);**  prints a string **str** right justfied in a total width of 20 characters.  **printf("%-20s", str);**  prints a string **str** left justfied in a total width of 20 characters.  Example: The output of  **printf("%20s\n", "My KFUPM");**  **printf("%-20s\n", "My KFUPM");**  is: | **fprintf(outptr, "%20s", str);**  prints a string **str** right justfied in a total width of 20 characters.  **fprintf(outptr, "%-20s", str);**  prints a string **str** left justfied in a total width of 20 characters |
| **printf(str)**  prints a string str without using format specifier %s | **fprintf(outptr, str);**  prints a string str without using format specifier %s |
| **gets(str);**  **gets** reads a string, that may include embedded white space, from the keyboard until **'\n'** is encountered. The string is then null terminated and **'\n'** IS NOT INCLUDED in the string. Leading white space is not skipped. The new line character DOES NOT REMAIN in the input buffer **stdin** | **fgets(str, n, inptr);**  **fgets** reads a string from the specified input file  until either **n - 1** characters have been read or **'\n'** is encountered. If **'\n'** is encountered it is INCLUDED in the string. The string is then null terminated.    The string must be big enough to include both the **'\n'** , if any, and the terminating **'\0'**.  **fgets** returns **NULL** on end of file |

|  |  |
| --- | --- |
| **puts(str);**  **puts** outputs a string on the monitor, WITHOUT MODIFYING THE STRING, it then generates a new line | **fputs(str, outptr);**  **fputs** prints a string to the specified output file WITHOUT generating new line. |

**Example:** The following program reads one line at a time from **inFile.txt** and writes that line to the file **outFile.txt**:

**#include <stdlib.h>**

**#include <string.h>**

**int main(void){**

**char str1[80];**

**FILE \*input, \*output;**

**if((input = fopen("inFile.txt", "r")) == NULL){**

**printf("Error in opening inFile.txt . . . ");**

**exit(1);**

**}**

**output = fopen("outFile.txt", "w");**

**while(fgets(str1, 80, input)!= NULL)**

**fputs(str1, output);**

**fclose(input);**

**fclose(output);**

**printf("Output written to outFile.txt\n");**

**return 0;**

**}**

**Example**: The program below reads a text-file of the form shown below and writes to an output file the average score of each student and the section average:

|  |
| --- |
| RASHID MUHAMMAD 9000010 50.0 80.5 67.5  AHMAD ZUBEIR 9000011 90.5 95.0 88.5  QASIM OMAR 9000012 45.0 50.0 60.0  MUSTAFA ABDALLAH 9000013 70.5 80.6 85.4  AMIN HASSAN 9000014 70.5 70.5 45.4  SAID ALI 9000015 80.5 81.6 65.5  ABDULKAREEM YUSUF 9000016 73.5 82.4 85.2  OMAR ABDALLAH 9000017 60.5 70.6 19.5  AMMAR RAJAB 9000018 70.5 85.3 35.7  MUHAMMAD SAID 9000019 92.5 89.5 46.9  ZAKARIAH TALHA 9000020 70.5 60.5 84.5  ABDULMAJED ATHMAN 9000021 45.5 80.9 95.6  SALEEM MUHAMMAD 9000022 70.5 87.3 80.2  ZULFIKAR KHAN 9000023 50.5 23.4 83.8  ABDULLATIF UBAIDA 9000024 70.5 90.2 75.6  BASHEER HUSSEIN 9000025 60.5 80.6 55.3  HASHEEM ABUBAKAR 9000026 70.5 70.5 25.7  ABDULRAZZAK MUNIR 9000027 80.5 90.6 77.5  AMJAAD SHAFEEQ 9000029 70.5 82.5 66.5  YAHYA MUHSIN 9000030 90.5 83.7 85.4 |

#include <stdio.h>

#include <stdlib.h>

#define NUMQUIZES 3

int main(void){

double grade, studentTotal, studentAverage, sectionTotal = 0.0, sectionAverage;

int m, n , studentID , totalNumberOfGrades = 0;

char name1[81], name2[81];

FILE \*quizInput, \*quizOutput;

if((quizInput = fopen("QuizGrades.txt", "r")) == NULL){

printf("Error in opening QuizGrades.txt\n");

exit(1);

}

quizOutput = fopen("QuizResults.txt", "w");

fprintf(quizOutput, "Name\t\t\tID#\t\tAverage\n\n");

while(fscanf(quizInput, "%s%s%d", name1, name2, &studentID) != EOF) {

studentTotal = 0.0;

for(n = 1; n <= NUMQUIZES; n++){

fscanf(quizInput, "%lf", &grade);

studentTotal += grade;

}

studentAverage = studentTotal / NUMQUIZES;

fprintf(quizOutput, "%-11s %-11s %d\t\t%.2f\n", name1, name2, studentID, studentAverage);

sectionTotal += studentTotal;

totalNumberOfGrades += NUMQUIZES;

}

sectionAverage = sectionTotal / totalNumberOfGrades;

fprintf(quizOutput, "\n\nSection Average = %.2f", sectionAverage);

fclose(quizInput);

fclose(quizOutput);

printf("Results written to QuizResults.txt\n");

return 0;

}

**5. Standard string functions:**

The prototypes of standard string functions are in the header file **string.h .** Some of these functions are:

|  |  |  |
| --- | --- | --- |
| Function | Description | Comment |
| **strlen(string1)** | returns the length of **string1** excluding the terminating NULL character |  |
| **strcat(string1, string2)** | appends **strings2** to **string1** | **string1** must be a string variable.If it is a constant string a run-time error occurs. |
| **strcpy(string1, string2)** | replaces **string1** with **string2** | **string1** must be a string variable.If it is a constant string a run-time error occurs. |
| **strcmp(string1, string2)** | compares string1 and string2 with case-sensitivity and returns:  **Negative integer** if string1 < string2  **0** if string1 = = string2  **positive integer** if string1 > string2. |  |
| **strchr(str, ch)** | searches for the occurrence of character **ch** in the string **str** | Returns a pointer to the first occurrence of **ch** in **str**, if the search is successful; otherwise NULL is returned\* |
| **strstr (string1, string2)** | searches for the occurrence of **string2** in **string1** | Returns a pointer to the first character of the first occurrence of **string1** in **string2**, if the search is successful; otherwise NULL is returned\* |
| **strtok(str, delimiterSting)** | Uses multiple calls to split string **str** into parts (tokens) based on the delimeter characters in the **delimiterString** .The string to be tokenized and the **delimiterString** are passed in the first call; subsequent calls are of the form:  **strtok(NULL, delimiterString)** | Returns NULL if there are no more tokens; otherwise returns a pointer of type **char\*** to the next token.  Each call, modifies the tokenized string is modified such that by replacing the current delimeter by '\0'  Note:   * **str** must be a string variable.If it is a constant string a run-time error occurs. * the delimiters in **strtok** can be different for one call to another. |

\* Note: The NULL , '\0', has ACII value 0, and a non-NULL pointer has a non-zero address ; thus the functions **strchr** and **strstr** can be used as boolean (logical) functions. Example:

**if(strstr(string1, string2))**

**printf("%s is a substring of %s\n", string2, string1);**

**else**

**printf("%s is not a substring of %s\n", string2, string1);**

**Example: String comparison**

# #include <string.h>

# #include <stdio.h>

# int main(void){

# char s1[81], s2[81];

printf(**"**Enter string1 and string2 on separate lines: \n**"**);

# gets(s1);

# gets(s2);

# int result = strcmp(s1, s2);

# if (result == 0)

# printf("The strings are equal\n");

else if(result > 0)

# printf("string1 is greater than string2\n");

else

printf("string2 is greater than string1\n");

# return 0;

# }

Example: String Tokenization

#include <stdio.h>

#include <string.h>

int main(void){

const char string[] = "words separated by spaces -- and, punctuation!";

char cp[80];

const char delimiters[] = " .,;:!-";

char \*token;

strcpy(cp,string); /\* strtok modifies the tokenized string, to

preserve that string make writable copy. \*/

token = strtok (cp, delimiters);

while(token != NULL){

printf("%s\n", token);

token = strtok (NULL, delimiters);

}

printf("Last returned token is %s\n", token);

return 0;

}

**6. Looping over a string**

To manipulate each character of a string **str**, a loop similar to any one of the following two loops may be used:

|  |  |
| --- | --- |
| **int k;**  **for(k = 0; k < strlen(str); k++){**  **// Manipulate str[k]**  **}** | **int k = 0;**  **while(str[k] != '\0'){**  **// Manipulate str[k]**  **k++;**  **}** |

**Example:** Count the number of lowercase alphabetic letters in a sting **str**:

|  |  |
| --- | --- |
| **int k, count = 0;**  **for(k = 0; k < strlen(str); k++){**  **if(str[k] >= 'a' && str[k] <= 'z')**  **count++;**  **}**  **// . . .** | **int k = 0, count = 0;**  **while(str[k] != '\0'){**  **if(str[k] >= 'a' && str[k] <= 'z')**  **count++;**  **k++;**  **}**  **// . . .** |

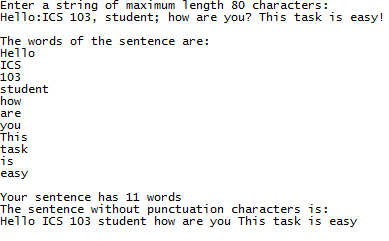
**Lab Tasks:**

#### Task 1:

Write a C program that prompts for and reads a sentence of maximum size 80 from the user. The program then:

* Displays the words in the sentence on separate lines and also displays their count.
* Places the words in a new string without the punctuation characters and in which each word is separated from the next word by a single space.
* Displays the new string.

Sample program run:



**Hint** : Use **strtok** and **strcat** functions. The delimiter string contains in addition to space, the punctuation characters: **, ; : ? and !**

#### Task 2:

A text-file **input.txt** contains English words that are separated by white-space characters and not by any punctuation character. Write a program that prompts for and reads a character, it then reads the textfile and print on an output file **output.txt** a table of the frequency of the character in each word of the text-file. Assume the maximum word length is 30 characters. The frequency of the character in each word must be computed in a function **getFrequency**. This function must have no scanf and printf calls.

Sample input file:

|  |
| --- |
| Dhahran is about four hundred  kilometers from Riyadh and about  one thousand two hundred kilometers  from Jeddah |

Sample output file if **a** is the character read from the user:

|  |
| --- |
| word frequency of a  Dhahran 2  is 0  about 1  four 0  hundred 0  kilometers 0  from 0  Riyadh 1  and 1  about 1  one 0  thousand 1  two 0  hundred 0  kilometers 0  from 0  Jeddah 1 |

Part of the program is written below. You need to complete it. Use fscanf to read the words from the input file since they are separated by white space characters only.

#include <stdio.h>

#include <stdlib.h>

int getFrequency(char word[], char ch);

int main(void){

FILE \*inptr, \*outptr;

inptr = fopen("input.txt", "r");

if(inptr == NULL){

printf("Error in opening input.txt\n");

exit(1);

}

outptr = fopen("output.txt", "w");

char ch, word[30];

int count;

**Task 3:**

A word is a palindrome if it reads the same forwards and backwords. For example, each of the following words is a palindrome: rotator, rotor, radar, level, noon, tenet, SMS, dad, madam, 202, pop. Whereas each of the following words is not a palindrome: school, apple, student, 103.

Write a C program that prompts for and reads a word, in which all the letters are of the same case, it then passes this word to **isPalindrome** boolean function (a function that return true (1) or false (0)). The function determines whether the word is a palindrome or not. Your main function then prints an appropriate message. Assume that a word does not have more than10 characters.

Sample program runs:

|  |
| --- |
|  |
|  |