CS 332/532 – 1G- Systems Programming Lab 3

Objectives

The objective of this lab is to introduce you to dynamic memory allocation and strings in C programming.

1. Go over the solution for Lab 02.
2. Illustrate how to use dynamic memory allocation and strings in C.

Lab Assignment #2

Modify the insertion sort program developed in Lab #2 such that the program now sorts strings (array of characters) instead of floats and uses dynamic memory allocation to allocate memory based on the number of elements in the array as well as the length of individual strings.

You can use scanf function to read the strings, determine the length of the string, and then allocate appropriate amount of memory.

Also use separate functions to read the strings, sort the strings, and display the sorted strings.

Use appropriate string functions provided by the C library. A list of functions and details about each function can be found at: https://en.cppreference.com/w/c/string/byte

Compile and test the program and upload the C source code in the lab assignment submission section.

Assignment Submission

The assignment submission in Canvas is required. No late submissions will be accepted.

Submission Checklist:

- Upload the C source file (.c file) to Canvas as part of this lab submission. Submissions through the Canvas "Comments" will not be accepted.
- Upload a README.md file which should include:
    - Instructions on how to compile your C source file into an executable.
    - How to run the executable program.
    - Any citation documentation.
    - A link to your GitHub repository.

Please do not upload executables or object files. Independent Completion Forms are not required for labs.

Lab Workbook

Task 1:

In this task first we will go over the solution of Lab 02 and compare the C and Java code.

- Download the C program `(insertionsort.c)`, and the Java program `(InsertionSort.java)`.
- Now understand the C program and compare it with the given Java code.

Task 2:

Create a function to perform the insertion sort. Now the program will perform the following steps:

1. Prompt the user to enter the number of array elements (say, N).
2. Read the number of elements (N).
3. Use dynamic memory allocation to allocate an array of N single precision floating-point elements (C type float).
4. Read the N single precision floating-points elements to the allocated array.
5. Invoke a function to sort the array using insertion sort (the insertion sort function should take the array reference and number of elements as the arguments and sort the array in-place). The insertion sort function should have the return type as void, i.e., it does not return anything.
6. Print the sorted array.

Code snippet:

1. Function Declaration

```
void displayArray(float *arr, int size);
void sortArray(float *arr, int size);
```

2. Function definition

```
void displayArray(float *arr, int size){
    printf("[");
    for (int i=0; i<size-1; i++) {
            printf("%f, ",arr[i]);
    }
    printf("%f]\n", arr[size-1]);
}
```

3. Function call/invocation

```
displayArray(arr, N);
sortArray(arr, N);
```

4. Dynamic memory allocation for array

```
float *arr = (float*) malloc(N * sizeof(float));
```

Task 3:

In C, there is no primitive datatype called string. Strings are implemented as an array of characters typically terminated with the null character (\0). For example:

```
char hello[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
// OR
char hello[] = "Hello\0";
```

Note: When you initialize an array of characters, it is a good idea to add the **null character - `\0'** as the terminating character.

Now let's see few helpful functions for character array manipulation. For this action you need to include the header file called "string.h".

- strlen: To get the length of string.
- strcpy: To copy one string to another.
- strcat: To concatenate one string to another.
- strcmp: To compare one string to another string. This will return the integer value based on the two strings compared. The return value is 0 if the two strings are equal, greater than 0 if first string is greater than the second string, and less than 0 if the first string is less than the second string and the comparison is performed using unsigned characters.
- strchr: To search for the first occurrence of a character in the given string. This will return either a pointer to the first occurrence or NULL if not found.
- strstr: To search for the first occurrence of a sub-string in a given string. This will return either a pointer to the first occurrence of the sub-string or NULL if not found.

Now, let's some of these functions. Also, you can find out more about these functions by typing man <functionname>. For example:

```
man strcpy
```

Implement following examples for string manipulations:

Example 1:

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

int main(int argc, char** argv) {
    char hello[20] = {'h','e','l','l','o',' ','\0'};
    char name[] = "students!\0";
    strcat(hello, name);
    printf("%s\n",hello);

    return 0;
}
```

Note: the strcat function concatenate second string (name) into first string (hello), also notice that hello has one extra space before string termination to print "hello students!" with space between "hello" and "students!".

Example 2:

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

int main(int argc, char** argv) {
    char student1[] = "Alice\0";
    char student2[] = "Bob\0";
    char student3[] = "bob\0";
    char student4[] = "Alice\0";

    printf("strcmp(student1, student2) = %d\n", strcmp(student1, student2));
    printf("strcmp(student2, student1) = %d\n", strcmp(student2, student1));
    printf("strcmp(student2, student3) = %d\n", strcmp(student2, student3));
    printf("strcmp(student1, student4) = %d\n", strcmp(student1, student4));

    return 0;
}
```

Note: The strcmp function performs a character-by-character lexicographic comparison to compare the two strings.

Lab Assignment #3 is on page 1.