Operating System
COM301T

Programming Preparation Assignment

Lab - 0

By :

Sai Kaushik S

CED18I044

Question 1:

Develop an application (using C & Command Line Arguments) for:
i) Simulate the behavior of cp command in linux. (you should not invoke cp command from your C source!). Also your application should validate right usage; if less or more number of arguments are passed to the executable the program should prompt a message to the user. File read and write function calls are allowed. Rename your executable as mycopy.
Example usage could be ./mycopy fact.c factcopy.c

Logic or commands used:

The destination can either be a file or a directory.

1. 1If the destination is a directory:
   A new file is created in the directory with a file name same as the source file using fopen, that opens an existing file or creates a file at a specific path.
2. If the destination is a file or after the file is created from the above step:
   Using the read and write functions, we copy the contents of the source file to the destination file. Close the source and destination files.

C code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <fcntl.h>
#include <errno.h>
#include <sys/stat.h>
#include <string.h>
#include <unistd.h>

// Size of the contents that are copied
#define BUFF_SIZE 1024

// Name of the file if the destination is a directory
char fileName[200];

void displayHelp();

// A function to revert the string
void reverse(char *x, int begin, int end){
   char c;
   // If the end is before begin, return null
   if (begin >= end)
```

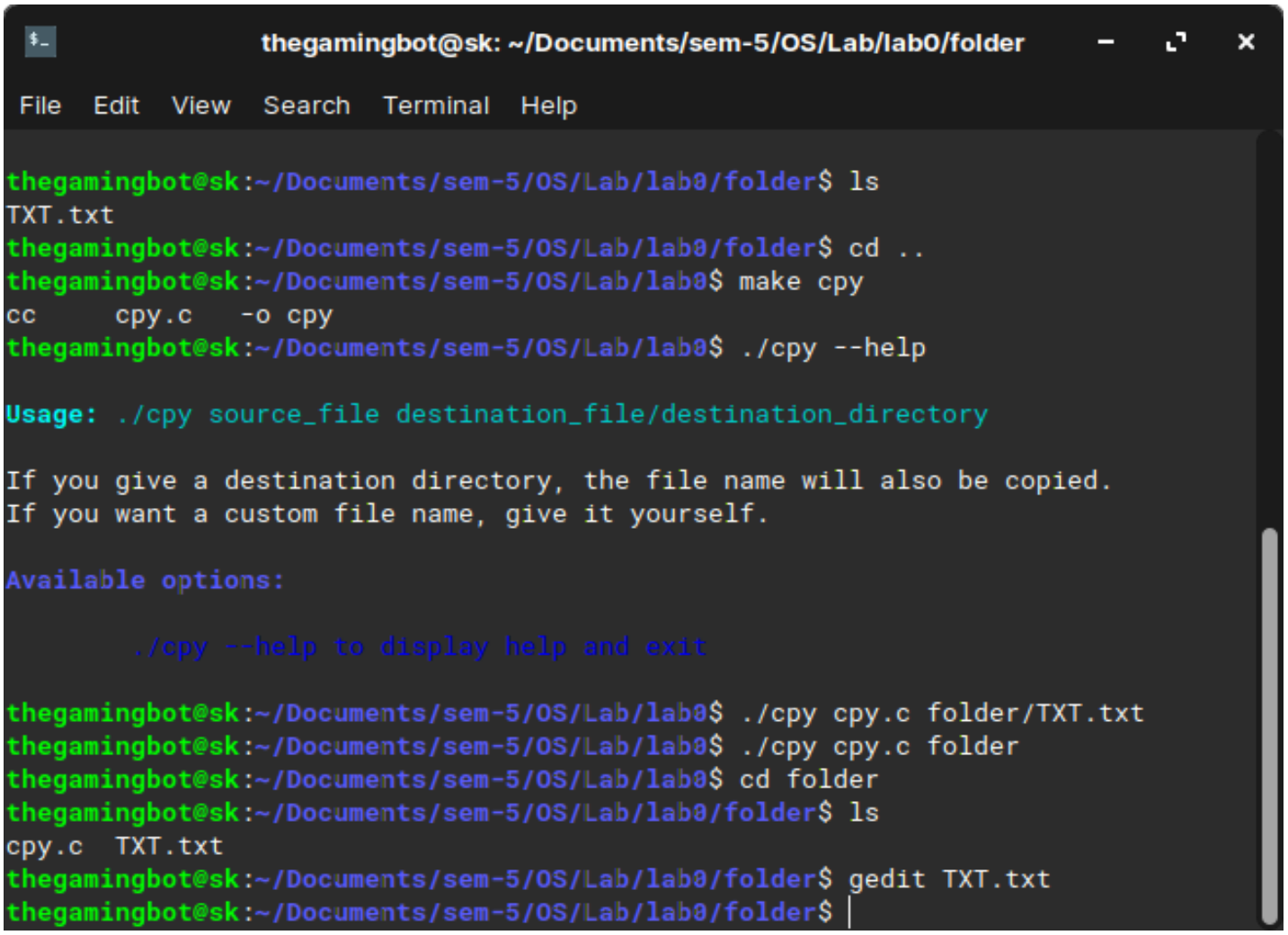
```
        return;
    // Swap the characters
    c = *(x+begin);
    *(x+begin) = *(x+end);
    *(x+end) = c;
    // Recursive call
    reverse(x, ++begin, --end);
}

// A function to get the file name, if the destination is a directory
char* getFileName(char *path){
    // Last index of the path
    int pathLen = strlen(path) - 1, i = 0;
    // While the index is greater than -1
    while(pathLen >= 0){
        // If a / is encountered break
        if (path[pathLen] == '/')
            break;
        // Copy the filename part of the path
        fileName[i++] = path[pathLen--];
    }
    // Revert the filename
    reverse(fileName, 0, strlen(path) - 1);
    //return the filename
    return fileName;
}

// Main driver program with arguments
int main(int argc, char* argv[]){
    int src, dest, bytesRead;
    char *buff[BUFF_SIZE];

    int status;
    struct stat st_buf;

    if(strncmp(argv[1], "--help", 6) == 0){
        // Print the instructions on how to use the command
        displayHelp();
```

```
        exit(EXIT_SUCCESS);
    }

    // If the length of the argument is not 3
    if(argc != 3){
        // Print the instructions on how to use the command
        printf("\033[1;31m");
        printf("Error: ");
        printf("\033[0;31m");
        printf("There must be one command, one source file and one
destination file.\n");
        displayHelp();
        exit(EXIT_FAILURE);
    }

    // Open the source file with read only permissions
    src = open(argv[1],O_RDONLY);
    // If open is unsuccessful, exit with a failure
    if(src == -1){
        printf("\nError opening file %s \nError number: %d\n", argv[1],
errno);
        exit(EXIT_FAILURE);
    }

    // Get the status of the second file/directory
    status = stat(argv[2], &st_buf);

    // If the second argument is a directory
    if (S_ISDIR (st_buf.st_mode)) {
        // Concat the filename to the path
        strcat(argv[2], "/");
        strcat(argv[2], getFileName(argv[1]));
        // Create a file at the destination path
        FILE* fp = fopen(argv[2], "w");
        // Close the file
        fclose(fp);
    }
```

```
    // Open destination file with respective flags & modes
    // O_CREAT & O_TRUNC is to truncate existing file or create a new file
    // S_IXXXX are file permissions for the user,groups & others
    dest = open(argv[2],O_WRONLY | O_CREAT | O_TRUNC, S_IRUSR | S_IWUSR |
S_IRGRP | S_IWGRP | S_IROTH | S_IWOTH);

    //If open is unsuccessful, exit with a failure
    if(dest == -1)  {
        printf("\nError opening file %s \nError number: %d\n", argv[2],
errno);
        exit(EXIT_FAILURE);
    }

    // While read is possible in src file
    while((bytesRead = read(src,buff,BUFF_SIZE)) > 0){
        // if the full string is not copied, exit with an error
        if(write(dest,buff,bytesRead) != bytesRead)
            printf("\nError in writing data to %s\n", argv[2]);
    }

    // If read in src file is possible
    if(bytesRead == -1)
        printf("\nError in reading data from %s\n", argv[1]);

    // If there is an error in closing src file
    if(close(src) == -1)
        printf("\nError in closing the source file\n");

    // If there is an error in closing dest file
    if(close(dest) == -1)
        printf("\nError in closing destination file\n");

    // Exit with success
    exit(EXIT_SUCCESS);
}

// A function to display help on usage of the command
void displayHelp(){
```

```
    printf("\033[1;36m");
    printf("\nUsage: ");
    printf("\033[0;36m");
    printf("./cpy source_file destination_file/destination_directory\n");
    printf("\033[0m");
    printf("\nIf you give a destination directory, the file name will also
be copied.\n");
    printf("If you want a custom file name, give it yourself.\n");
    printf("\033[1;34m");
    printf("\nAvailable options: ");
    printf("\033[0;34m");
    printf("\n\n\t./cpy --help to display help and exit\n\n");
}
```

Output:



[TXT.txt before ./cpy]
[TXT.txt after ./cpy]

ii) Extra Credits Qn – Extend the above application / develop an application to simulate the behavior of rm command in linux. rm command invoke from Source is not allowed! Other features as in earlier applications to be supported.

Logic or commands used:

./rmv has -r option for recursive removal of directories.
Regular ./rmv uses the remove function to delete the files.

./rmv -r opens up the directory, gets the contents of it and deletes them. The function gets called recursively if there are any directories within directories.

- readdir function returns dirent structures of files in the current working directory.
- struct dirent contains the file name, type of the file, length of the file, offset to the next dirent, inode number.
- S_ISDIR() checks whether the current source is a directory or not.

C code:

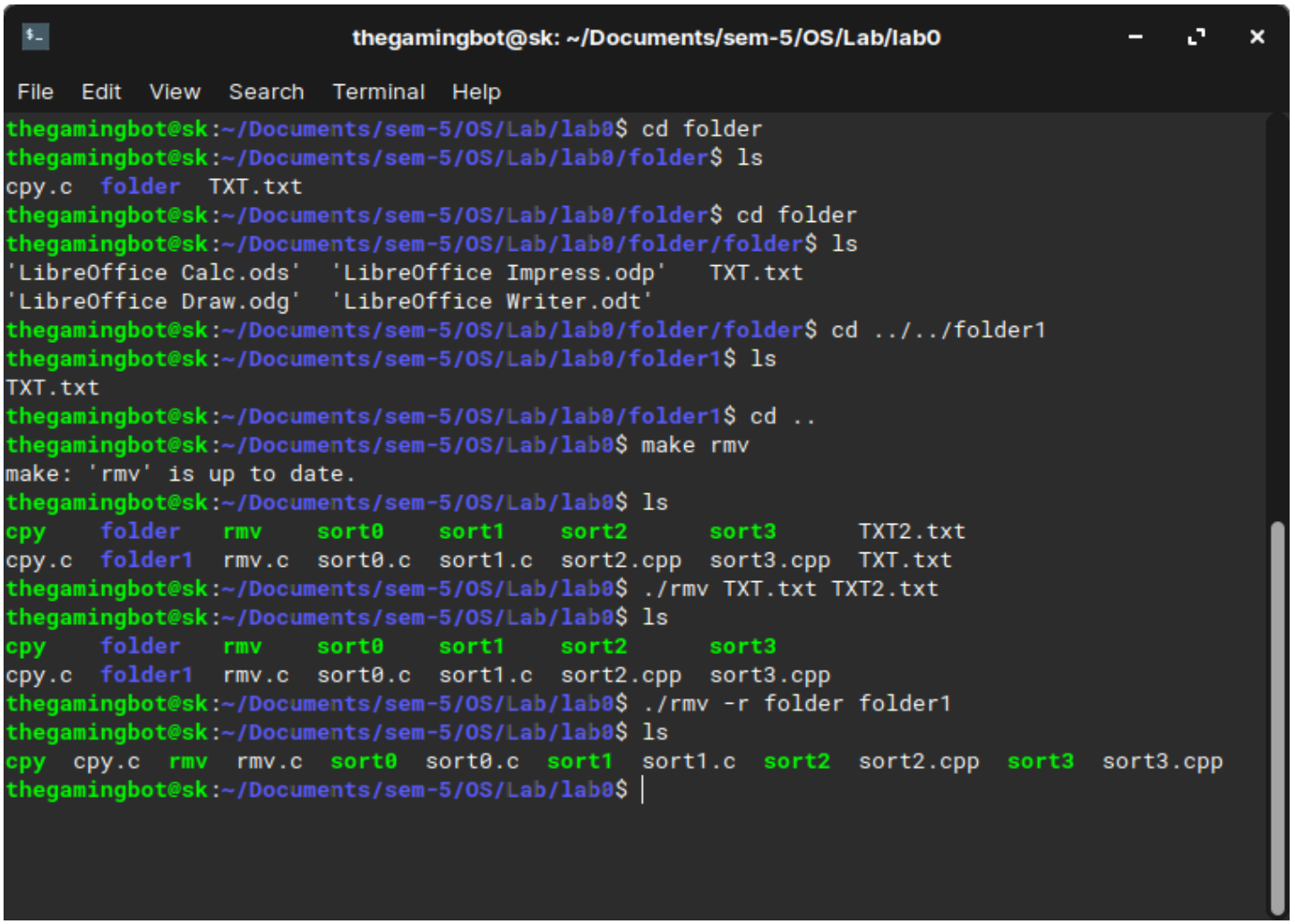
```c
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/stat.h>
#include <dirent.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define MAX 1024

void displayHelp();

// A function to recursively remove directories
int removeDir(char* path){
   struct dirent *de;
   char fname[300];
   // Open the directory
   DIR *dr = opendir(path);
   // If the directory does not exist
   if(dr == NULL){
```

```
        printf("No file or directory found\n");
        exit(EXIT_FAILURE);
    }
    // Read the directory and loop through it
    while((de = readdir(dr)) != NULL){
        int ret = -1;
        struct stat statbuf;
        // Concat the path to the file name
        sprintf(fname,"%s/%s",path,de->d_name);
        if (!strcmp(de->d_name, ".") || !strcmp(de->d_name, ".."))
            continue;
        // If the directory or file exists
        if(!stat(fname, &statbuf)){
            // If the content is a directory
            if(S_ISDIR(statbuf.st_mode)){
                // Unlink the directory
                ret = unlinkat(dirfd(dr),fname,AT_REMOVEDIR);
                // Recursively call the removeDir function
                if(ret != 0){
                    removeDir(fname);
                    ret = unlinkat(dirfd(dr),fname,AT_REMOVEDIR);
                }
            }
            // If the content is a file
            else
                // Unlink the file
                unlink(fname);
        }
    }
    // Close the directory
    closedir(dr);
    // Remove the directory
    rmdir(path);
}

// Main driver program with arguments
int main(int argc, char* argv[]){
    int status;
```

```
    struct stat st_buf;

    // If the user asks for help
    if(strncmp(argv[1], "--help", 6) == 0){
        // Print the instructions on how to use the command
        displayHelp();
        exit(EXIT_SUCCESS);
    }

    // If there are less arguments than required
    if(argc < 2){
        // Print the error and usage
        printf("\033[1;31m");
        printf("Error: ");
        printf("\033[0;31m");
        printf("There must be one command, and at least one file/directory
to be removed.\n");
        printf("\033[1;36m");
        displayHelp();
        exit(EXIT_FAILURE);
    }

    // Option -r recursive
    if(strncmp(argv[1], "-r", 2) == 0){
        // Loop through all the argv
        for(int i = 2; i < argc; i++){
            // Get the status of the content
            status = stat (argv[i], &st_buf);
            // If the content is a directory
            if (S_ISDIR (st_buf.st_mode)){
                // Call the removeDir function
                char path[MAX];
                getcwd(path, MAX);
                char* x = strcat(path, "/");
                x = strcat(x, argv[i]);
                removeDir(x);
            }
            // If the content is a file
```

```
            else
                // Remove the file
                remove(argv[i]);
        }
        exit(EXIT_SUCCESS);
    }
    // For normal rmv, loop through argv
    for (int i = 1; i < argc; i++){
        // remove the file
        remove(argv[i]);
    }
    exit(EXIT_SUCCESS);
}

// A function to display help on usage of the command
void displayHelp(){
    printf("\033[1;36m");
    printf("\nUsage: ");
    printf("\033[0;36m");
    printf("./rmv source_files\n");
    printf("\033[1;34m");
    printf("\nAvailable options: ");
    printf("\033[0;34m");
    printf("\n\n\t./rmv --help to display help and exit\n");
    printf("\t./rmv --r directory_paths to recursively remove the
directories.\n\n");
}
```

Output:


thegamingbot@sk: ~/Documents/sem-5/OS/Lab/lab0
File Edit View Search Terminal Help
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ cd folder
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder$ ls
cpy.c folder TXT.txt
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder$ cd folder
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder/folder$ ls
'LibreOffice Calc.ods' 'LibreOffice Impress.odp' TXT.txt
'LibreOffice Draw.odg' 'LibreOffice Writer.odt'
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder/folder$ cd ../../folder1
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder1$ ls
TXT.txt
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0/folder1$ cd ..
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ make rmv
make: 'rmv' is up to date.
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ ls
cpy folder rmv sort0 sort1 sort2 sort3 TXT2.txt
cpy.c folder1 rmv.c sort0.c sort1.c sort2.cpp sort3.cpp TXT.txt
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ ./rmv TXT.txt TXT2.txt
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ ls
cpy folder rmv sort0 sort1 sort2 sort3
cpy.c folder1 rmv.c sort0.c sort1.c sort2.cpp sort3.cpp
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ ./rmv -r folder folder1
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$ ls
cpy cpy.c rmv rmv.c sort0 sort0.c sort1 sort1.c sort2 sort2.cpp sort3 sort3.cpp
thegamingbot@sk:~/Documents/sem-5/OS/Lab/lab0$

Question 2:

Develop an application (using C & Command Line Arguments) for:
i) Sort an array of varying numbers of integers in ascending or descending order. The array and array size are passed at the command line. Invoke of linux command sort is not allowed. Use of atoi or itoa fns is allowed (need you should read online resources!). Let your program handle invalid usages as well!

Eg. ./mysort 5 1 50 40 30 20 1 here 5 is array size and 1 means ascending order sort and the rest of the input is the array to be sorted. Your code should handle descending order sort as well.

Logic or commands used:

Bubble sort is being used for sorting the elements.
The command line arguments are being converted to integers using the atoi() function.
To check if the given numbers are integers, the argv[] are converted to integer using atoi and are converted back to string using sprintf. Since 1.5 on atoi gives 1 and upon string comparison, if both don't match, it is not an integer.

C code:

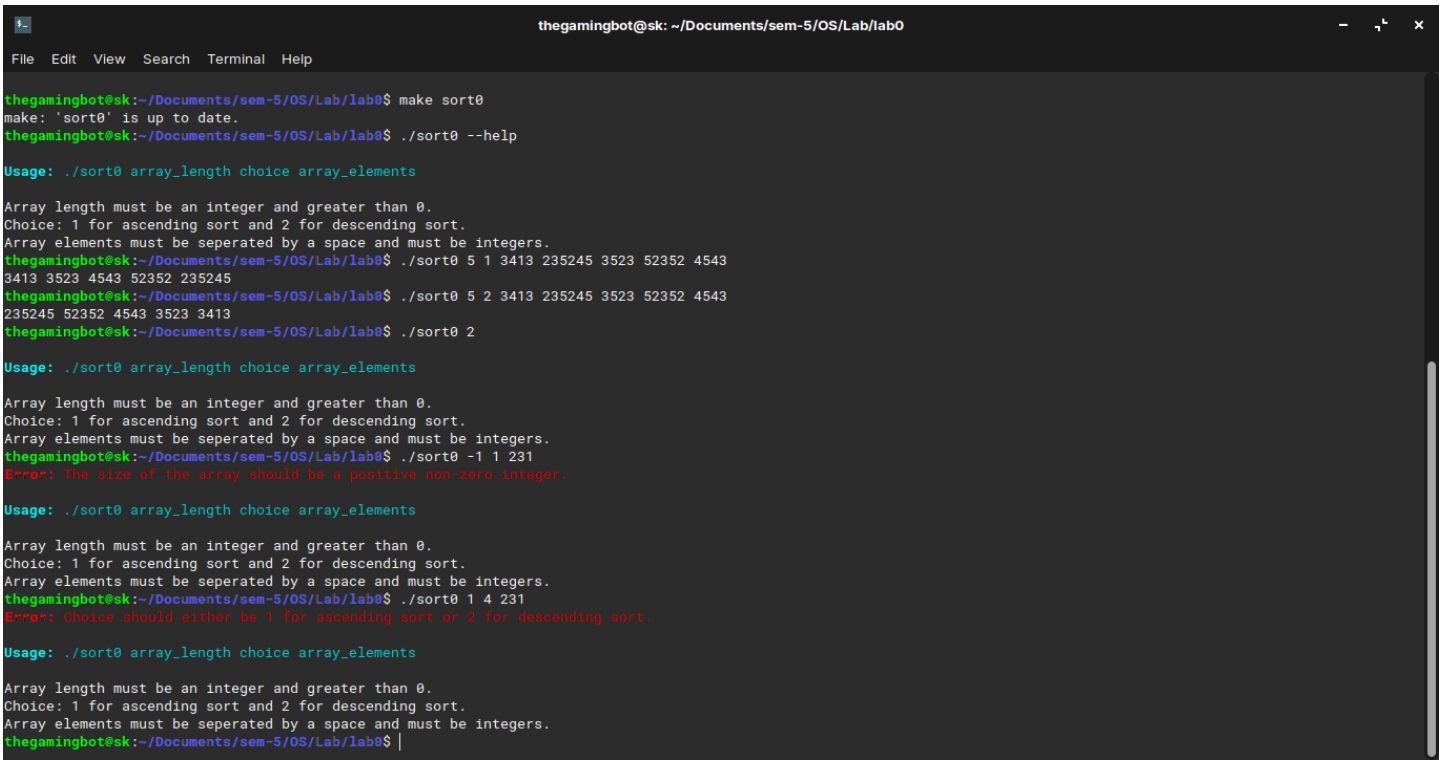
```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#define MAX 1024

void bubbleSortAsc(int* arr, int n);
void bubbleSortDesc(int* arr, int n);
void swap(int* a, int* b);
void print(int* arr, int n);
void displayHelp();

// Main driver program with arguments
int main(int argc, char* argv[]){
   // If the user ask for help on usage of the command
   if(strncmp(argv[1], "--help", 6) == 0 || argc < 4){
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_SUCCESS);
   }
```

```
   // Convert the size from string to int
   int size = atoi(argv[1]);
   char qw[] = ".";

    // If the size is negative or if there exists a floating point in the
size
   if(size <= 0 || strstr(argv[1], qw)){
       // Print an error
       printf("\033[1;31m");
       printf("Error: ");
       printf("\033[0;31m");
           printf("The size of the array should be a positive non-zero
integer.\n");
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_FAILURE);
   }

   // Convert the choice from string to int
   int choice = atoi(argv[2]);

   // If the choice is less than 1 or greater than two or if there exists
a floating point in the choice
   if(choice < 1 || choice > 2 || strstr(argv[2], qw)){
       // Print an error
       printf("\033[1;31m");
       printf("Error: ");
       printf("\033[0;31m");
          printf("Choice should either be 1 for ascending sort or 2 for
descending sort.\n");
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_FAILURE);
   }

   int arr[size];
   char a[MAX];
```

```
    // Loop through the test of the arguments
    for(int i = 3; i < argc; i++){
        // Convert the numbers from string to int
        arr[i - 3] = atoi(argv[i]);
        // Convert the numbers back to string from int
        sprintf(a, "%d", arr[i - 3]);
        // If the original arguments and converted arguments do not match
        if(strcmp(argv[i], a) != 0){
            printf("Enter only integers.\n");
            exit(EXIT_FAILURE);
        }
    }
    // If the choice is ascending
    if(choice == 1)
        bubbleSortAsc(arr, size);
    // If the choice is descending
    if(choice == 2)
        bubbleSortDesc(arr, size);
    // Print the sorted array
    print(arr, size);
    exit(EXIT_SUCCESS);
}

// A bubble sort function that sorts in ascending order
void bubbleSortAsc(int* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

// A bubble sort function that sorts in descending order
void bubbleSortDesc(int* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] < arr[j+1])
                swap(&arr[j], &arr[j+1]);
}
```

```
// A function that prints the contents of an int array
void print(int* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

// A function that swaps two integers
void swap(int* a, int* b){
   int c = *a;
   *a = *b;
   *b = c;
}

// A function to display help on usage of the command
void displayHelp(){
   printf("\033[1;36m");
   printf("\nUsage: ");
   printf("\033[0;36m");
   printf("./sort0 array_length choice array_elements\n");
   printf("\033[0m");
   printf("\nArray length must be an integer and greater than 0.\n");
   printf("Choice: 1 for ascending sort and 2 for descending sort.\n");
     printf("Array elements must be separated by a space and must be
integers.\n");
}
```

Output:



Extra Credits Qn: (I would advise everyone to try!)
Can you implement the above sorting (both ascending or descending) using only function internally for sorting logic (I mean bubble or insertion etc..)You should define the logic in your source code only once but the application should be able to handle both ascending or descending order sort!). Hint use function pointers!

Logic or commands used:

Bubble sort is being used for sorting the elements.
The command line arguments are being converted to integers using the atoi() function.
To check if the given numbers are integers, the argv[] are converted to integer using atoi and are converted back to string using sprintf. Since 1.5 on atoi gives 1 and upon string comparison, if both don't match, it is not an integer.
Function pointers are similar to variable pointers, but point to functions. Passing them as arguments to the bubble sort function allows us to call specific functions. If the user wants to sort in ascending order, pass the ascending function through the function call and use that function.

C code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX 1024
```

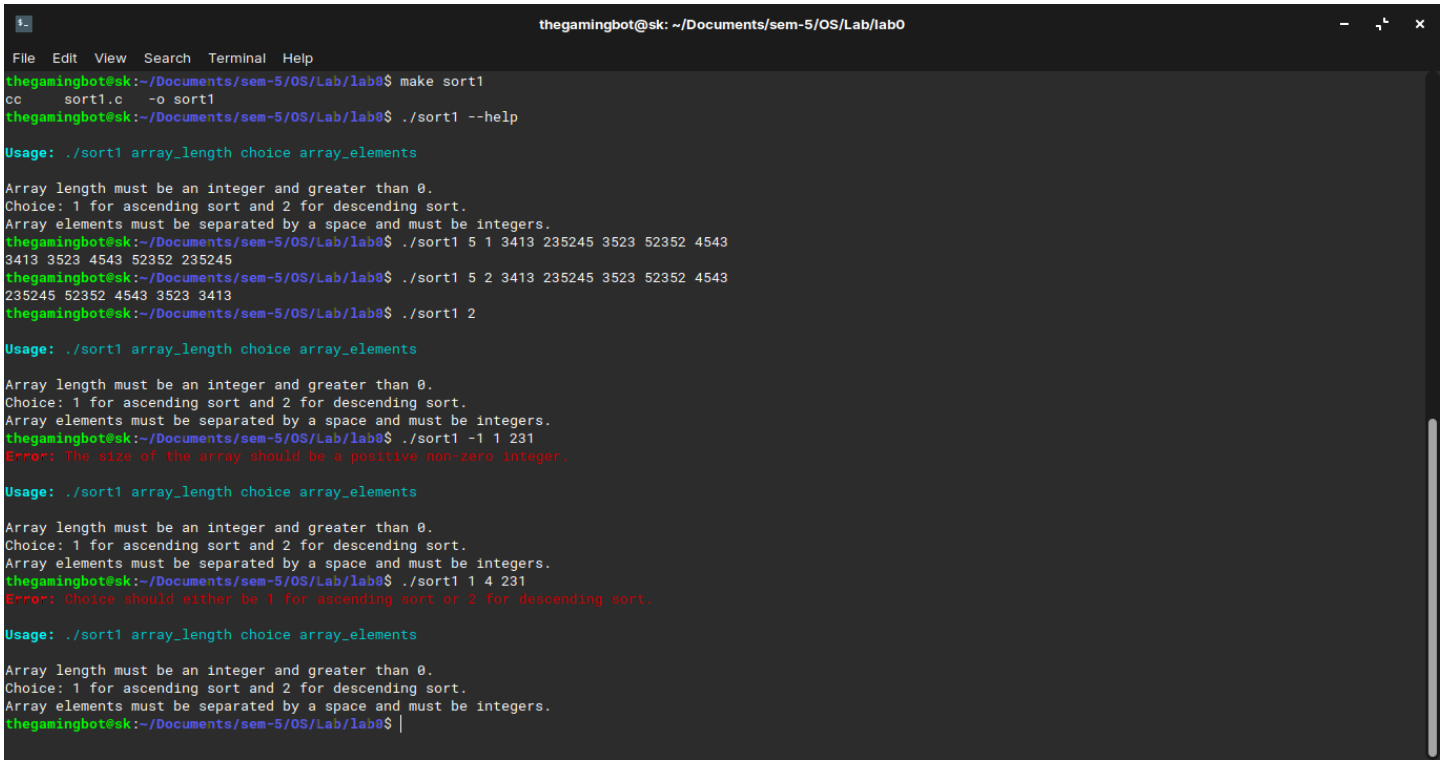
```
void bubbleSort(int* arr, int n, bool x, bool fun(const void*, const
void*));
void swap(int* a, int* b);
void print(int* arr, int n);
bool asc(const void* a, const void* b);
bool desc(const void* a, const void* b);
void displayHelp();

// Main driver program with arguments
int main(int argc, char* argv[]){
   // If the user ask for help on usage of the command
   if(strncmp(argv[1], "--help", 6) == 0 || argc < 4){
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_SUCCESS);
   }

   // Convert the size from string to int
   int size = atoi(argv[1]);
   char qw[1] = ".";

    // If the size is negative or if there exists a floating point in the
size
   if(size <= 0 || strstr(argv[1], qw)){
       // Print an error
       printf("\033[1;31m");
       printf("Error: ");
       printf("\033[0;31m");
           printf("The size of the array should be a positive non-zero
integer.\n");
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_FAILURE);
   }

   // Convert the choice from string to int
   int choice = atoi(argv[2]);
```

```
   // If the choice is less than 1 or greater than two or if there exists
a floating point in the choice
   if(choice < 1 || choice > 2 || strstr(argv[2], qw)){
       // Print an error
       printf("\033[1;31m");
       printf("Error: ");
       printf("\033[0;31m");
          printf("Choice should either be 1 for ascending sort or 2 for
descending sort.\n");
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_FAILURE);
   }

   int arr[size];
   char a[MAX];

   // Loop through the test of the arguments
   for(int i = 3; i < argc; i++){
       // Convert the numbers from string to int
       arr[i - 3] = atoi(argv[i]);
       // Convert the numbers back to string from int
       sprintf(a, "%d", arr[i - 3]);
       // If the original arguments and converted arguments do not match
       if(strcmp(argv[i], a) != 0){
           printf("Enter only integers.\n");
           exit(EXIT_FAILURE);
       }
   }
   // If the choice is ascending
   if(choice == 1)
       bubbleSort(arr, size, false, asc);
   // If the choice is descending
   if(choice == 2)
       bubbleSort(arr, size, true, desc);
   // Print the sorted array
   print(arr, size);
```

```
    exit(EXIT_SUCCESS);
}

// A bubble sort function with a function pointer input
void bubbleSort(int* arr, int n, bool x, bool fun(const void*, const
void*)){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (fun(&arr[j], &arr[j+1]))
                swap(&arr[j], &arr[j+1]);
}

// A function that prints the contents of an int array
void print(int* arr, int n){
    for(int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// A function to return the logic of ascending sort
bool asc(const void* a, const void* b){
    return *(int*)a > *(int*)b;
}

// A function to return the logic of descending sort
bool desc(const void* a, const void* b){
    return *(int*)a < *(int*)b;
}

// A function that swaps two integers
void swap(int* a, int* b){
    int c = *a;
    *a = *b;
    *b = c;
}

// A function to display help on usage of the command
void displayHelp(){
```

```
    printf("\033[1;36m");
    printf("\nUsage: ");
    printf("\033[0;36m");
    printf("./sort1 array_length choice array_elements\n");
    printf("\033[0m");
    printf("\nArray length must be an integer and greater than 0.\n");
    printf("Choice: 1 for ascending sort and 2 for descending sort.\n");
      printf("Array elements must be separated by a space and must be
integers.\n");
}
```

Output:



Question 3:

Develop an application (using C & Command Line Arguments) for:
a) Sorting an array of integers or floating point or characters passed at command line. Usage syntax you can follow a similar style as for the II question and also support validation logic in the code.
Read on function overloading – more than one function can carry the same name with different parameters!

Logic or commands used:

Bubble sort is being used for sorting the elements.
The command line arguments are being converted to integers using the atoi() function.
To distinguish integers, floating numbers and character, the below logic is used.
The spring can be either of length = 1 or length > 1

★ When the length is 1, the string can either be a character or an integer. Ex: 1, 2, 3 are both characters and integers. A, b,.. are characters too.
- 0 through 9 can be considered as integers, if all of the inputs are in this range.
- They are considered characters if any one of the input is a single length character, like a, A,..

★ When the length > 1, the string can be an integer or a floating number or a string.
- To check if the given numbers are integers, the argv[] are converted to integer using atoi and are converted back to string using sprintf. Upon string comparison, if both match, it is an integer.
- To check if the given numbers are floating numbers, the argv[] are converted to integer using atof and are converted back to string using sprintf. Upon string comparison, if both match, it is a float.
- Else, it is a string, give out an error.

Functions have the same same, same number of arguments but different data types. This feature is called function overloading.

C++ code:

```
#include <iostream>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include <sstream>
using namespace std;

void bubbleSortAsc(int* arr, int n);
void bubbleSortDesc(int* arr, int n);
void swap(int* a, int* b);
void print(int* arr, int n);
```

```
void bubbleSortAsc(float* arr, int n);
void bubbleSortDesc(float* arr, int n);
void swap(float* a, float* b);
void print(float* arr, int n);

void bubbleSortAsc(char* arr, int n);
void bubbleSortDesc(char* arr, int n);
void swap(char* a, char* b);
void print(char* arr, int n);

bool isAllLen1(int argc, char* argv[]);
bool isAnyAlpha(int argc, char* argv[]);
bool isAllInt(int argc, char* argv[]);
bool isAllFloat(int argc, char* argv[]);
void displayHelp();

// Main driver program with arguments
int main(int argc, char* argv[]){
   // If the user ask for help on usage of the command
   if(strncmp(argv[1], "--help", 6) == 0 || argc < 4){
       // Print the instructions on how to use the command
       displayHelp();
       exit(EXIT_SUCCESS);
   }

   // Convert the size from string to int
   int size = atoi(argv[1]);
   char qw[] = {'.'};

    // If the size is negative or if there exists a floating point in the
size
   if(size <= 0 || strstr(argv[1], qw)){
       // Print an error
       printf("\033[1;31m");
       printf("Error: ");
       printf("\033[0;31m");
           printf("The size of the array should be a positive non-zero
```

```
integer.\n");
        // Print the instructions on how to use the command
        displayHelp();
        exit(EXIT_FAILURE);
    }

    // Convert the choice from string to int
    int choice = atoi(argv[2]);

    // If the choice is less than 1 or greater than two or if there exists
a floating point in the choice
    if(choice < 1 || choice > 2 || strstr(argv[2], qw)){
        // Print an error
        printf("\033[1;31m");
        printf("Error: ");
        printf("\033[0;31m");
           printf("Choice should either be 1 for ascending sort or 2 for
descending sort.\n");
        // Print the instructions on how to use the command
        displayHelp();
        exit(EXIT_FAILURE);
    }

    int iarr[size];
    float farr[size];
    char carr[size];

    // If the length of all array elements are 1
    if(isAllLen1(argc, argv)){
        // If any one of them is an alphabet or character
        if(isAnyAlpha(argc, argv)){
            // Copy the contents to a char array
            for(int i = 3; i < argc; i++)
                carr[i - 3] = argv[i][0];
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc(carr, size);
            // If the choice is descending
```

```
            if(choice == 2)
                bubbleSortDesc(carr, size);
            // Print the sorted array
            print(carr, size);
            exit(EXIT_SUCCESS);
        }
        // If all of them are numbers
        else{
            // Copy the contents to an integer array
            for(int i = 3; i < argc; i++)
                iarr[i - 3] = atoi(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc(iarr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc(iarr, size);
            // Print the sorted array
            print(iarr, size);
            exit(EXIT_SUCCESS);
        }
    }
    // It the length of the content is more than one
    else{
        // If all the inputs are integers
        if(isAllInt(argc, argv)){
            // Copy the contents to an integer array
            for(int i = 3; i < argc; i++)
                iarr[i - 3] = atoi(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc(iarr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc(iarr, size);
            // Print the sorted array
            print(iarr, size);
            exit(EXIT_SUCCESS);
```

```
        }
        // If all the inputs are floating numbers
        else if(isAllFloat(argc, argv)){
            // Copy the contents to a floating array
            for(int i = 3; i < argc; i++)
                farr[i - 3] = atof(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc(farr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc(farr, size);
            // Print the sorted array
            print(farr, size);
            exit(EXIT_SUCCESS);
        }
        // If the numbers are not floating nor int nor characters
        else{
            // Print an error
            printf("\033[1;31m");
            printf("Error: ");
            printf("\033[0;31m");
                  printf("Enter a int array or float array or character
array.\n");
            // Display the usage
            displayHelp();
            exit(EXIT_FAILURE);
        }
    }
}

// A function that checks if length of all strings are 1
bool isAllLen1(int argc, char* argv[]){
    // Loop through the array
    for(int i = 3; i < argc; i++)
        // If any of the string length is not one, return false
        if(strlen(argv[i]) != 1)
            return false;
```

```
    return true;
}

// A function that checks if the contents contains an alphabet
bool isAnyAlpha(int argc, char* argv[]){
    // Loop through the array
    for(int i = 3; i < argc; i++)
        // If any of the content is an alphabet, return true
        if(isalpha(argv[i][0]) != 0)
            return true;
    return false;
}

// A function that checks if the contents are all integers
bool isAllInt(int argc, char* argv[]){
    int x;
    string a;
    // Loop through the contents
    for(int i = 3; i < argc; i++){
        // Convert the number from string to int
        x = atoi(argv[i]);
        // Convert the number back to string from int
        a = to_string(x);
        // Convert the original argument to a string
        string y = string(argv[i]);
            // If the original argument is different from the converted
argument, return false
        if(y.compare(a) != 0)
            return false;
    }
    return true;
}

// A function that checks if the contents are all floating numbers
bool isAllFloat(int argc, char* argv[]){
    double x;
    string a;
    // Loop through the contents
```

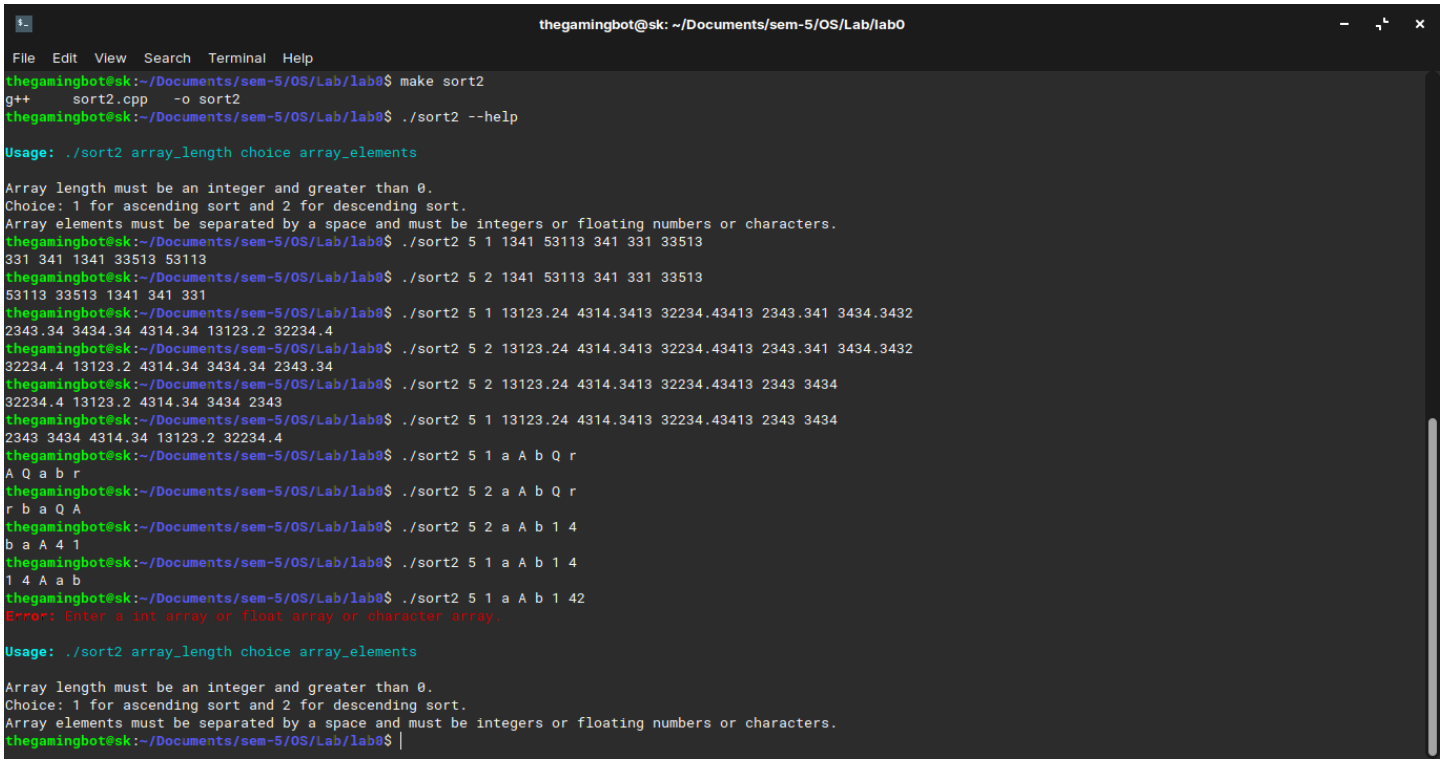
```
    for(int i = 3; i < argc; i++){
        // Convert the number from string to float
        x = atof(argv[i]);
        char* a;
        // Convert the number back to string from float
        sprintf(a, "%f", x);
        // Convert the original argument to a string
            // If the original argument is different from the converted
argument, return false
        if(strncmp(argv[i], a, strlen(argv[i])) != 0)
            return false;
    }
    return true;
}

// A bubble sort function that sorts integers in ascending order
void bubbleSortAsc(int* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

// A bubble sort function that sorts integers in descending order
void bubbleSortDesc(int* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] < arr[j+1])
                swap(&arr[j], &arr[j+1]);
}

// A bubble sort function that sorts floating numbers in ascending order
void bubbleSortAsc(float* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j+1])
                swap(&arr[j], &arr[j+1]);
}
```

```
// A bubble sort function that sorts floating numbers in descending order
void bubbleSortDesc(float* arr, int n){
   for (int i = 0; i < n - 1; i++)
       for (int j = 0; j < n - i - 1; j++)
           if (arr[j] < arr[j+1])
               swap(&arr[j], &arr[j+1]);
}

// A bubble sort function that sorts characters in ascending order
void bubbleSortAsc(char* arr, int n){
   for (int i = 0; i < n - 1; i++)
       for (int j = 0; j < n - i - 1; j++)
           if (arr[j] > arr[j+1])
               swap(&arr[j], &arr[j+1]);
}

// A bubble sort function that sorts characters in descending order
void bubbleSortDesc(char* arr, int n){
   for (int i = 0; i < n - 1; i++)
       for (int j = 0; j < n - i - 1; j++)
           if (arr[j] < arr[j+1])
               swap(&arr[j], &arr[j+1]);
}

// A function that prints the contents of an integer array
void print(int* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%d ", arr[i]);
   printf("\n");
}

// A function that prints the contents of a floating array
void print(float* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%g ", arr[i]);
   printf("\n");
}
```

```
// A function that prints the contents of a character array
void print(char* arr, int n){
   for(int i = 0; i < n; i++)
       printf("%c ", arr[i]);
   printf("\n");
}

// A function that swaps two integers
void swap(int* a, int* b){
   int c = *a;
   *a = *b;
   *b = c;
}

// A function that swaps two floats
void swap(float* a, float* b){
   float c = *a;
   *a = *b;
   *b = c;
}

// A function that swaps two characters
void swap(char* a, char* b){
   char c = *a;
   *a = *b;
   *b = c;
}

// A function to display help on usage of the command
void displayHelp(){
   printf("\033[1;36m");
   printf("\nUsage: ");
   printf("\033[0;36m");
   printf("./sort2 array_length choice array_elements\n");
   printf("\033[0m");
   printf("\nArray length must be an integer and greater than 0.\n");
   printf("Choice: 1 for ascending sort and 2 for descending sort.\n");
```

```
    printf("Array elements must be separated by a space and must be
integers or floating numbers or characters.\n");
}
```

Output:



Question 4:

Develop an application (using C & Command Line Arguments) for:
Same as above but you should define sort function only once internally and leave it to the compiler to generate data type specific functions. Clue is to use function templates feature in C. Read on it more!

Logic or commands used:

Bubble sort is being used for sorting the elements.
The command line arguments are being converted to integers using the atoi() function.
To distinguish integers, floating numbers and character, the below logic is used.
The spring can be either of length = 1 or length > 1

★ When the length is 1, the string can either be a character or an integer. Ex: 1, 2, 3 are both characters and integers. A, b,.. are characters too.
- 0 through 9 can be considered as integers, if all of the inputs are in this range.
- They are considered characters if any one of the input is a single length character, like a, A,..

★ When the length > 1, the string can be an integer or a floating number or a string.
- To check if the given numbers are integers, the argv[] are converted to integer using atoi and are converted back to string using sprintf. Upon string comparison, if both match, it is an integer.
- To check if the given numbers are floating numbers, the argv[] are converted to integer using atof and are converted back to string using sprintf. Upon string comparison, if both match, it is a float.
- Else, it is a string, give out an error.

Functions templates are used to reduce code redundancy. By declaring a new type of data type, which can be replaced during a function call. A function is created during the function call with a specified data type.

C++ Code:

```
#include <iostream>
#include <string.h>
#include <stdbool.h>
#include <ctype.h>
#include <sstream>
using namespace std;

// A bubble sort function that sorts a typename T in ascending order
template <typename T>
void bubbleSortAsc(T* arr, int n){
   for (int i = 0; i < n - 1; i++)
       for (int j = 0; j < n - i - 1; j++)
           if (arr[j] > arr[j+1]){
```

```
                T c = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = c;
            }
}

// A bubble sort function that sorts a typename T in descending order
template <typename T>
void bubbleSortDesc(T* arr, int n){
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] < arr[j+1]){
                T c = arr[j + 1];
                arr[j + 1] = arr[j];
                arr[j] = c;
            }
}

// A function that prints the contents on typename T array
template <typename T>
void print(T* arr, int n){
    for(int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

bool isAllLen1(int argc, char* argv[]);
bool isAnyAlpha(int argc, char* argv[]);
bool isAllInt(int argc, char* argv[]);
bool isAllFloat(int argc, char* argv[]);
void displayHelp();

// Main driver program with arguments
int main(int argc, char* argv[]){
    // If the user ask for help on usage of the command
    if(strncmp(argv[1], "--help", 6) == 0 || argc < 4){
        // Print the instructions on how to use the command
        displayHelp();
```

```
        exit(EXIT_SUCCESS);
    }

    // Convert the size from string to int
    int size = atoi(argv[1]);
    char qw[] = {'.'};

    // If the size is negative or if there exists a floating point in the
size
    if(size <= 0 || strstr(argv[1], qw)){
        // Print an error
        printf("\033[1;31m");
        printf("Error: ");
        printf("\033[0;31m");
            printf("The size of the array should be a positive non-zero
integer.\n");
        // Print the instructions on how to use the command
        displayHelp();
        exit(EXIT_FAILURE);
    }

    // Convert the choice from string to int
    int choice = atoi(argv[2]);

    // If the choice is less than 1 or greater than two or if there exists
a floating point in the choice
    if(choice < 1 || choice > 2 || strstr(argv[2], qw)){
        // Print an error
        printf("\033[1;31m");
        printf("Error: ");
        printf("\033[0;31m");
           printf("Choice should either be 1 for ascending sort or 2 for
descending sort.\n");
        // Print the instructions on how to use the command
        displayHelp();
        exit(EXIT_FAILURE);
    }
```

```
    int iarr[size];
    float farr[size];
    char carr[size];

    // If the length of all array elements are 1
    if(isAllLen1(argc, argv)){
        // If any one of them is an alphabet or character
        if(isAnyAlpha(argc, argv)){
            // Copy the contents to a char array
            for(int i = 3; i < argc; i++)
                carr[i - 3] = argv[i][0];
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc<char>(carr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc<char>(carr, size);
            // Print the sorted array
            print<char>(carr, size);
            exit(EXIT_SUCCESS);
        }
        // If all of them are numbers
        else{
            // Copy the contents to an integer array
            for(int i = 3; i < argc; i++)
                iarr[i - 3] = atoi(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc<int>(iarr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc<int>(iarr, size);
            // Print the sorted array
            print<int>(iarr, size);
            exit(EXIT_SUCCESS);
        }
    }
    // It the length of the content is more than one
```

```
    else{
        // If all the inputs are integers
        if(isAllInt(argc, argv)){
            // Copy the contents to an integer array
            for(int i = 3; i < argc; i++)
                iarr[i - 3] = atoi(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc<int>(iarr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc<int>(iarr, size);
            // Print the sorted array
            print<int>(iarr, size);
            exit(EXIT_SUCCESS);
        }
        // If all the inputs are floating numbers
        else if(isAllFloat(argc, argv)){
            // Copy the contents to a floating array
            for(int i = 3; i < argc; i++)
                farr[i - 3] = atof(argv[i]);
            // If the choice is ascending
            if(choice == 1)
                bubbleSortAsc<float>(farr, size);
            // If the choice is descending
            if(choice == 2)
                bubbleSortDesc<float>(farr, size);
            // Print the sorted array
            print<float>(farr, size);
            exit(EXIT_SUCCESS);
        }
        // If the numbers are not floating nor int nor characters
        else{
            // Print an error
            printf("\033[1;31m");
            printf("Error: ");
            printf("\033[0;31m");
                  printf("Enter a int array or float array or character
```

```
array.\n");
            // Display the usage
            displayHelp();
            exit(EXIT_FAILURE);
        }
    }
}

// A function that checks if length of all strings are 1
bool isAllLen1(int argc, char* argv[]){
    // Loop through the array
    for(int i = 3; i < argc; i++)
        // If any of the string length is not one, return false
        if(strlen(argv[i]) != 1)
            return false;
    return true;
}

// A function that checks if the contents contains an alphabet
bool isAnyAlpha(int argc, char* argv[]){
    // Loop through the array
    for(int i = 3; i < argc; i++)
        // If any of the content is an alphabet, return true
        if(isalpha(argv[i][0]) != 0)
            return true;
    return false;
}

// A function that checks if the contents are all integers
bool isAllInt(int argc, char* argv[]){
    int x;
    string a;
    // Loop through the contents
    for(int i = 3; i < argc; i++){
        // Convert the number from string to int
        x = atoi(argv[i]);
        // Convert the number back to string from int
        a = to_string(x);
```

```
        // Convert the original argument to a string
        string y = string(argv[i]);
            // If the original argument is different from the converted
argument, return false
        if(y.compare(a) != 0)
            return false;
    }
    return true;
}

// A function that checks if the contents are all floating numbers
bool isAllFloat(int argc, char* argv[]){
    double x;
    string a;
    // Loop through the contents
    for(int i = 3; i < argc; i++){
        // Convert the number from string to float
        x = atof(argv[i]);
        char* a;
        // Convert the number back to string from float
        sprintf(a, "%f", x);
        // Convert the original argument to a string
            // If the original argument is different from the converted
argument, return false
        if(strncmp(argv[i], a, strlen(argv[i])) != 0)
            return false;
    }
    return true;
}

// A function to display help on usage of the command
void displayHelp(){
    printf("\033[1;36m");
    printf("\nUsage: ");
    printf("\033[0;36m");
    printf("./sort3 array_length choice array_elements\n");
    printf("\033[0m");
    printf("\nArray length must be an integer and greater than 0.\n");
```

```
    printf("Choice: 1 for ascending sort and 2 for descending sort.\n");
      printf("Array elements must be separated by a space and must be
integers or floating numbers or characters.\n");
}
```

Output: