OS Lab Assignment: Pipe

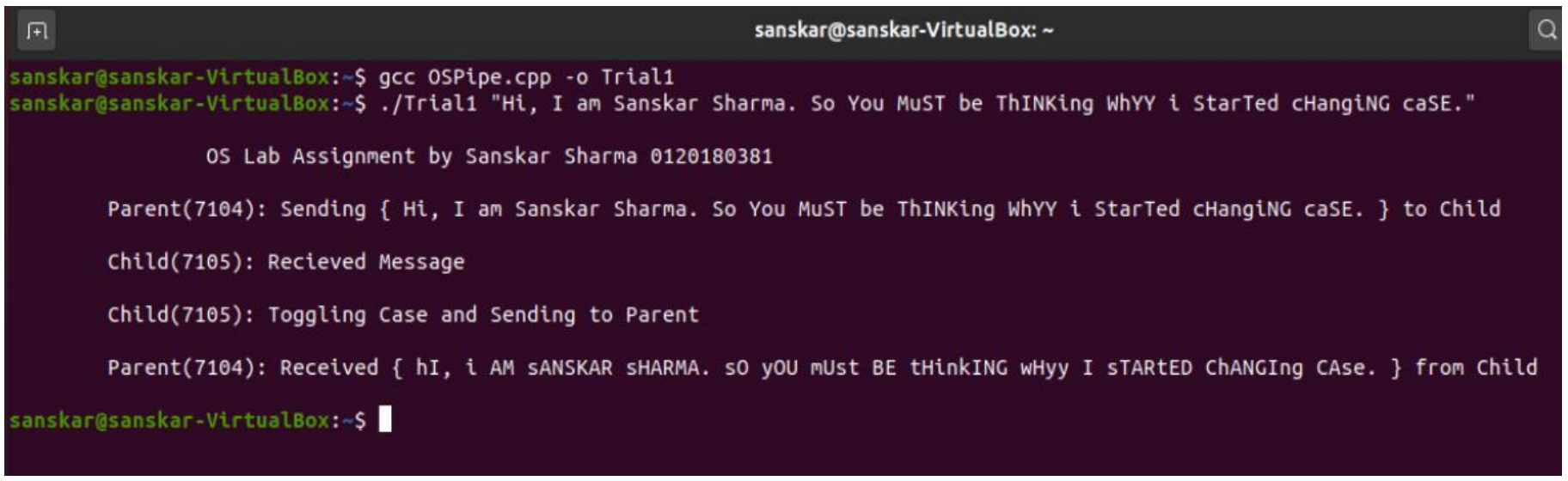
```c
//OS Assignment
//Submitted by: Sanskar Sharma
//PRN: 0120180381
/*Problem statement:
Design a program using ordinary pipes in which one process sends
a string message to a second process, and the second process reverses
the case of each character (lower to upper, upper to lower) in the
received message, and sends the reverse-case message to the
first process.
*/

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/wait.h>
#include <string.h>
#include <ctype.h>
#include <assert.h>
#include <stdbool.h>

//  Parent: reads from P1_READ, writes on P1_WRITE
//  Child:  reads from P2_READ, writes on P2_WRITE

#define P1_READ     0
#define P2_WRITE    1
#define P2_READ     2
#define P1_WRITE    3

// the total number of pipe *pairs* we need
#define NUM_PIPES   2


/*
    toggleString accepts an a pointer to char array,
    allocates size for the string to be toggled,
    copys the argument into a string, loops through
    the string and for every uppercase character
    we set it to its lower case counterpart and
    vice versa, returning the toggled string
*/
char *toggleString(char *argv){

    char *str = (char*)malloc(sizeof(argv[1]));
    /* Declare array sizeof input */

    strcpy(str, argv); /* Copy String to char array */

    for (int i = 0; str[i] != '\0'; i++)

    {

        if (islower(str[i]))
             str[i] = toupper(str[i]);

        else
             str[i] = tolower(str[i]);

    }
return str;

}

/*
    int inputValidation accept an integer
    (number of arugments) and a pointer to
    the cmd line input array
    We check to see if the command line input
    contains the minimal number of
    arugments and check to see whether or
    not the user input contains at least one
    reversible haracter, if all goes well we return 0.
*/
int inputValidation(int argc, char *argv[]){

    int i;          //Declare counter variable

    bool c = false; //Declare boolean flag using imported <stdbool.h>

    char str[strlen(argv[1])]; //Declare str

    strcpy(str, argv[1]); //copy argument into str

    if (argc != 2)  {     // check to see if we have enough arguments to continue
                          // Prompt user of correct usage
         fprintf(stderr, "\nUsage: %s <string> or <'string 1, string 2', ..., string n'> for multiple strings\n", argv[0]);

     exit(EXIT_FAILURE);    //Exit on improper input

    } else {
                  //loop through our string
         for(i=0;i<strlen(str);i++) {
                  //if any any char is a reversible character
         if(isalpha((int) str[i])){

            c = true; //set the flag to true

        }

    }
```

```
        if(c == false){ //If flag is false input does not contain any reversible charachters
           printf("\nSorry, The string you entered did NOT contain any Alphabetical Characters\nRun me again, with at least 1 Alphabetical character\n\n");
           exit(EXIT_FAILURE); //Exit on improper input
    }
      return (0);
    }
}
/*
Main takes input from command line, calls input validation to make sure of proper input,
then creates the pipes we will need and the forks the child process, Parent and Child
execute they're respective code
*/
int main(int argc,char *argv[]) {

    assert(argc>1);
    printf("\n\t\tOS Lab Assignment by Sanskar Sharma 0120180381\n");

    int fd[2*NUM_PIPES];    //Declare int[] of file descriptors

    int len, i;             //Declare length and integer for count

    pid_t pid;              //Declare process id

    char parent[strlen(argv[1])];   //Declare Parent array

    char child[strlen(argv[1])];    //Declare Child array

    if(inputValidation(argc, argv) == 0) /* Check for proper input */

    strcpy(parent, argv[1]);

    // create all the descriptor pairs we need
    for (i=0; i<NUM_PIPES; ++i)
    {
        if (pipe(fd+(i*2)) < 0)
        {
            perror("Failed to allocate pipes");
            exit(EXIT_FAILURE);
        }
    }

    // fork() returns 0 for child process, child-pid for parent process.
    if ((pid = fork()) < 0)
    {
        perror("Failed to fork process");
        return EXIT_FAILURE;
    }
    ////////////////////////////Childs Code BEGINS/////////////////////////////////
    ////////////////////////////////////////////////////////////////////////////////
    // if the pid is zero, this is the child process
    if (pid == 0)
    {
        // Child. Start by closing descriptors we
        // don't need in this process
        close(fd[P1_READ]);
        close(fd[P1_WRITE]);

        // used for output
        pid = getpid();

        // wait for parent to send us a value
        len = read(fd[P2_READ], &child, len);
        if (len < 0)
        {
            perror("Child: Failed to read data from pipe");
            exit(EXIT_FAILURE);
        }
        /*else if (len == 0)
        {
            // not an error, but certainly unexpected
            fprintf(stderr, "Child: Read EOF from pipe");
        }*/

        else
        {

              // report pid to console
            printf("\tChild(%d): Recieved Message\n\n\tChild(%d): Toggling Case and Sending to Parent\n",pid, pid);

            // send the message to toggleString and write it to pipe//
            if (write(fd[P2_WRITE], toggleString(parent), strlen(parent)) < 0)
            {
                perror("Child: Failed to write response value");
                exit(EXIT_FAILURE);

            }

        }

        // finished. close remaining descriptors.
        close(fd[P2_READ]);
        close(fd[P2_WRITE]);
```

```
    return EXIT_SUCCESS;
}
////////////////////////////Childs Code ENDS/////////////////////////////////

////////////////////////////Parent Code BEGINS/////////////////////////////////

// Parent. close unneeded descriptors
close(fd[P2_READ]);
close(fd[P2_WRITE]);

// used for output
pid = getpid();

// send a value to the child

printf("\n\tParent(%d): Sending { %s } to Child\n\n", pid, argv[1]);
if (write(fd[P1_WRITE], argv[1], strlen(argv[1])) != strlen(argv[1]))
{
    perror(
    exit(EXIT_FAILURE);
}

// now wait for a response
len = read(fd[P1_READ], &parent, strlen(parent));
if (len < 0)
{
    perror("Parent: failed to read value from pipe");
    exit(EXIT_FAILURE);
}
/*else if (len == 0)
{
    // not an error, but certainly unexpected
    fprintf(stderr, "Parent(%d): Read EOF from pipe", pid);
}*/

    else
    {
        // report what we received
        printf("\n\tParent(%d): Received { %s } from Child\n\n", pid, parent);
    }

    // close down remaining descriptors
    close(fd[P1_READ]);
    close(fd[P1_WRITE]);

    // wait for child termination
    wait(NULL);

    return EXIT_SUCCESS;

}
  ////////////////////////////Parent Code ENDS/////////////////////////////////

//End of Code
//Thanks for coming by!

/*
OUTPUT:
Trail 1::

sanskar@sanskar-VirtualBox:~$ gcc OSPipe.cpp -o Trial1
sanskar@sanskar-VirtualBox:~$ ./Trial1 "Hi, I am Sanskar Sharma. So You MuST be ThINKing WhYY i StarTed cHangiNG caSE."

        OS Lab Assignment by Sanskar Sharma 0120180381

    Parent(7296): Sending { Hi, I am Sanskar Sharma. So You MuST be ThINKing WhYY i StarTed cHangiNG caSE. } to Child

    Child(7297): Recieved Message

    Child(7297): Toggling Case and Sending to Parent

    Parent(7296): Received { hI, i AM sANSKAR sHARMA. sO yOU mUst BE tHinkING wHyy I sTARtED ChANGIng CAse. } from Child

sanskar@sanskar-VirtualBox:~$

Trial 2:

sanskar@sanskar-VirtualBox:~$ gcc OSPipe.cpp -o Trial2
sanskar@sanskar-VirtualBox:~$ ./Trial2 "i AM 20 YEARS OLD."

        OS Lab Assignment by Sanskar Sharma 0120180381

    Parent(7381): Sending { i AM 20 YEARS OLD. } to Child

    Child(7382): Recieved Message

    Child(7382): Toggling Case and Sending to Parent

    Parent(7381): Received { I am 20 years old. } from Child

sanskar@sanskar-VirtualBox:~$

Trial 3:

sanskar@sanskar-VirtualBox:~$ gcc OSPipe.cpp -o Trial3
sanskar@sanskar-VirtualBox:~$ ./Trial3 1+@#+@6+*

        OS Lab Assignment by Sanskar Sharma 0120180381

Sorry, The string you entered did NOT contain any Alphabetical Characters
```

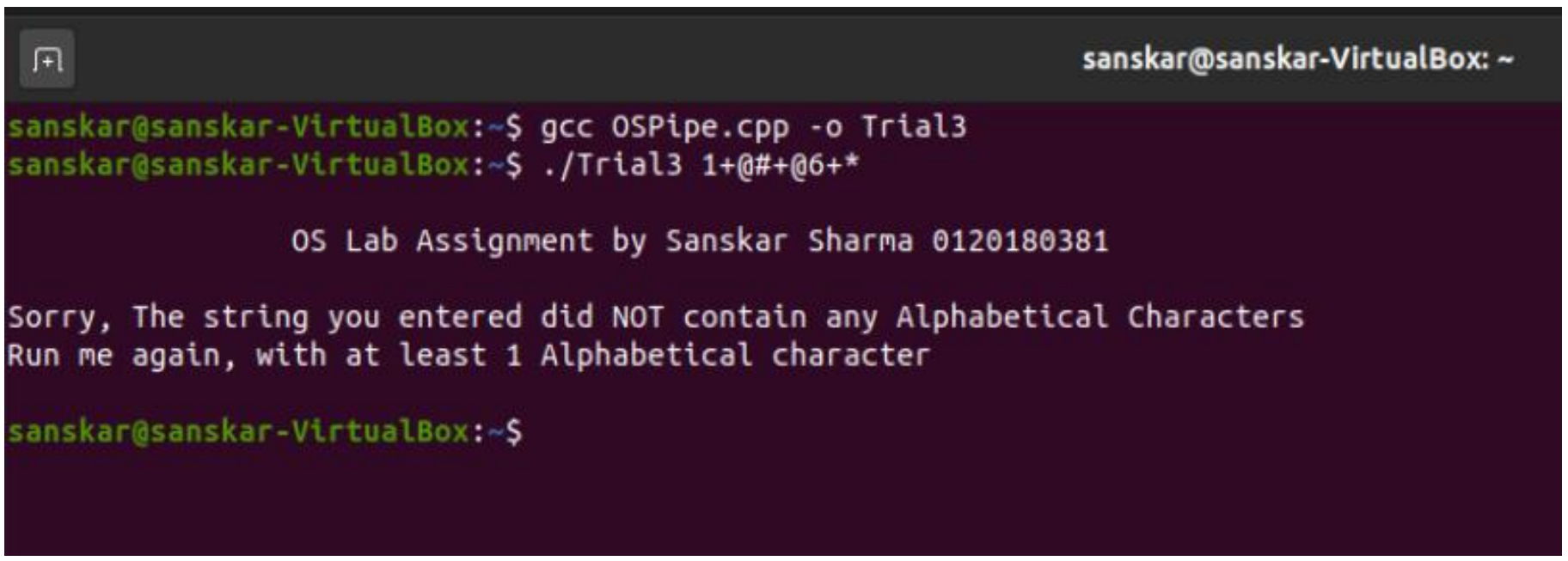
```
Run me again, with at least 1 Alphabetical character

sanskar@sanskar-VirtualBox:~$

Trial 4:

sanskar@sanskar-VirtualBox:~$ gcc OSPipe.cpp -o Trial4
sanskar@sanskar-VirtualBox:~$ ./Trial4 Okay another Try

        OS Lab Assignment by Sanskar Sharma 0120180381

Usage: ./Trial4 <string> or <'string 1, string 2', ..., string n'> for multiple strings
sanskar@sanskar-VirtualBox:~$

*/
```

Trial 1:

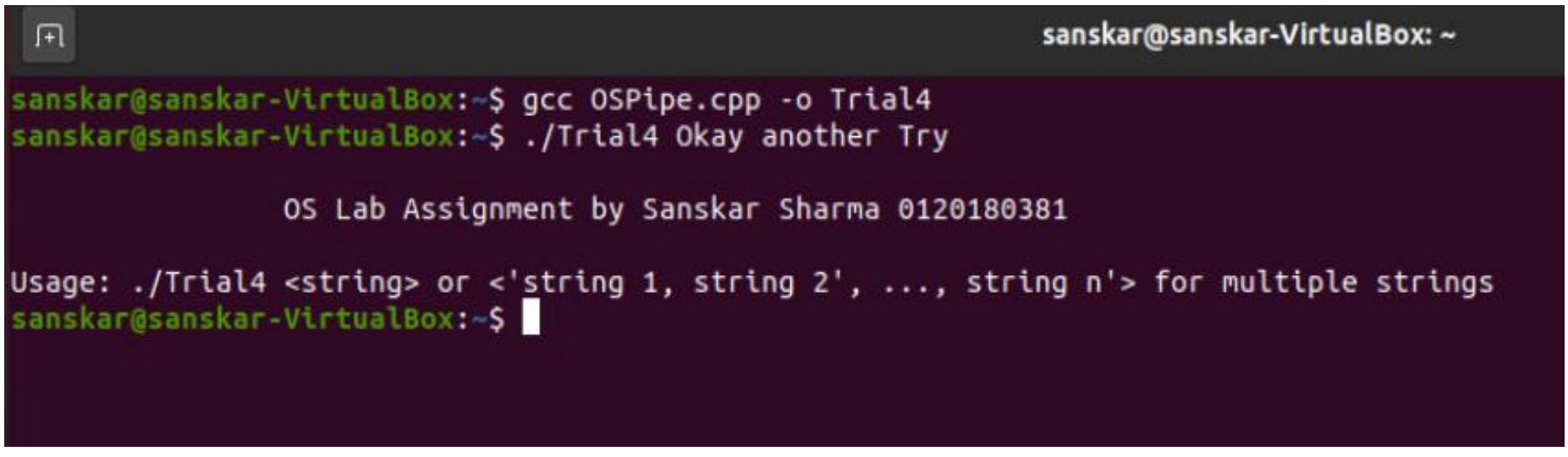


Trial 2:

Trial 3:



Trial 4:


sanskar@sanskar-VirtualBox: ~
sanskar@sanskar-VirtualBox:~$ gcc OSPipe.cpp -o Trial4
sanskar@sanskar-VirtualBox:~$ ./Trial4 Okay another Try
OS Lab Assignment by Sanskar Sharma 0120180381
Usage: ./Trial4 <string> or <'string 1, string 2', ..., string n'> for multiple strings
sanskar@sanskar-VirtualBox:~$