FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Compiler Construction - COMP 451 A
Spring 22

LAB - 01

Muhammad Sameed Gilani - 231488347

Introduction

TASK 1

For Task 1 we had to open a text file, whose name we received from a command line argument.
We then had to read the file and obtain the longest line on the file and display it on the console. We used builtin C string functions in this part. After reading the file line by line we could compare lengths of the lines through strlen() and use strcpy() to copy the longer string onto a predefined character array.

TASK 2

For Task 2 we did the same as Task 1, except we used user defined functions for strlen() and strcpy().

Major functions/library calls used in the program:

TASK 1

- **Command line arguments:**
  - ➔ int argc:

    This stores the number of arguments passed by the user when the run the program. This also includes the program name.

  - ➔ Char* argv[]:

    This is an array of character pointers, which point to the address of the command line arguments that were passed. argv[0] always stores the program name.

- **fopen() → <stdio.h>**

  - This is a C library function that uses a file pointer to open a file with in the chosen mode. In our case, “r” (read) was used.
  - It returns a file pointer if file opens successfully or Null when it fails.
  - We use this function to open and read the text file.

  Function Signature:

```
FILE *fopen(const char *file_name, const char
*mode_of_operation);
```

Sample from program:

```
FILE *fp;
fp = fopen(argv[1],"r");

if (fp == NULL){
     printf("Error opening file");
     exit(0);
}
```

- **malloc() → <stdlib.h>**
    - This function is used to dynamically allocate a single block of memory in the heap, with a size of our choice.
    - It returns a pointer to the allocated memory.
    - It does not initialize memory on execution.
    - We use this to allocate a block of memory to store the line by line that we read from the file.

Function Signature:

```
ptr = (cast-type*) malloc(byte-size)
```

Sample from program:

```
char* buff = (char*) malloc(sizeof(char)*1000);
```

- sizeof() returns the size of the data type in bytes. (2 for char). So this allocates a memory of 2000 bytes. With buff as the pointer.

- **fgets() → <stdio.h>**
    - This function reads line by line from the file. It keeps reading a line until a \n is encountered or the max number of characters is reached or end of file is reached.
    - The lines read are stored with a character pointer.
    - It returns the string/line that is read.

Function Signature:

```
char *fgets(char *str, int n, FILE *stream)
```

Sample from program:

```
while(fgets(buff,1000,fp) != NULL){}
```

- where buff (Memory from our malloc() call) stores the lines read, 1000 is the max character count and fp is the file pointer, from which it will read the lines.

- **strlen() → <string.h>**
  - This function returns the length of the string that is passed.
  - It does not count the '\0' character.
  - In our program we use to to get and compare the length of the line read from the file and the current largest line.

  Function Signature:

```
int strlen(const char *str);
```

  Sample from program:

```
if (strlen(buff) > strlen(longest_str)){
          strcpy(longest_str,buff);
}
```

- **strcpy() → <string.h>**
  - We use this function to copy a string to a character array.
  - It returns a pointer to the destination string.
  - In our program we use it to to copy the lines read from the file onto a per-defined character array.

  Function Signature:

```
char* strcpy(char* dest, const char* src);
```

  Sample from program:

```
if (strlen(buff) > strlen(longest_str)){
          strcpy(longest_str,buff);
}
```

  - We pass it a destination (longest_str) and a source (buff) in this case. The string in buff is copied to longest_str.

TASK 2

Same functions as Task 1 are used, except strlen() and strcpy() are user defined.

- get_length()
  - The function will return the length(int) of the string passed to it.
  - We pass the character pointer of the string to it, pointing to the start of the string.
  - We use a while loop that terminates once the character pointer is a '\n', newline characters i.e end of the current line
  - It increments the variable int len = 0, by 1 each loop
  - The character pointer s, is also incremented. But it increments by 2 bytes each loop as a char is 2 bytes.

Sample from program:

```
if (get_length(buff) > get_length(longest_str)){
          cpy_strings(longest_str,buff);
     }
```

Function:

```
int get_length(char *s){

int len = 0;

while(*s != '\n'){
     len ++;
     s++;
}
return len;
}
```

- cpy_strings()
    - This function does not return anything.
    - It copies a string from a source to a destination character array.
    - We pass two character pointers, one pointing to the start of the destination array and other at the sources.
    - We use a while loop that terminates once the source char pointer is a ‘\0’ terminating character for a string. Which tells us that the source string has ended.
    - First it copies the character from the source pointer to the destination pointer. Then both the source and character pointers are incremented. (+2 bytes as char is 2 bytes)
    - In our program we use it to copy the line from the file to a char array to store the longest string.

    Sample from program:

```
if (get_length(buff) > get_length(longest_str)){
          cpy_strings(longest_str,buff);
     }
```

    Function:

```
void cpy_strings(char *destination, char *source){

while (*source != '\0'){
     *destination = *source;
    destination++;
    source++;
}
```

The Program

TASK 1:

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



int main(int argc, char *argv[]){
     FILE *fp;
     char longest_str[100];


     //check number of args
     if(argc != 2){
     printf("This program needs a text file as argument\n");
     exit(0);
     }

     fp = fopen(argv[1],"r");

     if (fp == NULL){
          printf("Error opening file");
          exit(0);
     }


     char* buff = (char*) malloc(sizeof(char)*1000);

     while(fgets(buff,1000,fp) != NULL){

          if (strlen(buff) > strlen(longest_str)){
               strcpy(longest_str,buff);
          }
          //fputs(buff,stdout);
     }

     printf("Longest line in the file is: %s\n",longest_str);

     //your logic goes here
     return 0;
}
```

TASK 2:

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



int get_length(char *);
void cpy_strings(char *destination, char *source);
int main(int argc, char *argv[]){

     FILE *fp;
     char longest_str[100] = "ee\n";

     //check number of args
     if(argc != 2){
          printf("This program needs a text file as argument\
n");
          exit(0);
     }

     fp = fopen(argv[1],"r");

     if (fp == NULL){
          printf("Error opening file");
          exit(0);
     }


     char* buff = (char*) malloc(sizeof(char)*1000);

     while(fgets(buff,1000,fp) != NULL){
          //printf("%d\n",get_length(longest_str));


          if (get_length(buff) > get_length(longest_str)){
               cpy_strings(longest_str,buff);
          }
     }

     printf("Longest line in the file is: %s\n",longest_str);


     //your logic goes here
```

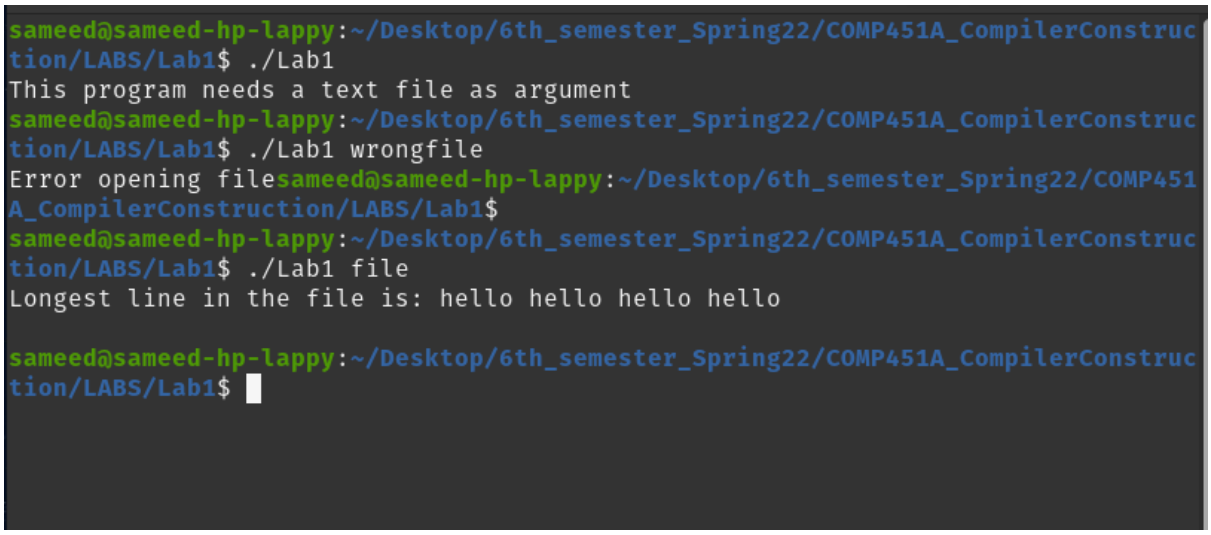
```
     return 0;

}



int get_length(char *s){

     int len = 0;

     while(*s != '\n'){
          len ++;
          s++;
     }
     return len;
}


void cpy_strings(char *destination, char *source){

     while (*source != '\0'){
          *destination = *source;
         destination++;
         source++;
     }

}
```

//////// END OF PROGRAM/////////

Output / Screen Shots

TASK 1

I. When wrong number of arguments are passed
II. when wrong file is passed
III. when correct number of arguments and correct file is passed



TASK 2

IV. When wrong number of arguments are passed
V. when wrong file is passed
VI. when correct number of arguments and correct file is passed