FORMAN CHRISTIAN COLLEGE
(A CHARTERED UNIVERSITY)

Compiler Construction - COMP 451 A

Spring 22

Assignment 1

Muhammad Sulaiman Sultan - 231453415

Muhammad Sameed Gilani - 231488347

INTRODUCTION:

A preprocessor accepts the source program C file, modifies it and gives its output file to the compiler. The preprocessor is only used if there are preprocessor directives in the source program. Preprocessor directives are usually defined by a '#' as the first character. These are the preprocessor directives that will be worked on,

- Macro Expansion
  - A #define directive is used to create a macro
  - The #define is followed by a macro head and macro body
  - The macro head is replaced with the body in the source file
- File inclusion
  - A header file is defined as, #include <filename.h>
  - The header files must be included to use their defined functions.
  - Header files are identified in the file and replaced with the contents of the header file, inside the source file

Furthermore, the preprocessor also removes all single line and block comments along with any remaining blank lines in the source file.

In this assignment these 3 tasks are performed using these functions,

- void stripOffComments(. . .)
- void macroExpansion(. . .)
- void include HeaderFiles(. . .)

LOGIC:

1. ***int main(int argc, char *argv[ ])*** ----------.> *Not complete*

- The main function will accept a command line argument, which will be the input/source program. A, C file. Relevant check is performed to see if argc received 2 arguments. (The program to be run & the input/source program)
- The name of the file is copied from argv[1] to a char array and the name is given as argument to , *stripOffComments(char fileArg[100]).*
- Then we call the 3 functions mentioned above, each using the output file of the previous function,
  - *stripOffComments(fileName);* → **Input:** source program. **Output:** out1.c
  - *macroExpansion();* → **Input:** out1.c. **Output:** out2.c
  - *includeHeaders();* → **Input:** out2.c. **Output:** final.c

2. ***stripOffComments(char fileArg[100])***

- ***removeComments(fileArg);,*** removes all the single and multi line comments.
- We use fgetc() to store the file into a buffer. We then iterate the buffer. If the first char is '/' and the second is either '/' or '*'.
- If the 2nd char is '/' then we skip chars until a '/n' character.
- If the 2nd char is '*' then we skip two chars to go ahead of the '/*'. After which we skip characters until we find a '*/'
- All characters beside that, I.e not comments are written to the out1.c using fputc.

- ***stripBlank();***, out1.c is opened to be read and a temp file is opened to be written to. out1.c is read using fgetc() and written to a temp file using fputc(). (out1.c is written to a temp file instead of a buffer, as reading the file line by line using fgets() and processing each line was easier then from a buffer)
- Then we open the the temp file to be read from and out1.c to be written to. Temp file is read line by line using fgets() and stored into a buffer. The buffer (single line) is read character by character.
- If a '\n' is the first character of the line. It is skipped.
- If a line starts with a space or tab. It will be read till there are spaces/tabs.
- If after those spaces/tabs there is a '\n' or '\0', it indicates the line was just empty spaces/tabs so it is skipped.
  If a a character is seen after those spaces/tabs, the line is written to out1.c, as it was not empty
- All other lines starting with characters are written to out1.c, using fprintf().
- At the end the files are closed and the temp file is deleted using remove()

3. ***void macroExpansion(. . .)***

- ***getMacroHeadBody();,*** Writes all the macro heads to file, TempHead and their bodies to TempBody. The head and body is mapped to the same line in their respective file.
- It identifies #define in the line being read from fgets(). These are further processed, the head and body are separately stored in their file.
  The macro is not written to the next file.
- The lines without #define are simply written to the next file, TempOut2.c
- ***macroExpansion();,*** uses the files created earlier and TempOut2.c to expand macros find in, out1.c

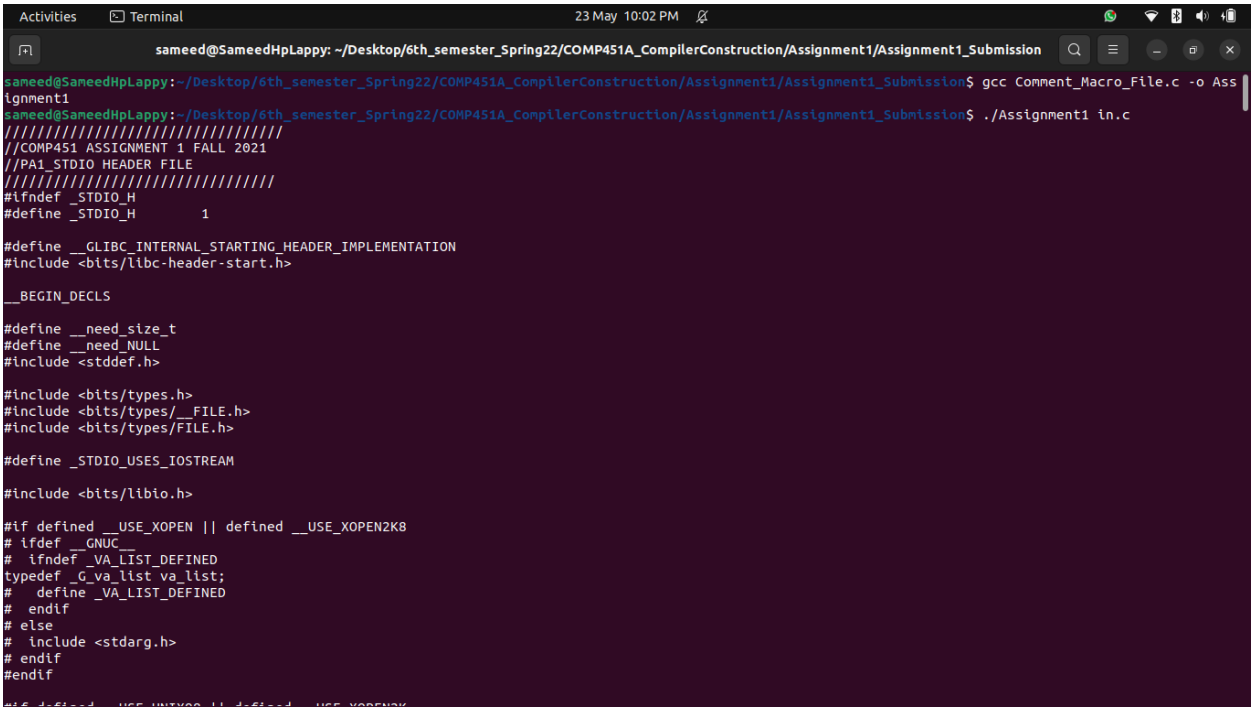
- So the general pattern is that, it gets a line from the source program, it checks that line for the usage of a macro. The macro is identified using the TempHead file and their respective body from TempBody.
  Then it replaces the macro head found in the line with its respective body. After which it writes that line to out2.c.
- A nested while loop is used with fgets()
- The outer loop gets the current line from out1.c, while the inner loop gets the macro heads from its file, this loop will run, based on how many total macros there are.
- The macro respective to the head, is identified using its first char (bodies are indexed), since the heads and bodies are mapped to the same lines in their files.
- The body and head are stored in respective buffers.
- Then using strstr(), we reach the pointer to the first char of the macro head in the line. It iterates through to the end of the macro head.
- From there the text after the macro head is stored in a buffer, for later use.
- Now that we have a pointer to the macro head, it uses strcpy() to copy the body onto the pointer. However text after the head is also removed.
- To correct that, we concat the new line with the body with the buffer in which we previously stored text that was after the macro head.
- Now we have a line with the macro head replaced with the body inside a buffer, which we write to out2.c
- Lines without macros are simply written to out2.c as well.
- At the end of each iteration of the inside loop, we use memset () to reset buffers and variables and rewind the body and head files.
- Now out2.c is available to be used by the next function.

4. ***void includeHeaderFiles(. . .)***

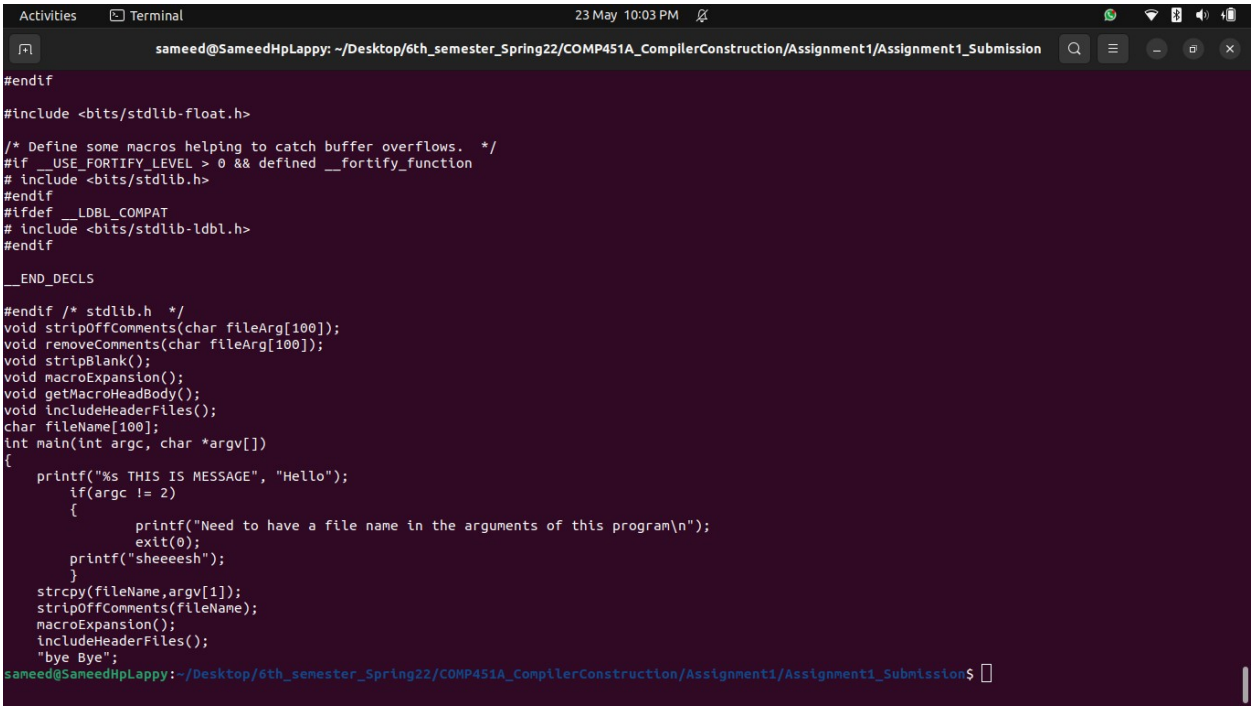
- We use fgets to iterate through the file line by line
- We use strstr() if the name of the header file is included in the current read line.
- If it is, then it is replaced with the contents of the header file using fputs() in the buffer
- At the end we simply write the buffer to final.c
- final.c is opened again and printed onto the console.

Sample Output:

Using in.c in submission folder.



Skipping till the end….



CODE:

```
/*
Muhammad Sulaiman Sultan: 231453415
Muhammad Sameed Gilani: 231488347
Compiler Cnstruction Section A
Assignment 1
Submitted to: M. Rauf Butt
*/

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

void stripOffComments(char fileArg[100]);
void removeComments(char fileArg[100]);
void stripBlank();
void macroExpansion();
void getMacroHeadBody();
void includeHeaderFiles();


char fileName[100]; // Will keep the file name, from argv[1]

int main(int argc, char *argv[])
{
                                if(argc != 2)
                                {
```

```
                                printf("Need to have a file name in
the arguments of this program\n");
                                exit(0);
                                }
    strcpy(fileName,argv[1]);


    stripOffComments(fileName);
    macroExpansion();
    includeHeaderFiles();




                                return 0;
}


void stripOffComments(char fileArg[100]){
    /*
        All comments and blank lines from the source file are removed
and written to out1.c
        The Source file is not changed
    */


                                removeComments(fileArg);
    stripBlank();


}


void removeComments(char fileArg[100])
{
                                FILE *SourceFile;
    FILE *outFile;
```

```
                                SourceFile = fopen(fileArg,"r");
    outFile = fopen("out1.c","w+");
                                char* buff = (char*)
malloc(sizeof(char)*100000);
                                char* buff2 = (char*)
malloc(sizeof(char)*100000);


                                char c = ' ';
                                int idx = 0;

                                while (c != EOF)
                                {
                                 c = fgetc(SourceFile);
                                 //printf("%c",c);
                                 buff[idx] = c;
                                 idx++;
                                }

                                int i,j = 0;
                                int flag = 0;
                                while(buff[i] != EOF)
                                {
                                 if(buff[i] == '/')
                                 {
                                       if(buff[i+1] == '/')
                                             while(buff[i] != '\n')
                                                   i++;
                                       if (buff[i+1] =='*')
                                       {
```

```
                                        i+=2;
                                        while(buff[i] != EOF)
                                        {
                                              if(buff[i-1] == '*'
&& buff[i] == '/')
                                              {
                                                    i++;
                                                    break;
                                              }
                                              i++;
                                        }
                                  }
                             }
                             fputc(buff[i],outFile);
                             i++;
                             j++;
                            }
                            fclose(SourceFile);
                            fclose(outFile);
}

void stripBlank(){
    /*
        Function will remove any blank lines from the file and updates
the out1.c file.
    */

    // Creating file handlers
    FILE *SourceFile;
    FILE *TempFile;
```

```
    FILE *OutFile;


    // Open the source file from previous function(For Reading) and
create a temp file(For Writing)
    SourceFile = fopen("out1.c","r");
    TempFile = fopen("Temp","w+");


    char* buffer = (char*) malloc(sizeof(char)*100000); // Memory
allocated


    // Reads the file character by character and writes it to the temp
file
    char currC = ' ';
    currC = fgetc(SourceFile);


    while(currC != EOF){
        fputc(currC,TempFile);
        currC = fgetc(SourceFile);
    }


    fclose(SourceFile);
    fclose(TempFile);




    // Opens the temp file(For reading) and the Out1.c file(For
writing)
    TempFile = fopen("Temp", "r");
    OutFile = fopen("out1.c","w+");


    // Reads the temp file and writes each line to the buffer
```

```
    while(fgets(buffer,100000,TempFile) != NULL){

        int i = 0;

        int check = 1; // Flag to see if there is text at any point in
the line


        if(buffer[0] == '\n' ) continue;



        if(buffer[0] == ' ' || buffer[0] == '\t' ){         // Will
see if the line starts with a space or tab

            while(buffer[i] == ' ' || buffer[i] == '\t' ){ //  It will
then see if the line has any text in it

                i++;


                if(buffer[i] == '\n' || buffer[i] == '\0'){ // Line
has no text in it so it is not written to out file

                    check = 0;

                    break;

                }


            }


            if(check == 1) fprintf(OutFile,"%s",buffer); // The line
has some text in it so it will be written to out file



        }

        else fprintf(OutFile,"%s",buffer); // If the line starts with
any text it is written to the out file

    }

    fclose(TempFile);
```

```
   // remove("Temp"); // Temp file is deleted
    fclose(OutFile);
}

void macroExpansion(){
    /*
        All macros are expanded in the file
    */

    getMacroHeadBody(); // files are created with the macro heads and
bodies respectively

    FILE *SourceFile;
    FILE *HeadFile;
    FILE *BodyFile;
    FILE *OutFile;

    SourceFile = fopen("TempOut2.c","r"); // Temp file from
getMacroHeadBody() is opened to be read
    OutFile = fopen("out2.c","w+");  // out2.c is created to be
written to
    HeadFile = fopen("TempHead","r");
    BodyFile = fopen("TempBody","r");

    char* buff = (char*) malloc(sizeof(char)*100000);
    char* buffAfterBodyStr = (char*) malloc(sizeof(char)*100000);

    char* buffHead = (char*) malloc(sizeof(char)*100000);
    char* buffHeadTemp = (char*) malloc(sizeof(char)*100000);
```

```
    char* buffBody = (char*) malloc(sizeof(char)*100000);
    char* buffBodyTemp = (char*) malloc(sizeof(char)*100000);



    int i = 0;
    int j = 0;
    int newLineFlag = 0;
    char *test;
    int val;
    char headLastChar = ' ';



    while(fgets(buff,100000,SourceFile) != NULL){ // Each line is read
from the source file
        // This will allow us to check each macro head, whether its
written in each single line thats being read


        while(fgets(buffHead,100000,HeadFile) != NULL){ // Each macro
head from the macro head file is read
            newLineFlag++;


            fgets(buffBody,100000,BodyFile); // We read the body file
and check its first character
            val = atoi(&buffBody[0]); // its first char is the number
of line its written on
                                      // This allows us to pick the
correct macro body, according to the macro head being considered
            while(val != newLineFlag){
                printf("%d",newLineFlag);
                fgets(buffBody,100000,BodyFile);
                val = atoi(&buffBody[0]);
```

```
            }


            i = 1;

            j = 0;

            while(buffBody[i] != '\n'){ // We remove the first
char(line number) and skip the \n and writes

                buffBodyTemp[j] = buffBody[i];// the body to a buffer

                i++;

                j++;

            }            // buffBodyTemp == BODY of macro in question



            i = 0;

            j = 0;

            while(buffHead[i] != '\n'){ // skips \n and writes the
macro head to a buffer

                buffHeadTemp[j] = buffHead[i];

                i++;

                j++;

            }

                        // buffHeadTemp == HEAD of macro in question


            headLastChar = buffHeadTemp[i-1]; // To keep track of the
last char of the macro head



            test = strstr(buff,buffHeadTemp); // Gives us a pointer to
the first char of the macro head, if it is in the,

            i = 0;                              // current line from
the source file

            j = 0;
```

```
            if(test){ // Current macro is in the current line


                while(test[i] != headLastChar){ // It gets to the last
character of the macro head

                    i++;

                }

                i++;

                while(test[i] != '\n'){ // now that the index (i) is
at the end of the macro head,

                    buffAfterBodyStr[j] = test[i]; // we then store
the chars after macro head in a buffer

                    i++;

                    j++;

                }

                buffAfterBodyStr[j] = '\n';


                strcpy(test,buffBodyTemp); // the pointer test is at
the start of the macro head

                                           // we copy the macro body onto
that pointer. So all the chars from the start

                                           // of the macro head till end
of line is replaced with the macro body


                strcat(test,buffAfterBodyStr); // To restore the text
after the macro body that was removed,

                                               // we concat the macro
body with the buffer, in which we previously stored the

                                               // text after the macro
head


            }
```

```
        // Buffers are reset to write the next iteration of items
        memset(buffHeadTemp,0,100000);
        memset(buffBodyTemp,0,100000);
        memset(buffAfterBodyStr,0,100000);



        j = 0;
        i = 0;


        }
        newLineFlag = 0;
        memset(buffBody,0,100000);


        // rewind the files to recheck every head with the new line
from fgets
        rewind(HeadFile);
        rewind(BodyFile);


        fputs(buff,OutFile); // buffer with the updated line, with
macro body replaced with head(if head was in the line)
                             // is written to out2.c


    }
    fclose(OutFile);
}


void getMacroHeadBody()
{
    /*
```

```
        This will write the macro Heads to a file and macro Bodys to a
seperate file.

        They will be placed in order in each file, so the head and
macro are mapped to the same lines.

    */

    FILE *SourceFile;

    FILE *TempOutFile;

    FILE *HeadFile;

    FILE *BodyFile;


    SourceFile = fopen("out1.c","r"); // out1.c from prev fucntion is
opened to be read

    TempOutFile = fopen("TempOut2.c","w+"); // a temp file is opened
to be written

    HeadFile = fopen("TempHead","w+");

    BodyFile = fopen("TempBody","w+");


    char* buff = (char*) malloc(sizeof(char)*100000);

    int i = 0;

    int bodyLineCount = 1; // Will track the line number the macro
body is stored on


    while(fgets(buff,100000,SourceFile) != NULL){


        // Reads the input file line by line. line with #define will
be caught

        if((buff[0] == '#' && buff[1] == 'd' && buff[2] == 'e' &&
buff[3] == 'f' && buff[4] == 'i' && buff[5] == 'n' && buff[6] == 'e')
|| (buff[0] == '#' && buff[1] == 'D' && buff[2] == 'E' && buff[3] ==
'F' && buff[4] == 'I' && buff[5] == 'N' && buff[6] == 'E')) {


            i = 7;
```

```
            while(buff[i] == ' ' || buff[i] == '\t'){ //Skips spaces
after #define
                i++;
            }

            while(buff[i] != ' '){ // Its at the Macro head now, itll
be written to its file
                fputc(buff[i],HeadFile);
                i++;
            }
            fputc('\n',HeadFile);

            while(buff[i] == ' ' ){ // Spaces after the macro head are
skipped
                i++;
            }
            fprintf(BodyFile,"%d",bodyLineCount);
            bodyLineCount++;

            while(buff[i] != '\n'){     // Its at the macro body.itll
written to its file
                fputc(buff[i],BodyFile);
                i++;
            }
            fputc('\n',BodyFile);
        }
        else{
            fputs(buff,TempOutFile); // It wont write the #define to
the the tempOut2 file
        }
```

```
    }
    fclose(SourceFile);
    fclose(TempOutFile);
    fclose(HeadFile);
    fclose(BodyFile);
}

void includeHeaderFiles()
{
                                    char* buff = (char*)
malloc(sizeof(char)*100000); // Will store each line read
                                    char* include = (char*)
malloc(sizeof(char)*100000); // Will store the include files

                                    FILE *fp;
                                    FILE *final;

                                    fp = fopen("out2.c","r");
                                    final = fopen("final.c","w+");

                                    while(fgets(buff,100000,fp) != NULL)
                                    {

                                     int i = 0;
                                     if (strstr(buff,"stdio.h") != NULL)
                                     {
                                           FILE *stdio;
                                           stdio = fopen("pa1_stdio", "r");
```

```
                                        while(fgets(include, 100000,
stdio) != NULL)
                                              fputs(include, final);
                                        fclose(stdio);
                                  }
                                  else if (strstr(buff,"stdlib.h") !=
NULL)
                                  {
                                        FILE *stdlib;
                                        stdlib = fopen("pa1_stdlib",
"r");
                                        while(fgets(include, 100000,
stdlib) != NULL)
                                              fputs(include, final);
                                        fclose(stdlib);
                                  }
                                  else if (strstr(buff,"string.h") !=
NULL)
                                  {
                                        FILE *string;
                                        string = fopen("pa1_string",
"r");
                                        while(fgets(include, 100000,
string) != NULL)
                                              fputs(include, final);
                                        fclose(string);
                                  }
                                  else
                                  {
                                        fputs(buff, final);
                                  }
                                 }
```

```
                                fclose(final);
    fclose(fp);

                                final = fopen("final.c", "r");
                                while(fgets(buff,100000,final) !=
NULL)
                                {
                                 printf("%s",buff);
                                }
                                fclose(final);
}
```