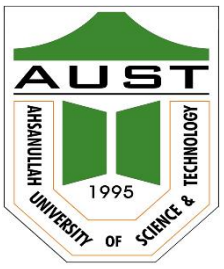

Ahsanullah University of Science and Technology (AUST)
Department of Computer Science and Engineering

Assignment 2

Course No.: CSE4130

Course Title: Formal Languages & Compilers Lab

Date of Submission-07.06.2023

Submitted To- Mr. Md. Aminur Rahman & Iffatur Nessa.

Submitted By-

MD Fardin Jaman Aranyak
190204093
Group: B2
Year- 4th
Semester- 1st
Session: Fall'22
Department- CSE

Answer:

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

char kws[20][10]={"int", "double", "float", "char", "for", "while", "do", "if", "else", "switch",
"case"};
char ops[] = "+-*/%=<>!|&";
char pars[] = "(){}[]", op[5];
char seps[]=",;'\"";
char ids[100][20];
char c,s[20];
int err=0;

FILE *rf,*wf;
int totid=1, totkw = 11, opflag=0;

//this function will take a file name input from user and open it in read mode
int read_file(){
   char filename[100];

   printf("\nEnter the filename: ");
   scanf("%s", filename);
   rf = fopen(filename, "r");
   if (rf == NULL)
   {
      printf("Error opening file.\n");
      return 1;
```

```
    }
}


//this plainC() function removes all newline, extra spaces and comments from a c source code file
//one problem occurs for this function is
//it end with a unknown character in the end on the generated file
void plainC(){
    FILE *p2;
    char c, c2 = ' ';
    p2 = fopen("lexemes.txt", "w");

    while((c = fgetc(rf))!= EOF)
    {
        if(c==' ' || c=='\n'){
            fputc(' ', p2);
            while((c=fgetc(rf)) == ' ' | c == '\n');
        }
        if((c=='/') && ((c2 = fgetc(rf))== '/'))
        {
            while( ((c=fgetc(rf))!='\n'));
        }
        else if((c=='/') && (c2=='*'))
        {
            while((c!='/') && (c2 != '*'))
            {
                c2 = c;
                c = fgetc(rf);
            }
```

```
        }
        else {fputc(c, p2);}
        c2 = c;
    }
    fclose(p2);
    fclose(rf);

    p2 = fopen("lexemes.txt", "r");
    while((c = fgetc(p2))!= EOF)
    {
        printf("%c", c);
    }
    fclose(p2);
}
//isoperator() function will check for a operators
int iskeyword(){
    for(int i=0; i<totkw; i++){
        if(strcmp(s,kws[i]) == 0){
            printf("[kw %s]", s);
            return 1;
        }
    }
    return 0;
}
int isoperator(){
    int len=strlen(ops);
    for(int i=0; i<len; i++){
        if(ops[i] == c){
```

```
            op[opflag] = c;
            opflag++;
            if((c=fgetc(rf))!=EOF){
               isoperator();
            }
            return 1;
         }
      }
      if(opflag>0) {
         op[opflag] = '\0';
         fseek(rf, -1, SEEK_CUR);
         opflag=0;
      }
      return 0;
}
//isparenthesis() function will check for a parenthesis
int isprenthesis(){
      int len=strlen(pars);
      for(int i=0; i<len; i++){
         if(pars[i] == c){
            //printf("[par %c] ",c);
            return 1;
         }
      }
      return 0;
}
//isseparator() function will check for a separator
int isseparator(){
```

```
    int len=strlen(seps);
    for(int i=0; i<len; i++){
        if(seps[i] == c){
            //printf("[sep %c] ",c);
            return 1;
        }
    }
    return 0;
}
//isidentifier() function find the valid keywords also label as id and if not valid then label as unkn
int isidentifier(){
    int i=0,idflag=0;
    for (int i = 1; i < totid; i++) {
        //this for loops checks if the identifier already declared
        //if declared then refers to the entry pointer in the symbol table of that identifier
        if(strcmp(s,ids[i]) == 0){
            printf("[id %d]", i);
            return 1;
        }
    }
    int len=strlen(s);
    //this if section checks if the word is a valid indentifier
    if(s[0]=='_' || isalpha(s[0])){
        for(i=1; i<len; i++){
            if(s[i]=='_' || isalnum(s[i])){
                idflag=1;
            }else return 0;
        }
```

```
        idflag=1;
    }
    if(idflag==1){
        strcpy(ids[totid++], s);
        printf("[id %s] ",s);
        idflag=0;
        return 1;
    }
    return 0;
}
//check if the word is a number or not
int isnumber(){
    int len=strlen(s);
    int i, nflag=0;
    for(i=0; i<len; i++){
        if( isdigit(s[i]) ){
            nflag=1;
        }
        else if(s[i]=='.'){
            nflag=2;
            i++;
            break;
        }
        else {
            return 0;
        }
    }
    if(nflag==2){
```

```
        while(i<len){
            if(isdigit(s[i])){
                nflag=1;
            }
            else { return 0; }
            i++;
        }
    }
    if(nflag==1){printf("[num %s] ",s); return 1;}
    return 0;
}
//this function analyses all the words and find the lexemes
//this function produces expected result but returns with a error
//cause of error: unknown
int lexemes(){
    //read a file to get the lexemes
    read_file();
    while((c=fgetc(rf)) != EOF){
        int i=0;
        //read letters and store the word
        for(i=0; !(isspace(c))&& !(isoperator()) && !(isprenthesis()) && !(isseparator()); i++){
            //store the letters until there is a space, operator, parenthesis or separator
            // if isoperator function called this will store the operators
            // other function will only return a positive value or 1
            s[i] = c;
            c=fgetc(rf);
        }
        s[i]='\0';
```

```
    int len = strlen(s);
    if(len>0){
        if(iskeyword());
        else if(isidentifier());
        else if(isnumber()){}
        else if(strncmp(s, "return", 5)==0){
            printf("[ret %s] ",s);
        }
        else{
            printf("[unkn %s] ",s);
            printf("\nerror %d: invalid lexemes \"%s\" \n", err++,s);
        }
        s[0] = '\0';
    }
    if(strlen(op)>0){
        //if there is a operator stored from the call of isoperator() function in line 168
        printf("[op %s] ",op);
        op[0]='\0';
    }
    else if(isprenthesis()){
        //call the isparanthesis function and print the paranthesis
        printf("[par %c] ",c);
    }
    else if(isseparator()){
        //call the isoperator function and print the operator
        printf("[sep %c] ",c);
    }
}
```

```
    fclose(rf);

}
int main()
{
    read_file();
    //to remove all the new line, extra white spaces and comments
    plainC();
    //get the lexemes and identify them
    lexemes();
    return 0;

}
```