Name: Mihir Thakkar

Class: TY A

Roll No: 59

Srn: 201901267

<u>CD Assignment-3</u>

Design a Lexical analyzer for the subset of C Language using LEX or FLEX to lookup and also dynamically add new tokens with first word on a line indicating the token class. Upload a single file with input , output and source code.

Input Text:

arithmetic: + -

relational: >=

logical: &&

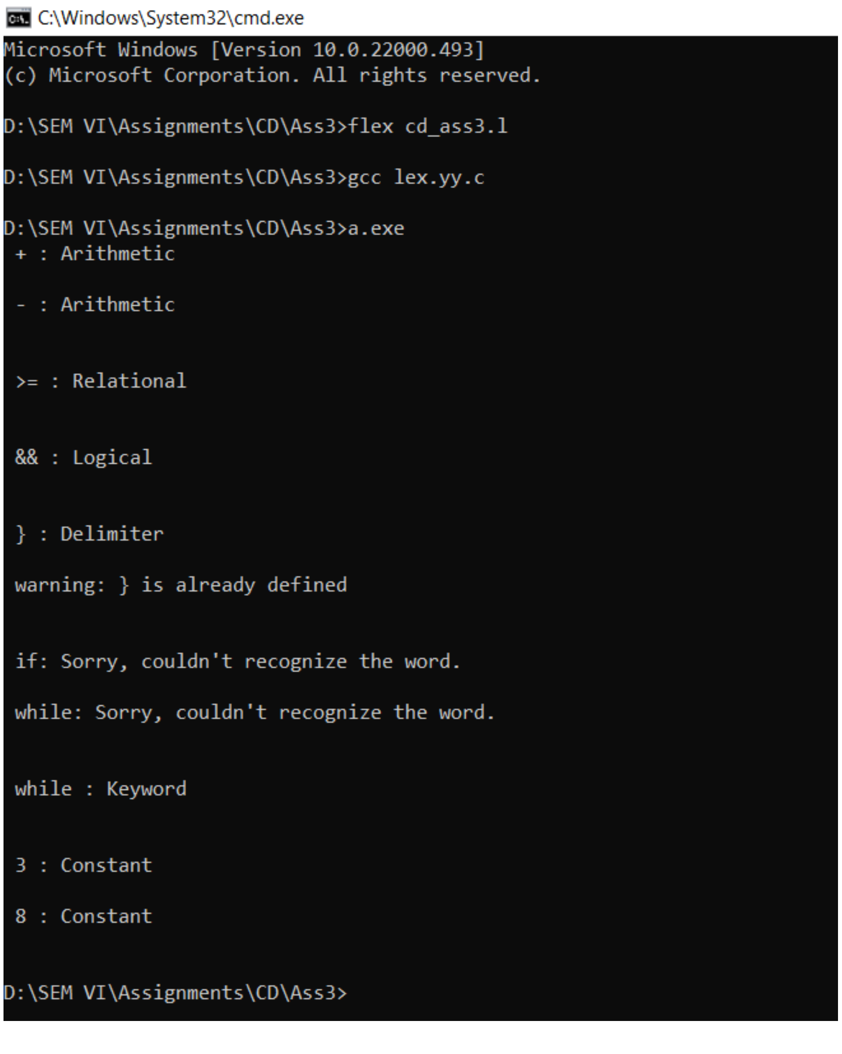
delimiter: } }

find: if while

keyword: while

constant: 3 8

Output:



C:\Windows\System32\cmd.exe

```
Microsoft Windows [Version 10.0.22000.493]
(c) Microsoft Corporation. All rights reserved.

D:\SEM VI\Assignments\CD\Ass3>flex cd_ass3.l

D:\SEM VI\Assignments\CD\Ass3>gcc lex.yy.c

D:\SEM VI\Assignments\CD\Ass3>a.exe
 + : Arithmetic

 - : Arithmetic


 >= : Relational


 && : Logical


 } : Delimiter

 warning: } is already defined


 if: Sorry, couldn't recognize the word.

 while: Sorry, couldn't recognize the word.


 while : Keyword


 3 : Constant

 8 : Constant


D:\SEM VI\Assignments\CD\Ass3>
```

Source Code (.l code):

```
%{

#include<stdio.h>
//#include<conio.h>
#include<string.h>

enum {
        LOOKUP = 0,
        KEYWORD,
        DELIMITER,
        RELATIONAL,
        ARITHMETIC,
        LOGICAL,
        ASSIGNMENT,
        CONSTANT
};

int state;

int add_word(int type, char *word);
int lookup_word(char *word);
void print(int,char*);

%}

%%

^find:   {state = LOOKUP; }

^keyword: { state = KEYWORD; }
```

```
^delimiter: { state = DELIMITER; }

^relational: { state = RELATIONAL; }

^arithmetic: { state = ARITHMETIC; }

^logical: { state = LOGICAL;}

^assignment: { state = ASSIGNMENT; }

^constant: { state = CONSTANT; }



"{"|"}"|"("|")"|";"|","|"["|"]"|"="|"=="|"<="|">="|"="|"<"|">"|"+"|"-"|"*"|"/"|"++"|"--
"|"%"|"&&"|"||"|[0-9]|[a-zA-Z]+ {

        //printf("%d",state);

          if (state != LOOKUP)

{

      if(add_word(state, yytext) == 1)

           print(state,yytext);

} else {

switch (lookup_word (yytext))

{


case KEYWORD: printf("%s: Keyword \n\n" , yytext);

break;

case DELIMITER: printf("%s: Delimiter \n\n" , yytext);

break;

case RELATIONAL: printf("%s: Relational \n\n" , yytext);

break;

case ARITHMETIC: printf("%s: Arithmetic \n\n", yytext);

break;

case LOGICAL: printf("%s: Logical \n\n" , yytext);

break;

case ASSIGNMENT: printf("%s: Assignment \n\n" , yytext);

break;
case CONSTANT: printf("%s: Constant \n\n" , yytext);
break;
default: printf("%s: Sorry, couldn't recognize the word. \n\n" , yytext);
break; }}}
%%
int yywrap()
{
        return 1;
}

struct word {
char *word_name;
int word_type;
struct word *next;
} ;
struct word *word_list;

void print(int state,char* name){

        switch(state)
        {
        case 0 : printf("Word in Lookup\n\n");
        break;
        case 1 : printf("%s : Keyword\n\n",name);
        break;
        case 2 : printf("%s : Delimiter\n\n",name);
        break;
        case 3 : printf("%s : Relational\n\n",name);
        break;
        case 4 : printf("%s : Arithmetic\n\n",name);
        break;
        case 5 : printf("%s : Logical\n\n",name);
        break;
        case 6 : printf("%s : Assignment\n\n",name);
        break;
        case 7 : printf("%s : Constant\n\n",name);
        break;
        default : printf("%s : NOT DEFINED\n\n",name);
        break;
        }
}

int main()
{
        yyin = fopen("cd_ass3_input.txt", "r");
        //printf("%s", *yyin);
        yylex();
        fclose(yyin);
}



/* first element in word list */
extern void *malloc();
int add_word(int type, char *word)
{
struct word *wp;
if(lookup_word(word) != LOOKUP)
{
//printf("%d\n\n",lookup_word(word));
printf("warning: %s is already defined \n\n" , word);
return 0;
}
```

```
wp = (struct word * ) malloc(sizeof (struct word) ) ;

wp->next=word_list;

/* have to copy the word itself as well */

wp->word_name = (char * ) malloc( strlen(word) +1) ;

strcpy (wp->word_name, word) ;

wp->word_type = type;

word_list = wp;

return 1;

}

int lookup_word (char* word)

{

struct word *wp = word_list;

/* search down the list looking for the word */

for(;wp; wp = wp->next)

        {

        if(strcmp (wp->word_name, word) == 0)

                return wp->word_type;

        }

        return LOOKUP;

        }
```