Experiment 1: Design a Lexical analyzer for the above language. The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.

#include<string.h>

#include<ctype.h>

#include<stdio.h>

void keyword(char str[10])

{

if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0||

strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||

strcmp("double",str)==0||strcmp("static",str)==0||strcmp("switch",str)==0||

strcmp("case",str)==0)

printf("\n%s is a keyword",str);

else

printf("\n%s is an identifier",str);

}

int main()

{

FILE \*f1,\*f2,\*f3;

char c,str[10],st1[10];

int num[100],lineno=0,tokenvalue=0,i=0,j=0,k=0;

printf("\nEnter the c program:\n");

//f1=fopen("input","w");

//while((c=getchar())!=EOF)

//putc(c,f1);

//fclose(f1);

f1=fopen("input","r");

f2=fopen("identifier","w");

f3=fopen("specialchar","w");

while((c=getc(f1))!=EOF)

{

if(isdigit(c))

{

tokenvalue=c-'0';

c=getc(f1);

while(isdigit(c))

{

tokenvalue\*=10+c-'0';

c=getc(f1);

}

num[i++]=tokenvalue;

ungetc(c,f1);

}

else if(isalpha(c))

{

putc(c,f2);

c=getc(f1);

while(isdigit(c)||isalpha(c)||c=='\_'||c=='$')

{

putc(c,f2);

c=getc(f1);

}

putc(' ',f2);

ungetc(c,f1);

}

else if(c==' '||c=='\t')

printf("");

else if(c=='\n')

lineno++;

else

putc(c,f3);

}

fclose(f2);

fclose(f3);

fclose(f1);

printf("\nThe no's in the program are");

for(j=0;j<i;j++)

printf("%d",num[j]);

printf("\n");

f2=fopen("identifier","r");

k=0;

printf("The keywords and identifiers are:");

while((c=getc(f2))!=EOF)

{

if(c!=' ')

str[k++]=c;

else

{

str[k]='\0';

keyword(str);

k=0;

}

}

fclose(f2);

f3=fopen("specialchar","r");

printf("\nSpecial characters are");

while((c=getc(f3))!=EOF) printf("%c",c);

printf("\n");

fclose(f3);

printf("Total no. of lines are:%d",lineno);

}

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**public** **class** Program {

**public** **static** **void** main(String[] args) **throws** IOException {

// **TODO** Auto-generated method stub

String[] words=**null**;

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));

String s;

**while**((s=br.readLine())!=**null**)

{

words=s.split(" "); //Split the word using space

**for** (String word : words)

{

**if**(*operator*(word) == **true**) {}

**else** **if**(*specialChars*(word) == **true**) {}

**else** **if**(*Keywords*(word)==**true**)

System.***out***.println(word+" is a keyword.");

**else**

System.***out***.println(word+" is a Identifier.");

}

}

}

**private** **static** **boolean** Keywords(String word)

{

**if**(word.equalsIgnoreCase("int") == **true** || word.equalsIgnoreCase("float") == **true** ||

word.equalsIgnoreCase("char") == **true** || word.equalsIgnoreCase("main") == **true** ||

word.equalsIgnoreCase("static") == **true**|| word.equalsIgnoreCase("printf") == **true**)

**return** **true**;

**return** **false**;

}

**private** **static** **boolean** identifier(String word)

{

**char** ch = word.charAt(0);

**if**((**int**)(ch) >= 0 && (**int**)(ch) <= 9)

{

**return** **false**;

}

**return** **true**;

}

**private** **static** **boolean** operator(String word)

{

**char** a = (word.length()==1) ? word.charAt(0) : '\0' ;

**char** match[]={'+','-','\*','/','=','%','^','|','<','>'};

**for**(**int** i=0;i<match.length && a!='\0';i++)

{

**if**(a == (match[i]))

{

System.***out***.println(a + " is a operator");

**return** **true**;

}

}

**return** **false**;

}

**private** **static** **boolean** specialChars(String word)

{

**char** a = (word.length()==1) ? word.charAt(0) : '\0' ;

**char** match[]={'(',')',';','{','}','[',']','&','!','%','^','|',','};

**for**(**int** i=0;i<match.length && a!='\0';i++)

{

**if**(a == (match[i]))

{

System.***out***.println(a + " is a special character");

**return** **true**;

}

}

**return** **false**;

}

}

//int main ( ) { int a , b , c = a + b ; }

---------------------------------------------------------------------------------------------------------------------------

Experiment 2: Write a C program to identify whether a given line is a comment or not.

#include<stdio.h>

#include<string.h>

int main(){

char arr[100];

int i,flag=0,len=0;

printf("Enter a string to check comments : ");

gets(arr);

len = strlen(arr);

for(i=0;i<len-1;i++){

if(arr[i] =='/' && arr[i+1] == '/' && arr[len-1]!='/' && arr[len-2]!='/' ){

flag = 1;

printf("It is a single line comment.");

break;

}

if(arr[i] =='/' && arr[i+1] == '\*'){

flag=2;

}

if (flag==2 && arr[i] =='\*' && arr[i+1] == '/' )

{

flag=1;

printf("It is a multi-line comments");

break;

}

}

if(flag == 0 || flag == 2 ){

printf("The string is not commented");

}

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 3: Write a C program to recognize strings under 'a', 'a\*b+', 'abb'

#include<stdio.h>

#include<string.h>

#include<stdlib.h>

void main()

{

char s[20],c;

int state=0,i=0;

printf("\n Enter a string:");

scanf("%s",s);

while(s[i]!='\0')

{

switch(state)

{

case 0: c=s[i++];

if(c=='a')

state=1 ;

else if(c=='b')

state=2;

else

state=6;

break ;

case 1: c=s[i++];

if(c=='a')

state=3 ;

else if(c=='b')

state=4 ;

else

state=6 ;

break;

case 2: c=s[i++];

if(c=='a')

state=6 ;

else if(c=='b')

state=2 ;

else

state=6 ;

break;

case 3: c=s[i++];

if(c=='a')

state=3 ;

else if(c=='b')

state=2 ;

else

state=6 ;

break;

case 4: c=s[i++];

if(c=='a')

state=6 ;

else if(c=='b')

state=5 ;

else

state=6 ;

break;

case 5: c=s[i++];

if(c=='a')

state=6 ;

else if(c=='b')

state=2;

else

state=6 ;

break;

case 6:

printf("\n %s is not recognised.",s);

}

}

if(state==1)

printf("\n %s is accepted under rule 'a'",s) ;

else if((state==2)||(state==4))

printf("\n %s is accepted under rule 'a\*b+'",s);

else if(state==5)

printf("\n %s is accepted under rule 'abb'",s) ;

else

printf("\n %s will not accept any rule.",s);

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 4: Write a C program to test whether a given identifier is valid or not.

#include<stdio.h>

#include <ctype.h>

void main()

{

char a[10];

int flag, i=1;

printf("\n Enter an identifier:");

scanf("%s",a);

if(isalpha(a[0]))

flag=1;

else

printf("\n Not a valid identifier");

while(a[i]!='\0') {

if(!isdigit(a[i]) && (!isalpha(a[i])))

{

flag=0;

break;

}

i++ ;

}

if(flag==1)

printf("\n Valid identifier");

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 5: Write a C program to simulate lexical analyzer for validating operators.

#include<stdio.h>

void main()

{

char s[5];

printf("\n Enter any operator:");

scanf("%s",s);

switch(s[0])

{

case'>':

if(s[1]=='=')

printf("\n Greater than or equal");

else

printf("\n Greater than");

break;

case'<':

if(s[1]=='=')

printf("\n Less than or equal");

else

printf("\nLess than");

break;

case'=':

if(s[1]=='=')

printf("\nEqual to");

else

printf("\nAssignment");

break;

case'!':

if(s[1]=='=')

printf("\nNot Equal");

else

printf("\n Bit Not");

break;

case'&':

if(s[1]=='&')

printf("\nLogical AND");

else

printf("\n Bitwise AND");

break;

case'|':

if(s[1]=='|')

printf("\nLogical OR") ;

else

printf("\nBitwise OR");

break;

case'+':

printf("\n Addition") ;

break;

case'-':

printf("\nSubstraction");

break;

case'\*':

printf("\nMultiplication") ;

break ;

case'/':

printf("\nDivision");

break;

case'%':

printf("Modulus");

break ;

default:

printf("\n Not a operator");

}

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 6: Implement the lexical analyzer using JLex, flex or lex or other lexical analyzer generating tools.

DIGIT [0-9]

ID [a-z][a-z0-9]\*

%%

char|int|float|double|main|void|real|printf|for|while|do

printf("keyword=%s\n",yytext);

{DIGIT} printf("interger %s",yytext);

{ID}+ printf("identifier %s",yytext);

{DIGIT}+"."{DIGIT}\* printf("floating point number:%s\n",yytext);

[=|!=|>=|<=|<|>|+|-|\*|/] printf("operator %s",yytext);

[(] printf("openparenthesis %s",yytext);

[)] printf("closeparenthesis %s",yytext);

[{] printf("open bbrace is %s",yytext);

[}] printf("close braceis %s",yytext);

[;] printf("eos %s",yytext);

[\t\n]+ /\*eat up whitesapaces \*/

["{"|^}\n]\*"}"] /\*eat up comments \*/

["|,|.] printf("punctaation :%s ",yytext);

%%

main()

{

yylex();

}

int yywrap()

{

return 1;

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 7: Write a C program for constructing recursive descent parsing.

#include<stdio.h>

char input[20];

int i=0;

void main()

{

int n;

display();

printf("\nEnter the input string:");

scanf("%s",input);

E();

printf("\nThe given input string is parsed.");

}

E( )

{

T( );

E1( );

}

E1( )

{

if(input[i]=='+')

{

i++;

T();

E1();

}

}

T( )

{

F( );

T1( );

}

T1( )

{

if(input[i]=='\*')

{

i++;

F( );

T1( );

}

}

F( )

{

if(input[i]=='(')

{

i++;

E();

if(input[i]==')')

i++;

else

error( );

}

else if(input[i]=='i')

i++;

else

error( );

}

error( )

{

printf("\nThe given input string is not parsed.");

}

display( )

{

printf("\nGrammar is:");

printf("\nE->TE1\nE1->+TE1/@\nT->FT1\nT1->\*FT1/@\nF- >(E)/i");

printf("\nNonTerminals:E,E1,T,T1,F");

printf("\nterminals:+,\*,(,),i\nEpsilon symbol- @Start symbol-E\n");

return;

}

---------------------------------------------------------------------------------------------------------------------------

Experiment 8 : Write a C program to implement operator precedence parsing.

#include<stdio.h>

#include<string.h>

void main()

{

char stack[20],ip[20],opt[10][10][1],ter[10];

int i,j,k,n,top=0,col,row;

for(i=0;i<10;i++)

{

stack[i]=NULL;

ip[i]=NULL;

for(j=0;j<10;j++)

{

opt[i][j][1]=NULL;

}

}

printf("Enter the no.of terminals :\n");

scanf("%d",&n);

printf("\nEnter the terminals :\n");

scanf("%s",&ter);

printf("\nEnter the table values :\n");

for(i=0;i<n;i++)

{

for(j=0;j<n;j++)

{

printf("Enter the value for %c %c:",ter[i],ter[j]);

scanf("%s",opt[i][j]);

}

}

printf("\n\*\*\*\* OPERATOR PRECEDENCE TABLE \*\*\*\*\n");

for(i=0;i<n;i++)

{

printf("\t%c",ter[i]);

}

printf("\n");

for(i=0;i<n;i++){printf("\n%c",ter[i]);

for(j=0;j<n;j++){printf("\t%c",opt[i][j][0]);}}

stack[top]='$';

printf("\nEnter the input string:");

scanf("%s",ip);

i=0;

printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");

printf("\n%s\t\t\t%s\t\t\t",stack,ip);

while(i<=strlen(ip))

{

for(k=0;k<n;k++)

{

if(stack[top]==ter[k])

col=k;

if(ip[i]==ter[k])

row=k;

}

if((stack[top]=='$')&&(ip[i]=='$')){

printf("String is accepted\n");

break;}

else if((opt[col][row][0]=='<') ||(opt[col][row][0]=='='))

{ stack[++top]=opt[col][row][0];

stack[++top]=ip[i];

printf("Shift %c",ip[i]);

i++;

}

else{

if(opt[col][row][0]=='>')

{

while(stack[top]!='<'){--top;}

top=top-1;

printf("Reduce");

}

else

{

printf("\nString is not accepted");

break;

}

}

printf("\n");

for(k=0;k<=top;k++)

{

printf("%c",stack[k]);

}

printf("\t\t\t");

for(k=i;k<strlen(ip);k++){

printf("%c",ip[k]);

}

printf("\t\t\t");

}

}

---------------------------------------------------------------------------------------------------------------------------