Ex.3 Evaluation of Arithmetic expression using Ambiguous Grammar(Use Lex and Yacc Tool)

**E-> E+E | E-E|E\*E | E/E| (E) | id**

**Lex file:**

%option noyywrap

%{

#include<stdio.h>

#include"y.tab.h"

void yyerror(char \*s);

extern int yylval;

%}

digit [0-9]

%%

{digit}+ {yylval=atoi(yytext);return NUM;}

[-+\*/\n] {return \*yytext;}

\( {return \*yytext;}

\) {return \*yytext;}

. {yyerror("syntax error");}

%%

**YACC file:**

%{

#include<stdio.h>

void yyerror(char\*);

extern int yylex(void);

%}

%token NUM

%%

S:

S E '\n' {printf("%d\n",$2);}

|

;

E:

E '+' E {$$=$1+$3;}

|E '-' E {$$=$1-$3;}

|E '\*' E {$$=$1\*$3;}

|E '/' E {$$=$1/$3;}

|'(' E ')' {$$=$2;}

|NUM {$$=$1;}

%%

void yyerror(char \*s)

{

printf("%s",s);

}

int main()

{

yyparse();

return 0;

}

Ex.4 Evaluation of Arithmetic expression using Unmbiguous Grammar(Use Lex and Yacc Tool)

**E-> E+T | E-T|T**

**T->T\*F | T/F|F**

**F-> (E) | id**

**Lex File:**

%option noyywrap

%{

#include<stdio.h>

#include"y.tab.h"

void yyerror(char \*s);

extern int yylval;

%}

digit [0-9]

%%

{digit}+ {yylval=atoi(yytext);return NUM;}

[-+\*/\n] {return \*yytext;}

\( {return \*yytext;}

\) {return \*yytext;}

. {yyerror("syntax error");}

%**%**

**YACC file:**

%{

#include<stdio.h>

void yyerror(char\*);

extern int yylex(void);

%}

%token NUM

%%

S:

S E '\n' {printf("%d\n",$2);}

|

;

E:

E '+' T {$$=$1+$3;}

|E '-' T {$$=$1-$3;}

|T {$$=$1;}

T:

T '\*' F {$$=$1\*$3;}

|T '/' F {$$=$1/$3;}

|F {$$=$1;}

F:

'(' E ')' {$$=$2;}

|NUM {$$=$1;}

%%

void yyerror(char \*s)

{

printf("%s",s);

}

int main()

{

yyparse();

return 0;

}

**Ex.5 Use LEX and YACC tool to implement Desktop Calculator.**

**E-> E+T | E-T|T**

**T->T\*F | T/F|F**

**F-> (E) | id**

**Lex File:**

%option noyywrap

%{

#include<stdio.h>

#include"y.tab.h"

void yyerror(char \*s);

extern int yylval;

%}

digit [0-9]

%%

{digit}+ {yylval=atoi(yytext);return NUM;}

[a-z] {yylval=toascii(\*yytext)-97;return ID;}

[A-Z] {yylval=toascii(\*yytext)-65;return ID;}

[-+\*/=\n] {return \*yytext;}

\( {return \*yytext;}

\) {return \*yytext;}

. {yyerror("syntax error");}

%%

**YACC file:**

%{

#include<stdio.h>

void yyerror(char\*);

extern int yylex(void);

int val[26];

%}

%token NUM ID

%%

S:

S E '\n' {printf("%d\n",$2);}

| S ID '=' E '\n' {val[$2]=$4;}

|

;

E:

E '+' T {$$=$1+$3;}

|E '-' T {$$=$1-$3;}

|T {$$=$1;}

T:

T '\*' F {$$=$1\*$3;}

|T '/' F {$$=$1/$3;}

|F {$$=$1;}

F:

'(' E ')' {$$=$2;}

|NUM {$$=$1;}

|ID {$$=val[$1];}

%%

void yyerror(char \*s)

{

printf("%s",s);

}

int main()

{

yyparse();

return 0;

}

**Ex. No. 6 RECURSIVE DESCENT PARSING**

#include<stdio.h>

#include<conio.h>

#include<string.h>

int i=0 ,f=0;

char str[30];

void E();

void Eprime();

void T();

void Tprime();

void F();

void E()

{

printf("\nE->TE'");

T();

Eprime();

}

void Eprime()

{

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

{

printf("\n\E'->+TE'");

i++;

T();

Eprime();

}

else if((str[i]==')')||(str[i]==’$’))

printf("\nE'->^");

}

void T()

{

printf("\nT->FT'");

F();

Tprime();

}

void Tprime()

{

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

{

printf("\nT'->\*FT'");

i++;

F();

Tprime();

}

else if((str[i]==')')||(str[i]==’+’)||(str[i]==’$’))

printf("\nT'->^");

}

void F()

{

if(str[i]=='a')

{

printf("\nF->a");

i++;

}

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

{

printf("\nF->(E)");

i++;

E();

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

i++;

}

else

f=1;

}

void main()

{

int len;

clrscr();

printf("Enter the string: ");

scanf("%s",str);

len=strlen(str);

str[len]='$';

E();

if((str[i]=='$')&&(f==0))

printf("\nStringsucessfully parsed!");

else

printf("\nSyntax Error!");

getch();

}

**Output 1**

Enter the string: a+a\*a

E->TE'

T->FT'

F->a

T'->^

E'->+TE'

T->FT'

F->a

T'->\*FT'

F->a

T'->^

E'->^

String Sucessfully parsed!

**Output 2**

Enter the string: a++

E->TE'

T->FT'

F->a

T'->^

E'->+TE'

T->FT'

T'->^

E'->+TE'

T->FT'

T'->^

E'->^

Syntax Error!

**Ex.No. 7-SHIFT REDUCE PARSER**

Source code:

#include<stdio.h>

#include<string.h>

int z,i,j,c;

char a[16],stk[15];

void reduce();

void main()

{

puts("GRAMMAR is E->E+E \n E->E\*E \n E->(E) \n E->a");

puts("enter input string ");

gets(a);

c=strlen(a);

a[c]='$';

stk[0]='$';

puts("stack \t input \t action");

for(i=1,j=0;j<c; i++,j++)

{

if(a[j]=='a')

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n%s\t\t%s\tshift->a",stk,a);

reduce();

}

else

{

stk[i]=a[j];

stk[i+1]='\0';

a[j]=' ';

printf("\n%s\t\t%s\tshift->%c",stk,a,stk[i]);

reduce();

}

}

if(a[j]=='$')

reduce();

if((stk[1]=='E')&&(stk[2]=='\0'))

printf("\n%s\t\t%s\tAccept",stk,a);

else

printf("\n%s\t\t%s\terror",stk,a);

}

void reduce()

{

for(z=1; z<=c; z++)

if(stk[z]=='a')

{

stk[z]='E';

stk[z+1]='\0';

printf("\n%s\t\t%s\tReduce by E->a",stk,a);

}

for(z=1; z<=c; z++)

if(stk[z]=='E' &&stk[z+1]=='+' &&stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n%s\t\t%s\tReduce by E->E+E",stk,a);

i=i-2;

}

for(z=1; z<=c; z++)

if(stk[z]=='E' &&stk[z+1]=='\*' &&stk[z+2]=='E')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n%s\t\t%s\tReduce by E->E\*E",stk,a);

i=i-2;

}

for(z=1; z<=c; z++)

if(stk[z]=='(' &&stk[z+1]=='E' &&stk[z+2]==')')

{

stk[z]='E';

stk[z+1]='\0';

stk[z+2]='\0';

printf("\n%s\t\t%s\tReduce by E->(E)",stk,a);

i=i-2;

}

}

Output 1:

GRAMMAR is E->E+E

E->E\*E

E->(E)

E->a

enter input string

a+a\*a

stack input action

$a +a\*a$ shift->a

$E +a\*a$ Reduce by E->a

$E+ a\*a$ shift->+

$E+a \*a$ shift->a

$E+E \*a$ Reduce by E->a

$E \*a$ Reduce by E->E+E

$E\* a$ shift->\*

$E\*a $ shift->a

$E\*E $ Reduce by E->a

$E $ Reduce by E->E\*E

$E $ Accept

**Ex. No. 8 Implement Operator Precedence Parser algorithm.**

**Algorithm:**

*Initialize*: Set *ip* to point to the first symbol of *w*$

*repeat*:

Let *X* be the top stack symbol, and ***a***the symbol pointed to by *ip*

**if** $ is on the top of the stack and ip points to $ **then return**

**else**

Let *a* be the top terminal on the stack, and *b* the symbol pointed to by *ip*

**if** *a* <· *b* **or** *a* =· *b* **then**

push *b* onto the stack

advance *ip* to the next input symbol

**else if** *a* ·> *b* **then**

# repeat

pop the stack

**until** the top stack terminal is related by <·

to the terminal most recently popped

**else** *error()*

# end

**Source code:**

#include<stdio.h>

#include<conio.h>

void main()

{

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

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

clrscr();

for(i=0;i<3;i++)

{

stack[i]=NULL;

ip[i]=NULL;

for(j=0;j<3;j++)

{

opt[i][j][0]=NULL;

}

}

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

scanf("%d",&n);

printf("\nEnter the terminals:");

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("\nOPERATOR 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\tINPUT STRING\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])

row=k;

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

col=k;

}

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

{

printf("String is accepted");

break;

}

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

{

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

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

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

i++;

}

else

{

if(opt[row][col][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");

}

getch();

}

Output 1:

Enter the no.of terminals:3

Enter the terminals:a+$

Enter the table values:

Enter the value for a a:e

Enter the value for a +:>

Enter the value for a $:>

Enter the value for + a:<

Enter the value for + +:>

Enter the value for + $:>

Enter the value for $ a:<

Enter the value for $ +:<

Enter the value for $ $:A

OPERATOR PRECEDENCE TABLE:

a + $

a e > >

+ < > >

$ < < A

Enter the input string:a+a$

STACK INPUT STRING ACTION

$ a+a$ Shift a

$<a +a$ Reduce

$ +a$ Shift +

$<+ a$ Shift a

$<+<a $ Reduce

$<+ $ Reduce

$ $ String is accepted