**Compiler all Experiments**

1. **Lex Program**

%{  
int count = 0;  
%}  
%%  
[A-Z] {printf("%s is a capital letter",yytext);  
count++;  
}  
. {printf("%s is not a capital letter",yytext);}  
\n {return 0;}  
%%  
int yywrap(){}  
int main()  
{  
yylex();  
printf("%d letters are capital", count);  
return 0;  
}

**2 . Lex Program for Calculator**

%{  
#include <stdio.h>  
#include <stdlib.h>  
  
int op = 0, i;  
float a, b;  
  
void digi();  
%}  
  
dig [0-9]+|([0-9]\*)"."([0-9]+)  
add "+"  
sub "-"  
mul "\*"  
div "/"  
pow "^"  
ln \n  
  
%%  
  
{dig} {digi();}  
{add} {op = 1;}  
{sub} {op = 2;}  
{mul} {op = 3;}  
{div} {op = 4;}  
{pow} {op = 5;}  
{ln} {printf("\n The Answer :%f\n\n", a);}  
  
  
%%  
  
void digi()  
{  
    if (op == 0)  
        a = atof(yytext);  
    else  
    {  
        b = atof(yytext);  
  
        switch (op)  
        {  
        case 1:  
            a = a + b;  
            break;  
  
        case 2:  
            a = a - b;  
            break;  
  
        case 3:  
            a = a \* b;  
            break;  
  
        case 4:  
            a = a / b;  
            break;  
  
        case 5:  
            for (i = a; b > 1; b--)  
                a = a \* i;  
            break;  
        }  
        op = 0;  
    }  
}  
  
int main(int argc, char \*argv[])  
{  
    yylex();  
    return 0;  
}  
  
int yywrap()  
{  
    return 1;  
}

**Q..Lex Program taking input from File**

%{  
#include <stdio.h>  
int countCapital = 0;  
int countDigits = 0;  
%}  
  
%%  
[A-Z] {  
    printf("%s is a capital letter\n", yytext);  
    countCapital++;  
}  
  
[0-9] {  
    printf("%s is a digit\n", yytext);  
    countDigits++;  
}  
  
. {  
    printf("%s is not a capital letter or digit\n", yytext);  
}  
  
\n {  
    printf("Total capital letters: %d\n", countCapital);  
    printf("Total digits: %d\n", countDigits);  
    return 0;  
}  
%%  
  
int yywrap() {}  
  
int main(int argc, char \*argv[]) {  
     
    FILE \*inputFile = fopen("input.txt", "r");  
     
  
    yyin = inputFile; // Set the file pointer for flex to read from  
  
    yylex();  
  
    fclose(inputFile);  
    return 0;  
}

**Commands**

~$ lex SimpleLex.l

~$ gcc lex.yy.c

~$ ./a.out

**4. Calculator Using Lex and Yacc**

**Lex Code**

%{

#include<stdio.h>

#include "y.tab.h"

extern int yylval;

%}

%%

[0-9]+ {

          yylval=atoi(yytext);

          return NUMBER;

       }

[\t] ;

[\n] return 0;

. return yytext[0];

%%

int yywrap()

{

return 1;

}

**Yacc Code**

%{

    #include<stdio.h>

    int flag=0;

%}

%token NUMBER

%left '+' '-'

%left '\*' '/' '%'

%left '(' ')'

%%

ArithmeticExpression: E{

         printf("\nResult=%d\n",$$);

         return 0;

        };

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

 |E'-'E {$$=$1-$3;}

 |E'\*'E {$$=$1\*$3;}

 |E'/'E {$$=$1/$3;}

 |E'%'E {$$=$1%$3;}

 |'('E')' {$$=$2;}

 | NUMBER {$$=$1;}

;

%%

void main()

{

   printf("\nEnter Any Arithmetic Expression which can have operations Addition, Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");

   yyparse();

  if(flag==0)

   printf("\nEntered arithmetic expression is Valid\n\n");

}

void yyerror()

{

   printf("\nEntered arithmetic expression is Invalid\n\n");

   flag=1;

}

**Commands**

**yacc -d filename.y**

**lex filename.l**

**gcc lex.yy.c y.tab.c -w**

**./a.out**

**5. Code for removing Left Recursion**

#include<stdio.h>

#include<string.h>

void main() {

char input[100],l[50],r[50],temp[10],tempprod[20],productions[25][50];

int i=0,j=0,flag=0,consumed=0;

printf("Enter the productions: ");

scanf("%1s->%s",l,r);

printf("%s",r);

while(sscanf(r+consumed,"%[^|]s",temp) == 1 && consumed <= strlen(r)) {

if(temp[0] == l[0]) {

flag = 1;

sprintf(productions[i++], "%s->%s%s'", l, temp+1, l);

}

else

sprintf(productions[i++], "%s'->%s%s'", l, temp, l);

consumed += strlen(temp)+1;

}

if(flag == 1) {

sprintf(productions[i++], "%s->ε", l);

printf("The productions after eliminating Left Recursion are:\n");

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

printf("%s\n",productions[j]);

}

else

printf("The Given Grammar has no Left Recursion");

}

Commands:

gcc Sample.c

~$ ./a.out

Input: E->E+E/T

**6. Left Factoring**

#include<stdio.h>

#include<string.h>

int main()

{

char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];

int i,j=0,k=0,l=0,pos;

printf("Enter Production : A->");

gets(gram);

for(i=0;gram[i]!='|';i++,j++)

part1[j]=gram[i];

part1[j]='\0';

for(j=++i,i=0;gram[j]!='\0';j++,i++)

part2[i]=gram[j];

part2[i]='\0';

for(i=0;i<strlen(part1)||i<strlen(part2);i++)

{

if(part1[i]==part2[i])

{

modifiedGram[k]=part1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;part1[i]!='\0';i++,j++){

newGram[j]=part1[i];

}

newGram[j++]='|';

for(i=pos;part2[i]!='\0';i++,j++){

newGram[j]=part2[i];

}

modifiedGram[k]='X';

modifiedGram[++k]='\0';

newGram[j]='\0';

printf("\n A->%s",modifiedGram);

printf("\n X->%s\n",newGram);

}

**Commands And Input**

~$ gcc Sample.c

~$ ./a.out

Enter Production : A->aE+bcD|aE+eIT

A->aE+X

X->bcD|eIT

**First and Follow**

// C program to calculate the First and

// Follow sets of a given grammar

#include <ctype.h>

#include <stdio.h>

#include <string.h>

// Functions to calculate Follow

void followfirst(char, int, int);

void follow(char c);

// Function to calculate First

void findfirst(char, int, int);

int count, n = 0;

// Stores the final result

// of the First Sets

char calc\_first[10][100];

// Stores the final result

// of the Follow Sets

char calc\_follow[10][100];

int m = 0;

// Stores the production rules

char production[10][10];

char f[10], first[10];

int k;

char ck;

int e;

int main(int argc, char\*\* argv)

{

int jm = 0;

int km = 0;

int i, choice;

char c, ch;

count = 8;

// The Input grammar

strcpy(production[0], "X=TnS");

strcpy(production[1], "X=Rm");

strcpy(production[2], "T=q");

strcpy(production[3], "T=#");

strcpy(production[4], "S=p");

strcpy(production[5], "S=#");

strcpy(production[6], "R=om");

strcpy(production[7], "R=ST");

int kay;

char done[count];

int ptr = -1;

// Initializing the calc\_first array

for (k = 0; k < count; k++) {

for (kay = 0; kay < 100; kay++) {

calc\_first[k][kay] = '!';

}

}

int point1 = 0, point2, xxx;

for (k = 0; k < count; k++) {

c = production[k][0];

point2 = 0;

xxx = 0;

// Checking if First of c has

// already been calculated

for (kay = 0; kay <= ptr; kay++)

if (c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

// Function call

findfirst(c, 0, 0);

ptr += 1;

// Adding c to the calculated list

done[ptr] = c;

printf("\n First(%c) = { ", c);

calc\_first[point1][point2++] = c;

// Printing the First Sets of the grammar

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

int lark = 0, chk = 0;

for (lark = 0; lark < point2; lark++) {

if (first[i] == calc\_first[point1][lark]) {

chk = 1;

break;

}

}

if (chk == 0) {

printf("%c, ", first[i]);

calc\_first[point1][point2++] = first[i];

}

}

printf("}\n");

jm = n;

point1++;

}

printf("\n");

printf("-----------------------------------------------"

"\n\n");

char donee[count];

ptr = -1;

// Initializing the calc\_follow array

for (k = 0; k < count; k++) {

for (kay = 0; kay < 100; kay++) {

calc\_follow[k][kay] = '!';

}

}

point1 = 0;

int land = 0;

for (e = 0; e < count; e++) {

ck = production[e][0];

point2 = 0;

xxx = 0;

// Checking if Follow of ck

// has already been calculated

for (kay = 0; kay <= ptr; kay++)

if (ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

// Function call

follow(ck);

ptr += 1;

// Adding ck to the calculated list

donee[ptr] = ck;

printf(" Follow(%c) = { ", ck);

calc\_follow[point1][point2++] = ck;

// Printing the Follow Sets of the grammar

for (i = 0 + km; i < m; i++) {

int lark = 0, chk = 0;

for (lark = 0; lark < point2; lark++) {

if (f[i] == calc\_follow[point1][lark]) {

chk = 1;

break;

}

}

if (chk == 0) {

printf("%c, ", f[i]);

calc\_follow[point1][point2++] = f[i];

}

}

printf(" }\n\n");

km = m;

point1++;

}

}

void follow(char c)

{

int i, j;

// Adding "$" to the follow

// set of the start symbol

if (production[0][0] == c) {

f[m++] = '$';

}

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

for (j = 2; j < 10; j++) {

if (production[i][j] == c) {

if (production[i][j + 1] != '\0') {

// Calculate the first of the next

// Non-Terminal in the production

followfirst(production[i][j + 1], i,

(j + 2));

}

if (production[i][j + 1] == '\0'

&& c != production[i][0]) {

// Calculate the follow of the

// Non-Terminal in the L.H.S. of the

// production

follow(production[i][0]);

}

}

}

}

}

void findfirst(char c, int q1, int q2)

{

int j;

// The case where we

// encounter a Terminal

if (!(isupper(c))) {

first[n++] = c;

}

for (j = 0; j < count; j++) {

if (production[j][0] == c) {

if (production[j][2] == '#') {

if (production[q1][q2] == '\0')

first[n++] = '#';

else if (production[q1][q2] != '\0'

&& (q1 != 0 || q2 != 0)) {

// Recursion to calculate First of New

// Non-Terminal we encounter after

// epsilon

findfirst(production[q1][q2], q1,

(q2 + 1));

}

else

first[n++] = '#';

}

else if (!isupper(production[j][2])) {

first[n++] = production[j][2];

}

else {

// Recursion to calculate First of

// New Non-Terminal we encounter

// at the beginning

findfirst(production[j][2], j, 3);

}

}

}

}

void followfirst(char c, int c1, int c2)

{

int k;

// The case where we encounter

// a Terminal

if (!(isupper(c)))

f[m++] = c;

else {

int i = 0, j = 1;

for (i = 0; i < count; i++) {

if (calc\_first[i][0] == c)

break;

}

// Including the First set of the

// Non-Terminal in the Follow of

// the original query

while (calc\_first[i][j] != '!') {

if (calc\_first[i][j] != '#') {

f[m++] = calc\_first[i][j];

}

else {

if (production[c1][c2] == '\0') {

// Case where we reach the

// end of a production

follow(production[c1][0]);

}

else {

// Recursion to the next symbol

// in case we encounter a "#"

followfirst(production[c1][c2], c1,

c2 + 1);

}

}

j++;

}

}

}

**Left Factoring**

#include<stdio.h>

#include<string.h>

int main()

{

char gram[20],part1[20],part2[20],modifiedGram[20],newGram[20],tempGram[20];

int i,j=0,k=0,l=0,pos;

printf("Enter Production : A->");

gets(gram);

for(i=0;gram[i]!='|';i++,j++)

part1[j]=gram[i];

part1[j]='\0';

for(j=++i,i=0;gram[j]!='\0';j++,i++)

part2[i]=gram[j];

part2[i]='\0';

for(i=0;i<strlen(part1)||i<strlen(part2);i++)

{

if(part1[i]==part2[i])

{

modifiedGram[k]=part1[i];

k++;

pos=i+1;

}

}

for(i=pos,j=0;part1[i]!='\0';i++,j++){

newGram[j]=part1[i];

}

newGram[j++]='|';

for(i=pos;part2[i]!='\0';i++,j++){

newGram[j]=part2[i];

}

modifiedGram[k]='X';

modifiedGram[++k]='\0';

newGram[j]='\0';

printf("\n A->%s",modifiedGram);

printf("\n X->%s\n",newGram);

}

**Commands**

~$ gcc Sample.c

~$ ./a.out

**Predictive Parsing Table**

#include<iostream>

#include<string>

#include<deque>

using namespace std;

int n,n1,n2;

int getPosition(string arr[], string q, int size)

{

   for(int i=0;i<size;i++)

   {

       if(q == arr[i])

           return i;

   }

   return -1;

}

int main()

{

   string prods[10],first[10],follow[10],nonterms[10],terms[10];

   string pp\_table[20][20] = {};

   cout<<"Enter the number of productions : ";

   cin>>n;

   cin.ignore();

   cout<<"Enter the productions"<<endl;

   for(int i=0;i<n;i++)

   {

       getline(cin,prods[i]);

       cout<<"Enter first for "<<prods[i].substr(3)<<" : ";

       getline(cin,first[i]);

   }

   cout<<"Enter the number of Terminals : ";

   cin>>n2;

   cin.ignore();

   cout<<"Enter the Terminals"<<endl;

   for(int i=0;i<n2;i++)

   {

       cin>>terms[i];

   }

   terms[n2] = "$";

   n2++;

   cout<<"Enter the number of Non-Terminals : ";

   cin>>n1;

   cin.ignore();

   for(int i=0;i<n1;i++)

   {

       cout<<"Enter Non-Terminal : ";

       getline(cin,nonterms[i]);

       cout<<"Enter follow of "<<nonterms[i]<<" : ";

       getline(cin,follow[i]);

   }

   cout<<endl;

   cout<<"Grammar"<<endl;

   for(int i=0;i<n;i++)

   {

       cout<<prods[i]<<endl;

   }

   for(int j=0;j<n;j++)

   {

       int row = getPosition(nonterms,prods[j].substr(0,1),n1);

       if(prods[j].at(3)!='#')

       {

           for(int i=0;i<first[j].length();i++)

           {

               int col = getPosition(terms,first[j].substr(i,1),n2);

               pp\_table[row][col] = prods[j];

           }

       }

       else

       {

           for(int i=0;i<follow[row].length();i++)

           {

               int col = getPosition(terms,follow[row].substr(i,1),n2);

               pp\_table[row][col] = prods[j];

           }

       }

   }

   //Display Table

   for(int j=0;j<n2;j++)

       cout<<"\t"<<terms[j];

   cout<<endl;

   for(int i=0;i<n1;i++)

   {

           cout<<nonterms[i]<<"\t";

           //Display Table

           for(int j=0;j<n2;j++)

           {

               cout<<pp\_table[i][j]<<"\t";

           }

           cout<<endl;

   }

   //Parsing String

   char c;

   do{

   string ip;

   deque<string> pp\_stack;

   pp\_stack.push\_front("$");

   pp\_stack.push\_front(prods[0].substr(0,1));

   cout<<"Enter the string to be parsed : ";

   getline(cin,ip);

   ip.push\_back('$');

   cout<<"Stack\tInput\tAction"<<endl;

   while(true)

   {

       for(int i=0;i<pp\_stack.size();i++)

           cout<<pp\_stack[i];

       cout<<"\t"<<ip<<"\t";

       int row1 = getPosition(nonterms,pp\_stack.front(),n1);

       int row2 = getPosition(terms,pp\_stack.front(),n2);

       int column = getPosition(terms,ip.substr(0,1),n2);

       if(row1 != -1 && column != -1)

       {

           string p = pp\_table[row1][column];

           if(p.empty())

           {

               cout<<endl<<"String cannot be Parsed."<<endl;

               break;

           }

           pp\_stack.pop\_front();

           if(p[3] != '#')

           {

               for(int x=p.size()-1;x>2;x--)

               {

                   pp\_stack.push\_front(p.substr(x,1));

               }

           }

           cout<<p;

       }

       else

       {

           if(ip.substr(0,1) == pp\_stack.front())

           {

               if(pp\_stack.front() == "$")

               {

                   cout<<endl<<"String Parsed."<<endl;

                   break;

               }

               cout<<"Match "<<ip[0];

               pp\_stack.pop\_front();

               ip = ip.substr(1);

           }

           else

           {

               cout<<endl<<"String cannot be Parsed."<<endl;

               break;

           }

       }

       cout<<endl;

   }

   cout<<"Continue?(Y/N) ";

   cin>>c;

   cin.ignore();

   }while(c=='y' || c=='Y');

   return 0;

}

**Commands:**

**~$ g++ Sample.cpp**

**~$ ./a.out**

Enter the Input

Enter the number of productions : 5

Enter the productions

S->aXYb

Enter first for aXYb : a

X->c

Enter first for c : c

X->#

Enter first for # : #

Y->d

Enter first for d : d

Y->#

Enter first for # : #

Enter the number of Terminals : 4

Enter the Terminals

a

b

c

d

Enter the number of Non-Terminals : 3

Enter Non-Terminal : S

Enter follow of S : $

Enter Non-Terminal : X

Enter follow of X : bd

Enter Non-Terminal : Y

Enter follow of Y : b

Grammar

S->aXYb

X->c

X->#

Y->d

Y->#

a b c d $

S S->aXYb

X X-># X->c X->#

Y Y-># Y->d

Enter the string to be parsed : acdb

Stack Input Action

S$ acdb$ S->aXYb

aXYb$ acdb$ Match a

XYb$ cdb$ X->c

cYb$ cdb$ Match c

Yb$ db$ Y->d

db$ db$ Match d

b$ b$ Match b

$ $

String Parsed.

**Intermediate Code Generation**

#include<stdio.h>

#include<ctype.h>

#include<stdlib.h>

#include<string.h>

void small();

void dove(int i);

int p[5]={0,1,2,3,4},c=1,i,k,l,m,pi;

char sw[5]={'=','-','+','/','\*'},j[20],a[5],b[5],ch[2];

void main()

{

printf("Enter the expression:");

scanf("%s",j);

printf("\tThe Intermediate code is:\n");

small();

}

void dove(int i)

{

a[0]=b[0]='\0';

if(!isdigit(j[i+2])&&!isdigit(j[i-2]))

{

a[0]=j[i-1];

b[0]=j[i+1];

}

if(isdigit(j[i+2])){

a[0]=j[i-1];

b[0]='t';

b[1]=j[i+2];

}

if(isdigit(j[i-2]))

{

b[0]=j[i+1];

a[0]='t';

a[1]=j[i-2];

b[1]='\0';

}

if(isdigit(j[i+2]) &&isdigit(j[i-2]))

{

a[0]='t';

b[0]='t';

a[1]=j[i-2];

b[1]=j[i+2];

sprintf(ch,"%d",c);

j[i+2]=j[i-2]=ch[0];

}

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

printf("\tt%d=%s\*%s\n",c,a,b);

if(j[i]=='/')

printf("\tt%d=%s/%s\n",c,a,b);

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

printf("\tt%d=%s+%s\n",c,a,b);if(j[i]=='-')

printf("\tt%d=%s-%s\n",c,a,b);

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

printf("\t%c=t%d",j[i-1],--c);

sprintf(ch,"%d",c);

j[i]=ch[0];

c++;

small();

}

void small()

{

pi=0;l=0;

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

{

for(m=0;m<5;m++)

if(j[i]==sw[m])

if(pi<=p[m])

{

pi=p[m];

l=1;

k=i;

}

}

if(l==1)

dove(k);

else

exit(0);}

**Commands:**

**~$ gcc exp10.c**

**~$ ./a.out**

**Input:**

**Enter the expression:A=B+C-D**