**Exercise1**

Design a lexical analyzer for given language and the lexical analyzer should ignore redundant spaces, tabs and new lines.

#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)||strcmp(“void”,str)==0

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

else

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

}

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");/\*gets(st1);\*/

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 identifiersare:");

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);

}

OUTPUT

Enter the c program

z=a+b\*60

^Z

The no's in the program are60

The keywords and identifiersare:

z is an identifier

a is an identifier

b is an identifier

Special characters are=+\*

Total no. of lines are:2

Process returned 0 (0x0) execution time : 14.694 s

Press any key to continue.

Exercise – 2

**Simulate FIRST and FOLLOW of a grammar**

#include<stdio.h>

#include<string.h>

int i,j,l,m,n=0,o,p,nv,z=0,x=0;

char str[10],temp,temp2[10],temp3[20],\*ptr;

struct prod

{

char lhs[10],rhs[10][10],ft[10],fol[10];

int n;

}pro[10];

void findter()

{

int k,t;

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

{

if(temp==pro[k].lhs[0])

{

for(t=0;t<pro[k].n;t++)

{

if( pro[k].rhs[t][0]<65 || pro[k].rhs[t][0]>90 )

pro[i].ft[strlen(pro[i].ft)]=pro[k].rhs[t][0];

else if( pro[k].rhs[t][0]>=65 && pro[k].rhs[t][0]<=90 )

{

temp=pro[k].rhs[t][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

break;

}

}

}

void findfol()

{

int k,t,p1,o1,chk;

char \*ptr1;

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

{

chk=0;

for(t=0;t<pro[k].n;t++)

{

ptr1=strchr(pro[k].rhs[t],temp);

if( ptr1 )

{

p1=ptr1-pro[k].rhs[t];

if(pro[k].rhs[t][p1+1]>=65 && pro[k].rhs[t][p1+1]<=90)

{

for(o1=0;o1<n;o1++)

if(pro[o1].lhs[0]==pro[k].rhs[t][p1+1])

{

strcat(pro[i].fol,pro[o1].ft);

chk++;

}

}

else if(pro[k].rhs[t][p1+1]=='\0')

{

temp=pro[k].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

chk++;

}

else

{

pro[i].fol[strlen(pro[i].fol)]=pro[k].rhs[t][p1+1];

chk++;

}

}

}

if(chk>0)

break;

}

}

int main()

{

FILE \*f;

//clrscr();

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

pro[i].n=0;

f=fopen("cdprog2.txt","r");

while(!feof(f))

{

fscanf(f,"%s",pro[n].lhs);

if(n>0)

{

if( strcmp(pro[n].lhs,pro[n-1].lhs) == 0 )

{

pro[n].lhs[0]='\0';

fscanf(f,"%s",pro[n-1].rhs[pro[n-1].n]);

pro[n-1].n++;

continue;

}

}

fscanf(f,"%s",pro[n].rhs[pro[n].n]);

pro[n].n++;

n++;

}

printf("\n\nTHE GRAMMAR IS AS FOLLOWS\n\n");

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

for(j=0;j<pro[i].n;j++)

printf("%s -> %s\n",pro[i].lhs,pro[i].rhs[j]);

pro[0].ft[0]='#';

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

{

for(j=0;j<pro[i].n;j++)

{

if( pro[i].rhs[j][0]<65 || pro[i].rhs[j][0]>90 )

{

pro[i].ft[strlen(pro[i].ft)]=pro[i].rhs[j][0];

}

else if( pro[i].rhs[j][0]>=65 && pro[i].rhs[j][0]<=90 )

{

temp=pro[i].rhs[j][0];

if(temp=='S')

pro[i].ft[strlen(pro[i].ft)]='#';

findter();

}

}

}

printf("\n\nFIRST\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].ft);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].ft[l]==pro[i].ft[j])

break;

if(l==-1)

printf("%c",pro[i].ft[j]);

}

}

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

temp2[i]=pro[i].lhs[0];

pro[0].fol[0]='$';

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

{

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

{

for(j=0;j<pro[i].n;j++)

{

ptr=strchr(pro[l].rhs[j],temp2[i]);

if( ptr )

{

p=ptr-pro[l].rhs[j];

if(pro[l].rhs[j][p+1]>=65 && pro[l].rhs[j][p+1]<=90)

{

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

if(pro[o].lhs[0]==pro[l].rhs[j][p+1])

strcat(pro[i].fol,pro[o].ft);

}

else if(pro[l].rhs[j][p+1]=='\0')

{

temp=pro[l].lhs[0];

if(pro[l].rhs[j][p]==temp)

continue;

if(temp=='S')

strcat(pro[i].fol,"$");

findfol();

}

else

pro[i].fol[strlen(pro[i].fol)]=pro[l].rhs[j][p+1];

}

}

}

}

printf("\n\nFOLLOW\n");

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

{

printf("\n%s -> ",pro[i].lhs);

for(j=0;j<strlen(pro[i].fol);j++)

{

for(l=j-1;l>=0;l--)

if(pro[i].fol[l]==pro[i].fol[j])

break;

if(l==-1)

printf("%c",pro[i].fol[j]);

}

}

printf("\n");

//getch();

}

OUTPUT

THE GRAMMAR IS AS FOLLOWS

S -> ABE

S -> a

A -> p

A -> t

B -> Aq

S -> f

A -> w

->

FIRST

S -> #pta

A -> pt

B -> pt

S -> f

A -> w

->

FOLLOW

S -> $

A -> ptq

B ->

S ->

A -> ptq

-> $pt

Process returned 0 (0x0) execution time : 0.134 s

Press any key to continue.

or

// C program to calculate the First and

// Follow sets of a given grammar

#include<stdio.h>

#include<ctype.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], "E=TR");

    strcpy(production[1], "R=+TR");

    strcpy(production[2], "R=#");

    strcpy(production[3], "T=FY");

    strcpy(production[4], "Y=\*FY");

    strcpy(production[5], "Y=#");

    strcpy(production[6], "F=(E)");

    strcpy(production[7], "F=i");

    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++;

        }

    }

}

OUTPUT

First(E) = { (, i, }

First(R) = { +, #, }

First(T) = { (, i, }

First(Y) = { \*, #, }

First(F) = { (, i, }

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

Follow(E) = { $, ), }

Follow(R) = { $, ), }

Follow(T) = { +, $, ), }

Follow(Y) = { +, $, ), }

Follow(F) = { \*, +, $, ), }

Process returned 0 (0x0) execution time : 0.055 s

Press any key to continue.

Exercise – 3

**Develop an Operator precedence parser for a given grammar.**

#include<stdio.h>

#include<string.h>

char \*input;

int i=0;

char lasthandle[6],stack[50],handles[][5]={")E(","E\*E","E+E","i","E^E"};

//(E) becomes )E( when pushed to stack

int top=0,l;

char prec[9][9]={

                            /\*input\*/

            /\*stack    +    -   \*   /   ^   i   (   )   $  \*/

            /\*  + \*/  '>', '>','<','<','<','<','<','>','>',

            /\*  - \*/  '>', '>','<','<','<','<','<','>','>',

            /\*  \* \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  / \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  ^ \*/  '>', '>','>','>','<','<','<','>','>',

            /\*  i \*/  '>', '>','>','>','>','e','e','>','>',

            /\*  ( \*/  '<', '<','<','<','<','<','<','>','e',

            /\*  ) \*/  '>', '>','>','>','>','e','e','>','>',

            /\*  $ \*/  '<', '<','<','<','<','<','<','<','>',

                };

int getindex(char c)

{

switch(c)

    {

    case '+':return 0;

    case '-':return 1;

    case '\*':return 2;

    case '/':return 3;

    case '^':return 4;

    case 'i':return 5;

    case '(':return 6;

    case ')':return 7;

    case '$':return 8;

    }

}

int shift()

{

stack[++top]=\*(input+i++);

stack[top+1]='\0';

}

int reduce()

{

int i,len,found,t;

for(i=0;i<5;i++)//selecting handles

{

    len=strlen(handles[i]);

    if(stack[top]==handles[i][0]&&top+1>=len)

        {

        found=1;

        for(t=0;t<len;t++)

            {

            if(stack[top-t]!=handles[i][t])

                {

                found=0;

                break;

                }

            }

        if(found==1)

            {

            stack[top-t+1]='E';

            top=top-t+1;

            strcpy(lasthandle,handles[i]);

            stack[top+1]='\0';

            return 1;//successful reduction

            }

        }

}

return 0;

}

void dispstack()

{

int j;

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

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

}

void dispinput()

{

int j;

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

printf("%c",\*(input+j));

}

void main()

{

int j;

input=(char\*)malloc(50\*sizeof(char));

printf("\nEnter the string\n");

scanf("%s",input);

input=strcat(input,"$");

l=strlen(input);

strcpy(stack,"$");

printf("\nSTACK\tINPUT\tACTION");

while(i<=l)

{

shift();

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tShift");

if(prec[getindex(stack[top])][getindex(input[i])]=='>')

{

while(reduce())

{

printf("\n");

dispstack();

printf("\t");

dispinput();

printf("\tReduced: E->%s",lasthandle);

}

}

}

if(strcmp(stack,"$E$")==0)

    printf("\nAccepted;");

else

    printf("\nNot Accepted;");

}

**OUTPUT**

Enter the string

i+i\*i

STACK INPUT ACTION

$i +i\*i$ Shift

$E +i\*i$ Reduced: E->i

$E+ i\*i$ Shift

$E+i \*i$ Shift

$E+E \*i$ Reduced: E->i

$E \*i$ Reduced: E->E+E

$E\* i$ Shift

$E\*i $ Shift

$E\*E $ Reduced: E->i

$E $ Reduced: E->E\*E

$E$ Shift

$E$ Shift

Accepted;

Process returned 10 (0xA) execution time : 5.967 s

Press any key to continue.

Exercise – 4

**Construct a recursive descent parser for an expression.**

#include<stdio.h>

#include<conio.h>

#include<string.h>

char input[100];

int i,l;

void main()

{

clrscr();

printf("\nRecursive descent parsing for the following grammar\n"); printf("\nE->TE'\nE'->+TE'/@\nT->FT'\nT'->\*FT'/@\nF->(E)/ID\n"); printf("\nEnter the string to be checked:"); gets(input);

if(E())

{

if(input[i+1]=='\0')

printf("\nString is accepted");

else

printf("\nString is not accepted");

}

else

printf("\nString not accepted");

getch();

}

E()

{

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

EP()

{

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

{

i++;

if(T())

{

if(EP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

T()

{

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

TP()

{

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

{

i++;

if(F())

{

if(TP())

return(1);

else

return(0);

}

else

return(0);

}

else

return(1);

}

F()

{

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

{

i++;

if(E())

{

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

{

i++;

return(1);

}

else

return(0);

}

else

return(0);

}

else if(input[i]>='a'&&input[i]<='z'||input[i]>='A'&&input[i]<='Z')

{

i++;

return(1);

}

else

return(0);

}

OUTPUT1

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->\*FT'/@

F->(E)/ID

Enter the string to be checked:a+b\*c

String is accepted

OUTPUT2

Recursive descent parsing for the following grammar

E->TE'

E'->+TE'/@

T->FT'

T'->\*FT'/@

F->(E)/ID

Enter the string to be checked:a++b

String not accepted

Exercise – 5

**Construct a LL(1) parser for an expression**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

#include<stdlib.h>

void followfirst(char , int , int);

void findfirst(char , int , int);

void follow(char c);

int count,n=0;

char calc\_first[10][100];

char calc\_follow[10][100];

int m=0;

char production[10][10], first[10];

char f[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;

printf("How many productions ? :");

scanf("%d",&count);

printf("\nEnter %d productions in form A=B where A and B are grammar symbols :\n\n",count);

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

{

scanf("%s%c",production[i],&ch);

}

int kay;

char done[count];

int ptr = -1;

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;

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

if(c == done[kay])

xxx = 1;

if (xxx == 1)

continue;

findfirst(c,0,0);

ptr+=1;

done[ptr] = c;

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

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

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;

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;

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

if(ck == donee[kay])

xxx = 1;

if (xxx == 1)

continue;

land += 1;

follow(ck);

ptr+=1;

donee[ptr] = ck;

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

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

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++;

}

char ter[10];

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

ter[k] = '!';

}

int ap,vp,sid = 0;

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

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

if(!isupper(production[k][kay]) && production[k][kay]!= '#' && production[k][kay] != '=' && production[k][kay] != '\0'){

vp = 0;

for(ap = 0;ap < sid; ap++){

if(production[k][kay] == ter[ap]){

vp = 1;

break;

}

}

if(vp == 0){

ter[sid] = production[k][kay];

sid ++;

}

}

}

}

ter[sid] = '$';

sid++;

printf("\n\t\t\t\t\t\t\t The LL(1) Parsing Table for the above grammer :-");

printf("\n\t\t\t\t\t\t\t^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^\n");

printf("\n\t\t\t=====================================================================================================================\n");

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

for(ap = 0;ap < sid; ap++){

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

}

printf("\n\t\t\t=====================================================================================================================\n");

char first\_prod[count][sid];

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

int destiny = 0;

k = 2;

int ct = 0;

char tem[100];

while(production[ap][k] != '\0'){

if(!isupper(production[ap][k])){

tem[ct++] = production[ap][k];

tem[ct++] = '\_';

tem[ct++] = '\0';

k++;

break;

}

else{

int zap=0;

int tuna = 0;

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

if(calc\_first[zap][0] == production[ap][k]){

for(tuna=1;tuna<100;tuna++){

if(calc\_first[zap][tuna] != '!'){

tem[ct++] = calc\_first[zap][tuna];

}

else

break;

}

break;

}

}

tem[ct++] = '\_';

}

k++;

}

int zap = 0,tuna;

for(tuna = 0;tuna<ct;tuna++){

if(tem[tuna] == '#'){

zap = 1;

}

else if(tem[tuna] == '\_'){

if(zap == 1){

zap = 0;

}

else

break;

}

else{

first\_prod[ap][destiny++] = tem[tuna];

}

}

}

char table[land][sid+1];

ptr = -1;

for(ap = 0; ap < land ; ap++){

for(kay = 0; kay < (sid + 1) ; kay++){

table[ap][kay] = '!';

}

}

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

ck = production[ap][0];

xxx = 0;

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

if(ck == table[kay][0])

xxx = 1;

if (xxx == 1)

continue;

else{

ptr = ptr + 1;

table[ptr][0] = ck;

}

}

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

int tuna = 0;

while(first\_prod[ap][tuna] != '\0'){

int to,ni=0;

for(to=0;to<sid;to++){

if(first\_prod[ap][tuna] == ter[to]){

ni = 1;

}

}

if(ni == 1){

char xz = production[ap][0];

int cz=0;

while(table[cz][0] != xz){

cz = cz + 1;

}

int vz=0;

while(ter[vz] != first\_prod[ap][tuna]){

vz = vz + 1;

}

table[cz][vz+1] = (char)(ap + 65);

}

tuna++;

}

}

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

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

if(calc\_first[k][kay] == '!'){

break;

}

else if(calc\_first[k][kay] == '#'){

int fz = 1;

while(calc\_follow[k][fz] != '!'){

char xz = production[k][0];

int cz=0;

while(table[cz][0] != xz){

cz = cz + 1;

}

int vz=0;

while(ter[vz] != calc\_follow[k][fz]){

vz = vz + 1;

}

table[k][vz+1] = '#';

fz++;

}

break;

}

}

}

for(ap = 0; ap < land ; ap++){

printf("\t\t\t %c\t|\t",table[ap][0]);

for(kay = 1; kay < (sid + 1) ; kay++){

if(table[ap][kay] == '!')

printf("\t\t");

else if(table[ap][kay] == '#')

printf("%c=#\t\t",table[ap][0]);

else{

int mum = (int)(table[ap][kay]);

mum -= 65;

printf("%s\t\t",production[mum]);

}

}

printf("\n");

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

printf("\n");

}

int j;

printf("\n\nPlease enter the desired INPUT STRING = ");

char input[100];

scanf("%s%c",input,&ch);

printf("\n\t\t\t\t\t===========================================================================\n");

printf("\t\t\t\t\t\tStack\t\t\tInput\t\t\tAction");

printf("\n\t\t\t\t\t===========================================================================\n");

int i\_ptr = 0,s\_ptr = 1;

char stack[100];

stack[0] = '$';

stack[1] = table[0][0];

while(s\_ptr != -1){

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

int vamp = 0;

for(vamp=0;vamp<=s\_ptr;vamp++){

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

}

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

vamp = i\_ptr;

while(input[vamp] != '\0'){

printf("%c",input[vamp]);

vamp++;

}

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

char her = input[i\_ptr];

char him = stack[s\_ptr];

s\_ptr--;

if(!isupper(him)){

if(her == him){

i\_ptr++;

printf("POP ACTION\n");

}

else{

printf("\nString Not Accepted by LL(1) Parser !!\n");

exit(0);

}

}

else{

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

if(ter[i] == her)

break;

}

char produ[100];

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

if(him == table[j][0]){

if (table[j][i+1] == '#'){

printf("%c=#\n",table[j][0]);

produ[0] = '#';

produ[1] = '\0';

}

else if(table[j][i+1] != '!'){

int mum = (int)(table[j][i+1]);

mum -= 65;

strcpy(produ,production[mum]);

printf("%s\n",produ);

}

else{

printf("\nString Not Accepted by LL(1) Parser !!\n");

exit(0);

}

}

}

int le = strlen(produ);

le = le - 1;

if(le == 0){

continue;

}

for(j=le;j>=2;j--){

s\_ptr++;

stack[s\_ptr] = produ[j];

}

}

}

printf("\n\t\t\t=======================================================================================================================\n");

if (input[i\_ptr] == '\0'){

printf("\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN ACCEPTED !!\n");

}

else

printf("\n\t\t\t\t\t\t\t\tYOUR STRING HAS BEEN REJECTED !!\n");

printf("\t\t\t=======================================================================================================================\n");

}

void follow(char c)

{

int i ,j;

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'){

followfirst(production[i][j+1],i,(j+2));

}

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

follow(production[i][0]);

}

}

}

}

}

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

{

int j;

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))

{

findfirst(production[q1][q2], q1, (q2+1));

}

else

first[n++]='#';

}

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

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

}

else {

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

}

}

}

}

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

{

int k;

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;

}

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

{

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

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

}

else{

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

follow(production[c1][0]);

}

else{

followfirst(production[c1][c2],c1,c2+1);

}

}

j++;

}

}

}

OUTPUT

How many productions ? :8

Enter 8 productions in form A=B where A and B are grammar symbols :

E=TR

R=+TR

R=#

T=FY

Y=\*FY

Y=#

F=(E)

F=i

First(E)= { (, i, }

First(R)= { +, #, }

First(T)= { (, i, }

First(Y)= { \*, #, }

First(F)= { (, i, }

Follow(E) = { $, ), }

Follow(R) = { $, ), }

Follow(T) = { +, $, ), }

Follow(Y) = { +, $, ), }

Follow(F) = { \*, +, $, ), }

The LL(1) Parsing Table for the above grammer :-

^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^

=====================================================================

| + \* ( ) i $

=====================================================================

E | E=TR E=TR

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

R | R=+TR R=# R=#

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

T | T=FY T=FY

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

Y | Y=# Y=\*FY Y=# Y=#

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

F | F=(E) F=i

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

Please enter the desired INPUT STRING = i+i\*i$

=====================================================================

Stack Input Action

=====================================================================

$E i+i\*i$ E=TR

$RT i+i\*i$ T=FY

$RYF i+i\*i$ F=i

$RYi i+i\*i$ POP ACTION

$RY +i\*i$ Y=#

$R +i\*i$ R=+TR

$RT+ +i\*i$ POP ACTION

$RT i\*i$ T=FY

$RYF i\*i$ F=i

$RYi i\*i$ POP ACTION

$RY \*i$ Y=\*FY

$RYF\* \*i$ POP ACTION

$RYF i$ F=i

$RYi i$ POP ACTION

$RY $ Y=#

$R $ R=#

$ $ POP ACTION

=====================================================================

YOUR STRING HAS BEEN ACCEPTED !!

=====================================================================

Exercise – 6

**Design predictive parser for the given language.**

**Aim:** To Write a Program for Implementation of Predictive Parser.

#include<stdio.h>

#include<string.h>

char prol[7][10]={"S","A","A","B","B","C","C"};

char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};

char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"};

char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};

char follow[7][10]={"$","$","$","a$","b$","c$","d$"};

char table[5][6][10];

numr(char c)

{

switch(c)

{

case 'S': return 0;

case 'A': return 1;

case 'B': return 2;

case 'C': return 3;

case 'a': return 0;

case 'b': return 1;

case 'c': return 2;

case 'd': return 3;

case '$': return 4;

}

return(2);

}

void main()

{

int i,j,k;

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

for(j=0;j<6;j++)

strcpy(table[i][j]," ");

printf("\nThe following is the predictive parsing table for the following grammar:\n");

for(i=0;i<7;i++)

printf("%s\n",prod[i]);

printf("\nPredictive parsing table is\n");

fflush(stdin);

for(i=0;i<7;i++)

{

k=strlen(first[i]);

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

if(first[i][j]!='@')

strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);

}

for(i=0;i<7;i++)

{

if(strlen(pror[i])==1)

{

if(pror[i][0]=='@')

{

k=strlen(follow[i]);

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

strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);

}

}

}

strcpy(table[0][0]," ");

strcpy(table[0][1],"a");

strcpy(table[0][2],"b");

strcpy(table[0][3],"c");

strcpy(table[0][4],"d");

strcpy(table[0][5],"$");

strcpy(table[1][0],"S");

strcpy(table[2][0],"A");

strcpy(table[3][0],"B");

strcpy(table[4][0],"C");

printf("\n--------------------------------------------------------\n");

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

for(j=0;j<6;j++)

{

printf("%-10s",table[i][j]);

if(j==5)

printf("\n--------------------------------------------------------\n");

}

}

OUTPUT

The following is the predictive parsing table for the following grammar:

S->A

A->Bb

A->Cd

B->aB

B->@

C->Cc

C->@

Predictive parsing table is

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

a b c d $

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

S S->A S->A S->A S->A

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

A A->Bb A->Bb A->Cd A->Cd

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

B B->aB B->@ B->@ B->@

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

C C->@ C->@ C->@

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

Process returned 0 (0x0) execution time : 0.168 s

Press any key to continue.

Exercise – 7

**Implementation of shift reduce parsing algorithm.**

**Aim:** Program to implement shift reduce parser

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

char ip\_sym[15],stack[15];

int ip\_ptr=0,st\_ptr=0,len,i;

char temp[2],temp2[2];

char act[15];

void check();

void main()

{

printf("\n\t\t SHIFT REDUCE PARSER\n");

printf("\n GRAMMER\n");

printf("\n E->E+E\n E->E/E");

printf("\n E->E\*E\n E->a/b");

printf("\n enter the input symbol:\t");

gets(ip\_sym);

printf("\n\t stack implementation table");

printf("\n stack \t\t input symbol\t\t action");

printf("\n\_\_\_\_\_\_\_\_\t\t\_\_\_\_\_\_\_\_\_\_\_\_\t\t\_\_\_\_\_\_\_\_\_\_\_\_\n");

printf("\n $\t\t%s$\t\t\t--",ip\_sym);

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

len=strlen(ip\_sym);

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

{

stack[st\_ptr]=ip\_sym[ip\_ptr];

stack[st\_ptr+1]='\0';

ip\_sym[ip\_ptr]=' ';

ip\_ptr++;

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

strcpy(act,"shift");

temp[0]=ip\_sym[ip\_ptr];

temp[1]='\0';

strcat(act,temp);

check();

st\_ptr++;

}

st\_ptr++;

check();

}

void check()

{

int flag=0;

temp2[0]=stack[st\_ptr];

temp2[1]='\0';

if((!strcmpi(temp2,"a"))||(!strcmpi(temp2,"b")))

{

stack[st\_ptr]='E';

if(!strcmpi(temp2,"a"))

printf("\n $%s\t\t%s$\t\t\tE->a",stack,ip\_sym);

else

printf("\n $%s\t\t%s$\t\t\tE->b",stack,ip\_sym);

flag=1;

}

if((!strcmpi(temp2,"+"))||(strcmpi(temp2,"\*"))||(!strcmpi(temp2,"/")))

{

flag=1;

}

if((!strcmpi(stack,"E+E"))||(!strcmpi(stack,"E\E"))||(!strcmpi(stack,"E\*E")))

{

strcpy(stack,"E");

st\_ptr=0;

if(!strcmpi(stack,"E+E"))

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\E"))

printf("\n $%s\t\t%s$\t\t\tE->E\E",stack,ip\_sym);

else

if(!strcmpi(stack,"E\*E"))

printf("\n $%s\t\t%s$\t\t\tE->E\*E",stack,ip\_sym);

else

printf("\n $%s\t\t%s$\t\t\tE->E+E",stack,ip\_sym);

flag=1;

}

if(!strcmpi(stack,"E")&&ip\_ptr==len)

{

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

exit(0);

}

if(flag==0)

{

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

exit(0);

}

return;

}

OUTPUT

SHIFT REDUCE PARSER GRAMMER

E->E+E

E->E/E

E->E\*E

E->a/b

enter the input symbol: a+b

stack implementation table

stack input symbol action

\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

$ a+b$ --

$a +b$ shift a

$E +b$ E->a

$E+ b$ shift +

$E+b $ shift b

$E+E $ E->b

$E $ E->E+E

$E $ ACCEPT

Process returned 0 (0x0) execution time : 9.427 s

Press any key to continue.

Exercise – 8

**Design a LALR bottom up parser for the given language**.

**Aim:** To Write a program to Design LALR Bottom up Parser.

#include<stdio.h>

#include<stdlib.h>

#include<string.h>

void push(char \*,int \*,char);

char stacktop(char \*);

void isproduct(char,char);

int ister(char);

int isnter(char);

int isstate(char);

void error();

void isreduce(char,char);

char pop(char \*,int \*);

void printt(char \*,int \*,char [],int);

void rep(char [],int);

struct action

{

char row[6][5];

};

const struct action A[12]={

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","emp","acc"},

{"emp","rc","sh","emp","rc","rc"},

{"emp","re","re","emp","re","re"},

{"sf","emp","emp","se","emp","emp"},

{"emp","rg","rg","emp","rg","rg"},

{"sf","emp","emp","se","emp","emp"},

{"sf","emp","emp","se","emp","emp"},

{"emp","sg","emp","emp","sl","emp"},

{"emp","rb","sh","emp","rb","rb"},

{"emp","rb","rd","emp","rd","rd"},

{"emp","rf","rf","emp","rf","rf"}

};

struct gotol

{

char r[3][4];

};

const struct gotol G[12]={

{"b","c","d"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"emp","emp","emp"},

{"i","c","d"},

{"emp","emp","emp"},

{"emp","j","d"},

{"emp","emp","k"},

{"emp","emp","emp"},

{"emp","emp","emp"},

};

char ter[6]={'i','+','\*',')','(','$'};

char nter[3]={'E','T','F'};

char states[12]={'a','b','c','d','e','f','g','h','m','j','k','l'};

char stack[100];

int top=-1;

char temp[10];

struct grammar

{

char left;

char right[5];

};

const struct grammar rl[6]={

{'E',"e+T"},

{'E',"T"},

{'T',"T\*F"},

{'T',"F"},

{'F',"(E)"},

{'F',"i"},

};

void main()

{

char inp[80],x,p,dl[80],y,bl='a';

int i=0,j,k,l,n,m,c,len;

printf(" Enter the input :");

scanf("%s",inp);

len=strlen(inp);

inp[len]='$';

inp[len+1]='\0';

push(stack,&top,bl);

printf("\n stack \t\t\t input");

printt(stack,&top,inp,i);

do

{

x=inp[i];

p=stacktop(stack);

isproduct(x,p);

if(strcmp(temp,"emp")==0)

error();

if(strcmp(temp,"acc")==0)

break;

else

{

if(temp[0]=='s')

{

push(stack,&top,inp[i]);

push(stack,&top,temp[1]);

i++;

}

else

{

if(temp[0]=='r')

{

j=isstate(temp[1]);

strcpy(temp,rl[j-2].right);

dl[0]=rl[j-2].left;

dl[1]='\0';

n=strlen(temp);

for(k=0;k<2\*n;k++)

pop(stack,&top);

for(m=0;dl[m]!='\0';m++)

push(stack,&top,dl[m]);

l=top;

y=stack[l-1];

isreduce(y,dl[0]);

for(m=0;temp[m]!='\0';m++)

push(stack,&top,temp[m]);

}

}

}

printt(stack,&top,inp,i);

}while(inp[i]!='\0');

if(strcmp(temp,"acc")==0)

printf(" \n accept the input ");

else

printf(" \n do not accept the input ");

getch();

}

void push(char \*s,int \*sp,char item)

{

if(\*sp==100)

printf(" stack is full ");

else

{

\*sp=\*sp+1;

s[\*sp]=item;

}

}

char stacktop(char \*s)

{

char i;

i=s[top];

return i;

}

void isproduct(char x,char p)

{

int k,l;

k=ister(x);

l=isstate(p);

strcpy(temp,A[l-1].row[k-1]);

}

int ister(char x)

{

int i;

for(i=0;i<6;i++)

if(x==ter[i])

return i+1;

return 0;

}

int isnter(char x)

{

int i;

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

if(x==nter[i])

return i+1;

return 0;

}

int isstate(char p)

{

int i;

for(i=0;i<12;i++)

if(p==states[i])

return i+1;

return 0;

}

void error()

{

printf(" error in the input ");

exit(0);

}

void isreduce(char x,char p)

{

int k,l;

k=isstate(x);

l=isnter(p);

strcpy(temp,G[k-1].r[l-1]);

}

char pop(char \*s,int \*sp)

{

char item;

if(\*sp==-1)

printf(" stack is empty ");

else

{

item=s[\*sp];

\*sp=\*sp-1;

}

return item;

}

void printt(char \*t,int \*p,char inp[],int i)

{

int r;

printf("\n");

for(r=0;r<=\*p;r++)

rep(t,r);

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

for(r=i;inp[r]!='\0';r++)

printf("%c",inp[r]);

}

void rep(char t[],int r)

{

char c;

c=t[r];

switch(c)

{

case 'a': printf("0");

break;

case 'b': printf("1");

break;

case 'c': printf("2");

break;

case 'd': printf("3");

break;

case 'e': printf("4");

break;

case 'f': printf("5");

break;

case 'g': printf("6");

break;

case 'h': printf("7");

break;

case 'm': printf("8");

break;

case 'j': printf("9");

break;

case 'k': printf("10");

break;

case 'l': printf("11");

break;

default :printf("%c",t[r]);

break;

}

}

OUTPUT

Enter the input :i+i\*i

stack input

0 i+i\*i$

0i5 +i\*i$

0F3 +i\*i$

0T2 +i\*i$

0E1 +i\*i$

0E1+6 i\*i$

0E1+6i5 \*i$

0E1+6F3 \*i$

0E1+6T9 \*i$

0E1+6T9\*7 i$

0E1+6T9\*7i5 $

0E1+6T9\*7F10 $

0E1+6T9 $

0E1 $

accept the input

Exercise – 9

**Implement the lexical analyzer using JLex, flex or lex or other lexical analyzer**

**generating tools**

/\* program name is lexp.l \*/

%{

/\* program to recognize a c program \*/

int COMMENT=0;

%}

identifier [a-zA-Z][a-zA-Z0-9]\*

%%

#.\* { printf("\n%s is a PREPROCESSOR DIRECTIVE",yytext);}

int |

float |

char |

double |

while |

for |

do |

if |

break |

continue |

void |

switch |

case |

long |

struct |

const |

typedef |

return |

else |

goto {printf("\n\t%s is a KEYWORD",yytext);}

"/\*" {COMMENT = 1;}

/\*{printf("\n\n\t%s is a COMMENT\n",yytext);}\*/

"\*/" {COMMENT = 0;}

/\* printf("\n\n\t%s is a COMMENT\n",yytext);}\*/

{identifier}\( {if(!COMMENT)printf("\n\nFUNCTION\n\t%s",yytext);}

\{ {if(!COMMENT) printf("\n BLOCK BEGINS");}

\} {if(!COMMENT) printf("\n BLOCK ENDS");}

{identifier}(\[[0-9]\*\])? {if(!COMMENT) printf("\n %s IDENTIFIER",yytext);}

\".\*\" {if(!COMMENT) printf("\n\t%s is a STRING",yytext);}

[0-9]+ {if(!COMMENT) printf("\n\t%s is a NUMBER",yytext);}

\)(\;)? {if(!COMMENT) printf("\n\t");ECHO;printf("\n");}

\( ECHO;

= {if(!COMMENT)printf("\n\t%s is an ASSIGNMENT OPERATOR",yytext);}

\<= |

\>= |

\< |

== |

\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL OPERATOR",yytext);}

%%

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

{

if (argc > 1)

{

FILE \*file;

file = fopen(argv[1],"r");

if(!file)

{

printf("could not open %s \n",argv[1]);

exit(0);

}

yyin = file;

}

yylex();

printf("\n\n");

return 0;

} int yywrap()

{

return 0;

}

**Input:**

$vi var.c

#include<stdio.h>

main()

{

int a,b;

}

**Output:**

$lex lex.l

$cc lex.yy.c

$./a.out var.c

#include<stdio.h> is a PREPROCESSOR DIRECTIVE

FUNCTION

main (

)

BLOCK BEGINS

int is a KEYWORD

a IDENTIFIER

b IDENTIFIER

BLOCK ENDS

**Exercise – 10**

**Write a program to perform loop unrolling.**

// This program does not uses loop unrolling.

#include<stdio.h>

int main(void)

{

    for (int i=0; i<5; i++)

        printf("Hello\n"); //print hello 5 times

    return 0;

}

|  |
| --- |
| // This program uses loop unrolling.  #include<stdio.h>    int main(void)  {      // unrolled the for loop in program 1      printf("Hello\n");      printf("Hello\n");      printf("Hello\n");      printf("Hello\n");      printf("Hello\n");        return 0;  } |

Output:

Hello

Hello

Hello

Hello

Hello

Or

#include<stdio.h>

#define TOGETHER (8)

int main(void)

{

int i = 0;

int entries = 50; /\* total number to process \*/

int repeat; /\* number of times for while.. \*/

int left = 0; /\* remainder (process later) \*/

/\* If the number of elements is not be divisible by BLOCKSIZE, \*/

/\* get repeat times required to do most processing in the while loop \*/

repeat = (entries / TOGETHER); /\* number of times to repeat \*/

left = (entries % TOGETHER); /\* calculate remainder \*/

/\* Unroll the loop in 'bunches' of 8 \*/

while (repeat--)

{

printf("process(%d)\n", i );

printf("process(%d)\n", i + 1);

printf("process(%d)\n", i + 2);

printf("process(%d)\n", i + 3);

printf("process(%d)\n", i + 4);

printf("process(%d)\n", i + 5);

printf("process(%d)\n", i + 6);

printf("process(%d)\n", i + 7);

/\* update the index by amount processed in one go \*/

i += TOGETHER;

}

/\* Use a switch statement to process remaining by jumping to the case label \*/

/\* at the label that will then drop through to complete the set \*/

switch (left)

{

case 7 : printf("process(%d)\n", i + 6); /\* process and rely on drop through \*/

case 6 : printf("process(%d)\n", i + 5);

case 5 : printf("process(%d)\n", i + 4);

case 4 : printf("process(%d)\n", i + 3);

case 3 : printf("process(%d)\n", i + 2);

case 2 : printf("process(%d)\n", i + 1); /\* two left \*/

case 1 : printf("process(%d)\n", i); /\* just one left to process \*/

case 0 : ; /\* none left \*/

}

}

**Output:**

Process(1)

Process(2)

Process(3) …………..Process(48)

**Exercise – 11**

**Convert the BNF rules into YACC form and write code to generate abstract syntax**

**Tree**

**Program**:

**Save this color part as int.l**

%{

#include"y.tab.h"

#include<stdio.h>

#include<string.h>

int LineNo=1;

%}

identifier [a-zA-Z][\_a-zA-Z0-9]\*

number [0-9]+|([0-9]\*\.[0-9]+)

%%

main\(\) return MAIN;

if return IF;

else return ELSE;

while return WHILE;

int |

char |

float return TYPE;

{identifier} {strcpy(yylval.var,yytext);

return VAR;}

{number} {strcpy(yylval.var,yytext);

return NUM;}

\< |

\> |

\>= |

\<= |

== {strcpy(yylval.var,yytext);

return RELOP;}

[ \t] ;

\n LineNo++;

. return yytext[0];

%%

**Save this color part as int.y**

%{

#include<string.h>

#include<stdio.h>

struct quad

{

char op[5];

char arg1[10];

char arg2[10];

char result[10];

}QUAD[30];

struct stack

{

int items[100];

int top;

}stk;

int Index=0,tIndex=0,StNo,Ind,tInd;

extern int LineNo;

%}

%union

{

char var[10];

}

%token <var> NUM VAR RELOP

%token MAIN IF ELSE WHILE TYPE

%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP

%left '-' '+'

%left '\*' '/'

%%

PROGRAM : MAIN BLOCK

;

BLOCK: '{' CODE '}'

;

CODE: BLOCK

| STATEMENT CODE

| STATEMENT

;

STATEMENT: DESCT ';'

| ASSIGNMENT ';'

| CONDST

| WHILEST

;

DESCT: TYPE VARLIST

;

VARLIST: VAR ',' VARLIST

| VAR

;

ASSIGNMENT: VAR '=' EXPR{

strcpy(QUAD[Index].op,"=");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,$1);

strcpy($$,QUAD[Index++].result);

}

;

EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}

| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}

| EXPR '\*' EXPR {AddQuadruple("\*",$1,$3,$$);}

| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}

| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}

| '(' EXPR ')' {strcpy($$,$2);}

| VAR

| NUM

;

CONDST: IFST{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

}

| IFST ELSEST

;

IFST: IF '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

BLOCK {

strcpy(QUAD[Index].op,"GOTO");

strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

};

ELSEST: ELSE{

tInd=pop();

Ind=pop();

push(tInd);

sprintf(QUAD[Ind].result,"%d",Index);

}

BLOCK{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

};

CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);

StNo=Index-1;

}

| VAR

| NUM

;

WHILEST: WHILELOOP{

Ind=pop();

sprintf(QUAD[Ind].result,"%d",StNo);

Ind=pop();

sprintf(QUAD[Ind].result,"%d",Index);

}

;

WHILELOOP: WHILE '(' CONDITION ')' {

strcpy(QUAD[Index].op,"==");

strcpy(QUAD[Index].arg1,$3);

strcpy(QUAD[Index].arg2,"FALSE");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

BLOCK {

strcpy(QUAD[Index].op,"GOTO");

strcpy(QUAD[Index].arg1,"");

strcpy(QUAD[Index].arg2,"");

strcpy(QUAD[Index].result,"-1");

push(Index);

Index++;

}

;

%%

extern FILE \*yyin;

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

{

FILE \*fp;

int i;

if(argc>1)

{

fp=fopen(argv[1],"r");

if(!fp)

{

printf("\n File not found");

exit(0);

}

yyin=fp;

}

yyparse();

printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2 Result" "\n\t\t

--------------------");

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

{

printf("\n\t\t %d\t %s\t %s\t %s\t

%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);

}

printf("\n\t\t -----------------------");

printf("\n\n");

return 0;

}

void push(int data)

{

stk.top++;

if(stk.top==100)

{

printf("\n Stack overflow\n");

exit(0);

}

stk.items[stk.top]=data;

}

int pop()

{

int data;

if(stk.top==-1)

{

printf("\n Stack underflow\n");

exit(0);

}

data=stk.items[stk.top--];

return data;

}

void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])

{

strcpy(QUAD[Index].op,op);

strcpy(QUAD[Index].arg1,arg1);

strcpy(QUAD[Index].arg2,arg2);

sprintf(QUAD[Index].result,"t%d",tIndex++);

strcpy(result,QUAD[Index++].result);

}

yyerror()

{

printf("\n Error on line no:%d",LineNo);

}

Input:

$vi test.c

main()

{

int a,b,c;

if(a<b)

{

a=a+b;

}

while(a<b)

{

a=a+b;

}

if(a<=b)

{

c=a-b;

}

else

{

c=a+b;

}

}

Output:

$lex int.l

$yacc –d int.y

$gcc lex.yy.c y.tab.c –ll –lm

$./a.out test.c

**OUTPUT**



**Exercise – 12**

**Write a program for constant propagation.**

#include <stdio.h>

int main()

{

int x, y, z;

x = 10;

y = x + 45;

z = y + 4;

printf("The value of z = %d", z);

return 0;

}

**Output:**

The value of z=54

**ADDITIONAL EXPERIMENT**

**Exercise - 13. Write a program to implement SLR parser**

//SLR paresr

#include<stdio.h>

#include<string.h>

**int** axn[][**6**][**2**]={

{{**100**,**5**},{-**1**,-**1**},{-**1**,-**1**},{**100**,**4**},{-**1**,-**1**},{-**1**,-**1**}},

{{-**1**,-**1**},{**100**,**6**},{-**1**,-**1**},{-**1**,-**1**},{-**1**,-**1**},{**102**,**102**}},

{{-**1**,-**1**},{**101**,**2**},{**100**,**7**},{-**1**,-**1**},{**101**,**2**},{**101**,**2**}},

{{-**1**,-**1**},{**101**,**4**},{**101**,**4**},{-**1**,-**1**},{**101**,**4**},{**101**,**4**}},

{{**100**,**5**},{-**1**,-**1**},{-**1**,-**1**},{**100**,**4**},{-**1**,-**1**},{-**1**,-**1**}},

{{-**1**,-**1**},{**101**,**6**},{**101**,**6**},{-**1**,-**1**},{**101**,**6**},{**101**,**6**}},

{{**100**,**5**},{-**1**,-**1**},{-**1**,-**1**},{**100**,**4**},{-**1**,-**1**},{-**1**,-**1**}},

{{**100**,**5**},{-**1**,-**1**},{-**1**,-**1**},{**100**,**4**},{-**1**,-**1**},{-**1**,-**1**}},

{{-**1**,-**1**},{**100**,**6**},{-**1**,-**1**},{-**1**,-**1**},{**100**,**1**},{-**1**,-**1**}},

{{-**1**,-**1**},{**101**,**1**},{**100**,**7**},{-**1**,-**1**},{**101**,**1**},{**101**,**1**}},

{{-**1**,-**1**},{**101**,**3**},{**101**,**3**},{-**1**,-**1**},{**101**,**3**},{**101**,**3**}},

{{-**1**,-**1**},{**101**,**5**},{**101**,**5**},{-**1**,-**1**},{**101**,**5**},{**101**,**5**}}

};//Axn Table

**int** gotot[**12**][**3**]={**1**,**2**,**3**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,**8**,**2**,**3**,-**1**,-**1**,-**1**,

-**1**,**9**,**3**,-**1**,-**1**,**10**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**,-**1**}; //GoTo table

**int** a[**10**];

**char** b[**10**];

**int** top=-**1**,btop=-**1**,i;

**void** **push**(**int** k)

{

**if**(top<**9**)

a[++top]=k;

}

**void** **pushb**(**char** k)

{

**if**(btop<**9**)

b[++btop]=k;

}

**char** **TOS**()

{

**return** a[top];

}

**void** **pop**()

{

**if**(top>=**0**)

top--;

}

**void** **popb**()

{

**if**(btop>=**0**)

b[btop--]='\0';

}

**void** **display**()

{

**for**(i=**0**;i<=top;i++)

printf("%d%c",a[i],b[i]);

}

**void** **display1**(**char** p[],**int** m) //Displays The Present Input String

{

**int** l;

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

**for**(l=m;p[l]!='\0';l++)

printf("%c",p[l]);

printf("**\n**");

}

**void** **error**()

{

printf("Syntax Error");

}

**void** **reduce**(**int** p)

{

**int** len,k,ad;

**char** src,\*dest;

**switch**(p)

{

**case** **1**:dest="E+T";

src='E';

**break**;

**case** **2**:dest="T";

src='E';

**break**;

**case** **3**:dest="T\*F";

src='T';

**break**;

**case** **4**:dest="F";

src='T';

**break**;

**case** **5**:dest="(E)";

src='F';

**break**;

**case** **6**:dest="i";

src='F';

**break**;

**default:**dest="**\0**";

src='\0';

**break**;

}

**for**(k=**0**;k<strlen(dest);k++)

{

pop();

popb();

}

pushb(src);

**switch**(src)

{

**case** 'E':ad=**0**;

**break**;

**case** 'T':ad=**1**;

**break**;

**case** 'F':ad=**2**;

**break**;

**default:** ad=-**1**;

**break**;

}

push(gotot[TOS()][ad]);

}

**int** **main**()

{

**int** j,st,ic;

**char** ip[**20**]="**\0**",an;

// clrscr();

printf("Enter any String**\n**");

+

scanf("%s",ip);

push(**0**);

display();

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

**for**(j=**0**;ip[j]!='\0';)

{

st=TOS();

an=ip[j];

**if**(an>='a'&&an<='z') ic=**0**;

**else** **if**(an=='+') ic=**1**;

**else** **if**(an=='\*') ic=**2**;

**else** **if**(an=='(') ic=**3**;

**else** **if**(an==')') ic=**4**;

**else** **if**(an=='$') ic=**5**;

**else** {

error();

**break**;

}

**if**(axn[st][ic][**0**]==**100**)

{

pushb(an);

push(axn[st][ic][**1**]);

display();

j++;

display1(ip,j);

}

**if**(axn[st][ic][**0**]==**101**)

{

reduce(axn[st][ic][**1**]);

display();

display1(ip,j);

}

**if**(axn[st][ic][**1**]==**102**)

{

printf("Given String is accepted **\n**");

// getch();

**break**;

}

/\* else

{

printf("Given String is rejected \n");

break;

}\*/

}

**return** **0**;

}

/\*

-----------------------OUTPUT-----------------------------

deepti@Inspiron-3542:~$ gcc slr.c

deepti@Inspiron-3542:~$ ./a.out

Enter any String

a+a\*a$

0 a+a\*a$

0a5 +a\*a$

0F3 +a\*a$

0T2 +a\*a$

0E1 +a\*a$

0E1+6 a\*a$

0E1+6a5 \*a$

0E1+6F3 \*a$

0E1+6T9 \*a$

0E1+6T9\*7 a$

0E1+6T9\*7a5 $

0E1+6T9\*7F10 $

0E1+6T9 $

0E1 $

Given String is accepted

\*/