**Experiment no:** 1

**Date:**

**LEXICAL ANALYSER**

**AIM:** To implement Lexical analyzer using C++.

**ALGORITHM:**

1. Get the command in a form of a string.
2. Identify the keywords in it.
3. Classify the keywords and numbers.
4. Classify the rest as special characters.
5. Output lexemes in the correct order.

**PROGRAM:**

#include<stdio.h>

#include<iostream.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

void main()

{

clrscr();

int i,n,temp,temp1,count=1;

char com[100],a[10];

cout<<"Give the command:";

gets(com);

cout<<endl;

n=strlen(com);

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

{

if(isalpha(com[i]))

{

temp=0;

while(isalpha(com[i]))

{

a[temp]=com[i];

temp++;

i++;

}a[temp]='\0';

if(!strcmp("for",a)||!strcmp("while",a))

{

cout<<"<"<<a<<">";

i--;

}

else

{

cout<<"<id,"<<count<<">";

count++;

i--;

}

}

else if(isdigit(com[i]))

{

temp1=0;

while(isdigit(com[i]))

{

a[temp1]=com[i];

temp1++;

i++;

}a[temp1]='\0';

cout<<"<"<<a<<">";

i--;

}

else if(com[i]=='>')

{

if(com[i+1]=='=')

{

cout<<"<>=>";i++;

}

else

cout<<"<>>";

}

else if(com[i]=='<')

{

if(com[i+1]=='=')

{

cout<<"<<=>";

i++;

}

else

cout<<"<<>";

}

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

{

if(com[i+1]=='=')

{

cout<<"<==>";

i++;

}

else

cout<<"<=>";

}

else if(com[i]!=' ')

cout<<"<"<<com[i]<<">";

}

getch();

}

**OUTPUT:**

Give the command: a=b+c\*60

<id, 1><=><id, 2><+><id, 3><\*><60>

**RESULT:** The program was executed successfully and output is verified.

**Experiment no:** 2

**Date:**

**SYNTAX ANALYSER**

**AIM:** To create a syntax analyzer using C++.

**ALGORITHM:**

1. Get the input from the user.
2. Save each token in a separate temporary 2-D array.
3. Search for the operators.
4. Print the operator and in the next line print the before element.
5. Do the same until end of string.

**PROGRAM:**

#include<iostream.h>

#include<conio.h>

#include<string.h>

#include<stdio.h>

#include<ctype.h>

void main()

{

int i=0,j=0;

int level=0,temp1=0;

char inputstatement[100];

char temp[100];

clrscr();

cout<<"\nEnter the Scanned Expression:\n";

gets(inputstatement);

cout<<"\nAbstract Syntax Tree:\n";

while(i<strlen(inputstatement))

{

if(inputstatement[i]=='='||inputstatement[i]=='+'||inputstatement[i]=='-'||inputstatement[i]=='/'||inputstatement[i]=='\*'||inputstatement[i]=='%')

{

if(strlen(temp)==0&&i!=0)

{

cout<<"\n---SYNTAX ERROR---\n";

break ;

}

temp[j]='\0';

while(temp1!=level)

{

cout<<"\t";

temp1++;

}

temp1=0;

cout<<"\t"<<inputstatement[i]<<"\n";

while(temp1!=level)

{

cout<<"\t";

temp1++;

}

level++;

cout<<temp;

j=0;

temp1=0;

}

else if(inputstatement[i]!='<'&& inputstatement[i]!='>'&& inputstatement[i]!=',')

{

temp[j]=inputstatement[i];

j++;

}

if(inputstatement[i]==','&& isdigit(inputstatement[i-1]))

i++;

i++;

}

if(i==strlen(inputstatement))

{

temp[j]='\0';

while(temp1!=level)

{

cout<<"\t";

temp1++;

}

cout<<temp;

}

getch(); }**OUTPUT:**

Enter the Scanned Expression:

<id,1><=><id,2><+><id,3>

Abstract Syntax Tree:

=

id1 +

id2 id3

**RESULT:** The program was executed successfully and output is verified.

**Experiment no:** 3

**Date:**

**FIRST AND FOLLOW**

**AIM:** To write a C++ program to find first and follow of a grammar.

**ALGORITHM:**

1. Check for the first element of a non-terminals production is a terminal.
2. If true, add it to first (NT).
3. For follow, check if the NT exists, on the RHS of that production. If next symbol is:
4. NT, then add first (NT) to follow.
5. T, add T to follow.
6. Nothing then add follow (source) to follow.
7. E, follow (source) is added to follow.

**PROGRAM:**

#include<iostream.h>

#include<stdio.h>

#include<conio.h>

#include<string.h>

#include<ctype.h>

int temp1=0,n;

char nt[10],pdt[10][50];

void addnterm(char);

void first(char);

void follow(char);

int flag=0;

void main()

{

int temp,i,j;

clrscr();

cout<<"\nEnter the number of productions:";

cin>>n;

cout<<"\nEnter the productions:";

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

gets(pdt[i]);

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

{

temp=strlen(pdt[i]);

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

{

if(isupper(pdt[i][j])&&pdt[i][j]!='î')

{

addnterm(pdt[i][j]);

}

}

}

for(i=0;i<temp1;i++)

{

cout<<"\nFirst("<<nt[i]<<")={";

first(nt[i]);

cout<<"\b}";

}

for(i=0;i<temp1;i++)

{

cout<<"\nFollow("<<nt[i]<<")={";

if(i==0)

cout<<"$,";

follow(nt[i]);

cout<<"\b}";

}

getch();

}

void addnterm(char a)

{

int i=0,flag=1;

if(temp1==0)

{

nt[temp1]=a;

temp1++;

}

else

{

while(i<temp1&&flag)

{

if(a==nt[i])

flag=0;

else

i++;

}

if(flag)

{

nt[temp1]=a;

temp1++;

}

}

}

void first(char a)

{

int i;

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

{

if(pdt[i][0]==a)

{

if((isdigit(pdt[i][2])||islower(pdt[i][2])||pdt[i][2]=='(' ||pdt[i][2]==')'||pdt[i][2]=='+'||pdt[i][2]=='-'||pdt[i][2]=='\*' ||pdt[i][2]=='/'||pdt[i][2]=='%') && pdt[i][2]!='î')

cout<<pdt[i][2]<<",";

else if(pdt[i][2]=='î' && !flag)

cout<<pdt[i][2]<<",";

else

first(pdt[i][2]);

}

}

}

void follow(char a)

{

int i,j;

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

{

j=2;

while(pdt[i][j]!='\0')

{

if(pdt[i][j]==a)

{

if((isdigit(pdt[i][j+1])||islower(pdt[i][j+1])||pdt[i][j+1]=='('||pdt[i][j+1]==')'||pdt[i][j+1]=='+' ||pdt[i][j+1]=='-'||pdt[i][j+1]=='\*'|| pdt[i][j+1]=='/'||pdt[i][j+1]=='%') && pdt[i][j+1]!='î')

cout<<pdt[i][j+1];

else if(pdt[i][j+1]=='\0')

{

if(i==0)

cout<<"$,";

if(pdt[i][j]==pdt[i][0]);

else

follow(pdt[i][0]);

}

else

{

flag=1;

first(pdt[i][j+1]);

}

}

j++;

}

}

}

**OUTPUT:**

Enter the number of productions: 6

Enter the productions=a

Z=XYZ

X=Y

X=b

Y=ε

Y=c

First (Z) = {a, ε, c, b}

First(X) = {ε, c, b}

First(Y) = {ε, c}

Follow (Z) = {$}

Follow(X) ={c}

Follow(Y) = {a, c, b, $}

**RESULT:** The program was executed successfully and output is verified.

**Experiment no:** 4

**Date:**

**LEFT RECURSION**

**AIM:** To eliminate left recursion of a grammar using C++.

**ALGORITHM:**

1. Get the number of non-terminals in the grammar.
2. Get the number of productions for each non-terminal.
3. Apply the rules of left recursion.
4. Print the result.

**PROGRAM:**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

void main()

{

char a[10],b[50][10]={""},d[50][10]={""},ch;

int i,n,c[10]={0},j,k,t,n1;

clrscr();

printf("\nEnter the left production(s) (NON TERMINALS) : ");

scanf("%s",a);

n=strlen(a);

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

{

printf("\nEnter the number of productions for %c : ",a[i]);

scanf("%d",&c[i]);

}

t=0;

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

{

printf("\nEnter the right productions for %c",a[i]);

k=t;

for(j=0;j<c[i];j++)

{

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

do

{

scanf("%s",b[k]);

k++;

}while(k<j);

}

t=t+10;

}

t=0;

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

{

if(a[i]==b[t][0])

{

n1=strlen(b[t]);

for(k=1;k<n1;k++)

d[t][k-1]=b[t][k];

}

t=t+10;

}

t=0;

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

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

{

if(a[i]==b[t][0])

{

for(j=1;j<c[i];j++)

printf("\n%c -> %s%c'",a[i],b[t+j],a[i]);

}

t=t+10;

}

t=0;

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

{

if(a[i]==b[t][0])

printf("\n%c' -> %s%c'|%c",a[i],d[t],a[i],(char)238);

else

for(j=0;j<c[i];j++)

printf("\n%c -> %s",a[i],b[t+j]);

t=t+10;

}

getch();

}

**OUTPUT:**

Enter the number of non-terminals: ETF

Number of productions for E: 2

Number of productions for T: 2

Number of productions for F: 2

E->E+T

E->T

T->T\*F

T->F

F->(E)

F->i

Resulting production:

E -> TE'

T -> FT'

E' -> +TE'|i

T' -> \*FT'|i

F -> (E)

F -> i

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 5

**Date:**

**LEFT FACTORING**

**AIM:** To eliminate left factoring of a grammar using C++.

**ALGORITHM:**

1. Get the number of terminals
2. Get the number of productions for each terminal
3. Apply the results of left factoring
4. Print the result

**PROGRAM:**

#include<string.h>

#include<stdio.h>

#include<stdlib.h>

#include<conio.h>

void main()

{

char ch,lhs[20][20],rhs[20][20][20],temp[20],temp1[20];

int n,n1,count[20],x,y,i,j,k,c[20];

clrscr();

printf("\nEnter the no. of productions : ");

scanf("%d",&n);

n1=n;

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

{

printf("\nProduction %d \nEnter the no. of productions : ",i+1);

scanf("%d",&c[i]);

printf("\nEnter LHS : ");

scanf("%s",lhs[i]);

for(j=0;j<c[i];j++)

{

printf("%s->",lhs[i]);

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

}

}

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

{

count[i]=1;

while(memcmp(rhs[i][0],rhs[i][1],count[i])==0)

count[i]++;

}

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

{

count[i]--;

if(count[i]>0)

{

strcpy(lhs[n1],lhs[i]);

strcat(lhs[i],"'");

for(k=0;k<count[i];k++)

temp1[k] = rhs[i][0][k];

temp1[k++] = '\0';

for(j=0;j<c[i];j++)

{

for(k=count[i],x=0;k<strlen(rhs[i][j]);x++,k++)

temp[x] = rhs[i][j][k];

temp[x++] = '\0';

if(strlen(rhs[i][j])==1)

strcpy(rhs[n1][1],rhs[i][j]);

strcpy(rhs[i][j],temp);

}

c[n1]=2;

strcpy(rhs[n1][0],temp1);

strcat(rhs[n1][0],lhs[n1]);

strcat(rhs[n1][0],"'");

n1++;

}

}

printf("\n\nThe resulting productions are : \n");

for(i=0;i<n1;i++)

{

if(i==0)

printf("\n %s -> %c|",lhs[i],(char)238);

else

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

for(j=0;j<c[i];j++)

{

printf(" %s ",rhs[i][j]);

if((j+1)!=c[i])

printf("|");

}

printf("\b\b\b\n");

}

getch();

}

**OUTPUT:**

Enter the no. of productions: 2

Production 1

Enter the no. of productions: 3

Enter LHS: S

S->iCtSeS

S->iCtS

S->a

Production 2

Enter the no. of productions: 1

Enter LHS: C

C->b

The resulting productions are:

S' -> î| eS | |

C -> b

S -> iCtSS' | a

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 6

**Date:**

**SHIFT REDUCE PARSING**

**AIM:** To perform shift reduce parsing on a set of productions using a C++ program

**ALGORITHM:**

1. Get the productions from the user.
2. Apply the rules for Shift reduce parser.
3. Form the table consisting of Stack,Input and Action.
4. Print the result.

**PROGRAM:**

#include<stdio.h>

#include<string.h>

#include<conio.h>

#include<dos.h>

int novar=0,sttop=1,intop=1,j=0,i=0,handlelength=0;

char ipstr1[100],ipstr[100],popped,var;

char prod[20][20],handle[100],stack[100]="#",input[100]="#";

struct grammar

{

char lhs,rhs[20][20];

int noprod;

}g[20];

int checkhandle()

{

int i,m,k;

char temp[2]={ ' ' , '\0'};

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

{

strcpy(handle,"");

for(m=i;m<=sttop-1;temp[0]=stack[m],strcat(handle,temp),m++);

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

{

for(k=0;k<g[m].noprod && strcmp(handle,g[m].rhs[k])!=0;k++);

if(k!=g[m].noprod)

{

var=g[m].lhs;

return strlen(handle);

}

}

}

return 0;

}

void print(char \*text,int textlen)

{

int i;

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

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

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

}

void printi(char \*text,int textlen)

{

int i;

for(i=textlen-1;i>=0;i--)

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

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

}

void main()

{

int n,m,k,len,j=0,v;

clrscr();

printf("\n Enter the productions of the grammar(END to end):\n");

do

{

scanf("%s",prod[i++]);

}while(strcmp(prod[i-1],"END")!=0);

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

{

m=0,k=0;

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

if(g[j].lhs==prod[n][0])

break;

if(j==novar)

g[novar++].lhs=prod[n][0];

for(k=3;k<strlen(prod[n])+1;k++)

{

if(prod[n][k]!='|' && prod[n][k]!='\0')

g[j].rhs[g[j].noprod][m++]=prod[n][k];

if(prod[n][k]=='|' || prod[n][k]=='\0')

{

g[j].rhs[g[j].noprod++][m]='\0';

m=0;

}

}

}

printf("\nENTER THE INPUT STRING:");

scanf("%s",ipstr);

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

for(i=strlen(ipstr)-1;i>=0;i--)

input[intop++]=ipstr[i];

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

printf(" STACK\t\t\tINPUT\t\t\tACTION\n");

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

print(stack,sttop);

printi(input,intop);

while(1)

{

int count=0;

while((handlelength=checkhandle())>0)

{

if(input[intop-1]=='\*' && count++==2)

break;

else if(input[intop-1]=='=' && count++==3)

break;

else if(input[intop-1]=='e' && count++==1)

break;

for(i=0;i<handlelength;i++,--sttop);

stack[sttop++]=var;

printf("REDUCE BY %c -> %s\n",var,handle);

print(stack,sttop);

printi(input,intop);

}

popped=input[--intop];

if(popped!='#')

stack[sttop++]=popped;

handlelength=checkhandle();

if(popped=='#' && (handlelength=checkhandle())==0)

break;

printf("SHIFT ' %c '\n",popped);

print(stack,sttop);

printi(input,intop);

}

if(sttop==2 && stack[1]==g[0].lhs)

printf("ACCEPT\n");

else

printf("ERROR\n");

printf("---------------------------------------------------------------");

getch();

}

**OUTPUT:**

Enter the productions (END to end):

S->iCtSeS

S->iCtS

S->a

C->b

END

ENTER THE INPUT STRING: ibtaea

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

STACK INPUT ACTION

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

# ibtaea# SHIFT ' i '

#i btaea# SHIFT ' b '

#ib taea# REDUCE BY C -> b

#Ic taea# SHIFT ' t '

#iCt aea# SHIFT ' a '

#iCta ea# REDUCE BY S -> a

#iCtS ea# SHIFT ' e '

#iCtSe a# SHIFT ' a '

#iCtSea # REDUCE BY S -> a

#iCtSeS # REDUCE BY S -> iCtSeS

#S # ACCEPT

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

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 7

**Date:**

**LR (0) SET OF ITEMS**

**AIM:** To compute LR (0) set of items for the given grammar.

**ALGORITHM:**

1. Get the productions from the user.
2. Convert it into augmented grammar.
3. Construct the set of items.
4. Print the result.

**PROGRAM:**

#include<iostream.h>

#include<conio.h>

#include<string.h>

char prod[20][20],listofvar[26]="ABCDEFGHIJKLMNOPQR";

int novar=1,i=0,j=0,k=0,n=0,m=0,arr[30];

int noitem=0;

struct Grammar

{

char lhs;

char rhs[8];

}g[20],item[20],clos[20][10];

int isvariable(char variable)

{

for(int i=0;i<novar;i++)

if(g[i].lhs==variable)

return i+1;

return 0;

}

void findclosure(int z, char a)

{

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

for(i=0;i<arr[z];i++)

{

for(j=0;j<strlen(clos[z][i].rhs);j++)

{

if(clos[z][i].rhs[j]=='.' && clos[z][i].rhs[j+1]==a)

{

clos[noitem][n].lhs=clos[z][i].lhs;

strcpy(clos[noitem][n].rhs,clos[z][i].rhs);

char temp=clos[noitem][n].rhs[j];

clos[noitem][n].rhs[j]=clos[noitem][n].rhs[j+1];

clos[noitem][n].rhs[j+1]=temp;

n=n+1;

}

}

}

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

{

for(j=0;j<strlen(clos[noitem][i].rhs);j++)

{

if(clos[noitem][i].rhs[j]=='.' && isvariable(clos[noitem][i].rhs[j+1])>0)

{

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

{

if(clos[noitem][i].rhs[j+1]==clos[0][k].lhs)

{

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

if(clos[noitem][l].lhs==clos[0][k].lhs && strcmp(clos[noitem][l].rhs,clos[0][k].rhs)==0)

break;

if(l==n)

{

clos[noitem][n].lhs=clos[0][k].lhs;

strcpy(clos[noitem][n].rhs,clos[0][k].rhs);

n=n+1;

}

}

}

}

}

}

arr[noitem]=n;

int flag=0;

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

{

if(arr[i]==n)

{

for(j=0;j<arr[i];j++)

{

int c=0;

for(k=0;k<arr[i];k++)

if(clos[noitem][k].lhs==clos[i][k].lhs && strcmp(clos[noitem][k].rhs,clos[i][k].rhs)==0)

c=c+1;

if(c==arr[i])

{

flag=1;

goto exit;

}

}

}

}

exit:;

if(flag==0)

arr[noitem++]=n;

}

void main()

{

clrscr();

cout<<"ENTER THE PRODUCTIONS OF THE GRAMMAR(0 TO END) :\n";

do

{

cin>>prod[i++];

}while(strcmp(prod[i-1],"0")!=0);

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

{

m=0;

j=novar;

g[novar++].lhs=prod[n][0];

for(k=3;k<strlen(prod[n]);k++)

{

if(prod[n][k] != '|')

g[j].rhs[m++]=prod[n][k];

if(prod[n][k]=='|')

{

g[j].rhs[m]='\0';

m=0;

j=novar;

g[novar++].lhs=prod[n][0];

}

}

}

for(i=0;i<26;i++)

if(!isvariable(listofvar[i]))

break;

g[0].lhs=listofvar[i];

char temp[2]={g[1].lhs,'\0'};

strcat(g[0].rhs,temp);

cout<<"\n\n augumented grammar \n";

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

cout<<endl<<g[i].lhs<<"->"<<g[i].rhs<<" ";

getch();

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

{

clos[noitem][i].lhs=g[i].lhs;

strcpy(clos[noitem][i].rhs,g[i].rhs);

if(strcmp(clos[noitem][i].rhs,"ε")==0)

strcpy(clos[noitem][i].rhs,".");

else

{

for(int j=strlen(clos[noitem][i].rhs)+1;j>=0;j--)

clos[noitem][i].rhs[j]=clos[noitem][i].rhs[j-1];

clos[noitem][i].rhs[0]='.';

}

}

arr[noitem++]=novar;

for(int z=0;z<noitem;z++)

{

char list[10];

int l=0;

for(j=0;j<arr[z];j++)

{

for(k=0;k<strlen(clos[z][j].rhs)-1;k++)

{

if(clos[z][j].rhs[k]=='.')

{

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

if(list[m]==clos[z][j].rhs[k+1])

break;

if(m==l)

list[l++]=clos[z][j].rhs[k+1];

}

}

}

for(int x=0;x<l;x++)

findclosure(z,list[x]);

}

cout<<"\n THE SET OF ITEMS ARE \n\n";

for(z=0;z<noitem;z++)

{

cout<<"\n I"<<z<<"\n\n";

for(j=0;j<arr[z];j++)

cout<<clos[z][j].lhs<<"->"<<clos[z][j].rhs<<"\n";

getch();

}

getch();

}

**OUTPUT:**

Enter the productions of grammar (0 to end):

E->E+T

E->T

T->T\*F

T->F

F-> (E)

F->i

Augmented grammar:

A->E

E->E+T

E->T

T->T\*F

T->F

F->(E)

F->i

I0:

A->.E

E->.E+T

E->.T

T->.T\*F

T->.F

F->.(E)

F->.i

I1:

A->E.

E->E.+T

I2:

E->T.

T->T.\*F

I3:

T->F.

I4:

F->(.E)

E->.E+T

E->.T

T->.T\*F

T->.F

F->.(E)

I5:

F->i.

I6:

E->E+.T

T->.T\*F

T->.F

F->.(E)

F->.i

I7

T->T\*.F

F->.(E)

F->.i

I8

F->(E.)

E->E.+T

I9

E->E+T.

T->T.\*F

I10

T->T\*F.

I11

F->(E).

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 8

**Date:**

**INTERMEDIATE CODE GENERATOR**

**AIM:** To generate intermediate code for the given grammar using C++.

**ALGORITHM:**

1. Get the productions from the user.
2. Apply the rules for First and Follow for all the steps.
3. Print the result.

**PROGRAM:**

#include<iostream.h>

#include<string.h>

#include<conio.h>

char reg[10][3]={"R0","R1","R2","R3","R4","R5"};

char stmt[10][10],code[15];

int nostmt=0,i=0,output[15];

void icode(char source[10],char dest[10],int out)

{

strcat(code,source);

strcat(code," ");

strcat(code,dest);

output[i]=out;

cout<<code<<endl;

getch();

}

void main()

{

clrscr();

cout<<" Enter the statements(END to end): \n";

do

{

cin>>stmt[nostmt++];

}while(strcmp(stmt[nostmt-1],"END")!=0);

nostmt=nostmt-1;

cout<<"\n THE INTERMEDIATE CODE IS\n\n";

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

{

strcpy(code,"");

int rd=-1,rs=-1,k;

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

{

if(stmt[j][0]==stmt[i][2])

rs=output[j];

if(stmt[j][0]==stmt[1][4])

rd=output[j];

}

if(rs==-1)

{

strcpy(code,"MOV ");

char temp[2]={stmt[i][2],'\0'};

icode(temp,reg[i],i);

}

if(stmt[i][3]=='+')

strcpy(code,"ADD ");

if(stmt[i][3]=='-')

strcpy(code,"SUB ");

if(stmt[i][3]=='\*')

strcpy(code,"MUL ");

if(stmt[i][3]=='/')

strcpy(code,"DIV ");

if(rd==-1)

{

char temp[2]={stmt[i][4],'\0'};

if(rs!=-1)

k=output[rs];

else

k=i;

icode(temp,reg[k],k);

}

if(rs!=-1 && rd!=-1)

{

int flag=0;

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

if(stmt[j][2]==stmt[i][2] || stmt[j][2]==stmt[i][4])

flag=1;

if(flag!=1)

icode(reg[output[rs]],reg[output[rd]],output[rd]);

if(flag==1)

icode(reg[output[rd]],reg[output[rs]],output[rs]);

}

}

strcpy(code,"MOV ");

char temp[2]={stmt[i-1][0],'\0'};

icode(reg[output[i-1]],temp,0);

}

**OUTPUT:**

Enter the statements (END to end):

t=a+b

t=a\*b

END

THE INTERMEDIATE CODE IS

MOV a R0

ADD b R0

MOV a R1

MUL b R1

MOV R1 t

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 9

**Date:**

**MACRO PROGRAM**

**AIM:** To write a C++ program to implement macros.

**ALGORITHM:**

1. Define the macro.
2. Input the macro function from user.
3. Link the other file.
4. Print the result.

**PROGRAM:**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

void main()

{

FILE \*f1,\*f2,\*f3,\*f4;

int x=0,y=0,brk=0,cnt=0,dec,use;

char p[20],q[20][20],r[20][20],s[20][20],t;

char fnam1[30],fnam2[30],ch,c,a;

clrscr();

printf("\nEnter the input file name : ");

scanf("%s",fnam1);

f1 = fopen(fnam1,"w");

printf("\nEnter the contents of the file : ");

while((ch=getchar())!=EOF)

putc(ch,f1);

fclose(f1);

printf("\nEnter the macro expanded file name : ");

scanf("%s",fnam2);

f3 = fopen(fnam2,"w");

f2 = fopen(fnam1,"r");

fscanf(f2,"%s",p);

while(!feof(f2))

{

if(strcmp(p,"#define")==0)

{

fscanf(f2,"%s",p);

strcpy(q[x++],p);

fscanf(f2,"%s",p);

strcpy(r[y++],p);

}

fscanf(f2,"%s",p);

}

cnt=y;

fseek(f2,0,SEEK\_SET);

do

{

fread(p,1,1,f2);

fprintf(f3,"%s",p);

}while(strcmp(p,"{")!=0);

do

{

t=getc(f2);

x=0;

y=0;

label:

if(y<strlen(q[x]) && x<=cnt)

{

dec = toascii(q[x][y]);

use = toascii(t);

if(dec==use)

{

y++;

t=getc(f2);

if(y==(strlen(q[x])-1))

brk=1;

goto label;

}

else

{

brk=0;

y=0;

x++;

goto label;

}

}

if(brk==1)

fprintf(f3,"%s",r[x]);

fprintf(f3,"%c",t);

}while(!feof(f2));

fclose(f3);

fclose(f2);

f4=fopen(fnam2,"r");

while(!feof(f4))

{

a=getc(f4);

printf("%c",a);

}

getch();

}

**OUTPUT:**

Enter input filename: macroip.c

Enter the contents of the file:

#include<stdio.h>

#define PI 3.14

#define sum(x,y) (x+y)

void main()

{

int x,y,z;

x=18;

y=24;

z=sum(x,y)\*PI;

}

^Z

Enter the macro expanded file name: macroop.c

#inclue<stdio.h>

#define PI 3.14

#define sum(x,y) (x+y)

void main()

{

int x,y,z;

x=18;

y=24;

z=(x+y)\*3.14;

}

**RESULT:** The program was executed successfully and output was verified

**Experiment no:** 10

**Date:**

**LINKER PROGRAM**

**AIM:** To write a C++ program to implement linker.

**ALGORITHM:**

1. Define the two files called external file and main file.
2. Get the input in the main file.
3. Define the functions in the external file for factorial and for changing the input value.
4. Call the functions in the main file and print the reverse.

**PROGRAM:**

**Link1.c**

#include<iostream.h>

#include<conio.h>

#include<link.h>

int fact(int i);

void main()

{

int a;

clrscr();

cout<<"\n Enter the value of n: ";

cin>>n;

change();

cout<<"\n The changed value of n is:"<< n;

cout<<"\n Enter the value to find factorial of: ";

cin>>a;

cout<<"\n The factorial of " <<a<< "is "<< fact(a);

}

**Link.h**

int n;

int fact(int i)

{

int j,fact=1;

if(i<0)

return -1;

else if(i==0)

return 1;

else

{

for(j=1;j<=i;j++)

fact\*=j;

return fact;

}

}

void change()

{

n=n\*10;

}

\*\*Note: Link.h must be saved in C:/TC/INCLUDE/

**OUTPUT:**

Enter the value of n:5

Changed value of n is : 50

Enter the value to find factorial: 3

Factorial is 6

**RESULT:** The program was executed successfully and output was verified