SRM
INSTITUTE OF SCIENCE & TECHNOLOGY
(Deemed to be University u/s 3 of UGC Act, 1956)

DEPT. Of Computer Science Engineering

SRM IST, Kattankulathur – 603 203

STUDENT PORTFOLIO

Name : Rahul Goel
Register Number : RA1911030010094
Mail ID : rg4994@srmist.edu.in
Department : CSE
Specialization : Cyber Security
Semester : 6

SUBJECT TITLE : 18CSC304J COMPILER DESIGN

HANDLED BY : MS.C.FANCY

COMPILER DESIGN ASSIGNMENT

Rahul Vaid
RA1911030010094

Q1) S → (L) | a
L → L,S | S

The given grammar is an operation precedence grammar.

Operator precedence table:

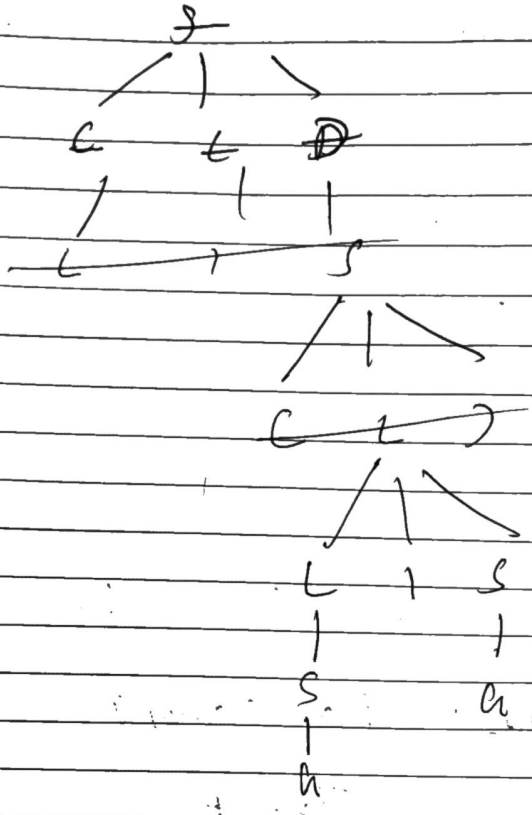
Terminal Symbols : {(,), ,, a, ε, $}

	(	)	,	a	$
(	<	=	<	<	
)	=	>	>	=	>
,	<	>	>	<	>
a	-	>	>	=	>
$	<	<	<	<	A

Parsing Table : (a, (a,a))

Stack	Input	Actions
$	(a, (a, a)) $	push (
$ (	a, (a, a)) $	push a
$ (a	, (a, a)) $	pop a
$ (	, (a, a)) $	push,
$ (,	(a, a)) $	push a
$ (, (	a, a)) $	pop a
$ (, (	, a)) $	push (,
$ (, (,	a)) $	push a
$ (, (,	)) $	pop a
$ (, (	)) $	push)

$ (, ( )	) $	pop)
$ (, (	) $	push)
$ (, b)	$	pop)
$ (, (	$	pop (
$ (, (	$	pop (
$ (,	$	pop ,
$ (	$	pop (
$	$	Accept.



Q2) $E \rightarrow E+E \mid E \times E \mid b$

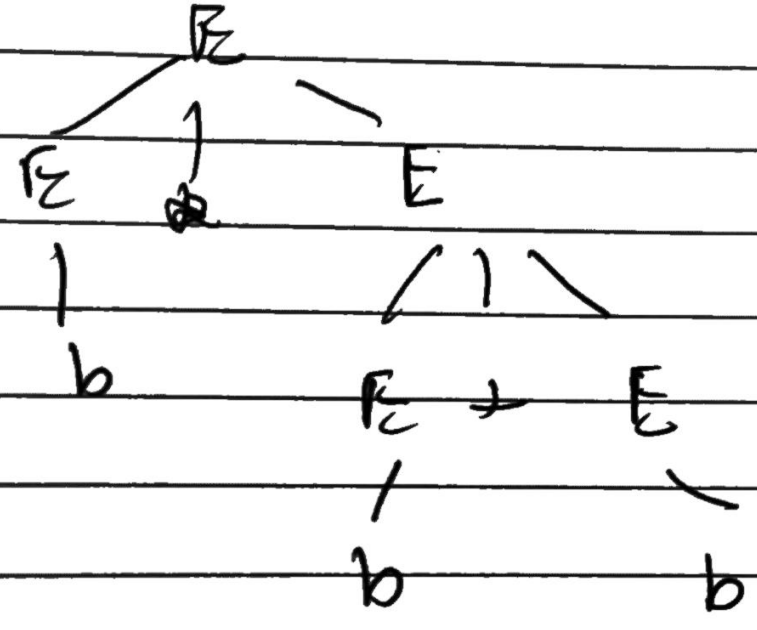
Solⁿ:

	b	+	*	$
b		>	>	>
+	<	>	<	>
*	<	>	>	>
$	<	<	<	A

b * b+b

Stack	Input	Action
$	b * b + b $	push b
$ b	* b + b $	pop b
$	* b + b $	push *
$ *	b + b $	pop b push b
$ * b	+ b $	pop * push +
$	+ b $	push b
$ +	b $	pop b
$ + b	$	pop +
$	$	
$	$	Accept

$E \rightarrow E + E \mid E * E \mid b$



Rahul Goel
RA1911030010094.

CO - Assignment - 4

Storage organisation

- The operating system maps the logical address into physical addresses.
- The storage layout is influenced by the addressing constraints of the target machine.
- A character array of length 10 needs only enough bytes to hold 10 characters a compiler may allocate 12 bytes to get alignment, leaving 2 bytes unused.
- This unused space due to alignment consideration is referred to as padding.
- Activation records maintain the Variable information.

From the prospective of the Compiler writer:

- The executing target program runs in

its own logical target address space in which each program value has a location.

- The Management and organization of this logical address space is shared between the compiler, operating system, and target machine.
- The operating system maps the logical addresses and physical address which are usually spread throughout memory.

Activation Records

- Procedure calls and returns are usually managed buy a run time stack called the Control stack. Each live activation has an activation record.
- The root of the activation tree will be at the bottom of the stack.
- The entire sequence of activation records on

the stack corresponding to the path in the activation tree to the activation where control currently resides.

- The latter activation has its record at the top of stack.

Actual Parameters
Returned Values
Control link
Access link
Saved machine status
Local data
Temporaries

(iii) Storage Allocation Strategies

The different ways to allocate memory are:

i) Static storage Allocation
ii) Stack storage Allocation
iii) Heap storage Allocation

Static storage Allocation

- In static allocation names are bound to storage locations.
- If memory is created at compile time then the memory will be created in static area and only once.
- Static allocation supports the dynamic data structure that means memory is created only at compile time and deallocated after completion of program.
- The drawback with static storage allocation is that size and position of data objects should be know at compile time.
- Another drawback is restriction of the recursion Procedure.

Stack Storage Allocation

- In static allocation, names are bound to storage locations.
- If memory is created at compile time then the memory will be created in static area and only once.

- Static allocation supports the dynamic data structure that means memory is created only at compile time and dellocated after completion of program.
- The drawback with static storage allocation is that size and position of data objects should be know at compile time.
- Another drawback is restriction of the recursion procedure.

Stack Storage Allocation

- In Stack storage allocation, storage is organized as stack.
- An activation record is pushed into the stack when activation begins and it is popped when activation end.
- Activation record contains the locals so that they are bound to fresh storage in each activation record. The value of locals is deleted when the activation ends.
- It works on the basis of last in first act (LIFO) and this allocation supports the recursion proces.

Heap Storage Allocation :-

- Heap Allocation is the most flexible allocation scheme.
- Allocation and deallocation of memory can be done at any time and at any place depending upon the users requirement.
- Heap allocation is used to allocate memory to variables dynamically and when the variables are no more used the claim it back.
- Heap storage allocation supports the recursion process.

13.4.22

Name : Rahul Goel
Reg no: RA1911030010094

Q Explain about cross compiler and its issues.

Ans A Cross Compiler is a compiler capable of creating executable code for a platform other than the one on which the compiler is running. For example, a compiler that runs on a PC but generates code that runs on an Android Smartphone is a Cross Compiler.

Uses of Cross Compiler :-

→ A cross Compiler is necessary to compile code for multiple platforms for one development host.

→ Direct compilation on the target platform might be infeasible, for example on embedded systems with limited computing resources.

→ Cross Compilers are distinct from Source-to-Source compilers. Across Compiler is for cross platform software generation of machine code, while a Source-to-Source compiler translates from one programming language to another in text code.

→ Both are programming tools.

→ Embedded computers where a device has extremely limited resources. For example, a microwave oven will have an extremely small computer to read its keypad and door sensor, provide output to a digital display and speaker and to control the machinery for cooking food. This computer is generally not powerful enough to run a compiler, a file system or development environment.

Issues with Cross Compiler :-

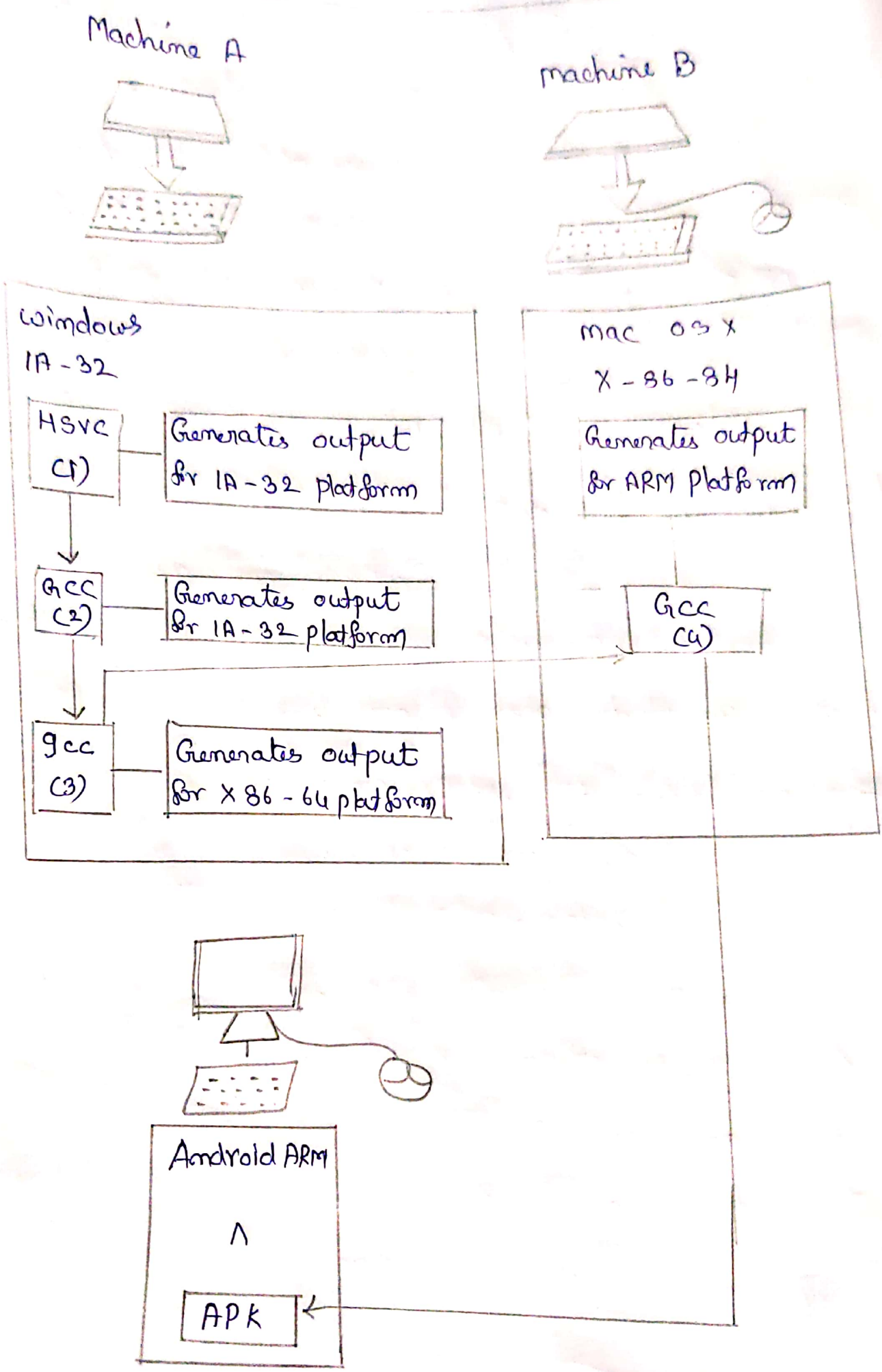
→ Keeping multiple builds contexts straight.

- Information leaks between the target and host.
- Each toolchain has its own headers and libraries.
- Different wirings in path depending on target or host.

- Hard to remove things the host has, but the target doesn't.
  - Install gzip on the host, but not in the target.
- Uname - m
- Machine type given by the host may not be equal target.
- Environment variable.
- Looking at /proc.
- Lying to configure.
  - The design to configure i.e fundamentally wrong for cross-compiling
  - A design to configure is fundamentally wrong for cross-compiling
- Hard to test the result
  - The test suite is often part of source. You didn't build the source on the target, so you have to copy the source tarball to the target

hardware. Then to run its test suite you need to cross-compile tcl/expect and install target's root file system.

- Bad Error Reporting
  - Using the wrong "strip" mostly works until you build for sh4 target.
  - The perl thing mentioned earlier shipped to a Costumer. It built, it ran, it just didn't work in this area.
- Not all packages cross-compile.
  - Most developers burely care about native compiling on non-x86, let alone cross-compiling.



SRM INSTITUTE OF SCIENCE AND TECHNOLOGY

SRM Nagar, Kattankulathur – 603 203,

Kancheepuram District

SCHOOL OF COMPUTING

DEPARTMENT OF NETWORKING & COMMUNICATIONS

LAB RECORD

Course Code : 18CSC304J

Course Name : Compiler Design

Submitted By:

Name : Rahul Goel

Reg No. : RA1911030010094

INDEX

EXPERIMENT – 1

LEXICAL ANALYSER

Aim:

To study and code a lexical analyser in any of the programming languages.

Language used: c++

Software used: online c++ compiler

Algorithm:

- Start.
- Get the input program from the file prog.txt.
- Read the program line by line and check if each word in a line is a keyword, identifier, constant or an operator.
- If the word read is an identifier, assign a number to the identifier and make an entry into the symbol table stored in sybol.txt.
- For each lexeme read, generate a token as follows:
    - If the lexeme is an identifier, then the token generated is of the form
    - If the lexeme is an operator, then the token generated is .
    - If the lexeme is a constant, then the token generated is .
    - If the lexeme is a keyword, then the token is the keyword itself.
- The stream of tokens generated are displayed in the console output.
- Stop.

Code:

Main.cpp

```
#include<bits/stdc++.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
```

```
using namespace std;

int isKeyword(char buffer[]){
   char keywords[32][10] =
   {"auto","break","case","char","const","continue","default",
                  "do","double","else","enum","extern","float","for","goto",
                  "if","int","long","register","return","short","signed",
                  "sizeof","static","struct","switch","typedef","union",
                  "unsigned","void","volatile","while"};
   int i, flag = 0;

   for(i = 0; i < 32; ++i){
      if(strcmp(keywords[i], buffer) == 0){
         flag = 1;
         break;
      }
   }

   return flag;
}

int main(){
   char ch, buffer[15],b[30], logical_op[] = "><",math_op[]="+-
*/=",numer[]=".0123456789",other[]=",;\(){}[]'':";
   int count=0;
   ifstream fin("Program.txt");
```

```
int mark[1000]={0};
int i,j=0,kc=0,ic=0,lc=0,mc=0,nc=0,oc=0,aaa=0;
vector < string > k;
vector<char >id;
vector<char>lo;
vector<char>ma;
vector<string>nu;
 vector<char>ot;
if(!fin.is_open()){
   cout<<"error while opening the file\n";
   exit(0);
}

while(!fin.eof()){
      ch = fin.get();
     for(i = 0; i < 12; ++i){
         if(ch == other[i]){
            int aa=ch;
          if(mark[aa]!=1){
            ot.push_back(ch);
            mark[aa]=1;
            ++oc;
          }
         }
   }
```

```
    for(i = 0; i < 5; ++i){
        if(ch == math_op[i]){
            int aa=ch;
         if(mark[aa]!=1){
            ma.push_back(ch);
            mark[aa]=1;
            ++mc;
         }
        }
    }
    for(i = 0; i < 2; ++i){
        if(ch == logical_op[i]){
            int aa=ch;
         if(mark[aa]!=1){
            lo.push_back(ch);
            mark[aa]=1;
            ++lc;
         }
        }


    }
    if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6' || ch=='7'
|| ch=='8' || ch=='9' || ch=='.' ||ch == ' ' || ch == '\n' || ch == ';'){


        if(ch=='0' || ch=='1' || ch=='2' || ch=='3' || ch=='4' || ch=='5' || ch=='6' ||
ch=='7' || ch=='8' || ch=='9' || ch=='.')b[aaa++]=ch;
        if((ch == ' ' || ch == '\n' || ch == ';') && (aaa != 0)){
```

```
            b[aaa] = '\0';
            aaa = 0;
            char arr[30];
             strcpy(arr,b);
                nu.push_back(arr);
             ++nc;

          }
      }


    if(isalnum(ch)){
       buffer[j++] = ch;
    }
    else if((ch == ' ' || ch == '\n') && (j != 0)){
            buffer[j] = '\0';
            j = 0;

            if(isKeyword(buffer) == 1){

               k.push_back(buffer);
               ++kc;
            }
            else{
```

```
            if(buffer[0]>=97 && buffer[0]<=122) {
               if(mark[buffer[0]-'a']!=1){
               id.push_back(buffer[0]);
               ++ic;
               mark[buffer[0]-'a']=1;
              }

            }

           }

      }

}

fin.close();
printf("Keywords: ");
 for(int f=0;f<kc;++f){
      if(f==kc-1){
         cout<<k[f]<<"\n";
         count++;
      }
      else {
         cout<<k[f]<<", ";
      }
```

```
    }
    cout<<"no.of Keywords "<<count+1<<endl;
    count=0;
    printf("\nIdentifiers: ");
     for(int f=0;f<ic;++f){
        if(f==ic-1){
              cout<<id[f]<<"\n";
              count++;
            }
            else {
              cout<<id[f]<<", ";
            }
    }
    cout<<"no.of Identifiers: "<<count+1<<endl;
    count=0;
    printf("\nMath Operators: ");
    for(int f=0;f<mc;++f){
            if(f==mc-1){
              cout<<ma[f]<<"\n";
              count++;
            }
            else {
              cout<<ma[f]<<", ";
            }
    }
    cout<<"no.of Math Operators: "<<count+1<<endl;
```

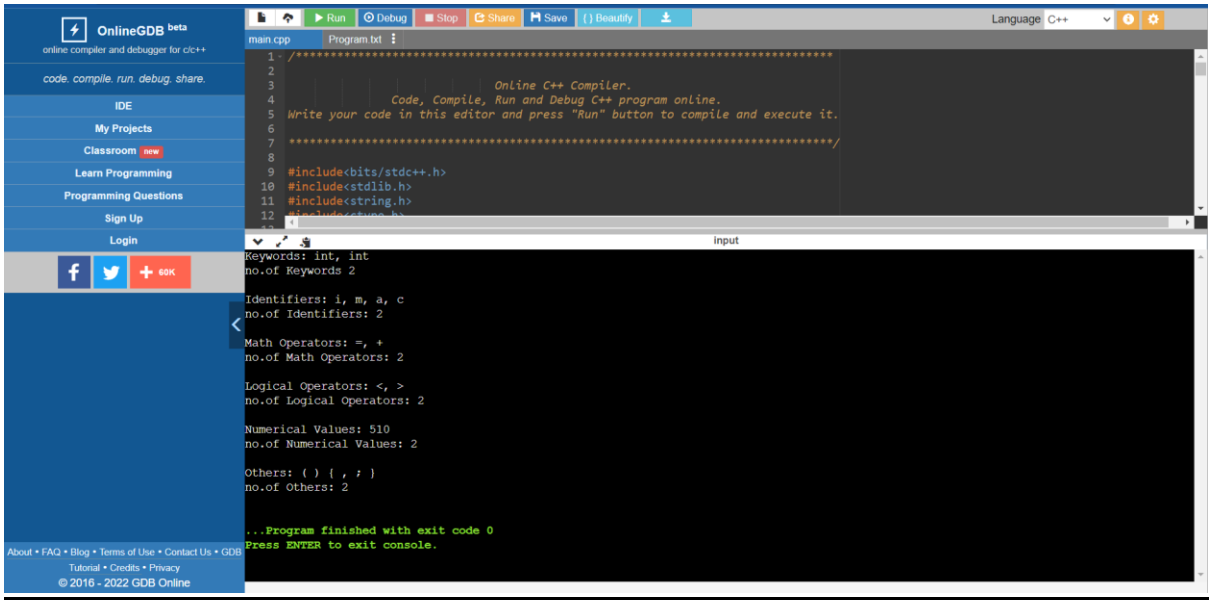
```
    count=0;
    printf("\nLogical Operators: ");
    for(int f=0;f<lc;++f){
        if(f==lc-1){
            cout<<lo[f]<<"\n";
            count++;
        }
        else {
            cout<<lo[f]<<", ";
        }

    }
     cout<<"no.of Logical Operators: "<<count+1<<endl;
    count=0;
    printf("\nNumerical Values: ");
    for(int f=0;f<nc;++f){
        if(f==nc-1){
            cout<<nu[f]<<"\n";
            count++;
        }
        else {
            cout<<nu[f]<<", ";
        }

    }
     cout<<"no.of Numerical Values: "<<count+1<<endl;
```

```
   count=0;
   printf("\nOthers: ");
   for(int f=0;f<oc;++f){
         if(f==oc-1){
            cout<<ot[f]<<"\n";
            count++;
         }
         else {
            cout<<ot[f]<<" ";
         }

   }
   cout<<"no.of Others: "<<count+1<<endl;

   return 0;
}
```

Program.txt:

```
#include<iostream>
int main()
{
   int a=5,b=10;
   cout<<a+b;
}
```

Output:



Result:

Lexical Analyser was studied and executed successfully in c++.

EXPERIMENT – 2

Regular Expression to NFA conversion

Aim:

To study and perform regular expression to NFA (non deterministic finite automata) conversion in any of the programming languages.

Language used: c++

Software used: online c++ compiler

Algorithm:

- Start
- Get the input from the user
- Initialise separate variables and functions for Postfix, Display and NFA.
- Create separate methods for different operators like +,*,.
- By using Switch case initialise different cases for the input
- For '.' operator initialise a separate method by using various stack functions. Do the same for other operators like *,+
- Regular expression is in the form of a.b(or) a+b
- Display the output
- Stop

Code:

```
#include<iostream>
#include<string.h>
int main()
{
   printf("Enter the regular expression: ");
        char reg[20];
```

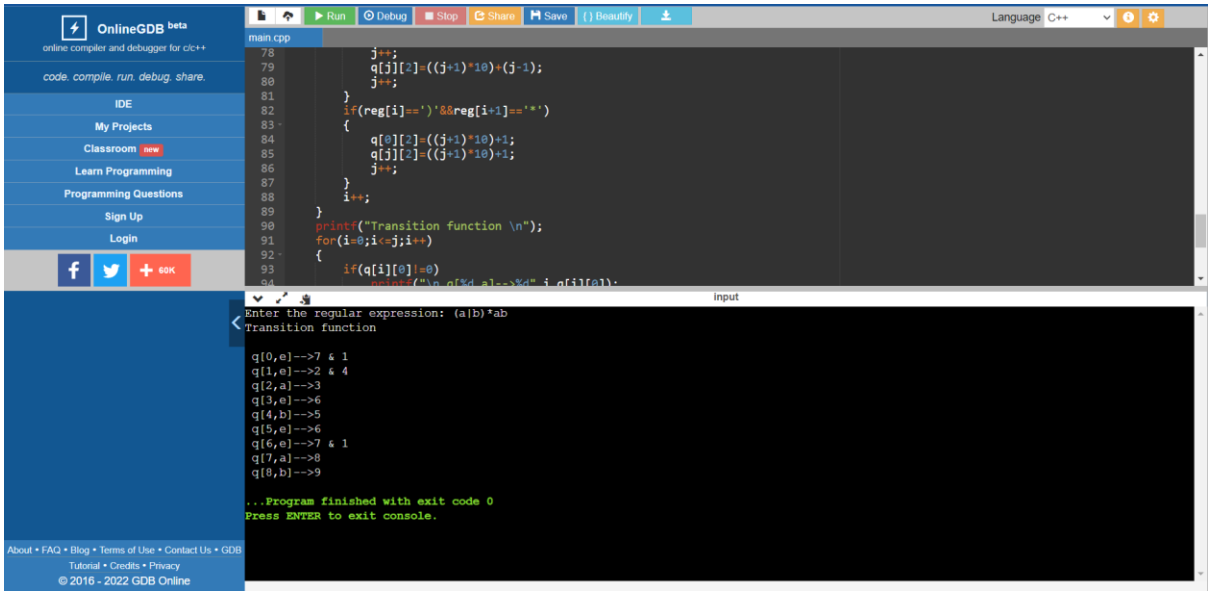
```
        int q[20][3],i,j,len,a,b;
        for(a=0;a<20;a++)
        {
                for(b=0;b<3;b++)
                {
                        q[a][b]=0;
                }
        }
        scanf("%s",reg);
        len=strlen(reg);
        i=0;
j=1;
        while(i<len)
        {
                if(reg[i]=='a'&&reg[i+1]!='|'&&reg[i+1]!='*')
                {
                        q[j][0]=j+1;
                        j++;
                }
                if(reg[i]=='b'&&reg[i+1]!='|'&&reg[i+1]!='*')
                {
                        q[j][1]=j+1;
                        j++;
                }
                if(reg[i]=='e'&&reg[i+1]!='|'&&reg[i+1]!='*')
                {
```

```
            q[j][2]=j+1;
            j++;
        }
        if(reg[i]=='a'&&reg[i+1]=='|'&&reg[i+2]=='b')
        {
            q[j][2]=((j+1)*10)+(j+3);
            j++;
            q[j][0]=j+1;
            j++;
            q[j][2]=j+3;
            j++;
            q[j][1]=j+1;
            j++;
            q[j][2]=j+1;
            j++;
            i=i+2;
        }
        if(reg[i]=='b'&&reg[i+1]=='|'&&reg[i+2]=='a')
        {
            q[j][2]=((j+1)*10)+(j+3);
            j++;
            q[j][1]=j+1;
            j++;
            q[j][2]=j+3;
            j++;
            q[j][0]=j+1;
```

```
            j++;
            q[j][2]=j+1;
            j++;
            i=i+2;
        }
        if(reg[i]=='a'&&reg[i+1]=='*')
        {
            q[j][2]=((j+1)*10)+(j+3);
            j++;
            q[j][0]=j+1;
            j++;
            q[j][2]=((j+1)*10)+(j-1);
            j++;
        }
        if(reg[i]=='b'&&reg[i+1]=='*')
        {
            q[j][2]=((j+1)*10)+(j+3);
            j++;
            q[j][1]=j+1;
            j++;
            q[j][2]=((j+1)*10)+(j-1);
            j++;
        }
        if(reg[i]==')'&&reg[i+1]=='*')
        {
            q[0][2]=((j+1)*10)+1;
```

```
                q[j][2]=((j+1)*10)+1;
                j++;
        }
        i++;
    }
    printf("Transition function \n");
    for(i=0;i<=j;i++)
    {
        if(q[i][0]!=0)
            printf("\n q[%d,a]-->%d",i,q[i][0]);
        if(q[i][1]!=0)
            printf("\n q[%d,b]-->%d",i,q[i][1]);
        if(q[i][2]!=0)
        {
            if(q[i][2]<10)
                printf("\n q[%d,e]-->%d",i,q[i][2]);
            else
                printf("\n q[%d,e]-->%d & %d",i,q[i][2]/10,q[i][2]%10);
        }
    }
    return 0;
}
```

Output:



Result:

The regular expression to NFA conversion was successfully executed in C++.

EXPERIMENT –3

NFA to DFA conversion

Aim:

To study and perform NFA to DFA conversion using any of the programming languages

Language used: c++

Software used: online c++ compiler

Algorithm:

1. In the main function we initialise a 3d-array vector to dynamically store the data for the transition table of NFA and then eventually convert it to DFA STEP
2. We take input for the total number of states in NFA and also total number of elements in the alphabet (no. of input symbols)
3. Then for each state we build nested for loops wherein for each state and for each input we take the number of output states STEP
4. Next whatever we get as a 3D array we insert it to the main 3D vector table STEP
5. We now call the print function wherein we display the 3d table elements
6. Next for the transition table of DFA we need to do the epsilon closure for that we need to compute 4 nested for loops within a while loop
7. Use the graph adjacency technique to track if for any of the state whether it is directing to a new state/adjacency location(this indicates state change according to given input symbol).If yes we take that row data and store it into another 3-d matrix which contains the solution
8. Now print the DFA table

Code:

```
#include<iostream>

#include<bits/stdc++.h>

using namespace std;

void print(vector<vector<vector<int> > > table){

cout<<" STATE/INPUT |";

char a='a';
```

```
for(int i=0;i<table[0].size()-1;i++){
cout<<" "<<a++<<" |";
}
cout<<" ^ "<<endl<<endl;
for(int i=0;i<table.size();i++){
cout<<" "<<i<<" ";
for(int j=0;j<table[i].size();j++){
cout<<" | ";
for(int k=0;k<table[i][j].size();k++){
cout<<table[i][j][k]<<" ";
}
}
cout<<endl;
}
}
void printdfa(vector<vector<int> > states, vector<vector<vector<int> > > dfa){
cout<<" STATE/INPUT ";
char a='a';
for(int i=0;i<dfa[0].size();i++){
cout<<"| "<<a++<<" ";
}
cout<<endl;
for(int i=0;i<states.size();i++){
cout<<"{ ";
for(int h=0;h<states[i].size();h++)
```

```
cout<<states[i][h]<<" ";
if(states[i].empty()){
cout<<"^ ";
}
cout<<"} ";
for(int j=0;j<dfa[i].size();j++){
cout<<" | ";
for(int k=0;k<dfa[i][j].size();k++){
cout<<dfa[i][j][k]<<" ";
}
if(dfa[i][j].empty()){
cout<<"^ ";
}
}
cout<<endl;
}
}
vector<int> closure(int s,vector<vector<vector<int> > > v){
vector<int> t;
queue<int> q;
t.push_back(s);
int a=v[s][v[s].size()-1].size();
for(int i=0;i<a;i++){
t.push_back(v[s][v[s].size()-1][i]);
//cout<<"t[i]"<<t[i]<<endl;
```

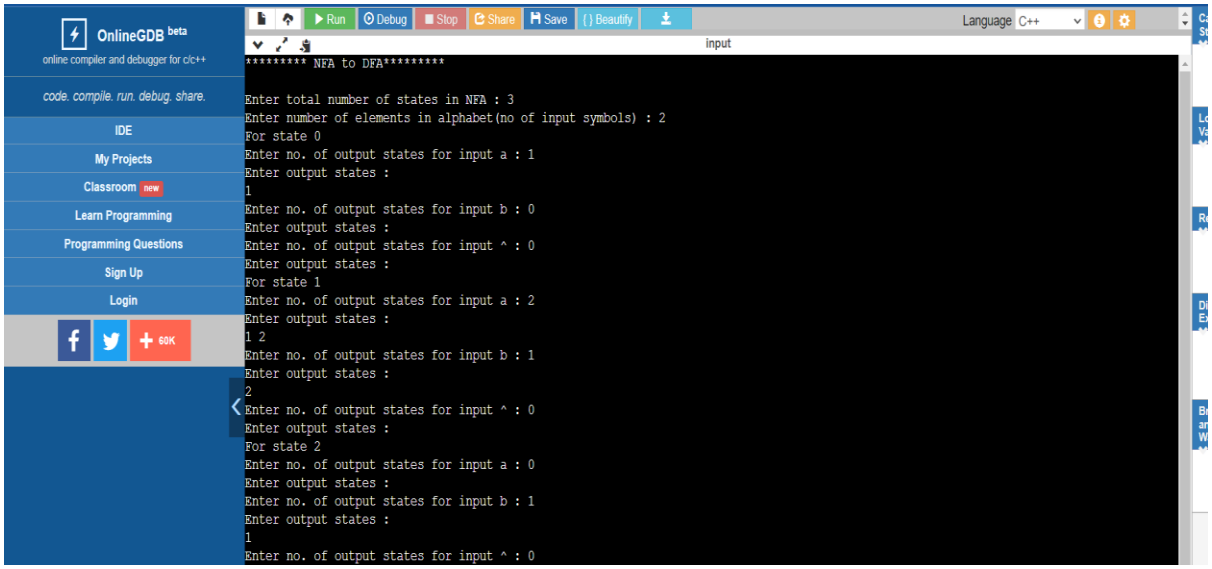
```
q.push(t[i]);
}
while(!q.empty()){
int f=q.front();
q.pop();
if(!v[f][v[f].size()-1].empty()){
int u=v[f][v[f].size()-1].size();
for(int i=0;i<u;i++){
int y=v[f][v[f].size()-1][i];
if(find(t.begin(),t.end(),y)==t.end()){
//cout<<"y"<<y<<endl;
t.push_back(y);
q.push(y);
}
}
}
}
return t;
}
int main(){
int n,alpha;
cout<<"********* NFA to DFA*********"<<endl<<endl;
cout<<"Enter total number of states in NFA : ";
cin>>n;
cout<<"Enter number of elements in alphabet(no of input symbols) : ";
```

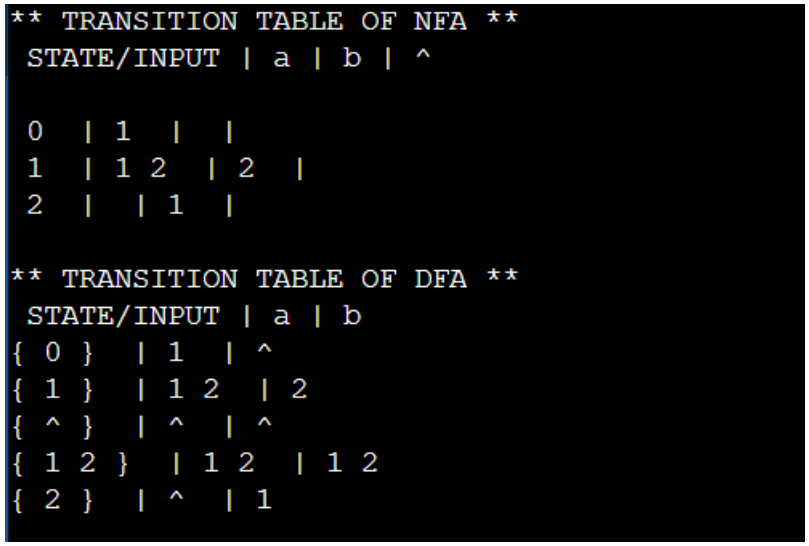
```
cin>>alpha;
vector<vector<vector<int> > > table;
for(int i=0;i<n;i++){
cout<<"For state "<<i<<endl;
vector< vector< int > > v;
char a='a';
int y,yn;
for(int j=0;j<alpha;j++){
vector<int> t;
cout<<"Enter no. of output states for input "<<a++<<" : ";
cin>>yn;
cout<<"Enter output states :"<<endl;
for(int k=0;k<yn;k++){
cin>>y;
t.push_back(y);
}
v.push_back(t);
}
vector<int> t;
cout<<"Enter no. of output states for input ^ : ";
cin>>yn;
cout<<"Enter output states :"<<endl;
for(int k=0;k<yn;k++){
cin>>y;
t.push_back(y);
```

```
}
v.push_back(t);
table.push_back(v);
}
cout<<"** TRANSITION TABLE OF NFA **"<<endl;
print(table);
cout<<endl<<"** TRANSITION TABLE OF DFA **"<<endl;
vector<vector<vector<int> > > dfa;
vector<vector<int> > states;
states.push_back(closure(0,table));
queue<vector<int> > q;
q.push(states[0]);
while(!q.empty()){
vector<int> f=q.front();
q.pop();
vector<vector<int> > v;
for(int i=0;i<alpha;i++){
vector<int> t;
set<int> s;
for(int j=0;j<f.size();j++){
for(int k=0;k<table[f[j]][i].size();k++){
vector<int> cl= closure(table[f[j]][i][k],table);
for(int h=0;h<cl.size();h++){
if(s.find(cl[h])==s.end())
s.insert(cl[h]);
```

```
}
}
}
for(set<int >::iterator u=s.begin(); u!=s.end();u++)
t.push_back(*u);
v.push_back(t);
if(find(states.begin(),states.end(),t)==states.end())
{
states.push_back(t);
q.push(t);
}
}
dfa.push_back(v);
}
printdfa(states,dfa);
}
```

Output:





Result:

The NFA to DFA conversion was successfully executed in C++.

EXPERIMENT - 4a

ELIMINATION OF LEFT RECURSION

Aim:

To study and code a program for elimination of left recursion.

Language used: c++

Software used: online c++ compiler

Algorithm:

1. Start the program.

2. Initialize the arrays for taking input from the user.

3. Prompt the user to input the no. of non-terminals having left recursion and no. of productions for these non-terminals.

4. Prompt the user to input the production for non-terminals.

5. Eliminate left recursion using the following rules:-

A->Aα1| Aα2 | . . . . . |Aαm

A->β1| β2| . . . . .| βn

Then replace it by

A-> βi A' i=1,2,3,…..m

A'-> αj A' j=1,2,3,…..n

A'-> Ɛ

6. After eliminating the left recursion by applying these rules, display the productions without left recursion.

7. Stop

Code:

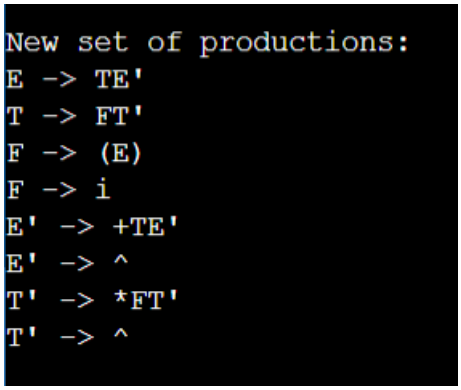
```
#include <iostream>
```

```
#include <vector>
#include <string>
using namespace std;

int main()
{
   int n;
   cout<<"\nEnter number of non terminals: ";
   cin>>n;
   cout<<"\nEnter non terminals one by one: ";
   int i;
   vector<string> nonter(n);
   vector<int> leftrecr(n,0);
   for(i=0;i<n;++i) {
         cout<<"\nNon terminal "<<i+1<<" : ";
      cin>>nonter[i];
   }
   vector<vector<string> > prod;
   cout<<"\nEnter '^' for null";
   for(i=0;i<n;++i) {
      cout<<"\nNumber of "<<nonter[i]<<" productions: ";
      int k;
      cin>>k;
      int j;
      cout<<"\nOne by one enter all "<<nonter[i]<<" productions";
      vector<string> temp(k);
```

```
      for(j=0;j<k;++j) {
         cout<<"\nRHS of production "<<j+1<<": ";
         string abc;
         cin>>abc;
         temp[j]=abc;

if(nonter[i].length()<=abc.length()&&nonter[i].compare(abc.substr(0,nonter[i].length())
)==0)
            leftrecr[i]=1;
      }
      prod.push_back(temp);
   }
   for(i=0;i<n;++i) {
      cout<<leftrecr[i];
   }
   for(i=0;i<n;++i) {
      if(leftrecr[i]==0)
         continue;
      int j;
      nonter.push_back(nonter[i]+"'");
      vector<string> temp;
      for(j=0;j<prod[i].size();++j) {

if(nonter[i].length()<=prod[i][j].length()&&nonter[i].compare(prod[i][j].substr(0,nonter[
i].length()))==0) {
            string abc=prod[i][j].substr(nonter[i].length(),prod[i][j].length()-
nonter[i].length())+nonter[i]+"'";
            temp.push_back(abc);
```

```
            prod[i].erase(prod[i].begin()+j);
            --j;
         }
         else {
            prod[i][j]+=nonter[i]+"'";
         }
      }
      temp.push_back("^");
      prod.push_back(temp);
   }
   cout<<"\n\n";
   cout<<"\nNew set of non-terminals: ";
   for(i=0;i<nonter.size();++i)
      cout<<nonter[i]<<" ";
   cout<<"\n\nNew set of productions: ";
   for(i=0;i<nonter.size();++i) {
      int j;
      for(j=0;j<prod[i].size();++j) {
         cout<<"\n"<<nonter[i]<<" -> "<<prod[i][j];
      }
   }
   return 0;
}
```

Output:



Result:

A proper working code has been studied, written and executed successfully in C++.

EXPERIMENT – 4b

ELIMINATION OF LEFT FACTORING

Aim:

To study and code a program for ELIMINATION OF LEFT FACTORING

Language used: c++

Software used: online c++ compiler

Algorithm:

1. Start

2. Ask the user to enter the set of productions

3. Check for common symbols in the given set of productions by comparing

with:

A->aB1|aB2

4. If found, replace the particular productions with:

A->aA'

A'->B1|B2| Ɛ

5. Display the output

6. Exit

Code:

```
#include<string.h>
#include <iostream>
#include<stdlib.h>
int 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];
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");
    }
    return 0;}
```

}**Output:**

Result: A proper working code has been studied, written and executed successfully in C++.

EXPERIMENT - 5

FIRST & FOLLOW COMPUTATION

Aim:

To compute FIRST() and FOLLOW() for the given language.

Language used: c++

Software used: online c++ compiler

Algorithm:

Store the grammar on a 2D character array production. findfirst function is for calculating the first of any non terminal. Calculation of first falls under two broad cases :

- If the first symbol in the R.H.S of the production is a Terminal then it can directly be included in the first set.
- If the first symbol in the R.H.S of the production is a Non-Terminal then call the findfirst function again on that Non-Terminal. To handle these cases like Recursion is the best possible solution. Here again, if the First of the new Non-Terminal contains an epsilon then we have to move to the next symbol of the original production which can again be a Terminal or a Non-Terminal.

Code:

main.cpp:

```
#include<bits/stdc++.h>
using namespace std;
set<char> ss;
bool dfs(char i, char org, char last, map<char,vector<vector<char>>> &mp){
   bool rtake = false;
```

```
    for(auto r : mp[i]){
        bool take = true;
        for(auto s : r){
            if(s == i) break;
            if(!take) break;
            if(!(s>='A'&&s<='Z')&&s!='e'){
                ss.insert(s);
                break;
            }
            else if(s == 'e'){
                if(org == i||i == last)
                ss.insert(s);
                rtake = true;
                break;
            }
            else{
                take = dfs(s,org,r[r.size()-1],mp);
                rtake |= take;
            }
        }
    }
    return rtake;
}
int main(){
    int i,j;
    ifstream fin("input.txt");
```

```
    string num;
    vector<int> fs;
    vector<vector<int>> a;
    map<char,vector<vector<char>>> mp;
    char start;
    bool flag = 0;
    cout<<"Grammar: "<<'\n';
    while(getline(fin,num)){
        if(flag == 0) start = num[0],flag = 1;
        cout<<num<<'\n';
        vector<char> temp;
        char s = num[0];
        for(i=3;i<num.size();i++){
            if(num[i] == '|'){
                mp[s].push_back(temp);
                temp.clear();
            }
            else temp.push_back(num[i]);
        }
        mp[s].push_back(temp);
    }
    map<char,set<char>> fmp;
    for(auto q : mp){
        ss.clear();
        dfs(q.first,q.first,q.first,mp);
        for(auto g : ss) fmp[q.first].insert(g);
```

```
    }
    cout<<'\n';
    cout<<"FIRST: "<<'\n';
    for(auto q : fmp){
        string ans = "";
        ans += q.first;
        ans += " = {";
        for(char r : q.second){
            ans += r;
            ans += ',';
        }
        ans.pop_back();
        ans+="}";
        cout<<ans<<'\n';
    }
    map<char,set<char>> gmp;
    gmp[start].insert('$');
    int count = 10;
    while(count--){
        for(auto q : mp){
            for(auto r : q.second){
                for(i=0;i<r.size()-1;i++){
                    if(r[i]>='A'&&r[i]<='Z'){
                        if(!(r[i+1]>='A'&&r[i+1]<='Z')) gmp[r[i]].insert(r[i+1]);
                        else {
                            char temp = r[i+1];
```

```
int j = i+1;
while(temp>='A'&&temp<='Z'){
   if(*fmp[temp].begin()=='e'){
      for(auto g : fmp[temp]){
         if(g=='e') continue;
         gmp[r[i]].insert(g);
      }
      j++;
      if(j<r.size()){
         temp = r[j];
         if(!(temp>='A'&&temp<='Z')){
            gmp[r[i]].insert(temp);
            break;
         }
      }
      else{
         for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
         break;
      }
   }
   else{
      for(auto g : fmp[temp]){
         gmp[r[i]].insert(g);
      }
      break;
   }
```

```
                    }
                }
            }
        }
        if(r[r.size()-1]>='A'&&r[r.size()-1]<='Z'){
            for(auto g : gmp[q.first]) gmp[r[i]].insert(g);
        }
      }
    }
  }
  cout<<'\n';
  cout<<"FOLLOW: "<<'\n';
  for(auto q : gmp){
    string ans = "";
    ans += q.first;
    ans += " = {";
    for(char r : q.second){
      ans += r;
      ans += ',';
    }
    ans.pop_back();
    ans+="}";
    cout<<ans<<'\n';
  }
  return 0;
}
```

Input:

input.txt:

main.cpp | input.txt

```
1  S->ACB|CbB|Ba
2  A->da|BC
3  B->g|e
4  C->h|e
```

Output:

main.cpp | input.txt

```
22                  take = dfs(s,org,r[r.size()-1],mp);
23                  rtake |= take;
24              }
25          }
26      }
27      return rtake;
28  }
29  int main(){
30      int i,j;
31      ifstream fin("input.txt");
32      string num;
33      vector<int> fs;
34      vector<vector<int>> a;
```

input

```
Grammar:
S->ACB|CbB|Ba
A->da|BC
B->g|e
C->h|e

FIRST:
A = {d,e,g,h}
B = {e,g}
C = {e,h}
S = {a,b,d,e,g,h}

FOLLOW:
A = {$,g,h}
B = {$,a,g,h}
C = {$,b,g,h}
S = {$}


...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

The FIRST() and FOLLOW() of the given grammar is computed.

EXPERIMENT - 6

Predictive Parsing Table

Aim:

To construct a predictive parsing table for a given grammar.

Language used: c++

Software used: online c++ compiler

Algorithm:

- Input the no of productions in the grammar
- Input the productions from the user in the form of a string
- Eliminate any left recursion or left factoring if present in the grammar
- compute the first for the given grammar using the following rules:
  1. If x is a terminal, then FIRST(x) = { 'x' }
  2. If x-> Є, is a production rule, then add Є to FIRST(x).
  3. If X->Y1 Y2 Y3….Yn is a production, FIRST(X) = FIRST(Y1)
  4. If FIRST(Y1) contains Є then FIRST(X) = { FIRST(Y1) – Є } U { FIRST(Y2) }
  5. If FIRST (Yi) contains Є for all i = 1 to n, then add Є to FIRST(X)
- Compute the follow for the given grammar using the following rules:
  1. FOLLOW(S) = { $ } // where S is the starting Non-Terminal
  2. If A -> pBq is a production, where p, B and q are any grammar symbols, then everything in FIRST(q) except Є is in FOLLOW(B)
  3. If A->pB is a production, then everything in FOLLOW(A) is in FOLLOW(B).
  4. If A->pBq is a production and FIRST(q) contains Є, then FOLLOW(B) contains { FIRST(q) – Є } U FOLLOW(A)
- Print the first and follow productions derived
- Print the predictive parsing table using the first and follow produced

Code:

```
#include <iostream>
#include<string.h>
int main()
{
char fin[10][20],st[10][20],ft[20][20],fol[20][20];
int a=0,e,i,t,b,c,n,k,l=0,j,s,m,p;
printf("enter the no. of productions\n");
scanf("%d",&n);
printf("enter the productions in a grammar\n");
for(i=0;i<n;i++)
scanf("%s",st[i]);
for(i=0;i<n;i++)
fol[i][0]='\0';
for(s=0;s<n;s++)
{
for(i=0;i<n;i++)
{
j=3;
l=0;
a=0;
l1:if(!((st[i][j]>64)&&(st[i][j]<91)))
{
for(m=0;m<l;m++)
{
if(ft[i][m]==st[i][j])
```

```
goto s1;
}
ft[i][l]=st[i][j];
l=l+1;
s1:j=j+1;
}
else
{
if(s>0)
{
while(st[i][j]!=st[a][0])
{
a++;
}
b=0;
while(ft[a][b]!='\0')
{
for(m=0;m<l;m++)
{
if(ft[i][m]==ft[a][b])
goto s2;
}
ft[i][l]=ft[a][b];
l=l+1;
s2:b=b+1;
}
```

```
}
}
while(st[i][j]!='\0')
{
if(st[i][j]=='|')
{
j=j+1;
goto l1;
}
j=j+1;
}
ft[i][l]='\0';
}
}
printf("first productions\n");
for(i=0;i<n;i++)
printf("FIRS[%c]=%s\n",st[i][0],ft[i]);
fol[0][0]='$';
for(i=0;i<n;i++)
{
k=0;
j=3;
if(i==0)
l=1;
else
l=0;
```

```
k1:while((st[i][0]!=st[k][j])&&(k<n))
{
if(st[k][j]=='\0')
{
k++;
j=2;
}
j++;
}
j=j+1;
if(st[i][0]==st[k][j-1])
{
if((st[k][j]!='|')&&(st[k][j]!='\0'))
{
a=0;
if(!((st[k][j]>64)&&(st[k][j]<91)))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==st[k][j])
goto q3;
}
fol[i][l]=st[k][j];
l++;
q3:p++;
}
```

```
else
{
while(st[k][j]!=st[a][0])
{
a++;
}
p=0;
while(ft[a][p]!='\0')
{
if(ft[a][p]!='e')
{
for(m=0;m<l;m++)
{
if(fol[i][m]==ft[a][p])
goto q2;
}
fol[i][l]=ft[a][p];
l=l+1;
}
else
e=1;
q2:p++;
}
if(e==1)
{
e=0;
```

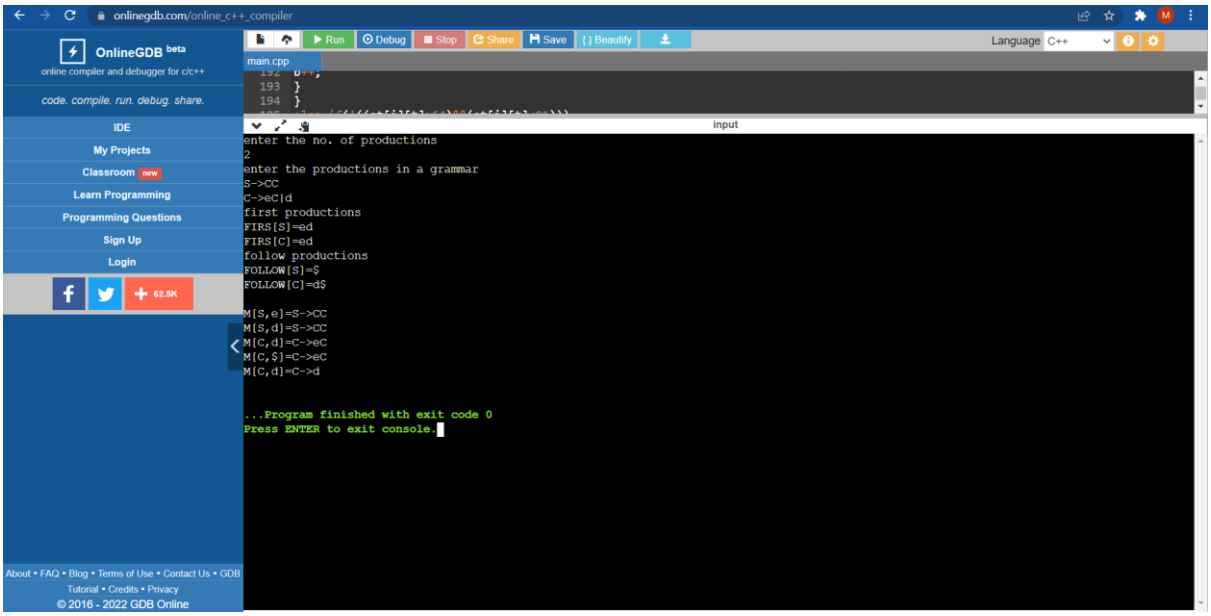
```
goto a1;
}
}
}
else
{
a1:c=0;
a=0;
while(st[k][0]!=st[a][0])
{
a++;
}
while((fol[a][c]!='\0')&&(st[a][0]!=st[i][0]))
{
for(m=0;m<l;m++)
{
if(fol[i][m]==fol[a][c])
goto q1;
}
fol[i][l]=fol[a][c];
l++;
q1:c++;
}
}
goto k1;
}
```

```
fol[i][l]='\0';
}
printf("follow productions\n");
for(i=0;i<n;i++)
printf("FOLLOW[%c]=%s\n",st[i][0],fol[i]);
printf("\n");
s=0;
for(i=0;i<n;i++)
{
j=3;
while(st[i][j]!='\0')
{
if((st[i][j-1]=='|')||(j==3))
{
for(p=0;p<=2;p++)
{
fin[s][p]=st[i][p];
}
t=j;
for(p=3;((st[i][j]!='|')&&(st[i][j]!='\0'));p++)
{
fin[s][p]=st[i][j];
j++;
}
fin[s][p]='\0';
if(st[i][t]=='e')
```

```
{
b=0;
a=0;
while(st[a][0]!=st[i][0])
{
a++;
}
while(fol[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],fol[a][b],fin[s]);
b++;
}
}
else if(!((st[i][t]>64)&&(st[i][t]<91)))
printf("M[%c,%c]=%s\n",st[i][0],st[i][t],fin[s]);
else
{
b=0;
a=0;
while(st[a][0]!=st[i][3])
{
a++;
}
while(ft[a][b]!='\0')
{
printf("M[%c,%c]=%s\n",st[i][0],ft[a][b],fin[s]);
```

```
b++;
}
}
s++;
}
if(st[i][j]=='|')
j++;
}
}
return 0;
}
```

Output:



Result:

Predictive Parsing Table has been constructed successfully for the given grammar using c++.

EXPERIMENT - 7

SHIFT REDUCE PARSER

Aim:

To implement a shift reduce parser.

Language used: c++

Software used: online c++ compiler

Algorithm:

1. Start the program
2. Input the number of productions
3. Input the string to be parsed
4. Perform the following basic operations on the string:

- SHIFT: involves moving the symbols from the input buffer onto the stack
- REDUCE: if the handle appears on top of the stack then, its reduction by using production rule is done i.e. RHS of a production rule is pushed onto the stack.
- ACCEPT: if only the start symbol is present in the stack and the input buffer is empty

then, the parsing action is called accept. When accepted action is obtained, it means

successful parsing is done.

- ERROR: this is the situation in which the parser can neither perform shift action nor

reduce action and not even accept action

5. Start matching the character with the given productions

- If only start symbol is left, string is accepted
- Else string is rejected

6. Stop the execution

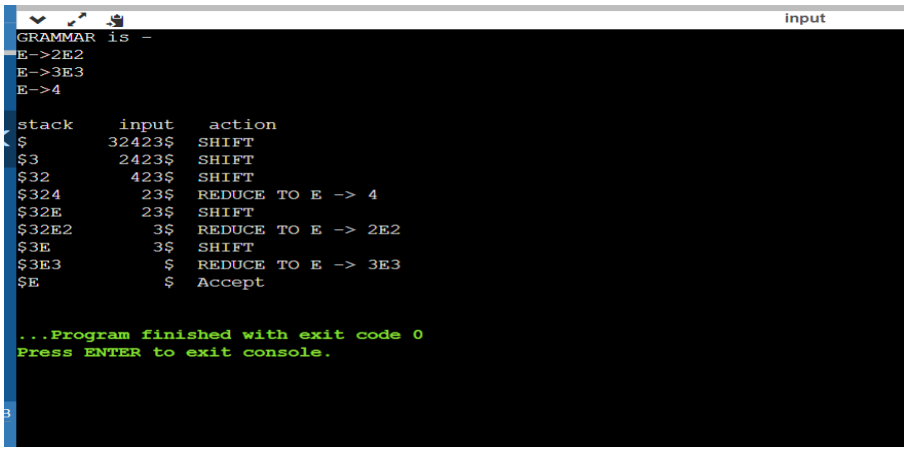
Code:

```
#include <bits/stdc++.h>
using namespace std;
int z = 0, i = 0, j = 0, c = 0;
char a[16], ac[20], stk[15], act[10];
 void check()
{
 strcpy(ac,"REDUCE TO E -> ");
 for(z = 0; z < c; z++)
 {
 if(stk[z] == '4')
 {
 printf("%s4", ac);
 stk[z] = 'E';
 stk[z + 1] = '\0';
 printf("\n$%s\t%s$\t", stk, a);
 } }
for(z = 0; z < c - 2; z++)
 {
 if(stk[z] == '2' && stk[z + 1] == 'E' &&
 stk[z + 2] == '2')
 {
 printf("%s2E2", ac);
 stk[z] = 'E';
 stk[z + 1] = '\0';
 stk[z + 2] = '\0';
```

```
 printf("\n$%s\t%s$\t", stk, a);
 i = i - 2;
 }
 }
 for(z = 0; z < c - 2; z++)
 {
 if(stk[z] == '3' && stk[z + 1] == 'E' &&
 stk[z + 2] == '3')
 {
 printf("%s3E3", ac);
 stk[z]='E';
 stk[z + 1]='\0';
 stk[z + 1]='\0';
 printf("\n$%s\t%s$\t", stk, a);
 i = i - 2;
 }
 }
 return ;
}
int main()
{
 printf("GRAMMAR is -\nE->2E2 \nE->3E3 \nE->4\n");
 strcpy(a,"32423");
 c=strlen(a);
 strcpy(act,"SHIFT");
 printf("\nstack \t input \t action");
```

```
 printf("\n$\t%s$\t", a);
 for(i = 0; j < c; i++, j++)
 {
 printf("%s", act);
 stk[i] = a[j];
 stk[i + 1] = '\0';
 a[j]=' ';
 printf("\n$%s\t%s$\t", stk, a);
 check();
 }
 check();
 if(stk[0] == 'E' && stk[1] == '\0')
 printf("Accept\n");
 else //else reject
 printf("Reject\n");
}
```

Output:



Result: A proper working code was executed successfully in C++.

EXPERIMENT - 8

Leading and Trailing

AIM : A program to implement Leading and Trailing

ALGORITHM :

1. For Leading, check for the first non-terminal.
2. If found, print it.
3. Look for next production for the same non-terminal.
4. If not found, recursively call the procedure for the single non-terminal present before the

comma or End Of Production String.

5. Include it's results in the result of this non-terminal.
6. For trailing, we compute same as leading but we start from the end of the production to the beginning.
7. Stop

CODE :

```
#include<iostream
.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
int vars,terms,i,j,k,m,rep,count,temp=-1;
char
var[10],term[10],lead[10][10],trail[10][1
0]; struct grammar
```

```
  {
int prodno;

char lhs,rhs[20][20];
  }gram[50
  ]; void
  get()
  {
cout<<"\n---------- LEADING AND TRAILING ----------\n";
cout<<"\nEnter the no. of variables : ";
cin>>vars;
cout<<"\nEnter the variables : \n";
for(i=0;i<vars;i++)
{
                cin>>gram[i].lh
                s;
                var[i]=gram[i].l
                hs;
}
cout<<"\nEnter the no. of terminals : ";
cin>>terms;
cout<<"\nEnter the terminals : ";
for(j=0;j<terms;j++)
                cin>>term[j];
cout<<"\n---------- PRODUCTION DETAILS ----------\n";
for(i=0;i<vars;i++)
{
                cout<<"\nEnter the no. of production of
```

```
                "<<gram[i].lhs<<":"; cin>>gram[i].prodno;
                for(j=0;j<gram[i].prodno;j++)
                {
                        cout<<gram[i].lhs<<"-
                        >";
                        cin>>gram[i].rhs[j];
                }
}
  }
  void leading()
  {
for(i=0;i<vars;i++)
{
                for(j=0;j<gram[i].prodno;j++)
                {
                        for(k=0;k<terms;k++)
                        {
                                if(gram[i].rhs[j][0]==term
                                        [k]) lead[i][k]=1;
                                els
                                e
                                {
                                        if(gram[i].rhs[j][1]==term
                                                [k]) lead[i][k]=1;
                                }
                        }
                }
        }


for(rep=0;rep<vars;rep++)
{
```

```
                for(i=0;i<vars;i++)
                {
                        for(j=0;j<gram[i].prodno;j++)
                        {
                                for(m=1;m<vars;m++)
                                {
                                        if(gram[i].rhs[j][0]==var[m])
                                        {
                                                temp=
                                                  m;
                                                goto
                                                 out;
                                        }

out: for(k=0;k<terms;k++)
                                {
                                     if(lead[temp][k]==1)
                                             lead[i][k]=1;
                                }
                        }
                }
}
  }
  void trailing()
  {
for(i=0;i<vars;i++)
{
                for(j=0;j<gram[i].prodno;j++)
                {
                        count=0;
```

```
while(gram[i].rhs[j][count]!='
\x0')
        count++;
for(k=0;k<terms;k
++)
{
        if(gram[i].rhs[j][count-
                1]==term[k])
                trail[i][k]=1;
```

```
                        else
                        {



                        }
                }
        }
}
    if(gram[i].rhs[j][count-2]==term[k]) trail[i][k]=1;
for(rep=0;rep<vars;rep++)
{
        for(i=0;i<vars;i++)
        {
                for(j=0;j<gram[i].prodno;j++)
                {
                        count=0;
                        while(gram[i].rhs[j][count]!='\x0
                        ')
                                count++;
                        for(m=1;m<vars;m++)
                        {
                                if(gram[i].rhs[j][count-
                                        1]==var[m]) temp=m;
                        }
                        for(k=0;k<terms;k++)
                        {
                               if(trail[temp][k]==1)
                                        trail[i][k]=1;
                        }
```

```
                        }
                }
}
  }
  void display()
  {
for(i=0;i<vars;i++)
{
                cout<<"\nLEADING("<<gram[i].lhs<<") = ";
                for(j=0;j<terms;j++)
                {
                        if(lead[i][j]==1)
                                cout<<term[j]<<",";
                }
}
cout<<endl; for(i=0;i<vars;i++)
{
                cout<<"\nTRAILING("<<gram[i].lhs<<") = ";
                for(j=0;j<terms;j++)
                {
                        if(trail[i][j]==1)
                                cout<<term[j]<<",";
                }
}
  }
  void main()
  {
clrscr();
get(); leading();
trailing();
```

```
display();
getch();
  }
```

OUTPUT :

```
LEADING AND TRAILING

Enter the no. of variables : 3

Enter the variables :
A
S
B

Enter the no. of terminals : 4

Enter the terminals : +
-
*
i

PRODUCTION DETAILS

Enter the no. of production of A:2
A->A+S
A->S
```

```
Enter the no. of production of S:2
S->S*F
S->B

Enter the no. of production of B:2
B->-E
B->I

LEADING(A) = +,-,*,
LEADING(S) = -,*,
LEADING(B) = -,

TRAILING(A) = +,-,*,
TRAILING(S) = -,*,
TRAILING(B) = -,
```

RESULT :

The program was successfully compiled and run.

EXPERIMENT - 9

Computation of LR(0) Items

Aim: A program to implement LR(0) items

Algorithm:-

1. Start.
2. Create structure for production with LHS and RHS.
3. Open file and read input from file.
4. Build state 0 from extra grammar Law S' -> S $ that is all start symbol of grammar and one Dot (.) before S symbol.
5. If Dot symbol is before a non-terminal, add grammar laws that this non-terminal is in Left Hand Side of that Law and set Dot in before of first part of Right Hand Side.
6. If state exists (a state with this Laws and same Dot position), use that instead.
7. Now find set of terminals and non-terminals in which Dot exist in before.
8. If step 7 Set is non-empty go to 9, else go to 10.
9. For each terminal/non-terminal in set step 7 create new state by using all grammar law that Dot position is before of that terminal/non-terminal in reference state by increasing Dot point to next part in Right Hand Side of that laws.
10. Go to step 5.
11. End of state building.
12. Display the output.
13. End.

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 THE GRAMMAR(0 TO END) :
E->T+E
E->T
T->id
0


 augumented grammar

A->E
E->T+E
E->T
T->id
 THE SET OF ITEMS ARE


 I0

A->.E
E->.T+E
E->.T
T->.id

 I1

A->E.
```

```
 I2

E->T.+E
E->T.

 I3

T->i.d

 I4

E->T+.E
E->.T+E
E->.T
T->.id

 I5

T->id.

 I6

E->T+E.
```

Result:-

The program was successfully compiled and run.

Experiment – 10

Intermediate code Generation for Postfix and Prefix expression

Aim: A program to implement Intermediate code generation – Postfix, Prefix.

Algorithm:

1. Declare set of operators.
2. Initialize an empty stack.
3. To convert INFIX to POSTFIX follow the following steps
4. Scan the infix expression from left to right.
5. If the scanned character is an operand, output it.
6. Else, If the precedence of the scanned operator is greater than the precedence of the operator in the stack(or the stack is empty or the stack contains a '('), push it.
7. Else, Pop all the operators from the stack which are greater than or equal to in precedence than that of the scanned operator. After doing that Push the scanned operator to the stack.
8. If the scanned character is an '(', push it to the stack.
9. If the scanned character is an ')', pop the stack and output it until a '(' is encountered, and discard both the parenthesis.
10. Pop and output from the stack until it is not empty.
11. To convert INFIX to PREFIX follow the following steps
12. First, reverse the infix expression given in the problem.
13. Scan the expression from left to right.
14. Whenever the operands arrive, print them.
15. If the operator arrives and the stack is found to be empty, then simply push the operator into the stack.
16. Repeat steps 6 to 9 until the stack is empty.

Code:

```
#include<stdio.h>

#include<stdlib.h>

#include<ctype.h>

#include<string.h>
```

```
#define SIZE 100



char stack[SIZE];
int top = -1;

void push(char item)
{
        if(top >= SIZE-1)
        {
                printf("\nStack Overflow.");
        }
        else
        {
                top = top+1;
                stack[top] = item;
        }
}
char pop()
{
        char item ;

        if(top <0)
        {
                printf("stack under flow: invalid infix expression");
                getchar();
```

```
                exit(1);
        }
        else
        {
                item = stack[top];
                top = top-1;
                return(item);
        }
}
int is_operator(char symbol)
{
        if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' || symbol
=='-')
        {
                return 1;
        }
        else
        {
        return 0;
        }
}
int precedence(char symbol)
{
        if(symbol == '^')
        {
                return(3);
        }
```

```
        else if(symbol == '*' || symbol == '/')
        {
                return(2);
        }
        else if(symbol == '+' || symbol == '-')
        {
                return(1);
        }
        else
        {
                return(0);
        }
}

void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
        int i, j;
        char item;
        char x;

        push('(');
        strcat(infix_exp,")");

        i=0;
        j=0;
        item=infix_exp[i];
```

```
        while(item != '\0')
        {
                if(item == '(')
                {
                        push(item);
                }
                else if( isdigit(item) || isalpha(item))
                {
                        postfix_exp[j] = item;
                        j++;
                }
                else if(is_operator(item) == 1)
                {
                        x=pop();
                        while(is_operator(x) == 1 && precedence(x)>=
precedence(item))
                        {
                                postfix_exp[j] = x;
                                j++;
                                x = pop();
                        }
                        push(x);

                        push(item);
                }
                else if(item == ')')
                {
```

```
                x = pop();
                while(x != '(')
                {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                }
        }
        else
        {
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }
        i++;


        item = infix_exp[i];
}
if(top>0)
{
        printf("\nInvalid infix Expression.\n");
        getchar();
        exit(1);
}
if(top>0)
{
```

```
                printf("\nInvalid infix Expression.\n");
                getchar();
                exit(1);
        }


        postfix_exp[j] = '\0';
}
int main()
{
        char infix[SIZE], postfix[SIZE];
        printf("ASSUMPTION: The infix expression contains single letter variables and single digit constants only.\n");
        printf("\nEnter Infix expression : ");
        gets(infix);

        InfixToPostfix(infix,postfix);
        printf("Postfix Expression: ");
        puts(postfix);

        return 0;
}
```

Output:

```
Given expression: {()[]{}([{}]
Brackets are not balanced.

Given expression: ({[]({}})
Brackets are balanced.

Before reversal: abcdefghij
After reversal: jihgfedcba

Infix expression: A+B+C+D
Postfix expression: AB+C+D+

Infix expression: A*B+C*D
Postfix expression: AB*CD*+
```

Result: The code was successfully implemented and output was recorded.

Experiment – 11

Intermediate Code Generation – Quadruple, Triple, Indirect Triple

Aim: A program to implement intermediate code generation - Quadruple, Triple, Indirect triple.

Algorithm:

The algorithm takes a sequence of three-address statements as input. For each three address statements of the form a:= b op c perform the various actions. These are as follows:

1. Invoke a function getreg to find out the location L where the result of computation b op c should be stored.
2. Consult the address description for y to determine y'. If the value of y currently in memory and register both then prefer the register y' . If the value of y is not already in L then generate the instruction MOV y' , L to place a copy of y in L.
3. Generate the instruction OP z' , L where z' is used to show the current location of z. if z is in both then prefer a register to a memory location. Update the address descriptor of x to indicate that x is in location L. If x is in L then update its descriptor and remove x from all other descriptors.
4. If the current value of y or z have no next uses or not live on exit from the block or in register then alter the register descriptor to indicate that after execution of x : = y op z those register will no longer contain y or z.

Program:

```
#include<iostream>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
using namespace std;
```

```
 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];
 int main()
 {
printf("Enter the expression : ");
scanf("%s",j);
printf("The 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]))
{
```

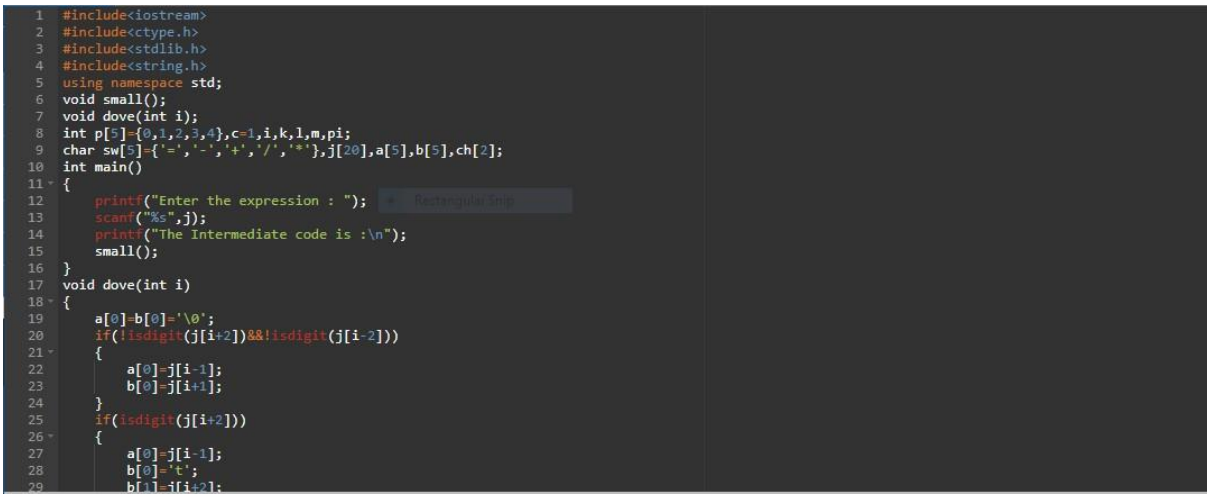
```
        b[0]=j[i+1];
        a[0]='t';
        a[1]=j[i-2];
        b[1]='\0';
}
if(isdigit(j[i+2]) &&isdigit(j[i-2]))
{

        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("t%d=%s*%s\n",
        c,a,b);
    if(j[i]=='/')
        printf("t%d=%s/%s\n",
        c,a,b);
    if(j[i]=='+')
        printf("t%d=%s+%s\n",c,a,b);if(j[
        i]=='-')
printf("t%d=%s-
%s\n",c,a,b); if(j[i]=='=')
   printf("%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);
 }
```

Input:

a=b+c-d

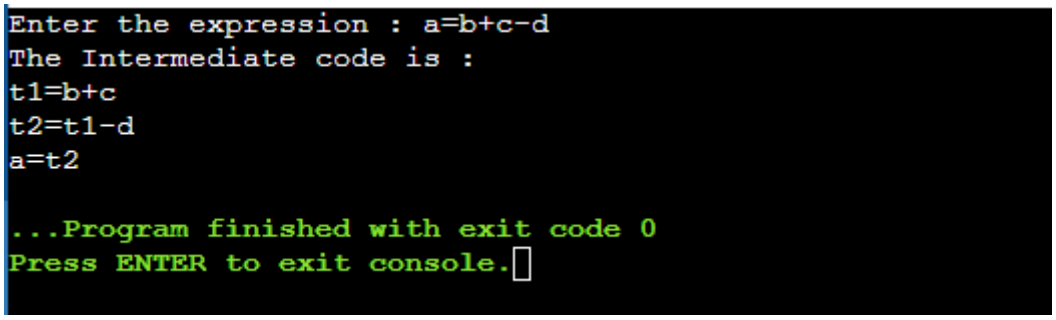
Code Screenshots:



```
#include<iostream>
#include<ctype.h>
#include<stdlib.h>
#include<string.h>
using namespace std;
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];
int main()
{
    printf("Enter the expression : ");
    scanf("%s",j);
    printf("The 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]=i[i+2];
```

```
    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);
}
```

Output:



```
Enter the expression : a=b+c-d
The Intermediate code is :
t1=b+c
t2=t1-d
a=t2

...Program finished with exit code 0
Press ENTER to exit console.
```

Result:

A program to implement intermediate code generation - Quadruple, Triple, Indirect triple has been compiled and run successfully.