COMSATS UNIVERSITY ISLAMABAD, LAHORE CAMPUS

DEPARTMENT OF COMPUTER SCIENCE

CSC459: LAB-COMPILER DESIGN AND CONSTRUCTION

LAB MANUAL

INSTRUCTOR: M Mudassar

Contents

Objective of Compiler Design and Construction Lab

Compiler is a **System Software** that converts High level language to low level language. We human beings can't program in machine language (low level lang.) understood by Computers so we program. In high level language and compiler is the software which bridges the gap between user and computer

1. Lexical Analysis
2. Syntax Analysis
3. Semantic Analysis
4. Intermediate Code Generation
5. Code Optimization
6. Code Generation

In the lab sessions students implement Lexical Analyzers and code for each phase to understand compiler software working and its coding in detail.

Lab 1& 2 – An Introduction

Objectives

The lab objective is to explore some basic concepts related to Language, Programming Languages, Syntax and Semantics, Compiler, Interpreter, Difference of Compiler and Interpreter,

Lab Tasks

What is a language?

Language is the human capacity for acquiring and using complex systems of communication, and a language is any specific example of such a system. The scientific study of language is called linguistics.

What is Programming language?

A programming language is a formal constructed language designed to

communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

Types of Programming Languages

There are mainly two types of programming languages: Low Level Languages and High Level Languages.

Low Level Languages

In computer science, a low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture. Generally this refers to either machine code or assembly language. The word "low" refers to the small or nonexistent amount of abstraction between the language and machine language; because of this, low-level languages are sometimes described as being "close to the hardware".

High Level Languages

In computer science, a high-level programming language is a programming language with strong abstraction from the details of the computer. In comparison to low-level programming languages, it may use natural language elements, be easier to use, or may automate (or even hide entirely) significant areas of computing systems (e.g. memory management), making the process of developing a program simpler and more understandable relative to a lower-level language. The amount of abstraction provided defines how "high-level" a programming language is.

What is a Compiler?

A compiler is a computer program (or set of programs) that transforms source code written in a programming language (the source language) into another computer language (the target language, often having a binary form known as object code).The most common reason for wanting to transform source code is to create an executable program.

What is Interpreter?

In computer science, an interpreter is a computer program that directly executes, i.e. performs, instructions written in a programming or scripting language, without previously compiling them into a machine language program. An interpreter generally uses one of the following strategies for program execution:

1. parse the source code and perform its behavior directly
2. translate source code into some efficient intermediate representation and immediately execute this
3. explicitly execute stored precompiled code made by a compiler which is part of the interpreter system

What is the difference between compiler and interpreter?

A Compiler and Interpreter both carry out the same purpose – convert a high level language (like C, Java) instructions into the binary form which is understandable by computer hardware. They are the software used to execute the high level programs and codes to perform various tasks. Specific compilers/interpreters are designed for different high level languages. However both compiler and interpreter have the same objective but they differ in the way they accomplish their task i.e. convert high level language into machine language.

The interpreter takes one statement then translates it and executes it and then takes another statement. While the compiler translates the entire program in one go and then executes it.

- Compiler generates the error report after the translation of the entire page while an interpreter will stop the translation after it gets the first error.

- Compiler takes a larger amount of time in analyzing and processing the high level language code comparatively interpreter takes lesser time in the same process.

- Besides the processing and analyzing time the overall execution time of a code is faster for compiler relative to the interpreter.

What are different phases of compiler?

A compiler is a translator whose source language is a high-level language and whose object language are close to the machine language of an actual computer. The typical compiler consists of several phases each of which passes its output to the next phase;

- The *lexical phase* (scanner) groups characters into lexical units or tokens. The input to the lexical phase is a character stream. The output is a stream of tokens. Regular expressions are used to define the tokens recognized by a scanner (or lexical analyzer). The scanner is implemented as a finite state machine.

- The *parser* groups tokens into syntactical units. The output of the parser is a parse tree representation of the program. Context-free grammars are used to define the program structure recognized by a parser. The parser is implemented as push-down automata.

- The *semantic analysis* phase analyzes the parse tree for context-sensitive information often called the static semantics. The output of the semantic analysis phase is an annotated parse tree. Attribute grammars are used to describe the static semantics of a program. This phase is often combined with the parser. During the parse, information concerning variables and other objects is stored in a symbol table. The information is utilized to perform the context-sensitive checking.

- The *optimizer* applies semantics preserving transformations to the annotated parse tree to simplify the structure of the tree and to facilitate the generation of more efficient code.

- The *code generator* transforms the simplified annotated parse tree into object code using rules which denote the semantics of the source language. The code generator may be integrated with the parser.

- The *peep-hole* optimizer examines the object code, a few instructions at a time, and attempts to do machine dependent code improvements.

Lab 3 & 4 – Review of C/C++

Objectives

In these labs, students will recall the basic concepts of C/C++ programming language with compiler construction perspective.

Lab Task 1

Write a C/C++ program to count no of space, line, character and tabs.

Solution

```
#include<stdio.h>
#include<stdlib.h>

#include<string.h> int
main()
{
        FILE *fp1;
char ch;
```

```
            int space=0;
            int lines=0;
            int tabs=0;

            int chars=0;

    fp1=fopen("doc.c","r");
    while(!feof(fp1))
            {

            ch=fgetc(fp1);

            if(isgraph(ch)||ch==' '||ch=='\n'||ch=='\t')
            {

            chars++;
            }

            if(ch==' ')
            {

            space++;
            }

            if(ch=='\n')
            {

            lines++;
            }

            if(ch=='\t')
            {

            tabs++;
                        }

            }
            printf("spaces\t-->%d\n",space); printf("lines\t
            -->%d\n",lines+1); printf("tabs\t
            -->%d\n",tabs);
            printf("chars\t -->%d\n",chars);

            fclose(fp1);

            return 0;
    }
```

Input:

```
#include<stdio.h> void

main()
```

```
{
        printf("Hello World");
}
```

Output:

```
spaces -->2 lines
       -->5 tabs
       -->1
chars  -->57
```

Lab Task 2

Write a program which will take as input a C program consisting of single and multi-line comments and multiple blank spaces and produces as output the C program without comments and single blank space.

Solution

```
#include <stdio.h>

#include <fcntl.h>

#include<conio.h>

int main(void)

{
 clrscr();

   FILE *in, *out,*out2;

   in=fopen("try2.txt","w");

   char c;

   out2=fopen("final.txt","w");

   out=fopen("temp.c","r");
   char temp='a';
```

```
int flag=0;
if(!out)

        printf("\nfile cannot be opened");

else

        {

        flag=1;

        temp=c;

        while((c=getc(out))!=EOF)

                {

                                                printf("%c",c);

                if(temp=='/' && c=='*')
 do

                                                        {
 temp=c;
 c=getc(out);
 printf("%c",c);
                            if(temp=='*' && c=='/')

                                                        {
 c=getc(out);
 break;

                                                        }

                            }while(c!=EOF);
                if(temp=='/' && c=='/')
 do

                                                        {
 temp=c;


                                                c=getc(out);
 printf("%c",c);
 if(c=='\n')

                                                                        {
                                                 c=getc(out);
 break;

                                                                        }

                                      }while(c!=EOF);
                      if(c==' ' && temp==' ')
 continue;

                      temp=c;
 if(c!='/')

                                                                putc(c,out2);

                      }

            }

            fclose(out2);

            fopen("final.txt","r");

            while((c=getc(out2))!=EOF)

                                                               printf("%c",c);

            fclose(out2);

   fclose(out);

getch();

}
```

Input:

```
// This is a single line comment


            /* This is a
            Multiline Comment */
#include <stdio.h> void
main ()
{
 printf("Hello World");
}
```

Output:

```
#include<stdio.h> void
main()
{
printf("Hello World");

}
```

Lab 5& 6 – Review of File Handling Concept in C/C++

Objectives

In these labs, students will recall the basic concepts of file handling in C/C++ like opening a file, writing into file, reading from file, copy data into file, deleting data from file and searching specific information from file.

Lab Task 1

Write a C program for Students Management System. System should perform following features;

- Add new Student Record in File
- View Existing Records
- Update a record in File
- Delete a Record from File
- Exit

Enter Your Choice?

Solution

```
#include<stdio.h>

#include<stdlib.h>

int search_record(FILE *fPtr, int

roll_no); int menu();

int main(){
 int choice, roll_no;

            FILE *studentfPtr;

            studentfPtr=fopen("student.txt","a");

            if(studentfPtr==NULL){

                        printf("Sorry program could not continue");

                        exit(EXIT_FAILURE);

            }
 else{

                        choice=menu();

                        while(choice!= 0){

                                    switch(choice){
 case 1:
 add_record(studentfPtr);
 case 2:
 view_record(studentfPtr);
 case 3:

                                                            printf("\nPlease enter student's roll no:");

                                                            scanf("%d",&roll_no);


                                                             update_record(studentfPtr,roll_no);
 case 4:
```

```
                                                printf("\nPlease enter student's roll no:");
                                                scanf("%d",&roll_no);

                                                del_record(studentfPtr,roll_no);
 default:
 printf("\nWrong choice");

                                }
 choice=menu();

                        }

        }

        return 0;

}

int menu(){

        int choice;

        printf("\n1. Add student record

        \n2. View student record


        \n3. Update student record

        \n4. Delete student record

        \n0. Exit\nPlease enter your choice:");

        scanf("%d",&choice);

        return choice;

}

void add_record(FILE *fPtr){

        char name[20];

        int roll_no;
```

```
            printf("\nEnter Student's Name: "); scanf("%s",

            &name);

            printf("\nEnter student's roll_no"); scanf("%d",&roll_no);

            fprintf(fPtr,"\n Name: %s",name);

            fprintf(fPtr,"\n roll number: %d\n", roll_no);

}

void view_record(FILE *fPtr){ char
            c;

            rewind(fPtr);

            c=fgetc(fPtr);

            printf("\n");

            while(c!=EOF){
 printf("%c",c);

            }

            printf("\nrecords finished"); rewind(fPtr);


}

void del_record(FILE *fPtr, int roll_no){

}

void update_record(FILE *fPtr, int roll_no){

}

int search_record(FILE *fPtr, int roll_no){
            int compare;
            rewind(fPtr);

            while(fscanf(fPtr,"%d", &compare)!=EOF){ if(roll_no=compare){
 printf("\nRecord found");

                        }

            }
 return 0;

}
```

Lab Task 2

Write a program, which will read a program from file written in C, recognize all of the keywords in that program and print them in upper case letters.

Solution

```
#include <stdio.h>

#include <fcntl.h>

#include<conio.h>

#include<string.h>

#include<stdio.h>

#include<iostream.h>

#include<ctype.h>
char line[80];

char key[][6]={"VOID","MAIN","INT","CHAR","IF","ELSE","WHILE","FOR"};

void findkey()

            {
```

```
int i,j,k,l=strlen(line),p,pos; char
            temp[10];

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

                        {
p=0,pos=i;
while(!isalpha(line[i]))

                                    {
i++;
pos++;

                                    }
while(isalpha(line[i]))
temp[p++]=line[i++];
temp[p]='\0';
cout<<"\t*"<<temp<<"*";
for(j=0;j<8;j++)
if(strcmpi(temp,key[j])==0)

                                                {

                                                //cout<<"\nkeyword found is "<<key[j];

                                                int t=0;

                                                for(k=pos;k<pos+strlen(key[j]);k++)
line[k]=key[j][t++];
break;

                                                }

                        }

                        //cout<<"\n"<<line;
            }

int main(void)

{
clrscr();
```

```
    FILE *in, *out,*out2;

    in=fopen("try2.txt","w");

    char c;

    out2=fopen("final.txt","w");

    out=fopen("temp.c","r");

    char temp='a';

    int flag=0;
    if(!out)

            printf("\nfile cannot be opened");

    else

            {

            flag=1;

            //temp=c;

            while(1)

                        {

                                                                    int p=0;

                         while((c=getc(out))!='\n')

                                                                        {
  line[p++]=c;
  if(c==EOF)
                                         break;


                                                                        }
  line[p++]='\n';

                                   line[p++]='\0';
                        findkey();
```

```
 for(int i=0;i<strlen(line);i++)
                        putc(line[i],out2);
 putc('\n',out2);



                                                              if(c==EOF)
 break;

                        }

            }

            fclose(out2);

            cout<<"\n\n\n\n the modified prog is \n";



            fopen("final.txt","r");

            while((c=getc(out2))!=EOF)

                                                            printf("%c",c);

            fclose(out2);

   fclose(out);

getch();

}
```

Input:

```
#include<stdio.h> void
main()
{
            int a;
            printf("Hello World");
}
```

Output:

```
VOID

MAIN

INT
```

Lab 7– Identifying Identifiers

Objectives

In this lab, students will learn how to identify the identifiers of C language. After this lab student will be able to decide identifier of their own language and learn how to implement identifier rules to identify them in a program.

Lab Tasks

Write a program to identify whether the given string is C/C++ identifier or not.

Solution

```
#include<stdio.h>
#include<conio.h>

int   isiden(char*);
int  second(char*);
int   third();   void
main()
{
            char *str; int
            i = -1;
            clrscr();
            printf("\n\n\t\tEnter the desired String:
            "); do
            {
            ++i;
                        str[i] = getch(); if(str[i]!=10
                        &&       str[i]!=13)
                        printf("%c",str[i]);
                        if(str[i] == '\b')
                        {
                                    --i; printf("
                                    \b");
            }
            }while(str[i] != 10 && str[i] !=
            13); if(isident(str)) printf("\n\n\t\tThe given strig is an
            identifier"); else printf("\n\n\t\tThe given string is not an
            identifier");

getch();
}
```

```
//To Check whether the given string is identifier or not
//This function acts like first stage of dfa

int isident(char *str)
{
if((str[0]>='a' && str[0]<='z') || (str[0]>='A' && str[0]<='Z'))
        {
        return(second(str+1));
        } else
return 0;
}
//This function acts as second stage of dfa
int second(char *str)
{
        if((str[0]>='0' && str[0]<='9') || (str[0]>='a' && str[0]<='z') || (str[0]>='A' &&
        str[0]<='Z')) return(second(str+1)); //Implementing the loop from second stage t

        second stage
        } else
        { if(str[0] == 10 || str[0] == 13)
        {
        return(third(str));
        } else {
        return 0;
        }
        }
}

//This function acts as third stage of dfa int
third()
{ return 1; //Being final stage reaching it mean the string is identified
}
```

Input:

Enter the desired string: 123

Output:

The given string is not an identifier

Input:

Enter the desired string: ab123

Output:

The given string is an identifier

Lab 8 – Identify Keywords

Objectives

In this lab, students will learn how to identify keywords/reserved words of C language.

Lab Tasks

Write a program to identify that the enter word is a keyword of C/C++ language or not.

Solution

```
#include<stdio.h>

#include<conio.h>


#include<string.h> void
main()
{


            int i,flag=0,m;

            char s[5][10]={"if","else","goto","continue","return"},st[10];
clrscr();
```

```
            printf("\n enter the string :");
            gets(st);
            for(i=0;i<5;i++)
            {
            m=strcmp(st,s[i]);
            if(m==0)
            flag=1;
            }
            if(flag==0)
            printf("\n it is not a keyword");
else
            printf(―\n It is a keyword‖);      getch():
            }
```

Input:

Enter the string: return

Output:

It is a keyword

Input:

Enter the string: hello

Output:

It is not a keyword

Lab 9: Count Occurrence of If-Condition in a C program

Objective

In this lab, Student will practice of C/C++ basic concepts and learn how to scan and count a particular condition from a file.

Lab Task

Write a C/C++ program to find if condition from a given file and count the occurrence of IfCondition in file.

Solution

```
#include<stdio.h>

#include<stdlib.h>

#include <conio.h>

#define MAX 50

int countIf(FILE *fileptr); int
main(){

        char fileName[MAX];

        FILE *fileptr;

        int ifCount;
```

```
            printf("\nEnter the Required file name:");

            fgets(fileName,MAX,stdin);


            fileptr=fopen(fileName,"r");


            if (fileptr==NULL){
    printf("\nUnable to open the File");

                      return 0;


            }



            else {

                       printf("\n%s successfully opened",fileName);

                       ifCount=countIf(fileptr);
    printf(—\n %d No. of If-Condition in File are:‖, ifCount);


            }


            return 0;


            }


    int countIf(FILE *fileptr){

            int


            ifCount=0; char c;


            while ((c=fgetc(fileptr))!=EOF){ if

             (c==' '){

             if((c=fgetc(fileptr))=='i'){


             if((c=fgetc(fileptr))=='f'){
    if(((c=fgetc(fileptr))==' ') ||

            ((c=fgetc(fileptr))=='('){ ifCount++;
          }
}
}
}
}
          return ifCount++;
}
```

Input

Enter the Required file Name: Test.C

Output

No of If-Condition in File are: 3

Lab 10-11 Research Session: Scanner

Objectives

In these labs, students will read TYNI/MINI C Language, its feature, and first phase of compiler for TYNI Language. After these labs, Students will be able to choose three different languages and extracting some features of these languages to design their own language

Lab Tasks

Read Appendix of text book ―Principles , Techniques and Tools‖ by Aho, Lam, Sethi, Ullman.

A.1 Source Language

A.2 Main

A.3 Lexical Analyzer

Lab 12 – Basic Scanner Concepts

Objectives

The objective is to build a scanner for MINI-C language. After this lab, students will be able to build a scanner for their own designed language.

Lab Tasks

Write a program for a standalone scanner for C Language which read a C program and identify tokens of the C Language and generate the Symbol Table.

Solution

```
#include<stdio.h>

#include<ctype.h>
#include<string.h>


int main()
{
      FILE *input,

           *output; int l=1;

      int t=0;
      int j=0;

           int i,flag;

           char
ch,str[20];
      input = fopen("input.txt","r");

      output = fopen("output.txt","w"); char keyword[30][30] =
      {"int","main","if","else","do","while"}; fprintf(output,"Line no.
      \t Token no. \t Token \t Lexeme\n\n");

           while(!feof(input))

           {

           i=0;

           flag=0;

           ch=fgetc(input);
```

```
if( ch=='+' || ch== '-' || ch=='*' || ch=='/' )

{
        fprintf(output,"%7d\t\t %7d\t\t Operator\t %7c\n",l,t,ch);

t++;

}

else if( ch==';' || ch=='{' || ch=='}' || ch=='(' || ch==')' || ch=='?' || ch=='@' || ch=='!' || ch=='%')

{
        fprintf(output,"%7d\t\t %7d\t\t Special symbol\t %7c\n",l,t,ch);

t++;
  } else
  if(isdigit(ch))
   {

        fprintf(output,"%7d\t\t %7d\t\t Digit\t\t

%7c\n",l,t,ch); t++;
}
else if(isalpha(ch))
{
str[i]=ch;          i++;
        ch=fgetc(input);

while(isalnum(ch) && ch!=' ')
{

str[i]=ch;

i++;

ch=fgetc(input);
}
str[i]='\0';

for(j=0;j<=30;j++)
{

if(strcmp(str,keyword[j])==0)
{

flag=1;

break;
}
}

if(flag==1)
```

```
            {
                        fprintf(output,"%7d\t\t %7d\t\t Keyword\t
            %7s\n",l,t,str); t++;
            }
            else
            {
                        fprintf(output,"%7d\t\t %7d\t\t Identifier\t
            %7s\n",l,t,str); t++;
            }
            }
            else if(ch=='\n')
            {
l++;
            }
            }
            fclose(input);
        fclose(output);
            return 0;
}
```

Input:

//input.txt

```
#include<stdio.h>
void main()
{
            printf("Hello World");
```

```
                                    Lexeme
                                    include
                                     stdio h
                                     void
                                     main
```

```
}
```

Output:

//output.txt

```
 Line no.        Token no.        Token
                                  Identifier
                                  Identifier
                                  Identifier
                                  Identifier
                                  Keyword
    1               0
    1               1
    1               2
    2               3
    2               4             Special symbol  )
    2               5             Special symbol
    3               6             {
    4               7             Identifier      printf
    4               8             Identifier      Hello
    4               9             Identifier World
    4               10            Special symbol  )
    4               11            Special symbol  ;
    5               12            Special symbol  }
```

Lab 13-14 Research Session: Parser

Objectives

In these labs, students will read second phase of compiler construction of TYNI/MINI C Language and its feature. After these labs, Students will be able to understand different parsing techniques and implementation of parsing algorithms.

Lab Tasks

Read Appendix of text book ―Principles , Techniques and Tools‖ by Aho, Lam, Sethi, Ullman.

A.4 Symbol Tables and Types

A.8 Parser

A.9 Creating the Front End

Lab 15 – Implementing Simple CFG

Objectives

In this lab, students are assigned a task to understand how to implement a CFG in C/C++ language. After this lab, student will be able to implement CFG of their own designed language in second phase of compiler construction which is Parser.

Lab Tasks

Write a program for the given grammar, input a string to check whether it is acceptable by the grammar or not.

Grammar: ab?c*

Solution

```
#include<stdio.h>
#include<string.h>

intcheckGrammar(char a[]);


int main(){
        char string[100],exit;
        int result;

        printf("\nPlease enter a string for validation check:
        "); gets(string);
        result=checkGrammar(string);

        if(result==0){
        printf("\n%s is a valid string",string);
        }
        else{
                printf("\n%s is nto a valid string", string); }

        printf("\npress any key to exit");
        scanf("%c",&exit);
        return 0;
}

intcheckGrammar(chara[]){
 int length, i;
 length=(int)strlen(a);
```

```
            /*printf("%d",length);*/

            31

            if(length < 1)
            {
                        printf("\nYou did not not enter a string"); return
                        1;
            } else{
 for(i=0;i<=length-1;i++){


            if(i==0){
                                                if(a[i]=='a'){}
                                                else{return 1;}
                                    }


            if(i==1){
                                                if(a[i]=='b'||a[i]=='c' || a[i]=='\0'){} else{return
                                                1;}
                                    }


            if(i==2 || i>2 ){
                                                if(a[i]=='c' || a[i]=='\0'){}
                                                else{return 1;}

                                    }
                        }

            }

            return 0;
}
```

Input:
Please enter a string for validation check: abcccc

Output:
The string is valid

Input:
Please enter a string for validation check: bc

Output:

```
The string is not valid
```

Lab 16 – Top-Down Parser (Recursive Decent Parsing Algorithm)

Objectives

The main objective of this lab is to understand the basics of a parser and implementation of topdown parsing algorithm.

Lab Tasks

Write a C/C++ program for recursive decent parsing algorithm.

Solution

```
#include<stdio.h>
#include<string.h>

void E(),E1(),T(),T1(),F();
 int ip=0;
static char s[10];


int  main()
{
        char k; int
        l;
             ip=0;
        printf("Enter the
        string:\n"); scanf("%s",s); printf("The
        string is:
            %s",s); E();
if(s[ip]=='$')
 printf("\nString is accepted.\nThe length of the string is %d\n",strlen(s)-1);  else
              printf("\nString not accepted.\n");
            return 0;
}

void E()
{
            T();
            E1();

return;
}
void E1()
{
            if(s[ip]=='+')
            {
            ip++;
            T();
            E1();
            }
            return;
}

void T()
{
            F();
 T1();
            return;
}

void T1()
{
            if(s[ip]=='*')
            {
            ip++;
            F();
            T1();
            }
            return;
}

void F()
{
            if(s[ip]=='(')
            {
            ip++;
            E();
            if(s[ip]==')')
            {
            ip++;
            }
            }
        else if(s[ip]=='i')
            ip++;
else
               printf("\nId expected ");
```

```
            return;
}
```

Input:

```
Enter the string:
(i+i*i)$
```

Output:

```
The string is: (i+i*i)$
String is accepted.
The length of the string is 7
```

Lab 17 – Revision of Stack Implementation

Objectives

The objective of this lab is to practice stack implementation to recall the concepts of it.

Lab Tasks

Write a program in C/C++ which takes a mathematics infix expression as an input and convert it into postfix expression using stack.

Solution

```
#define SIZE 50

#include <ctype.h> #include

<stdio.h>


char s[SIZE];

int top = -1;



void push(char elem) {



s[++top] = elem;

}



char pop() {

return (s[top--]);

}
```

```
int pr(char elem) {


switch (elem) { case
'#':

 return 0;

case '(':

return 1; case

'+': case '-':

return 2; case

'*': case '/':

return 3;

}

}


void main() {

char infx[50], pofx[50], ch,

elem; int i = 0, k = 0;

printf("\n\nRead the Infix Expression ? ");

scanf("%s", infx);

push('#');

while ((ch = infx[i++]) != '\0')
  { if (ch == '(')

   push(ch);
```

```
else if (isalnum(ch))
pofx[k++] = ch; else
if (ch == ')') { while

(s[top] != '('){

  pofx[k++] = pop();
```

```
    }
    elem = pop();
   } else {
    while (pr(s[top]) >= pr(ch)){
                pofx[k++] = pop();}

    push(ch);
   }
}
while (s[top] != '#') {
            pofx[k++] = pop();}
pofx[k] = '\0';
printf("\n\nGiven Infix Expression: %s Postfix Expression: %s\n", infx, pofx);
}
```

Input

```
Read the infix Expression? 2+3/4
```

Output

```
Given infix Expression: 2+3/4   Postfix Expression: 234/+
```

Lab 18 – Implementation of FIRST Sets of Grammar

Objectives

The objective of this lab is to practice how to find FIRST Sets of a given grammar. After that students will be able to implement bottom-up parsing techniques

Lab Tasks

Write a program to compute the FIRST Sets of a given grammar.

Solution

```
#include<stdio.h>
#include<ctype.h>

int main()
{
            int i,n,j,k;           char
str[10][10],f; printf("Enter the
number of
 productions\n"); scanf("%d",&n); printf("Enter
       grammar\n");
            for(i=0;i<n;i++)
scanf("%s",&str[i]);       for(i=0;i<n;i++)
```

```
            {
            f= str[i][0];
            int temp=i;
            if(isupper(str[i][3]))
            { repeat:


            for(k=0;k<n;k++)
            {
            if(str[k][0]==str[i][3])
            {
            if(isupper(str[k][3]))
            {
            i=k;
            goto repeat;
            }
            else
            {
            printf("First(%c)=%c\n",f,str[k][3]);
            }
            }
            }
            }
            else
            {
    printf("First(%c)=%c\n",f,str[i][3]);
            }
            i=temp;
            }
    }
```

Input:

```
Enter the number of productions
3
Enter grammar
S ->AB
A ->a
B->b
```

Output:

```
First(S)=a

First(A)=a
First(B)=b
```

Lab 19 – Implementation of FOLLOW Sets of Grammar

Objectives

The objective of this lab is to practice how to find FOLLOW Sets of a given grammar. After that students will be able to implement bottom-up parsing techniques.

Lab Tasks

Write a program to compute the FOLLOW Sets of the given grammar.

Solution

```
#include<stdio.h>

main()
{
        int np,i,j,k;
        char prods[10][10],follow[10][10],Imad[10][10]; printf("enter
        no. of productions\n");
        scanf("%d",&np);
        printf("enter grammar\n");
        for(i=0;i<np;i++)
        {
        scanf("%s",&prods[i]);
        }

        for(i=0; i<np; i++)
        {
        if(i==0)
        {
        printf("Follow(%c) = $\n",prods[0][0]);//Rule1
        }
        for(j=3;prods[i][j]!='\0';j++)
        {
        int temp2=j;
                        //Rule-2: production A->xBb then everything in first(b) is in
        follow(B) if(prods[i][j] >= 'A' && prods[i][j] <= 'Z')
        {
        if((strlen(prods[i])-1)==j)
        {
        printf("Follow(%c)=Follow(%c)\n",prods[i][j],prods[i][0]);
        }
        int temp=i;
                                char f=prods[i][j];
```

```
            if(!isupper(prods[i][j+1])&&(prods[i][j+1]!='\0'))
            printf("Follow(%c)=%c\n",f,prods[i][j+1]);         if(isupper(prods[i][j+1]))
            {
            repeat:
            for(k=0;k<np;k++)
            {
            if(prods[k][0]==prods[i][j+1])
            {
            if(!isupper(prods[k][3]))
            {
            printf("Follow(%c)=%c\n",f,prods[k][3]);
            }          else
            {          i=k;      j=2;
            goto repeat;
            }
            }
            }          }
            i=temp;
            }
            j=temp2;
            }
            }
    }
```

Input:

```
enter   no.   of
productions  3
               enter
grammar
S ->AB
A ->a
B->b
```

Output:

Follow(S) = $

Follow(A)=b

Follow(B)=Follow(S)

Lab 20-21 – Implementing Parsing Algorithm-SLR(1)

Objectives

The objective of this practice is to understand the SLR(1) parsing algorithm and Implement the algorithm for a small grammar. After this practice, student will be able to choose a parsing algorithm for their project.

Lab Tasks

Write a C/C++ program for SLR parsing algorithm.

Solution

```
#include<stdio.h>
#include<string.h>

char a[8][5],b[7][5]; int
c[12][5];
int   w=0,e=0,x=0,y=0;
int   st2[12][2],st3[12];
char sta[12],ch;

void v1(char,int);
void v2(char,int,int,int);

main()
{
        int i,j,k,l=0,m=0,p=1,f=0,g,v=0,jj[12]; printf("\n\n\t*******Enter
        the Grammar Rules
        (max=3)*******\n\t"); for(i=0;i<3;i++)
```

```
{
gets(a[i]);          printf("\t");
}
for(i=0;i<3;i++)
{
for(j=0;j<strlen(a[i]);j++)
{
for(k=0;k<strlen(a[i]);k++)
{
if(p==k)
{
            b[l][m]='.';
            m+=1;
            b[l][m]=a[i][k];
            m+=1;
            }          else
            {
            b[l][m]=a[i][k];
            m++;
            }
            }
            p++;      l++;
            m=0;      }
            p=1;
}
i=0; p=0;
while(l!=i)
{
            for(j=0;j<strlen(b[i]);j++)
            {
            if(b[i][j]=='.')
            {
            p++;
```

```
		}
		}
		if(p==0)
		{
		b[i][strlen(b[i])]='.';
		}
		i++;
		p=0;
	}
	i=0;
	printf("\n\t*******Your States will be*******\n\t");
	while(l!=i)
	{
		printf("%d--> ",i);
		puts(b[i]);
	i++;
		printf("\t");
	}
	printf("\n");
```

```
v1('A',l);


p=c[0][0];
         m=0;


while(m!=6)
{
        for(i=0;i<st3[m];i++)
        {
        for(j=0;j<strlen(b[p]);j++)
        {
                        if(b[p][j]=='.' && ((b[p][j+1]>=65 &&
 b[p][j+1]<=90) ||(b[p][j+1]>=97&&b[p][j+1]<=122)))
        {
        st2[x][0]=m;
        sta[x]=b[p][j+1];
        v2(b[p][j+1],j,l,f);
        x++;
        //str();
        }
        else
        {
        if(b[p][j]=='.')
        {
        st2[x][0]=m;     sta[x]='S';
        st2[x][1]=m;
        x++;
        }
        }
        }
        p=c[m][i+1];
        }
        m++;
```

```
                p=c[m][0];
        }
        g=0; p=0; m=0;x=0; while(p!=11)

        {
                for(i=0;i<st3[p];i++)
                {
                for(k=0;k<p;k++)
                {
                for(j=0;j<3;j++)
                {
                                        if(c[k][j]==c[p][j])


                                                {
        m++;
        }       }
        if(m==3)
        {
                                        m=0;
goto ac;
        }       m=0;
        }
        if(m!=3)
        {
        if(v==0)
        {
        printf("\tI%d=",g);
        v++;    jj[g]=p;
        }
        printf("%d",c[p][i]);
```

```
            }
            }
            printf("\n");
            g++;  ac:
            p++;      v=0;
            }
            printf("\t*******Your DFA will be *******");
            for(i=0;i<9;i++)
            {
                                                printf("\n\t%d",st2[i][0]);
            printf("-->%c",sta[i]);
            }
            getchar();
}

void v1(char ai,int kk)
{  int i,j;
            for(i=0;i<kk;i++)
            {
            if(b[i][2]==ai&&b[i][1]=='.')
                        {
                                    c[w][e]=i;
            e++;
            if(b[i][2]>=65 && b[i][2]<=90)
            {
            for(j=0;j<kk;j++)
            {
            if(b[j][0]==ai && b[j][1]=='.')
            {
            c[w][e]=j;
            e++;
            }
            }
            }
            }
            }
            st3[w]=e;
            w++;
            e=0;
}

void v2(char ai,int ii,int kk,int tt)
{
            int i,j,k;
            for(i=0;i<kk;i++)
            {
            if(b[i][ii]=='.'&& b[i][ii+1]==ai)
            {
            for(j=0;j<kk;j++)
            {
            if(b[j][ii+1]=='.' && b[j][ii]==ai)
            {
            c[w][e]=j;          e++;
            st2[tt][1]=j;
            if(b[j][ii+2]>=65 && b[j][ii+1]<=90)
            {
            for(k=0;k<kk;k++)
            {
            if(b[k][0]==b[j][ii+2] && b[k][1]=='.')
            {
            c[w][e]=k;
            e++;
            }
```

```
                                                }
                                                }
            }
            }
 if((b[i][ii+1]>=65 && b[i][ii+1]<=90) && tt==1)
            {
            for(k=0;k<kk;k++)
            {
            if(b[k][0]==ai && b[k][1]=='.')
            {
            c[w][e]=k;
            e++;
            }
            }
            }
            }
            }
            st3[w]=e;
            w++;
            e=0;
}
```

Input:

```
            *******Enter the Grammar Rules (max=3)*******
            EAB
            Aa
       Bb
```

Output:

```
            *******Your States will be*******
            0--> E.AB
```

```
        1--> EA.B
    2--> EAB.
        3--> A.a
        4--> Aa.
        5--> B.b
      6--> Bb.

        I0=03
        I1=15
        I2=4
        I3=2
      I4=6



      *******Your DFA will be *******
      0 -->A
      0-->a
      1-->B
      1-->b
      2-->S
      3-->S
      4-->S
      0 -->
      0-->
```

Lab 22-23 – Implementing Parsing Algorithm-LR(0)

Objectives

The objective of this practice is to understand the LR(0) parsing algorithm and Implement the algorithm for a small grammar.

Lab Tasks

Write a Program to find CANONICAL LR (0) Collections. (Closure)

Solution

//closure.c

```
#include<stdio.h>
#include<string.h>

char a[8][5],b[7][5]; int
c[12][5];
int  w=0,e=0,x=0,y=0;  int
st2[12][2],st3[12];     char
sta[12],ch;              void
v1(char,int);            void
v2(char,int,int,int);

int main()
{
        int i,j,k,l=0,m=0,p=1,f=0,g,v=0,jj[12];
        //clrscr(); printf("\n\n\t*******Enter the
        Grammar Rules
            (max=3)*******\n\t"); for(i=0;i<3;i++)
            {
            gets(a[i]);
printf("\t");
            }
            for(i=0;i<3;i++)
            {
            for(j=0;j<strlen(a[i]);j++)
            {
```

```
    for(k=0;k<strlen(a[i]);k++)
    {
    if(p==k)
    {
                                b[l][m]='.';
    m+=1;
    b[l][m]=a[i][k];
    m+=1;
    }       else
    {
    b[l][m]=a[i][k];
    m++;
    }
    }
    p++;    l++;
    m=0;
    }
    p=1;
}
i=0; p=0;
while(l!=i)
{
    for(j=0;j<strlen(b[i]);j++)
    {
    if(b[i][j]=='.')
    {       p++;
}
    }
```

```
        if(p==0)
        {
        b[i][strlen(b[i])]='.';
        }
        i++;
        p=0;
    }
    i=0;

    printf("\n\t*******Your States will be*******\n\t");
    while(l!=i)
    {
          printf("%d--> ",i);
        puts(b[i]);
    i++;
        printf("\t");
    }
    printf("      \n");
    v1('A',l); p=c[0][0];
    m=0; while(m!=6)
    { for(i=0;i<st3[m];i++)
        {
        for(j=0;j<strlen(b[p]);j++)
        {
                                    if(b[p][j]=='.' && ((b[p][j+1]>=65 &&
b[p][j+1]<=90)|| (b[p][j+1]>=97&&b[p][j+1]<=122)))
        {
        st2[x][0]=m;
        sta[x]=b[p][j+1];           v2(b[p][j+1],j,l,f);
        x++;
        //str();
        }
        else
        {
        if(b[p][j]=='.')
        {
            st2[x][0]=m;      sta[x]='S';
st2[x][1]=m;
            x++;
            }
            }
            }
            p=c[m][i+1];
            }
            m++;
            p=c[m][0];
            }
            g=0;
            p=0;
            m=0;
            x=0;
        getchar();
            return 0;
}

void v1(char ai,int kk)
{  int i,j;
        for(i=
        0;i<kk
        ;i++)
        { if(b[i][2]==ai&&b[i][1]=='.')
                { c[w][e]=i;
            e++;
            if(b[i][2]>=65 && b[i][2]<=90)
            {
```

```
            for(j=0;j<kk;j++)
            {
            if(b[j][0]==ai && b[j][1]=='.')
            {
            c[w][e]=j;
            e++;
            }
            }
            }
            }
            }
            st3[w]=e;
            w++;
            e=0;
}

void v2(char ai,int ii,int kk,int tt)
{  int i,j,k;
            for(i=0;i<kk;i++)
            {
            if(b[i][ii]=='.'&& b[i][ii+1]==ai)
            {
            for(j=0;j<kk;j++)
            {
            if(b[j][ii+1]=='.' && b[j][ii]==ai)
            {
            c[w][e]=j;          e++;
            st2[tt][1]=j;
            if(b[j][ii+2]>=65 && b[j][ii+1]<=90)
            {
            for(k=0;k<kk;k++)
            {
 if(b[k][0]==b[j][ii+2] && b[k][1]=='.')
                                                {
                                        c[w][e]=k;
                                        e++;
```

```
                                                            }
                                                            }
                                    }
            }
            }
 if((b[i][ii+1]>=65 && b[i][ii+1]<=90) && tt==1)
            {
            for(k=0;k<kk;k++)
            {
            if(b[k][0]==ai && b[k][1]=='.')
            {
            c[w][e]=k;
            e++;
            }
            }
            }
            }
            }
            st3[w]=e;
            w++;
            e=0;
}
```

Input:

```
*******Enter the Grammar Rules (max=3) *******
    EAB
    Aa
    Bb
```

Output:

```
*******Your States will be*******

          0--> E.AB

          1--> EA.B
2--> EAB.

          3--> A.a

          4--> Aa.

          5--> B.b
      6--> Bb.
```

Lab 24-25 – Implementing Parsing Algorithm-LR(0)

Objectives

The objective of this practice is to understand the LR(0) parsing algorithm and Implement the algorithm for a small grammar.

Lab Tasks

Write a Program to find CANONICAL LR(0) Collections. (goto)

Solution

//goto.c

```
#include<stdio.h>
#include<string.h>

char a[8][5],b[7][5]; int
c[12][5];
```

```
int    w=0,e=0,x=0,y=0;
int    st2[12][2],st3[12];
char sta[12],ch;


void v1(char,int);
void v2(char,int,int,int);


int main()
{
        int i,j,k,l=0,m=0,p=1,f=0,g,v=0,jj[12];
        //clrscr(); printf("\n\n\t*******Enter the
        Grammar Rules
             (max=3)*******\n\t"); for(i=0;i<3;i++)
             {
             gets(a[i]);
printf("\t");
             }
             for(i=0;i<3;i++)
             {
             for(j=0;j<strlen(a[i]);j++)
             {
             for(k=0;k<strlen(a[i]);k++)
             {
                         if(p==k)
             {
             b[l][m]='.';
             m+=1;
             b[l][m]=a[i][k];
             m+=1;  }
```

```
    else
    {
    b[l][m]=a[i][k];
    m++;    }          }
    p++;
l++;
m=0;        }
    p=1;
}
i=0;


p=0;


while(l!=i)
{
    for(j=0;j<strlen(b[i]);j++)
    {
    if(b[i][j]=='.')
    {          p++;
}
    }
    if(p==0)
    {
    b[i][strlen(b[i])]='.';
    }
    i++;
    p=0;
}
i=0;

printf("\n\t*******Your States will be*******\n\t");
while(l!=i)
{
      printf("%d--> ",i);
    puts(b[i]);
i++;
      printf("\t");


}
printf("      \n");
v1('A',l);
p=c[0][0];
m=0;
while(m!=6)
{
    for(i=0;i<st3[m];i++)
    {
    for(j=0;j<strlen(b[p]);j++)
    {
                                    if(b[p][j]=='.' && ((b[p][j+1]>=65 &&
 b[p][j+1]<=90)|| (b[p][j+1]>=97&&b[p][j+1]<=122)))
    {
    st2[x][0]=m;      sta[x]=b[p][j+1];
v2(b[p][j+1],j,l,f);
    x++;
    //str();
    }
    else
    {
    if(b[p][j]=='.')
```

```
            {
            st2[x][0]=m;      sta[x]='S';
            st2[x][1]=m;
            x++;
                                 }

            }
            }
            p=c[m][i+1];
            }
            m++;
            p=c[m][0];
        }
        g=0;
         p=0;
         m=0;
         x=0;


        while(p!=11)
        {
              for(i=0;i<st3[p];i++)


              { for(k=0;k<p;k++)
            {
            for(j=0;j<3;j++)
            {
            if(c[k][j]==c[p][j])
            {
            m++;
            }         }
            if(m==3)
            {
                                m=0;
goto ac;
            }
            m=0;    }
            if(m!=3)
            {
            if(v==0)
            {
            printf("\tI%d=",g);
            v++;    jj[g]=p;
            }
            printf("%d",c[p][i]);
            }
            }
            printf("\n");
            g++;
ac:
            p++;
v=0;
            }
        getchar();
            return 0;
}

void v1(char ai,int kk)
{  int i,j;
            for(i=0;i<kk;i++)
            {
              if(b[i][2]==ai&&b[i][1]=='.')
              { c[w][e]=i;
            e++;
```

```
          if(b[i][2]>=65 && b[i][2]<=90)
          {
          for(j=0;j<kk;j++)
          {
          if(b[j][0]==ai && b[j][1]=='.')
          {
          c[w][e]=j;
          e++;
          }
          }
          }
          }
          }
          st3[w]=e;
          w++;
          e=0;
}

void v2(char ai,int ii,int kk,int tt)
{  int i,j,k;
          for(i=0;i<kk;i++)
          {
          if(b[i][ii]=='.'&& b[i][ii+1]==ai)
          {
          for(j=0;j<kk;j++)
          {
          if(b[j][ii+1]=='.' && b[j][ii]==ai)
          {
          c[w][e]=j;
          e++;
          st2[tt][1]=j;
          if(b[j][ii+2]>=65 && b[j][ii+1]<=90)
          {
          for(k=0;k<kk;k++)
          {
```

```
 if(b[k][0]==b[j][ii+2] && b[k][1]=='.')
            {
            c[w][e]=k;
            e++;
                                                         }


                                                         }
                                     }
            }
            }
 if((b[i][ii+1]>=65 && b[i][ii+1]<=90) && tt==1)
            {
            for(k=0;k<kk;k++)
            {
            if(b[k][0]==ai && b[k][1]=='.')
            {
            c[w][e]=k;
            e++;
            }
            }
            }
            }
            }
            st3[w]=e;
            w++;
            e=0;
}
```

Input:

```
      *******Enter the Grammar Rules (max=3)*******

      EAB

      Aa
   Bb
```

Output:

```
      *******Your States will be*******

      0--> E.AB

      1--> EA.B
2--> EAB.
      3--> A.a

      4--> Aa.

      5--> B.b
   6--> Bb.


      I0=03

      I1=15

      I2=4

      I3=2
   I4=6
```

Lab 26-27 – Implementing Parsing Algorithm-LL(1)

Objectives

The objective of this practice is to understand the LL(1) parsing algorithm and Implement the algorithm for a small grammar.

Lab Tasks

Write a Program to construct Predictive LL(1) TABLE

Solution

```
#include<stdio.h>

#include<conio.h>

#include<string.h          >
#include<process.h>

char prod[10][20],start[2];
char nonterm[10],term[10];
char input[10],stack[50];
int table[10][10];            int
te,nte; int n;

void main()
{
            clrscr();

            init();
parse();

            getch();
}
 init()


{
            int i,j;

            printf("\nNOTE:\n");

            printf("The terminals should be entered in single lower case letters,special symbol and\n");
```

```
        printf("non-terminals should be entered in single upper case letters.\n"); printf("extends
        to symbol is '->' and epsilon symbol is '@'
        \n"); printf("\nEnter the no. of terminals:");
        scanf("%d",&te);
for(i=0;i<te;i++)
        {
        fflush(stdin);
                printf("Enter the terminal
        %d:",i+1); scanf("%c",&term[i]);
        }
        term[i]='$';
        printf("\nEnter the no. of non terminals:");
        scanf("%d",&nte);
for(i=0;i<nte;i++)
        {
        fflush(stdin);
                printf("Enter the non-terminal
        %d:",i+1); scanf("%c",&nonterm[i]);
        }
        printf("\nEnter the no. of
        productions:"); scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("Enter the production
        %d:",i+1); scanf("%s",prod[i]);
        }
        fflush(stdin);
        printf("\nEnter the start symbol:");
        scanf("%c",&start[0]); printf("\nEnter
        the input string:"); scanf("%s",input);
        input[strlen(input)]='$';
        printf("\n\nThe productions are:");
        printf("\nProductionNo. Production");
        for(i=0;i<n;i++)
```

```
                printf("\n %d %s",i+1,prod[i]);
        printf("\n\nEnter the parsing table:"); printf("\n Enter the production number in the
        required entry as mentioned above.");
        printf("\n Enter the undefined entry or error of table as '0'\n\n");
        for(i=0;i<nte;i++)
        {
        for(j=0;j<=te;j++)
        {
        fflush(stdin);
                printf("Entry of table[%c,%c]:",nonterm[i],term[j]);
        scanf("%d",&table[i][j]);
        }
        }
    }

parse()
{
        int i,j,prodno;
        int top=-1,current=0;
        stack[++top]='$';
        stack[++top]=start[0];
        do
        {
        if((stack[top]==input[current])&&(input[current]=='$'))
        {
                        printf("\nThe given input string is
        parsed"); getch();
        exit(0);
        }
        else if(stack[top]==input[current])
        {
                top--;
        current++;
        }
        else if(stack[top]>='A'&&stack[top]<='Z')
```

```
            {
                                                  for(i=0;i<nte;i++)
            if(nonterm[i]==stack[top]) break;

                                  for(j=0;j<=te;j++)
    if(term[j]==input[current]) break;             prodno=table[i][j];
            if(prodno==0)
            {
                                         printf("\nThe given input string is not
            parsed"); getch();
            exit(0);
            }
            else
            {
 for(i=strlen(prod[prodno-1])-1;i>=3;i--)
            {
                                                 if(prod[prodno-1][i]!='@')
            stack[top++]=prod[prodno-1][i];
            }
            top--;
            }
            }
            else
            {
                                  printf("\nThe given input string is not parsed");
            getch();
            exit(0);
            }
            }while(1);
}
```

Input:

NOTE:

```
The terminals should be entered in single lower case letters,special symbol and non
            -terminals should be entered in single upper case letters.
extends to symbol is '->' and epsilon symbol is '@'

Enter the no. of terminals:2
Enter the terminal 1:a
Enter the terminal 2:b

Enter the no. of non terminals:3
Enter the non    -terminal 1:S
Enter the non    -terminal 2:A
Enter the non-terminal 3:B

Enter the no. of productions:7
Enter the production 1:S          ->aAB
Enter the production 2:S          ->bA
Enter the production 3:S          ->@
Enter the production 4:A          ->aAB
Enter the production 5:A          ->@
Enter the production 6:B          ->bB
Enter the production 7:B->@

Enter the start symbol:S

Enter the input string:aab$

The productions are:
 ProductionNo. Production
  1             S->aAB
  2             S->bA
  3             S->@
  4             A->aAB
  5             A->@
  6             B->bB
  7             B->@
```

```
Enter the parsing table:
 Enter the production number in the required entry as mentioned above.
Enter the undefined entry or error of table as '0'



Entry of table[S,a]:1

Entry of table[S,b]:2

Entry of table[S,$]:3

Entry of table[A,a]:4

Entry of table[A,b]:5

Entry of table[A            ,$]:5

Entry of table[B,a]:0

Entry of table[B,b]:6
Entry of table[B,$]:7
```

Output:

```
The given input string is parsed
```

Lab 28-29 – Left Recursion

Objectives

The objective of these labs is to find the left recursion from a grammar and then remove it.

Lab Tasks

Write a program which reads a regular grammar, and evaluate it either it has Left recursion or not. If the grammar is Left Recursive then remove the Left Recursion

Solution

```
#include<stdio.h>

#include<conio.h>

#include<string.h>



#define SIZE 20



void main () {
 char non_terminal;

                         char beta,alpha;

                         char production[SIZE];
                          int index=3;



             /* starting of the string following "->" */
                         clrscr();

                         printf("Enter the grammar:\n");

                          scanf("%s",production);

                          non_terminal=production[0];


 if(non_terminal==production[index])
```

```
{
                    alpha=production[index+1]; printf("Grammaris
                    left recursive.\n");

while(production[index]!=0 &&production[index]!='|')
                    {
index++;
if(production[index]!=0)
                              {
                              beta=production[index+1]; printf("Grammar without
                              left recursion:\n");

                              printf("%c->%c%c\'",non_terminal,beta,non_terminal);
                              printf("\n%c\'->%c%c\'|E\n",non_terminal,alpha,non_terminal);
                              }
else
                              {
printf("Grammar can't be reduced\n");
                              }
                    }
          }
          else
          {
          printf("Grammar is not leftrecursive.\n");
```

```
                }
            getch();




}
```

Input

S-->Sa

S-->a

Output

The grammar is left Recursive.

Grammar without left recursion:

S->S’

S’->aS’|E

Grammar can’t be reduced

Lab 30 – Semantic Analyzer-Attribute Grammar

Objectives

The objective of this lab is to give introduction to student about semantic analysis of a grammar.

Lab Tasks

What is Semantic Analysis?

Parsing only verifies that the program consists of tokens arranged in a syntactically valid combination. The next step is to move forward to semantic analysis, where we delve even deeper to check whether they form a sensible set of instructions in the programming language. Whereas any old noun phrase followed by some verb phrase makes a syntactically correct English sentence, a semantically correct one has subject-verb agreement, proper use of gender, and the components go together to express an idea that makes sense. For a program to be semantically valid, all variables, functions, classes, etc. must be properly defined, expressions and variables must be used in ways that respect the type system, access control must be respected, and so forth. Semantic analysis is the front end's penultimate phase and the compiler's last chance to weed out incorrect programs. Here is the need to ensure the program is sound enough to carry on to code generation.

A large part of semantic analysis consists of tracking variable/function/type declarations and type checking. In many languages, identifiers have to be declared before they're used. As the compiler encounters a new declaration, it records the type information assigned to that identifier. Then, as it continues examining the rest of the program, it verifies that the type of an identifier is respected in terms of the operations being performed. For example, the type of the right side expression of an assignment statement should match the type of the left side, and the left side needs to be a properly declared and assignable identifier. The parameters of a function should match the arguments of a function call in both number and type. The language may require that identifiers be unique, thereby forbidding two global declarations from sharing the same name. Arithmetic operands will need to be of numeric—perhaps even the exact same type (no automatic *int-to-double* conversion, for instance). These are examples of the things checked in the semantic analysis phase.

Some semantic analysis might be done right in the middle of parsing. As a particular construct is recognized, say an addition expression, the parser action could check the two operands and verify they are of numeric type and compatible for this operation. In fact, in a one-pass compiler, the code is generated right then and there as well. In a compiler that runs in more than one pass (such as the one we are building for Decaf), the first pass digests the syntax and builds a parse tree representation of the program. A second pass traverses the tree to verify that the program respects all semantic rules as well. The single-pass strategy is typically more efficient, but multiple passes allow for better modularity and flexibility (i.e., can often order things arbitrarily in the source program).

Types and Declarations

A type is a set of values and a set of operations operating on those values. There are three categories of types in most programming languages:

- *Base types:* int, float, double, char, bool, etc. These are the primitive types provided directly by the underlying hardware. There may be a facility for user-defined variants on the base types (such as C enums).
- *Compound types*: arrays, pointers, records, structs, unions, classes, and so on. These types are constructed as aggregations of the base types and simple compound types.

- *Complex types:* lists, stacks, queues, trees, heaps, tables, etc. You may recognize these as abstract data types. A language may or may not have support for these sort of higherlevel abstractions.

In many languages, a programmer must first establish the name and type of any data object (e.g., variable, function, type, etc). In addition, the programmer usually defines the lifetime. A declaration is a statement in a program that communicates this information to the compiler. The basic declaration is just a name and type, but in many languages it may include modifiers that control visibility and lifetime (i.e., static in C, private in Java). Some languages also allow declarations to initialize variables, such as in C, where you can declare and initialize in one statement. The following C statements show some example declarations:

```
double calculate(int a, double b); // function prototype

int x = 0; // global variables available


throughout double y; // the program


int main()


{


int m[3]; // local variables available only in main

char *n;
 ...

}
```

Function declarations or prototypes serve a similar purpose for functions that variable declarations do for variables. Function and method identifiers also have a type, and the compiler can use ensure that a program is calling a function/method correctly. The compiler uses the prototype to check the number and types of arguments in function calls. The location and qualifiers establish the visibility of the function (Is the function global? Local to the module? Nested in another procedure? Attached to a class?) Type declarations (e.g., C typedef, C++ classes) have similar behaviors with respect to declaration and use of the new typename.

What is Attribute Grammar?

An attribute grammar is a formal way to define attributes for the productions of a formal grammar, associating these attributes to values. The evaluation occurs in the nodes of the abstract syntax tree, when the language is processed by some parser or compiler.

The attributes are divided into two groups: synthesized attributes and inherited attributes. The synthesized attributes are the result of the attribute evaluation rules, and may also use the values of the inherited attributes. The inherited attributes are passed down from parent nodes.

Why we use Attribute Grammar?

Synthesized attributes are used to pass semantic information up the parse tree, while inherited attributes help pass semantic information down and across it. For instance, when constructing a language translation tool, such as a compiler, it may be used to assign semantic values to syntax constructions. Also, it is possible to validate semantic checks associated with a grammar, representing the rules of a language not explicitly imparted by the syntax definition.

Attribute grammars can also be used to translate the syntax tree directly into code for some specific machine, or into some intermediate language.

One strength of attribute grammars is that they can transport information from anywhere in the abstract syntax tree to anywhere else, in a controlled and formal way.