DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

COMPILER DESIGN

LAB MANUAL

(R22A0588)

B. TECH CSE

(III YEAR – I SEM)

R22 REGULATION
(2024-25)

Name : _____

Roll no : _____

Section : _____

Year : _____

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
(Autonomous Institution – UGC, Govt. of India)
Recognized under 2(f) and 12 (B) of UGC ACT 1956
(Affiliated to JNTUH, Hyderabad, Approved by AICTE - Accredited by NBA & NAAC – 'A' Grade - ISO 9001:2015 Certified)
Maisammaguda, Dhulapally (Post Via. Hakimpet), Secunderabad – 500100, Telangana State, India

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision

To acknowledge quality education and instill high patterns of discipline making the students technologically superior and ethically strong which involves the improvement in the quality of life in human race.

Mission

- To achieve and impart holistic technical education using the best of infrastructure, outstanding technical and teaching expertise to establish the students in to competent and confident engineers.
- Evolving the center of excellence through creative and innovative teaching learning practices for promoting academic achievement to produce internationally accepted competitive and world class professionals.

PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)

PEO1 – ANALYTICAL SKILLS

- To facilitate the graduates with the ability to visualize, gather information, articulate, analyze, solve complex problems, and make decisions. These are essential to address thechallenges of complex and computation intensive problems increasing their productivity.

PEO2 – TECHNICAL SKILLS

- To facilitate the graduates with the technical skills that prepare them for immediate employment and pursue certification providing a deeper understanding of the technology in advanced areas of computer science and related fields, thus encouraging to pursue higher education and research based on their interest.

PEO3 – SOFT SKILLS

- To facilitate the graduates with the soft skills that include fulfilling the mission, setting goals, showing self-confidence by communicating effectively, having a positive attitude, get involved in team-work, being a leader, managing their career and their life.

PEO4 – PROFESSIONAL ETHICS

- To facilitate the graduates with the knowledge of professional and ethical responsibilities by paying attention to grooming, being conservative with style, following dress codes, safety codes, and adapting themselves to technological advancements.

PROGRAM SPECIFIC OUTCOMES (PSOs)

After the completion of the course, B. Tech Computer Science and Engineering, the graduates will have the following Program Specific Outcomes:

1.Fundamentals and critical knowledge of the Computer System:- Able to Understand the working principles of the computer System and its components , Apply the knowledge to build, asses, and analyze the software and hardware aspects of it .

2.The comprehensive and Applicative knowledge of Software Development: Comprehensive skills of Programming Languages, Software process models, methodologies, and able to plan, develop, test, analyze, and manage the software and hardware intensive systems in heterogeneous platforms individually or working in teams.

3.Applications of Computing Domain & Research: Able to use the professional, managerial, interdisciplinary skill set, and domain specific tools in development processes, identify the research gaps, and provide innovative solutions to them.

PROGRAM OUTCOMES (POs)

Engineering Graduates should possess the following:

1. Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design / development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multi disciplinary environments.

12 .Life- long learning: Recognize the need for,and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

MALLA REDDY COLLEGE OF ENGINEERING & TECHNOLOGY
Maisammaguda, Dhulapally Post, Via Hakimpet, Secunderabad – 500100

GENERAL LABORATORY INSTRUCTIONS

1. Students are advised to come to the laboratory at least 5 minutes before (to the starting time), those who come after 5 minutes will not be allowed into the lab.
2. Plan your task properly much before to the commencement, come prepared to the labwith the synopsis / program / experiment details.
3. Student should enter into the laboratory with:
    a. Laboratory observation notes with all the details (Problem statement, Aim, Algorithm, Procedure, Program, Expected Output, etc.,) filled in for the lab session.
    b. Laboratory Record updated up to the last session experiments and other utensils (if any) needed in the lab.
    c. Proper Dress code and Identity card.
4. Sign in the laboratory login register, write the TIME-IN, and occupy the computer system allotted to you by the faculty.
5. Execute your task in the laboratory, and record the results / output in the lab observation note book, and get certified by the concerned faculty.
6. All the students should be polite and cooperative with the laboratory staff, must maintain the discipline and decency in the laboratory.
7. Computer labs are established with sophisticated and high end branded systems, which should be utilized properly.
8. Students / Faculty must keep their mobile phones in SWITCHED OFF mode during the lab sessions. Misuse of the equipment, misbehaviors with the staff and systems etc., will attract severe punishment.
9. Students must take the permission of the faculty in case of any urgency to go out; if anybody found loitering outside the lab / class without permission during workinghours will be treated seriously and punished appropriately.
10. Students should LOG OFF/ SHUT DOWN the computer system before he/she leaves the lab after completing the task (experiment) in all aspects. He/she must ensure the system / seat is kept properly.

HEAD OF THE DEPARTMENT **PRINCIPAL**

INDEX

S.No	Date	Name of the Activity/Experiment	Grade/ Marks	Faculty Signature

INDEX

WEEK 1

AIM : Write a C Program to Scan and Count the number of characters, words, and lines in a file

ALGORITHM/PROCEDURE:

1. Start
2. Read the input file/text
3. Initialize the counters for characters, words, lines to zero
4. Scan the characters, words, lines and
5. Increment the respective counters
6. Display the counts
7. End

PROGRAM:

```
#include<stdio.h>
#include<stdlib.h>
int main()
{
  FILE *fp;
  char fname[10],ch;
  int c_count=0,w_count=0,l_count=0;
/* Input Surce File name */
  printf("\nEnter the file name: ");
  scanf("%s",fname);
/* Open Source file in Read mode */
  fp=fopen(fname,"r");
/* Check if the file existing */
if(fp==NULL)
   {
   printf(" Unable to Open %s file",fname);
   printf("\n Please check if it exists!");
   exit(EXIT_FAILURE);
    }
while((ch=fgetc(fp))!=EOF)
    {
   c_count++;
     /* Check for newline chars */
        if(ch == '\n'|| ch == ' ')
           l_count++;
     /* Word count */
   if(ch ==' '|| ch == '\t' || ch == '\n')
          w_count++;
    }
printf("\n The file Statistics \n");
 printf(" The No of Characters: %d", c_count);
 printf("\n The No of Words: %d", w_count);
 printf(" \n The No of Lines: %d", l_count);
/* Close file to release resource */
fclose(fp);
} //End of main()
```

```
Input: Enter the Identifier input string/ file : 1.txt

aaa bbb ccc
ddd eee fff
ggg hhh iii
```

Output:

ubuntu@DESKTOP-FHDVQ08:~/lab$ ./a.out

Enter the file name: 1.txt

The file Statistics
The No of Characters: 36
The No of Words: 9
The No of Lines: 3

[Viva Questions]

1. What is Compiler?
2. List various language Translators.
3. Is it necessary to translate a HLL program? Explain.
4. List out the phases of a compiler?
5. Which phase of the compiler is called an optional phase? Why?

WEEK 2

AIM : Write a C Program to implement DFAs that recognize identifiers, constants, and operators of the mini language

ALGORITHM/ PROCEDURE:

1 Start
2 Design the DFAs(M)to recognize Identifiers, Constants, and Operators
3 Read the input string **w** give it as input to the DFAM
4 DFA processes the input and outputs “Yes” if wbЄL(M),“No” otherwise
5 Display the output
6 End

PROGRAM:

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
// Returns '1' if the character is a DELIMITER.
int isDelimiter(char ch)
{
if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == ',' || ch == ';' || ch == '>' ||
ch == '<' || ch == '=' || ch == '(' || ch == ')' ||
ch == '[' || ch == ']' || ch == '{' || ch == '}')
return (1);
return (0);
}
// Returns '1' if the character is an OPERATOR.
int isOperator(char ch)
{
if (ch == '+' || ch == '-' || ch == '*' ||
ch == '/' || ch == '>' || ch == '<' ||
ch == '=')
return (1);
return (0);
}
// Returns '1' if the string is a VALID IDENTIFIER.
int validIdentifier(char* str)
{
if (str[0] == '0' || str[0] == '1' || str[0] == '2' ||
str[0] == '3' || str[0] == '4' || str[0] == '5' ||
str[0] == '6' || str[0] == '7' || str[0] == '8' ||
str[0] == '9' || isDelimiter(str[0]) == 1)
return (0);
return (1);
}
// Returns '1' if the string is a KEYWORD.
int isKeyword(char* str)
```

```
{
if (!strcmp(str, "if") || !strcmp(str, "else") ||
!strcmp(str, "while") || !strcmp(str, "do") ||
!strcmp(str, "break") ||
!strcmp(str, "continue") || !strcmp(str, "int")
|| !strcmp(str, "double") || !strcmp(str, "float")
|| !strcmp(str, "return") || !strcmp(str, "char")
|| !strcmp(str, "case") || !strcmp(str, "char")
|| !strcmp(str, "sizeof") || !strcmp(str, "long")
|| !strcmp(str, "short") || !strcmp(str, "typedef")
|| !strcmp(str, "switch") || !strcmp(str, "unsigned")
|| !strcmp(str, "void") || !strcmp(str, "static")
|| !strcmp(str, "struct") || !strcmp(str, "goto"))
return (1);
return (0);
}
// Returns '1' if the string is an INTEGER.
int isInteger(char* str)
{
int i, len = strlen(str);
if (len == 0)
return (0);
for (i = 0; i < len; i++) {
if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
&& str[i] != '9' || (str[i] == '-' && i > 0))
return (0);
}
return (1);
}
// Returns '1' if the string is a REAL NUMBER.
int isRealNumber(char* str)
{
int i, len = strlen(str);
int hasDecimal = 0;
if (len == 0)
return (0);
for (i = 0; i < len; i++) {
if (str[i] != '0' && str[i] != '1' && str[i] != '2'
&& str[i] != '3' && str[i] != '4' && str[i] != '5'
&& str[i] != '6' && str[i] != '7' && str[i] != '8'
&& str[i] != '9' && str[i] != '.' ||
(str[i] == '-' && i > 0))
return (0);
if (str[i] == '.')
hasDecimal = 1;
}
return (hasDecimal);
}
```

```
// Extracts the SUBSTRING.
char* subString(char* str, int left, int right)
{
int i;
char* subStr = (char*)malloc(
sizeof(char) * (right - left + 2));
for (i = left; i <= right; i++)
subStr[i - left] = str[i];
subStr[right - left + 1] = '\0';
return (subStr);
}
// Parsing the input STRING.
void parse(char* str)
{
int left = 0, right = 0;
int len = strlen(str);
while (right <= len && left <= right) {
if (isDelimiter(str[right]) == 0)
right++;
if (isDelimiter(str[right]) == 1 && left == right) {
if (isOperator(str[right]) == 1)
printf("'%c' IS AN OPERATOR\n", str[right]);
right++;
left = right;
} else if (isDelimiter(str[right]) == 1 && left != right
|| (right == len && left != right)) {
char* subStr = subString(str, left, right - 1);
if (isKeyword(subStr) == 1)
printf("'%s' IS A KEYWORD\n", subStr);
else if (isInteger(subStr) == 1)
printf("'%s' IS AN INTEGER\n", subStr);
else if (isRealNumber(subStr) == 1)
printf("'%s' IS A REAL NUMBER\n", subStr);
else if (validIdentifier(subStr) == 1
&& isDelimiter(str[right - 1]) == 0)
printf("'%s' IS A VALID IDENTIFIER\n", subStr);
else if (validIdentifier(subStr) == 0
&& isDelimiter(str[right - 1]) == 0)
printf("'%s' IS NOT A VALID IDENTIFIER\n", subStr);
left = right;
}
}
return;
}
// DRIVER FUNCTION
void main()
{
// maximum length of string is 100 here
char str[100] = "int a = c + y; ";
//clrscr();
```

```
parse(str); // calling the parse function
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ nano week2.c
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ gcc week2.c
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ ./a.out
'int' IS A KEYWORD
'a' IS A VALID IDENTIFIER
'=' IS AN OPERATOR
'c' IS A VALID IDENTIFIER
'+' IS AN OPERATOR
'y' IS A VALID IDENTIFIER
```

[Viva Questions]

1. What is an Interpreter?
2. What are the other language processors you know?
3. What is the difference between DFA/ minimum DFA?
4. Write the difference between the interpreter and Compiler

WEEK 3

AIM : Write a lex program to implement simple calculator

PROGRAM:

The YACC program code for a simple calculator is given below: filename 88.y

```
%{
   #include<stdio.h>
   int flag=0;
%}
%name parse
%token NUMBER
%left '*' '/' '%'
%left '+' '-'
%left '(' ')'
%%
ArithmeticExpression: E{printf("\nResult=%d\n",$$); return 0;   };
E:E'+'E {$$=$1+$3;}
 |E'-'E {$$=$1-$3;}
 |E'*'E {$$=$1*$3;}
 |E'/'E {$$=$1/$3;}
 |E'%'E {$$=$1%$3;}
 |'('E')' {$$=$2;}
 | NUMBER {$$=$1;}
;
%%
void main()
{
  printf("\nEnter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus and Round brackets:\n");
  yyparse();
 if(flag==0)
  printf("\nEntered arithmetic expression is Valid\n\n");
}
void yyerror()
{
  printf("\nEntered arithmetic expression is Invalid\n\n");
  flag=1;
}
```

The lex file of the program is given below: filename 88.l

```
   %{
   #include<stdio.h>
   #include "y.tab.h"
   extern int yylval;
   %}
   %%
   [0-9]+ {yylval=atoi(yytext);return NUMBER;    }
```

```
[\t] ;
[\n] return 0;
. return yytext[0];
%%
int yywrap()
{
return 1;
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~$ yacc -d 88.y
ubuntu@DESKTOP-FHDVQ08:~$ lex 88.l
ubuntu@DESKTOP-FHDVQ08:~$ gcc lex.yy.c y.tab.c -w
ubuntu@DESKTOP-FHDVQ08:~$ ./a.out

Enter Any Arithmetic Expression which can have operations Addition,
Subtraction, Multiplication, Divison, Modulus and Round brackets:
3+4*5

Result=35

Entered arithmetic expression is Valid
```

[Viva Questions]

1. What is a Preprocessor and what is its role in compilation?
2. Which language is both compiled and interpreted?
3. List out the languages that are interpreted?
4. Explain the working of a NFA?
5. When do you prefer to design an NFA to DFA?

WEEK 4

AIM: Design a Lexical analyzer for the given language.
{Note-The lexical analyzer should ignore redundant spaces, tabs and newlines. It should also ignore comments. Although the syntax specification states that identifiers can be arbitrarily long, you may restrict the length to some reasonable value.}

ALGORITHM/ PROCEDURE:

We make use of the following two functions in the process.

lookup()–it takes string as argument t and checks its presence in the symbol table. If the string is found then returns the address else it returns NULL.

Insert()–it take string as its argument and the same is inserted into the symbol table and the corresponding address is returned.

1. Start
2. Declare an array of characters, an input file to store the input;
3. Read the character from the input file and put it into character type of variable, say 'c'.
4. If 'c' is blank then do nothing.
5. If 'c' is newline character line = line+1.
6. If 'c' is digit, set token Val, the value assigned for a digit and return the 'NUMBER'.
7. If 'c' is proper token then assign the token value.
8. Print the complete table with Token entered by the user, Associated token value.
9. Stop

PROGRAM:

```
//remove spaces lex program file name space2.l
%{
#include<stdio.h>
%}
%%
"/*".+"*/"     {}
"//".*  {}
%%
int main()
{
      yyin=fopen("f1.c","r");
      yyout=fopen("f2.c","w");
      yylex();
      fclose(yyin);
      fclose(yyout);
```

```
    return 0;
}
int yywrap()
{
return 1;
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~$ lex space2.l
ubuntu@DESKTOP-FHDVQ08:~$ gcc lex.yy.c
ubuntu@DESKTOP-FHDVQ08:~$ ./a.out
ubuntu@DESKTOP-FHDVQ08:~$ cat input.txt
hello     how
  are
you?
ubuntu@DESKTOP-FHDVQ08:~$ cat Output.txt
hellohowareyou?
```

//remove comments lex program file name comment.l

```
%{
#include<stdio.h>
int sl=0;
int ml=0;
%}
%%
"/*"[a-zA-Z0-9' '\t\n]+"*/"     ml++;
"//".*  sl++;
%%

int main()
{
    yyin=fopen("c1.c","r");
    yyout=fopen("f2.c","w");
    yylex();
    fclose(yyin);
    fclose(yyout);
    printf("\n Number of single line comments are = %d\n",sl);
    printf("\nNumber of multiline comments are =%d\n",ml);
    return 0;
}
int yywrap()
{
return 1;
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~$ nano comment.l
ubuntu@DESKTOP-FHDVQ08:~$ lex comment.l
ubuntu@DESKTOP-FHDVQ08:~$ gcc lex.yy.c
ubuntu@DESKTOP-FHDVQ08:~$ ./a.out

 Number of single line comments are = 1

Number of multiline comments are =1

input file: c1.c

// program
#include<stdio.h>
void main()
{
/* djcj
cskkk */
printf("hello world");
}

output file: f2.c

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

printf("hello world");
}
```

[Viva Questions]

1. What is lexical analyzer?
2. Which compiler is used for lexical analysis?
3. What is the output of Lexical analyzer?

WEEK 5

AIM : Implement the lexical analyzer using JLex, flex, flex or lex or other lexical analyzer generation tools

//lex program to accept the language contain all the strings over an alphabet {a,b}

PROGRAM:

```
%{
#include<stdio.h>
%}
%%
[a+b]+ {printf("valid string");}
.* {printf("invalid");}
[\n] return 0;
%%
int main()
{
printf("enter string to accept language\n");
yylex();
return 0;
}
int yywrap()
{
return 1;
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ lex dfa.l
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ gcc lex.yy.c
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ ./a.out
enter string to accept language
abaaa
valid string
```

[Viva Questions]

1. Which Finite state machines are used in lexical analyzer design?
2. What is the role of regular expressions, grammars in Lexical Analizer?

WEEK 6

AIM : write a yacc program to accept given Grammer s- >aA|a,A->a.

PROGRAM:

```
%{
#include<stdio.h>
int flag=0;
%}
%name parse
%token 'a'
%%
S:'a' A
|'a'
A:'a'
;
%%
int main()
{
 printf("\nEnter Any String:\n");
   yyparse();
  if(flag==0)
   printf("\nEntered string is Valid\n\n");
return 0;
}
void yyerror()
{
 printf("\nEntered string is Invalid\n\n");
   flag=1;
}
int yylex()
{
char c;
c=getchar();
if(c=='\n')
return 0;
else
return c;
}
int yywrap()
{
return 1;
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ yacc p.y
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ gcc y.tab.c –w
ubuntu@DESKTOP-FHDVQ08:~/IIIA$ ./a.out

Enter Any String:
aba

Entered string is Valid
```

[Viva Questions]

1. What are the functions of a Scanner?
2. What is Token?
3. What is lexeme, Pattern?
4. What is purpose of Lex?
5. What are the other tools used in Lexical Analysis?

WEEK 7

AIM: Write a Program for construction of LR Parsing table using C

PROGRAM:

```
#include<stdio.h>
#include<string.h>
int i,j,k,m,n=0,o,p,ns=0,tn=0,rr=0,ch=0;
char
read[15][10],gl[15],gr[15][10],temp,templ[15],tempr[15][10],*ptr,temp2[5],dfa[15][15];
struct states
{
   char lhs[15],rhs[15][10];
   int n;
}I[15];
int compstruct(struct states s1,struct states s2)
{
   int t;
   if(s1.n!=s2.n)
      return 0;
   if( strcmp(s1.lhs,s2.lhs)!=0 )
      return 0;
   for(t=0;t<s1.n;t++)
      if( strcmp(s1.rhs[t],s2.rhs[t])!=0 )
         return 0;
   return 1;
}
void moreprod()
{
   int r,s,t,l1=0,rr1=0;
   char *ptr1,read1[15][10];
   for(r=0;r<I[ns].n;r++)
```

```
{
   ptr1=strchr(I[ns].rhs[l1],'.');
   t=ptr1-I[ns].rhs[l1];
   if( t+1==strlen(I[ns].rhs[l1]) )
   {
      l1++;
      continue;
   }
   temp=I[ns].rhs[l1][t+1];
   l1++;
   for(s=0;s<rr1;s++)
      if( temp==read1[s][0] )
         break;
   if(s==rr1)
   {
      read1[rr1][0]=temp;
      rr1++;
   }
   else
      continue;
   for(s=0;s<n;s++)
   {
      if(gl[s]==temp)
      {
         I[ns].rhs[I[ns].n][0]='.';
         I[ns].rhs[I[ns].n][1]=NULL;
         strcat(I[ns].rhs[I[ns].n],gr[s]);
         I[ns].lhs[I[ns].n]=gl[s];
         I[ns].lhs[I[ns].n+1]=NULL;
         I[ns].n++;
```

```
            }
        }
    }
}
void canonical(int l)
{
    int t1;
    char read1[15][10],rr1=0,*ptr1;
    for(i=0;i<I[l].n;i++)
    {
        temp2[0]='.';
        ptr1=strchr(I[l].rhs[i],'.');
        t1=ptr1-I[l].rhs[i];
        if( t1+1==strlen(I[l].rhs[i]) )
            continue;
        temp2[1]=I[l].rhs[i][t1+1];
        temp2[2]=NULL;
        for(j=0;j<rr1;j++)
            if( strcmp(temp2,read1[j])==0 )
                break;
        if(j==rr1)
        {
            strcpy(read1[rr1],temp2);
            read1[rr1][2]=NULL;
            rr1++;
        }
        else
            continue;
        for(j=0;j<I[0].n;j++)
        {
```

```
    ptr=strstr(I[l].rhs[j],temp2);
    if( ptr )
    {
       templ[tn]=I[l].lhs[j];
       templ[tn+1]=NULL;
       strcpy(tempr[tn],I[l].rhs[j]);
       tn++;
    }
  }
  for(j=0;j<tn;j++)
  {
    ptr=strchr(tempr[j],'.');
    p=ptr-tempr[j];
    tempr[j][p]=tempr[j][p+1];
    tempr[j][p+1]='.';
    I[ns].lhs[I[ns].n]=templ[j];
    I[ns].lhs[I[ns].n+1]=NULL;
    strcpy(I[ns].rhs[I[ns].n],tempr[j]);
    I[ns].n++;
  }
  moreprod();
  for(j=0;j<ns;j++)
  {
    //if ( memcmp(&I[ns],&I[j],sizeof(struct states))==1 )
    if( compstruct(I[ns],I[j])==1 )
    {
       I[ns].lhs[0]=NULL;
       for(k=0;k<I[ns].n;k++)
          I[ns].rhs[k][0]=NULL;
       I[ns].n=0;
```

```
                dfa[l][j]=temp2[1];
                break;
            }
        }
        if(j<ns)
        {
            tn=0;
            for(j=0;j<15;j++)
            {
                templ[j]=NULL;
                tempr[j][0]=NULL;
            }
            continue;
        }
        dfa[l][j]=temp2[1];
        printf("\n\nI%d :",ns);
        for(j=0;j<I[ns].n;j++)
            printf("\n\t%c -> %s",I[ns].lhs[j],I[ns].rhs[j]);
        getch();
        ns++;
        tn=0;
        for(j=0;j<15;j++)
        {
            templ[j]=NULL;
            tempr[j][0]=NULL;
        }
    }
}
void main()
{
```

```
FILE *f;
int l;
clrscr();
for(i=0;i<15;i++)
{
   I[i].n=0;
   I[i].lhs[0]=NULL;
   I[i].rhs[0][0]=NULL;
   dfa[i][0]=NULL;
}
f=fopen("tab6.txt","r");
while(!feof(f))
{
   fscanf(f,"%c",&gl[n]);
   fscanf(f,"%s\n",gr[n]);
   n++;
}
printf("THE GRAMMAR IS AS FOLLOWS\n");
for(i=0;i<n;i++)
   printf("\t\t\t\t%c -> %s\n",gl[i],gr[i]);
I[0].lhs[0]='Z';
strcpy(I[0].rhs[0],".S");
I[0].n++;
l=0;
for(i=0;i<n;i++)
{
   temp=I[0].rhs[l][1];
   l++;
   for(j=0;j<rr;j++)
      if( temp==read[j][0] )
```

```
            break;
    if(j==rr)
    {
       read[rr][0]=temp;
       rr++;
    }
    else
       continue;
    for(j=0;j<n;j++)
    {
       if(gl[j]==temp)
       {
          I[0].rhs[I[0].n][0]='.';
          strcat(I[0].rhs[I[0].n],gr[j]);
          I[0].lhs[I[0].n]=gl[j];
          I[0].n++;
       }
    }
}
ns++;
printf("\nI%d :\n",ns-1);
for(i=0;i<I[0].n;i++)
    printf("\t%c -> %s\n",I[0].lhs[i],I[0].rhs[i]);
for(l=0;l<ns;l++)
    canonical(l);
printf("\n\n\t\tPRESS ANY KEY FOR DFA TABLE");
getch();
clrscr();
printf("\t\t\tDFA TABLE IS AS FOLLOWS\n\n\n");
for(i=0;i<ns;i++)
```

```
    {
       printf("I%d : ",i);
       for(j=0;j<ns;j++)
          if(dfa[i][j]!='\0')
             printf("'%c'->I%d | ",dfa[i][j],j);
       printf("\n\n\n");
    }
    printf("\n\n\n\t\tPRESS ANY KEY TO EXIT");
    getch();
}
```

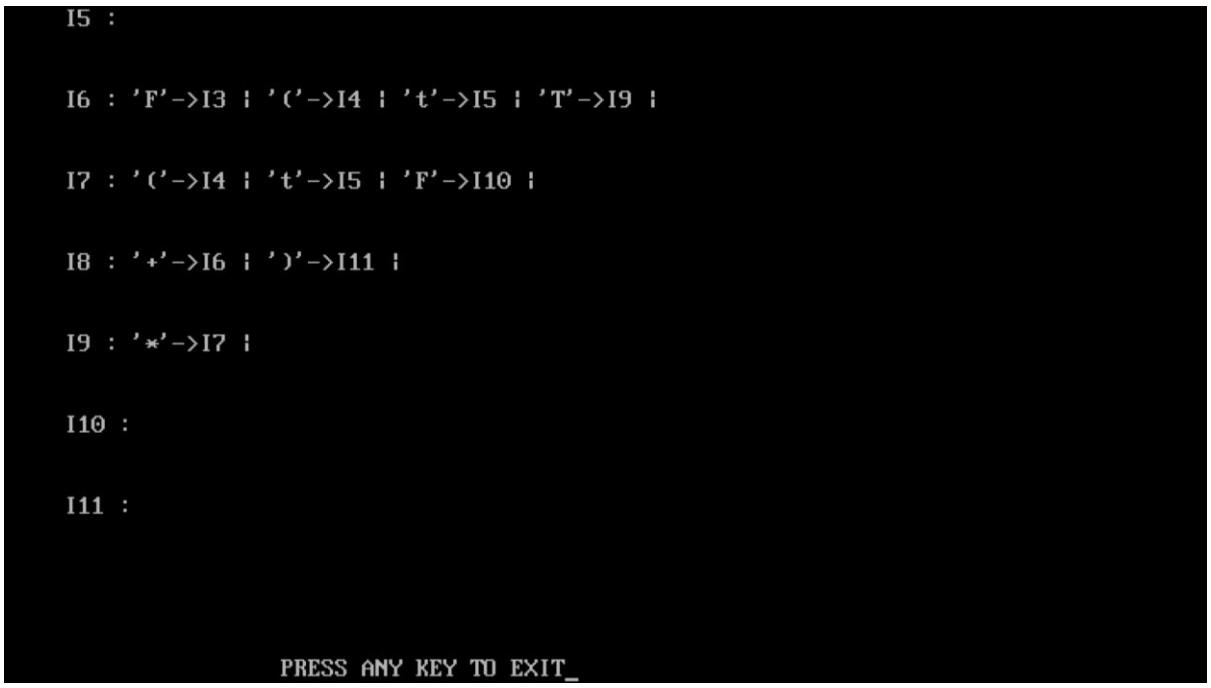
Output:

```
THE GRAMMAR IS AS FOLLOWS
                                S -> S+T
                                S -> T
                                T -> T*F
                                T -> F
                                F -> (S)
                                F -> t

I0 :
        Z -> .S
        S -> .S+T
        S -> .T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I1 :
        Z -> S.
        S -> S.+T
```

```
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I1 :
        Z -> S.
        S -> S.+T

I2 :
        S -> T.
        T -> T.*F

I3 :
        T -> F.

I4 :
        F -> (.S)
        S -> .S+T
        S -> .T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t
```

```
        F -> .(S)
        F -> .t

I5 :
        F -> t.

I6 :
        S -> S+.T
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I7 :
        T -> T*.F
        F -> .(S)
        F -> .t

I8 :
        F -> (S.)
        S -> S.+T

I9 :
        S -> S+T.
        T -> T.*F
```

```
        T -> .T*F
        T -> .F
        F -> .(S)
        F -> .t

I7 :
        T -> T*.F
        F -> .(S)
        F -> .t

I8 :
        F -> (S.)
        S -> S.+T

I9 :
        S -> S+T.
        T -> T.*F

I10 :
        T -> T*F.

I11 :
        F -> (S).

                PRESS ANY KEY FOR DFA TABLE_
```


I5 :
I6 : 'F'->I3 | '('->I4 | 't'->I5 | 'T'->I9 |
I7 : '('->I4 | 't'->I5 | 'F'->I10 |
I8 : '+'->I6 | ')'->I11 |
I9 : '*'->I7 |
I10 :
I11 :
PRESS ANY KEY TO EXIT_

[Viva Questions]

1. What is a parser and state the Role of it?
2. Types of parsers? Examples to each
3. What are the Tools available for implementation?
4. How do you calculate FIRST(), FOLLOW() sets used in Parsing Table construction?

WEEK 8

AIM: Design a Predictive Parser for the following grammar G: {E->TE', E' -> +TE' | 0, T->FT' , T'-> *FT'|0 , F-> (E) | id }

//C program for implementing the functionalities of predictive parser

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
char prol[7][10]={"S","A","A","B","B","C","C"};
char pror[7][10]={"A","Bb","Cd","aB","@","Cc","@"};
char prod[7][10]={"S->A","A->Bb","A->Cd","B->aB","B->@","C->Cc","C->@"};
char first[7][10]={"abcd","ab","cd","a@","@","c@","@"};
char follow[7][10]={"$","$","$","a$","b$","c$","d$"};
char table[5][6][10];
numr(char c)
{
switch(c)
{
case 'S': return 0;
case 'A': return 1;
case 'B': return 2;
case 'C': return 3;
case 'a': return 0;
case 'b': return 1;
case 'c': return 2;
case 'd': return 3;
case '$': return 4;
}
return(2);
}
void main()
{
int i,j,k;
clrscr();
for(i=0;i<5;i++)
for(j=0;j<6;j++)
strcpy(table[i][j]," ");
printf("\nThe following is the predictive parsing table for the following grammar:\n");
for(i=0;i<7;i++)
printf("%s\n",prod[i]);
printf("\nPredictive parsing table is\n");
fflush(stdin);
for(i=0;i<7;i++)
{
k=strlen(first[i]);
for(j=0;j<10;j++)
if(first[i][j]!='@')
strcpy(table[numr(prol[i][0])+1][numr(first[i][j])+1],prod[i]);
```

```
}
for(i=0;i<7;i++)
{
if(strlen(pror[i])==1)
{
if(pror[i][0]=='@')
{
k=strlen(follow[i]);
for(j=0;j<k;j++)
strcpy(table[numr(prol[i][0])+1][numr(follow[i][j])+1],prod[i]);
}
}
}
strcpy(table[0][0]," ");
strcpy(table[0][1],"a");
strcpy(table[0][2],"b");
strcpy(table[0][3],"c");
strcpy(table[0][4],"d");
strcpy(table[0][5],"$");
strcpy(table[1][0],"S");
strcpy(table[2][0],"A");
strcpy(table[3][0],"B");
strcpy(table[4][0],"C");
printf("\n---------------------------------------------------\n");
for(i=0;i<5;i++)
for(j=0;j<6;j++)
{
printf("%-10s",table[i][j]);
if(j==5)
printf("\n---------------------------------------------------\n");
}
getch();
}
```

Output:

```
The following is the predictive parsing table for the following grammar:
S->A
A->Bb
A->Cd
B->aB
B->@
C->Cc
C->@

Predictive parsing table is

------------------------------------------------------------
          a         b         c         d         $
------------------------------------------------------------
S         S->A      S->A      S->A      S->A
------------------------------------------------------------
A         A->Bb     A->Bb     A->Cd     A->Cd
------------------------------------------------------------
B         B->aB     B->@      B->@                B->@
------------------------------------------------------------
C                             C->@      C->@      C->@
------------------------------------------------------------

_
```

[Viva Questions]

1. What is yacc? Are there any other too ls available for parser generation?
2. How do you use it?
3. Structure of parser specification program
4. How many ways we can generate the Parser

WEEK 9

AIM: Design LALR bottom-up parser for the above language using tools or C

PROGRAM:

// yacc program to check whether the string is accepted by given grammer or not.

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%name parse
%token '*''+''(')''i'%%
E:E'+'T
|T
T:T'*'F
|F
F:'('E')'
|'i'
;
%%
int main()
{
yyparse();
printf("accepted");
return 0;
}
int yylex()
{
char c;
c=getchar();
if(c=='\n')
return 0;
else
return c;
}
yyerror()
{
printf("error");
exit(1);
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~$ yacc s1.y
ubuntu@DESKTOP-FHDVQ08:~$ gcc y.tab.c -w
ubuntu@DESKTOP-FHDVQ08:~$ ./a.out
i+i*i
accepted
```

[Viva Questions]

1. What is abstract syntax tree?
2. What is quadruple?
3. What is the difference between Triples and Indirect Triple?

WEEK 10

AIM: Write program to generate machine code from the abstract syntax tree generated by the parser. The following instruction set may be considered as target code

PROGRAM:

Bnf.l

```
%option noyywrap
%{
#include"y.tab.h"
#include<stdio.h>
#include<string.h>
int LineNo=1;
%}
identifier [a-zA-Z][_a-zA-Z0-9]*
number [0-9]+|([0-9]*\.[0-9]+)
%%
main\(\) return MAIN;
if return IF;
else return ELSE;
while return WHILE;
int |
char |
float return TYPE;
{identifier} {strcpy(yylval.var,yytext);
return VAR;}
{number} {strcpy(yylval.var,yytext);
return NUM;}
\< |
\> |
\>= |
\<= |
== {strcpy(yylval.var,yytext);
return RELOP;}
[ \t] ;
\n LineNo++;
. return yytext[0];
%%
```

Bnf.y

```
%{
#include<string.h>
#include<stdio.h>
struct quad
```

```
{
char op[5];
char arg1[10];
char arg2[10];
char result[10];
}QUAD[30];
struct stack
{
int items[100];
int top;
}stk;
int Index=0,tIndex=0,StNo,Ind,tInd;
extern int LineNo;
%}
%union { char var[10]; }
%token <var> NUM VAR RELOP
%token MAIN IF ELSE WHILE TYPE
%type <var> EXPR ASSIGNMENT CONDITION IFST ELSEST WHILELOOP
%left '-' '+'
%left '*' '/'
%%
PROGRAM : MAIN BLOCK
;
BLOCK: '{' CODE '}'
;
CODE: BLOCK
| STATEMENT CODE
| STATEMENT
;
STATEMENT: DESCT ';'
| ASSIGNMENT ';'
| CONDST
| WHILEST
;
DESCT: TYPE VARLIST
;
VARLIST: VAR ',' VARLIST
| VAR
;
ASSIGNMENT: VAR '=' EXPR{
strcpy(QUAD[Index].op,"=");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,$1);
strcpy($$,QUAD[Index++].result);
}
;
```

```
EXPR: EXPR '+' EXPR {AddQuadruple("+",$1,$3,$$);}
| EXPR '-' EXPR {AddQuadruple("-",$1,$3,$$);}
| EXPR '*' EXPR {AddQuadruple("*",$1,$3,$$);}
| EXPR '/' EXPR {AddQuadruple("/",$1,$3,$$);}
| '-' EXPR {AddQuadruple("UMIN",$2,"",$$);}
| '(' EXPR ')' {strcpy($$,$2);}
| VAR
| NUM
;
CONDST: IFST{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
| IFST ELSEST
;
IFST: IF '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
};
ELSEST: ELSE{
tInd=pop();
Ind=pop();
push(tInd);
sprintf(QUAD[Ind].result,"%d",Index);
}
BLOCK{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
};
CONDITION: VAR RELOP VAR {AddQuadruple($2,$1,$3,$$);
StNo=Index-1;
}
| VAR
```

```
| NUM
;
WHILEST: WHILELOOP{
Ind=pop();
sprintf(QUAD[Ind].result,"%d",StNo);
Ind=pop();
sprintf(QUAD[Ind].result,"%d",Index);
}
;
WHILELOOP: WHILE '(' CONDITION ')' {
strcpy(QUAD[Index].op,"==");
strcpy(QUAD[Index].arg1,$3);
strcpy(QUAD[Index].arg2,"FALSE");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
BLOCK {
strcpy(QUAD[Index].op,"GOTO");
strcpy(QUAD[Index].arg1,"");
strcpy(QUAD[Index].arg2,"");
strcpy(QUAD[Index].result,"-1");
push(Index);
Index++;
}
;
%%
extern FILE *yyin;
int main()
{
FILE *fp;
int i;
if(1)
{
fp=fopen("test.c","r");
if(!fp)
{
printf("\n File not found");
exit(0);
}
yyin=fp;
}
yyparse();
printf("\n\n\t\t ----------------------------""\n\t\t Pos Operator Arg1 Arg2 Result" "\n\t\t-
------------------");
for(i=0;i<Index;i++)
{
```

```
printf("\n\t\t %d\t %s\t %s\t
%s\t%s",i,QUAD[i].op,QUAD[i].arg1,QUAD[i].arg2,QUAD[i].result);
}
printf("\n\t\t ----------------------");
printf("\n\n");
return 0;
}
void push(int data)
{
stk.top++;
if(stk.top==100)
{
printf("\n Stack overflow\n");
exit(0);
}
stk.items[stk.top]=data;
}
int pop()
{
int data;
if(stk.top==-1)
{
printf("\n Stack underflow\n");
exit(0);
}
data=stk.items[stk.top--];
return data;
}
void AddQuadruple(char op[5],char arg1[10],char arg2[10],char result[10])
{
strcpy(QUAD[Index].op,op);
strcpy(QUAD[Index].arg1,arg1);
strcpy(QUAD[Index].arg2,arg2);
sprintf(QUAD[Index].result,"t%d",tIndex++);
strcpy(result,QUAD[Index++].result);
}
yyerror()
{
printf("\n Error on line no:%d",LineNo);
}
```

Test.c

```
main()
{
int a,b,c;
if(a<b)
```

```
{
a=a+b;
}
while(a<b)
{
a=a+b;
}
if(a<=b)
{
c=a-b;
}
else
{
c=a+b;
}
}
```

Output:

```
ubuntu@DESKTOP-FHDVQ08:~$ nano bnf.l
ubuntu@DESKTOP-FHDVQ08:~$ nano bnf.y
ubuntu@DESKTOP-FHDVQ08:~$ nano test.c
ubuntu@DESKTOP-FHDVQ08:~$ lex bnf.l
ubuntu@DESKTOP-FHDVQ08:~$ yacc -d bnf.y
bnf.y:19 parser name defined to default :"parse"
bnf.y:36:  warning:  type clash ('' 'var') on default action
bnf.y:42:  warning:  type clash ('' 'var') on default action
bnf.y:43:  warning:  type clash ('' 'var') on default action
bnf.y:68:  warning:  type clash ('' 'var') on default action
ubuntu@DESKTOP-FHDVQ08:~$ gcc lex.yy.c y.tab.c -w
ubuntu@DESKTOP-FHDVQ08:~$ ./a.out
```

```
----------------------------
Pos Operator Arg1 Arg2 Result
--------------------
0     <     a     b     t0
1     ==     t0     FALSE  5
2     +     a     b     t1
3     =     t1          a
4     GOTO               5
5     <     a     b     t2
6     ==     t2     FALSE  10
7     +     a     b     t3
8     =     t3          a
9     GOTO               5
10    <=     a     b     t4
11    ==     t4     FALSE  15
12    -     a     b     t5
13    =     t5          c
14    GOTO               17
15    +     a     b     t6
16    =     t6          c
----------------------
```

[Viva Questions]

1. State different forms of Three address statements.
2. What are different intermediate code forms?