NEHRU COLLEGE OF ENGINEERING AND RESEARCH CENTRE

(Accredited by NAAC, ISO 9001-2015 Certified Institution *Approved by AICTE New Delhi, Affiliated to APJKTU)*

Pampady, Thiruvilwamala(PO), Thrissur(DT), Kerala 680 588

DEPARTMENT OF COMPUTER SCEINCE AND ENGINEERING

LAB MANUEL

CS431 COMPILER DESIGN LAB

VISION OF THE INSTITUTION

To mould true citizens who are millennium leaders and catalysts of change through excellence in education.

MISSION OF THE INSTITUTION

NCERC is committed to transform itself into a center of excellence in Learning and Research in Engineering and Frontier Technology and to impart quality education to mould technically competent citizens with moral integrity, social commitment and ethical values.

We intend to facilitate our students to assimilate the latest technological know-how and to imbibe discipline, culture and spiritually, and to mould them in to technological giants, dedicated research scientists and intellectual leaders of the country who can spread the beams of light and happiness among the poor and the underprivileged.

ABOUT DEPARTMENT

- Established in: 2002

- Course offered : B.Tech in Computer Science and Engineering

  M.Tech in Computer Science and Engineering

  M.Tech in Cyber Security
- Approved by AICTE New Delhi and Accredited by NAAC
- Affiliated to the University of A P J Abdul Kalam Technological University.

DEPARTMENT VISION

Producing Highly Competent, Innovative and Ethical Computer Science and Engineering Professionals to facilitate continuous technological advancement.

DEPARTMENT MISSION

1. To Impart Quality Education by creative Teaching Learning Process
2. To Promote cutting-edge Research and Development Process to solve real world problems with emerging technologies.
3. To Inculcate Entrepreneurship Skills among Students.
4. To cultivate Moral and Ethical Values in their Profession.

PROGRAMME EDUCATIONAL OBJECTIVES

PEO1. Graduates will be able to Work and Contribute in the domains of Computer Science and Engineering through lifelong learning.

PEO2: Graduates will be able to analyse, design and development of novel Software Packages Web Services, System Tools and Components as per needs and specifications.

PEO3: Graduates will be able to demonstrate their ability to adapt to a rapidly changing environment by learning and applying new technologies.

PEO4: Graduates will be able to adopt ethical attitudes, exhibit effective communication skills, teamwork and leadership qualities.

PROGRAM OUTCOMES (POS)

Engineering Graduates will be able to:

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 multidisciplinary 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.

PROGRAM SPECIFIC OUTCOMES (PSO)

PSO1: Ability to Formulate and Simulate Innovative Ideas to provide software solutions for Real-time Problems and to investigate for its future scope.

PSO2: Ability to learn and apply various methodologies for facilitating development of high quality System Software Tools and Efficient Web Design Models with a focus on performance optimization.

PSO3: Ability to inculcate the Knowledge for developing Codes and integrating hardware/software products in the domains of Big Data Analytics, Web Applications and Mobile Apps to create innovative career path and for the socially relevant issues.

COURSE OUTCOME

C431.1	To Implement Lexical Analyser for a given language.
C431.2	To Apply the knowledge of Lex and Yacc tools to develop programs.
C431.3	To Develop different parsers for a given language.
C431.4	To Apply code optimization techniques for programs.
C431.5	To Demonstrate intermediate and machine level code generation for programs.
C431.6	To generate machine level code.

CO'S	PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8	PO9	PO10	PO11	PO12
C431.1	3	2	3	2	3	-	-	3	3	3	-	3
C431.2	3	3	3	-	3	-	-	3	3	3	-	3
C431.3	3	3	3	3	2	-	-	3	3	3	-	3
C431.4	3	-	2	-	3	-	-	3	3	3	-	3
C431.5	3	2	2	-	3	-	-	3	3	3	-	3
C431.6	3	2	2	-	3	-	-	3	3	3	-	3
C431	3	2.4	2.5	2.5	2.83			3	3	3		3

CO'S	PSO1	PSO2	PSO3
C431.1	3	2	-
C431.2	3	2	-
C431.3	3	2	-
C431.4	3	2	-
C431.5	3	2	-
C431.6	3	2	-
C431	3	2	-

Note: Highly correlated=3, Medium correlated=2, Less correlated=1

PREPARATION FOR THE LABORATORY SESSION
GENERAL INSTRUCTIONS TO STUDENTS

1. You should come prepared for your lab session properly to utilize the maximum time of lab session

2. You should attempt all lab experiments given in the list

3. You may seek assistance in doing the lab experiments from the available lab Instructor.

4. There should be proper comments for description of the problem, requirement of class, function etc. Proper comments are to be provided as and when necessary in the programming. This will develop a good practice in you for the rest of your professional life
5. Your program should be interactive and properly documented with real input/output data.
6. Proper management of file of lab record is necessary. Completed lab experiments should be submitted in the form of lab records

7. Maintain silence, order and discipline inside the lab. Don't use cell phones inside the laboratory

AFTER THE LABORATORY SESSION

1 Shut down the system before you leave.

2. Arrange your chair.

3. Make sure you understand what kind of report is to be prepared and due submission of record is next lab class.

MAKE-UPS AND LATE WORK

Students must participate in all laboratory exercises as scheduled. They must obtain permission from the faculty member for absence, which would be granted only under justifiable circumstances. In such an event, a student must make arrangements for a make-up laboratory, which will be scheduled when the time is available after completing one cycle.

LABORATORY POLICIES

1. Food, beverages & mobile phones are not allowed in the laboratory at any time.

2. Do not sit or place anything on instrument benches.

3. Organizing laboratory experiments requires the help of laboratory technicians and staff. Be punctual.

SYLLABUS

Course code	Course Name	L-T-P - Credits	Year of Introduction
CS431	COMPILER DESIGN LAB	0-0-3-1	2016

Pre-requisite : CS331 System Software Lab

Course Objectives:

- To implement the different Phases of compiler.
- To implement and test simple optimization techniques.
- To give exposure to compiler writing tools.

List of Exercises/Experiments :

1. Design and implement a lexical analyzer for given language using C and the lexical analyzer should ignore redundant spaces, tabs and new lines.
2. Implementation of Lexical Analyzer using Lex Tool
3. Generate YACC specification for a few syntactic categories.
   a) Program to recognize a valid arithmetic expression that uses operator +, - , * and /.
   b) Program to recognize a valid variable which starts with a letter followed by any number of letters or digits.
   c) Implementation of Calculator using LEX and YACC
   d) Convert the BNF rules into YACC form and write code to generate abstract syntax tree
4. Write program to find ε - closure of all states of any given NFA with ε transition.
5. Write program to convert NFA with ε transition to NFA without ε transition.
6. Write program to convert NFA to DFA
7. Write program to minimize any given DFA.
8. Develop an operator precedence parser for a given language.
9. Write program to find Simulate First and Follow of any given grammar.
10. Construct a recursive descent parser for an expression.
11. Construct a Shift Reduce Parser for a given language.
12. Write a program to perform loop unrolling.
13. Write a program to perform constant propagation.
14. Implement Intermediate code generation for simple expressions.
15. Implement the back end of the compiler which takes the three address code and produces the 8086 assembly language instructions that can be assembled and run using an 8086 assembler. The target assembly instructions can be simple move, add, sub, jump etc.

Expected Outcome:

The Student will be able to :

i. Implement the techniques of Lexical Analysis and Syntax Analysis.
ii. Apply the knowledge of Lex & Yacc tools to develop programs.
iii. Generate intermediate code.
iv. Implement Optimization techniques and generate machine level code.

INDEX

FINAL VERIFICATION BY THE HOD

EXPERIMENT – 1
IMPLEMENTATION OF LEXICAL ANALYZER USING C

AIM

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

PROGRAM /PROCEDURE

```
#include<stdio.h>
#include<string.h>
void main()
{
        FILE *f1;
        char c;
        char str[20];
        int i=0,num,linecount=1f;
        f1=fopen("input.txt","r");
        while((c=getc(f1))!=EOF) // TO READ THE GIVEN FILE
        {
                if(isdigit(c)) // TO RECOGNIZE NUMBERS
                {
                 num=c-48;
                 c=getc(f1);
                 while(isdigit(c))
                 {
                  num=num*10+(c-48);
```

```
   c=getc(f1);
  }
  printf("%d is a number \n",num);
  ungetc(c,f1);
 }
 else if(isalpha(c)) // TO RECOGNIZE KEYWORDS AND IDENTIFIERS
 {
  str[i++]=c;
  c=getc(f1);
  while(isdigit(c)||isalpha(c)||c=='_'||c=='$')
  {
    str[i++]=c;
    c=getc(f1);
  }
  str[i++]='\0';
  if(strcmp("for",str)==0||strcmp("while",str)==0||strcmp("do",str)==0|
|
strcmp("int",str)==0||strcmp("float",str)==0||strcmp("char",str)==0||
strcmp("double",str)==0||strcmp("static",str)==0||
strcmp("switch",str)==0||strcmp("case",str)==0)
    {
          printf("%s is a keyword\n",str);
    }
  else
          printf("%s is a identifier\n",str);
  ungetc(c,f1);
```

```
                i=0;
               }
               else if(c==' '||c=='\t') // TO IGNORE THE SPACE printf("\n");
               {}
               else if(c=='\n') // TO COUNT LINE NUMBER lineno++;
                 linecount++;
               else // TO FIND SPECIAL SYMBOL
                 printf("%c is a special symbol\n",c);
        }
printf("Total no. of lines are: %d\n",linecount);
fclose(f1);
getch();
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

OUTPUT:

```
The numbers in the program are: 10 20
The keywords and identifiers are:
int is a keyword
main is an identifier
int is a keyword
a is an identifier
char is a keyword
ch is an identifier
```

float is a keyword

f is an identifier

Special characters are ( ) { = , ; ; ; }

Total no. of lines are:5

Viva questions

1. The process of forming tokens from an input stream of characters is called
2. Which grammar defines Lexical Syntax?
3. Two Important lexical categories are
4. When expression sum=3+2 is tokenized then what is the token category of 3?
5. What is the output of a lexical analyzer?

EXPERIMENT 2

IMPLEMENTATION OF LEXICAL ANALYZER USING LEX TOOL

AIM

Design and implement a lexical analyzer using Lex Tool

PROGRAM/ PROCEDURE :

```
%{
#include<stdio.h>
%}
delim [\t]
ws {delim}+
letter [A-Za-z]
digit [0-9]
id {letter}({letter}|{digit})*
num {digit}+(\.{digit}+)?(E[+|-]?{digit}+)?
%%
ws {printf("no action");}
if|else|then {printf("%s is a keyword",yytext);} // TYPE 32 KEYWORDS
{id} {printf("%s is a identifier",yytext);}
{num} {printf("%s is a number",yytext);}
"<" {printf("It is a relational operator less than");}
"<=" {printf("It is a relational operator less than or equal");}
">" {printf("It is a relational operator greater than");}
">=" {printf("It is a relational operator greater than or equal");}
```

```
"==" {printf("It is a relational operator equal");}
"<>" {printf("It is a relational operator not equal");}
(.)* {printf("Unacceptable\n");}
%%
int yywrap()
{
return 0;
}
int main()
{
printf("\n Enter String:");
yylex();
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

OUTPUT

```
lexlexicalfile.l
cc lex.yy.c -ll
if
if is a keyword
```

num

num is a identifier

254

It is a number

<>

it is a relational operator not equal

Viva questions

1. The process of forming tokens from an input stream of characters is called
2. Which grammar defines Lexical Syntax?
3. Two Important lexical categories are
4. When expression sum=3+2 is tokenized then what is the token category of 3?
5. What is the output of a lexical analyzer?

EXPERIMENT – 3

PROGRAM TO RECOGNIZE A VALID ARITHMETIC EXPRESSION

AIM

Program to recognize a valid arithmetic expression that uses operators +, -, * and /.

PROGRAM/PROCEDURE

Lex Program:

```
%{
#include "y.tab.h"
%}
%%
[0-9]+ { return NUMBER; }
[_a-zA-Z][_a-zA-Z0-9]* { return ID; }
\n { return 0; }
. { return yytext[0]; }
%%
```

Yacc Program:

```
%{
#include<stdio.h>
#include<stdlib.h>
%}
%token NUMBER ID
%left '+''-''*''/'
%%
```

```
exp : exp'+'exp
| exp'-'exp
| exp'*'exp
| exp'/'exp
| '('exp')'
| NUMBER
| ID ;
%%
int main(int argc, char *argv[]) {
printf("Enter the expression: ");
yyparse();
printf("Valid Expression!\n");
return 0;
}
int yyerror() {
printf("Invalid Expression!\n");
exit(1);
}
int yywrap() {
return 1;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Execution:

lex exp3.l

yacc -d exp3.y

gcc lex.yy.c y.tab.c

Output:

Test Case #1: Valid Expression

Enter the expression: 12+23-8

Valid Expression!

Test Case #2: Invalid Expression

Enter the expression: a+*

Invalid Expression!

Viva questions

1. Explain lex and yacc tools:-
2. Give the structure of the lex program
3. Explain yytext
4. Explain yyleng?
5. Why we have to include 'y.tab.h' in lex?

EXPERIMENT -4

PROGRAM TO RECOGNIZE A VALID VARIABLE WHICH STARTS WITH A LETTER FOLLOWED BY ANY NUMBER OF LETTERS OR DIGITS

AIM

Program to recognize a valid variable, which starts with a letter, followed by any number of letters or digits.

PROGRAM/ PROCEDURE:

Lex Program:

```
%{
#include "y.tab.h"
%}
%%
[0-9] { return DIGIT; }
[a-zA-Z] { return ALPHA; }
\n { return 0; }
. { return yytext[0]; }
%%
```

Yacc Program:

```
%{
#include <stdio.h>
#include <stdlib.h>
%}
%token DIGIT ALPHA
%%
var : ALPHA
| var ALPHA
| var DIGIT ;
```

```
%%
int main(int argc, char *argv[]) {
printf("Enter a variable name: ");
yyparse();
printf("Valid Variable!\n");
return 0;
}
int yyerror() {
printf("Invalid Variable!\n");
exit(1);
}
int yywrap() {
return 1;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Execution:

```
lex exp4.l
yacc -d exp4.y
gcc lex.yy.c y.tab.c
```

Output:

Test Case #1: Valid Variable

```
Enter a variable name: a123
```

Valid Variable!

Test Case #2: Invalid Variable

Enter a variable name: 12aa

Invalid Variable!

Viva Questions

1. Explain lex and yacc tools:-
2. Give the structure of the lex program
3. Explain yytext
4. Explain yyleng?
5. Why we have to include 'y.tab.h' in lex?

EXPERIMENT -5
IMPLEMENTATION OF CALCULATOR USING LEX AND YACC

AIM

Implementation of calculator using lex and yacc

.

PROGRAM/PROCEDURE:

Lex Program:

```
%{
#include "y.tab.h"
#include<stdio.h>
extern int yylval;
%}
%%

[0-9]+ {yylval=atoi(yytext); return NUMBER;}
. {return yytext[0];}
[\t]+ ;
\n {return 0;}
%%
```

Yacc Program:

```
%{
#include<stdio.h>
%}
%token NUMBER
%left '+' '-'
%left '*' '/'
%%
st: exp {printf("sum::%d",$$);}
```

```
   ;
exp: exp '+' exp {$$ = $1 + $3;}
   |exp '-' exp {$$ = $1 - $3;}
   |exp '*' exp {$$ = $1 * $3;}
   |exp '/' exp {$$ = $1 / $3;}
   |'('exp')' {$$ = $2;}
   |NUMBER {$$ = $1;}
   ;
%%
int main()
{
      yyparse();
      return 0;
}
yyerror(char *s)
{
      printf("error:%s",s);
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Execution:

lex exp5.l

yacc -d exp5.y

gcc lex.yy.c y.tab.c

Output:

Enter the expression(5+2)*(3-1)/(2)

Answer=7

Viva Questions

1. Explain lex and yacc tools:-
2. The yacc produced by parser is
3. What is lexical analyzer?
4. Explain yyleng?
5. Differentiate the commands cp and mv?

EXPERIMENT – 6
CONVERSION OF NFA TO DFA

AIM

Program to convert nfa to dfa

PROGRAM/PROCEDURE:

```
// C Program to illustrate how to convert e-nfa to DFA

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_LEN 100

char NFA_FILE[MAX_LEN];
char buffer[MAX_LEN];
int zz = 0;

// Structure to store DFA states and their
// status ( i.e new entry or already present)
struct DFA {
  char *states;
  int count;
} dfa;

int last_index = 0;
FILE *fp;
int symbols;

/* reset the hash map*/
```

```
void reset(int ar[], int size) {
  int i;

  // reset all the values of
  // the mapping array to zero
  for (i = 0; i < size; i++) {
    ar[i] = 0;
  }
}

// Check which States are present in the e-closure

/* map the states of NFA to a hash set*/
void check(int ar[], char S[]) {
  int i, j;

  // To parse the individual states of NFA
  int len = strlen(S);
  for (i = 0; i < len; i++) {

    // Set hash map for the position
    // of the states which is found
    j = ((int)(S[i]) - 65);
    ar[j]++;
  }
```

```
}

// To find new Closure States
void state(int ar[], int size, char S[]) {
 int j, k = 0;

 // Combine multiple states of NFA
 // to create new states of DFA
 for (j = 0; j < size; j++) {
  if (ar[j] != 0)
   S[k++] = (char)(65 + j);
 }

 // mark the end of the state
 S[k] = '\0';
}

// To pick the next closure from closure set
int closure(int ar[], int size) {
 int i;

 // check new closure is present or not
 for (i = 0; i < size; i++) {
  if (ar[i] == 1)
   return i;
```

```
 }
 return (100);
}

// Check new DFA states can be
// entered in DFA table or not
int indexing(struct DFA *dfa) {
 int i;

 for (i = 0; i < last_index; i++) {
  if (dfa[i].count == 0)
   return 1;
 }
 return -1;
}

/* To Display epsilon closure*/
void Display_closure(int states, int closure_ar[],
                char *closure_table[],
                char *NFA_TABLE[][symbols + 1],
                char *DFA_TABLE[][symbols]) {
 int i;
 for (i = 0; i < states; i++) {
  reset(closure_ar, states);
  closure_ar[i] = 2;
```

```
  // to neglect blank entry
  if (strcmp(&NFA_TABLE[i][symbols], "-") != 0) {

   // copy the NFA transition state to buffer
   strcpy(buffer, &NFA_TABLE[i][symbols]);
   check(closure_ar, buffer);
   int z = closure(closure_ar, states);

   // till closure get completely saturated
   while (z != 100)
   {
    if (strcmp(&NFA_TABLE[z][symbols], "-") != 0) {
     strcpy(buffer, &NFA_TABLE[z][symbols]);

     // call the check function
     check(closure_ar, buffer);
    }
    closure_ar[z]++;
    z = closure(closure_ar, states);
   }
  }

  // print the e closure for every states of NFA
  printf("\n e-Closure (%c) :\t", (char)(65 + i));
```

```
    bzero((void *)buffer, MAX_LEN);
    state(closure_ar, states, buffer);
    strcpy(&closure_table[i], buffer);
    printf("%s\n", &closure_table[i]);
  }
}

/* To check New States in DFA */
int new_states(struct DFA *dfa, char S[]) {

  int i;

  // To check the current state is already
  // being used as a DFA state or not in
  // DFA transition table
  for (i = 0; i < last_index; i++) {
    if (strcmp(&dfa[i].states, S) == 0)
      return 0;
  }

  // push the new
  strcpy(&dfa[last_index++].states, S);

  // set the count for new states entered
```

```
  // to zero
  dfa[last_index - 1].count = 0;
  return 1;
}

// Transition function from NFA to DFA
// (generally union of closure operation )
void trans(char S[], int M, char *clsr_t[], int st,
           char *NFT[][symbols + 1], char TB[]) {
  int len = strlen(S);
  int i, j, k, g;
  int arr[st];
  int sz;
  reset(arr, st);
  char temp[MAX_LEN], temp2[MAX_LEN];
  char *buff;

  // Transition function from NFA to DFA
  for (i = 0; i < len; i++) {

    j = ((int)(S[i] - 65));
    strcpy(temp, &NFT[j][M]);

    if (strcmp(temp, "-") != 0) {
      sz = strlen(temp);
```

```
    g = 0;


    while (g < sz) {
     k = ((int)(temp[g] - 65));
     strcpy(temp2, &clsr_t[k]);
     check(arr, temp2);
     g++;
    }
  }
 }


 bzero((void *)temp, MAX_LEN);
 state(arr, st, temp);
 if (temp[0] != '\0') {
  strcpy(TB, temp);
 } else
  strcpy(TB, "-");
}


/* Display DFA transition state table*/
void Display_DFA(int last_index, struct DFA *dfa_states,
            char *DFA_TABLE[][symbols]) {
 int i, j;
 printf("\n\n*****************************************************\n\n");
 printf("\t\t DFA TRANSITION STATE TABLE \t\t \n\n");
```

```
 printf("\n STATES OF DFA :\t\t");

 for (i = 1; i < last_index; i++)
  printf("%s, ", &dfa_states[i].states);
 printf("\n");
 printf("\n GIVEN SYMBOLS FOR DFA: \t");

 for (i = 0; i < symbols; i++)
  printf("%d, ", i);
 printf("\n\n");
 printf("STATES\t");

 for (i = 0; i < symbols; i++)
  printf("|%d\t", i);
 printf("\n");

 // display the DFA transition state table
 printf("--------+----------------------\n");
 for (i = 0; i < zz; i++) {
  printf("%s\t", &dfa_states[i + 1].states);
  for (j = 0; j < symbols; j++) {
   printf("|%s \t", &DFA_TABLE[i][j]);
  }
  printf("\n");
 }
```

```
}

// Driver Code
int main() {
  int i, j, states;
  char T_buf[MAX_LEN];

  // creating an array dfa structures
  struct DFA *dfa_states = malloc(MAX_LEN * (sizeof(dfa)));
  states = 6, symbols = 2;

  printf("\n STATES OF NFA :\t\t");
  for (i = 0; i < states; i++)

    printf("%c, ", (char)(65 + i));
  printf("\n");
  printf("\n GIVEN SYMBOLS FOR NFA: \t");

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

    printf("%d, ", i);
  printf("eps");
  printf("\n\n");
  char *NFA_TABLE[states][symbols + 1];
```

```
// Hard coded input for NFA table
char *DFA_TABLE[MAX_LEN][symbols];
strcpy(&NFA_TABLE[0][0], "FC");
strcpy(&NFA_TABLE[0][1], "-");
strcpy(&NFA_TABLE[0][2], "BF");
strcpy(&NFA_TABLE[1][0], "-");
strcpy(&NFA_TABLE[1][1], "C");
strcpy(&NFA_TABLE[1][2], "-");
strcpy(&NFA_TABLE[2][0], "-");
strcpy(&NFA_TABLE[2][1], "-");
strcpy(&NFA_TABLE[2][2], "D");
strcpy(&NFA_TABLE[3][0], "E");
strcpy(&NFA_TABLE[3][1], "A");
strcpy(&NFA_TABLE[3][2], "-");
strcpy(&NFA_TABLE[4][0], "A");
strcpy(&NFA_TABLE[4][1], "-");
strcpy(&NFA_TABLE[4][2], "BF");
strcpy(&NFA_TABLE[5][0], "-");
strcpy(&NFA_TABLE[5][1], "-");
strcpy(&NFA_TABLE[5][2], "-");
printf("\n NFA STATE TRANSITION TABLE \n\n\n");
printf("STATES\t");

for (i = 0; i < symbols; i++)
  printf("|%d\t", i);
```

```
 printf("eps\n");

 // Displaying the matrix of NFA transition table
 printf("--------+-------------------------------------\n");
 for (i = 0; i < states; i++) {
  printf("%c\t", (char)(65 + i));

  for (j = 0; j <= symbols; j++) {
   printf("|%s \t", &NFA_TABLE[i][j]);
  }
  printf("\n");
 }
 int closure_ar[states];
 char *closure_table[states];

 Display_closure(states, closure_ar, closure_table, NFA_TABLE, DFA_TABLE);
 strcpy(&dfa_states[last_index++].states, "-");

 dfa_states[last_index - 1].count = 1;
 bzero((void *)buffer, MAX_LEN);

 strcpy(buffer, &closure_table[0]);
 strcpy(&dfa_states[last_index++].states, buffer);

 int Sm = 1, ind = 1;
```

```
 int start_index = 1;


 // Filling up the DFA table with transition values
 // Till new states can be entered in DFA table
 while (ind != -1) {
  dfa_states[start_index].count = 1;
  Sm = 0;
  for (i = 0; i < symbols; i++) {


   trans(buffer, i, closure_table, states, NFA_TABLE, T_buf);


   // storing the new DFA state in buffer
   strcpy(&DFA_TABLE[zz][i], T_buf);


   // parameter to control new states
   Sm = Sm + new_states(dfa_states, T_buf);
  }
  ind = indexing(dfa_states);
  if (ind != -1)
   strcpy(buffer, &dfa_states[++start_index].states);
  zz++;
 }
 // display the DFA TABLE
 Display_DFA(last_index, dfa_states, DFA_TABLE);
```

```
 return 0;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:

Test case 1:

```
STATES OF NFA :          A, B, C, D, E, F,

GIVEN SYMBOLS FOR NFA: 0, 1, eps

NFA STATE TRANSITION TABLE

STATES    |0    |1    eps
--------+-------------------------------------
A    |FC   |-    |BF
B    |-    |C    |-
C    |-    |-    |D
D    |E    |A    |-
E    |A    |-    |BF
F    |-    |-    |-
```

e-Closure (A) : ABCF

e-Closure (B) : BC

e-Closure (C) : C

e-Closure (D) : D

e-Closure (E) : ABCEF

e-Closure (F) : ABCF

****

DFA TRANSITION STATE TABLE

STATES OF DFA : ABCF, C,

GIVEN SYMBOLS FOR DFA: 0, 1,

STATES |0 |1

--------+----------

ABCF |ABCF |C

C |- |-

Test case 2:

Input : 6
2
FC - BF
- C -
- - D
E A -
A - BF
- - -

STATES OF NFA : A, B, C, D, E, F,

GIVEN SYMBOLS FOR NFA: 0, 1, eps

NFA STATE TRANSITION TABLE

STATES |0 |1 eps
--------+-----------------------------------
A |FC |- |BF
B |- |C |-
C |- |- |D
D |E |A |-
E |A |- |BF
F |- |- |-

e-Closure (A) : ABF

e-Closure (B) : B

e-Closure (C) : CD

e-Closure (D) : D

e-Closure (E) : BEF

e-Closure (F) : F

****

DFA TRANSITION STATE TABLE

STATES OF DFA : ABF, CDF, CD, BEF,

GIVEN SYMBOLS FOR DFA: 0, 1,

STATES	0	1
ABF	CDF	CD
CDF	BEF	ABF
CD	BEF	ABF
BEF	ABF	CD

Test case 3:

Input :

```
9
2
- - BH
- - CE
D - -
- - G
- F -
- - G
- - BH
I - -
- -  -
```

STATES OF NFA : A, B, C, D, E, F, G, H, I,

GIVEN SYMBOLS FOR NFA: 0, 1, eps

NFA STATE TRANSITION TABLE

```
STATES   |0   |1   eps
--------+-----------------------------------
A   |-    |-    |BH
B   |-    |-    |CE
C   |D    |-    |-
D   |-    |-    |G
E   |-    |F    |-
F   |-    |-    |G
G   |-    |-    |BH
H   |I    |-    |-
I   |-    |-    |-
```

e-Closure (A) : ABCEH

e-Closure (B) : BCE

e-Closure (C) : C

e-Closure (D) : BCDEGH

e-Closure (E) : E

e-Closure (F) : BCEFGH

e-Closure (G) : BCEGH

e-Closure (H) : H

e-Closure (I) : I

****

DFA TRANSITION STATE TABLE

STATES OF DFA : ABCEH, BCDEGHI, BCEFGH,

GIVEN SYMBOLS FOR DFA: 0, 1,

```
STATES   |0   |1
--------+----------------------
ABCEH   |BCDEGHI   |BCEFGH
BCDEGHI   |BCDEGHI   |BCEFGH
BCEFGH   |BCDEGHI   |BCEFGH
```

Viva questions

1. An automaton that presents output based on previous state or current input
2. If NFA of 6 states excluding the initial state is converted into DFA, maximum possible number of states for the DFA is ?
3. If n is the length of Input string and m is the number of nodes, the running time of DFA is x that of NFA.Find x?
4. The construction time for DFA from an equivalent NFA (m number of node)is:

EXPERIMENT NO 7
DFA MINIMIZATION

AIM

Program to minimize any given dfa.

PROGRAM/PROCEDURE:

```
#include <stdio.h>
#include <string.h>
#define STATES  99
#define SYMBOLS 20

int N_symbols;  /* number of input symbols */
int N_DFA_states;   /* number of DFA states */
char *DFA_finals;   /* final-state string */
int DFAtab[STATES][SYMBOLS];

char StateName[STATES][STATES+1];   /* state-name table */

int N_optDFA_states;    /* number of optimized DFA states */
int OptDFA[STATES][SYMBOLS];
char NEW_finals[STATES+1];

/*
   Print state-transition table.
   State names: 'A', 'B', 'C', ...
*/
void print_dfa_table(
   int tab[][SYMBOLS], /* DFA table */
   int nstates,    /* number of states */
   int nsymbols,   /* number of input symbols */
```

```
    char *finals)
{
    int i, j;

    puts("\nDFA: STATE TRANSITION TABLE");

    /* input symbols: '0', '1', ... */
    printf("     | ");
    for (i = 0; i < nsymbols; i++) printf("  %c  ", '0'+i);

    printf("\n-----+--");
    for (i = 0; i < nsymbols; i++) printf("-----");
    printf("\n");

    for (i = 0; i < nstates; i++) {
        printf("  %c  | ", 'A'+i);  /* state */
        for (j = 0; j < nsymbols; j++)
            printf("  %c  ", tab[i][j]);    /* next state */
        printf("\n");
    }
    printf("Final states = %s\n", finals);
}

/*
    Initialize NFA table.
```

```
*/
void load_DFA_table()
{

   DFAtab[0][0] = 'B'; DFAtab[0][1] = 'C';
   DFAtab[1][0] = 'E'; DFAtab[1][1] = 'F';
   DFAtab[2][0] = 'A'; DFAtab[2][1] = 'A';
   DFAtab[3][0] = 'F'; DFAtab[3][1] = 'E';
   DFAtab[4][0] = 'D'; DFAtab[4][1] = 'F';
   DFAtab[5][0] = 'D'; DFAtab[5][1] = 'E';

   DFA_finals = "EF";
   N_DFA_states = 6;
   N_symbols = 2;
}

/*
   Get next-state string for current-state string.
*/
void get_next_state(char *nextstates, char *cur_states,
   int dfa[STATES][SYMBOLS], int symbol)
{
   int i, ch;

   for (i = 0; i < strlen(cur_states); i++)
```

```
        *nextstates++ = dfa[cur_states[i]-'A'][symbol];
    *nextstates = '\0';
}

/*
    Get index of the equivalence states for state 'ch'.
    Equiv. class id's are '0', '1', '2', ...
*/
char equiv_class_ndx(char ch, char stnt[][STATES+1], int n)
{
    int i;

    for (i = 0; i < n; i++)
        if (strchr(stnt[i], ch)) return i+'0';
    return -1;  /* next state is NOT defined */
}

/*
    Check if all the next states belongs to same equivalence class.
    Return value:
        If next state is NOT unique, return 0.
        If next state is unique, return next state --> 'A/B/C/...'
    's' is a '0/1' string: state-id's
*/
char is_one_nextstate(char *s)
```

```
{
   char equiv_class;   /* first equiv. class */

   while (*s == '@') s++;
   equiv_class = *s++; /* index of equiv. class */

   while (*s) {
      if (*s != '@' && *s != equiv_class) return 0;
      s++;
   }

   return equiv_class; /* next state: char type */
}

int state_index(char *state, char stnt[][STATES+1], int n, int *pn,
   int cur)    /* 'cur' is added only for 'printf()' */
{
   int i;
   char state_flags[STATES+1]; /* next state info. */

   if (!*state) return -1; /* no next state */

   for (i = 0; i < strlen(state); i++)
      state_flags[i] = equiv_class_ndx(state[i], stnt, n);
   state_flags[i] = '\0';
```

```
    printf("   %d:[%s]\t--> [%s] (%s)\n",
        cur, stnt[cur], state, state_flags);

    if (i=is_one_nextstate(state_flags))
        return i-'0';   /* deterministic next states */
    else {
        strcpy(stnt[*pn], state_flags); /* state-division info */
        return (*pn)++;
    }
}

/*
    Divide DFA states into finals and non-finals.
*/
int init_equiv_class(char statename[][STATES+1], int n, char *finals)
{
    int i, j;

    if (strlen(finals) == n) {  /* all states are final states */
        strcpy(statename[0], finals);
        return 1;
    }

    strcpy(statename[1], finals);   /* final state group */
```

```
    for (i=j=0; i < n; i++) {
        if (i == *finals-'A') {
            finals++;
        } else statename[0][j++] = i+'A';
    }
    statename[0][j] = '\0';

    return 2;
}

/*
    Get optimized DFA 'newdfa' for equiv. class 'stnt'.
*/
int get_optimized_DFA(char stnt[][STATES+1], int n,
    int dfa[][SYMBOLS], int n_sym, int newdfa[][SYMBOLS])
{
    int n2=n;       /* 'n' + <num. of state-division info> */
    int i, j;
    char nextstate[STATES+1];

    for (i = 0; i < n; i++) {    /* for each pseudo-DFA state */
        for (j = 0; j < n_sym; j++) {    /* for each input symbol */
            get_next_state(nextstate, stnt[i], dfa, j);
            newdfa[i][j] = state_index(nextstate, stnt, n, &n2, i)+'A';
```

```
        }
    }

    return n2;
}

/*
    char 'ch' is appended at the end of 's'.
*/
void chr_append(char *s, char ch)
{
    int n=strlen(s);

    *(s+n) = ch;
    *(s+n+1) = '\0';
}

void sort(char stnt[][STATES+1], int n)
{
    int i, j;
    char temp[STATES+1];

    for (i = 0; i < n-1; i++)
        for (j = i+1; j < n; j++)
            if (stnt[i][0] > stnt[j][0]) {
```

```
            strcpy(temp, stnt[i]);
            strcpy(stnt[i], stnt[j]);
            strcpy(stnt[j], temp);
          }
}

/*
   Divide first equivalent class into subclasses.
      stnt[i1] : equiv. class to be segmented
      stnt[i2] : equiv. vector for next state of stnt[i1]
   Algorithm:
      - stnt[i1] is splitted into 2 or more classes 's1/s2/...'
      - old equiv. classes are NOT changed, except stnt[i1]
      - stnt[i1]=s1, stnt[n]=s2, stnt[n+1]=s3, ...
   Return value: number of NEW equiv. classses in 'stnt'.
*/
int split_equiv_class(char stnt[][STATES+1],
   int i1, /* index of 'i1'-th equiv. class */
   int i2, /* index of equiv. vector for 'i1'-th class */
   int n,  /* number of entries in 'stnt' */
   int n_dfa)  /* number of source DFA entries */
{
   char *old=stnt[i1], *vec=stnt[i2];
   int i, n2, flag=0;
   char newstates[STATES][STATES+1];   /* max. 'n' subclasses */
```

```
    for (i=0; i < STATES; i++) newstates[i][0] = '\0';

    for (i=0; vec[i]; i++)
        chr_append(newstates[vec[i]-'0'], old[i]);

    for (i=0, n2=n; i < n_dfa; i++) {
        if (newstates[i][0]) {
            if (!flag) {    /* stnt[i1] = s1 */
                strcpy(stnt[i1], newstates[i]);
                flag = 1;   /* overwrite parent class */
            } else  /* newstate is appended in 'stnt' */
                strcpy(stnt[n2++], newstates[i]);
        }
    }

    sort(stnt, n2); /* sort equiv. classes */

    return n2;  /* number of NEW states(equiv. classes) */
}

/*
    Equiv. classes are segmented and get NEW equiv. classes.
*/
int set_new_equiv_class(char stnt[][STATES+1], int n,
```

```
   int newdfa[][SYMBOLS], int n_sym, int n_dfa)
{
   int i, j, k;

   for (i = 0; i < n; i++) {
      for (j = 0; j < n_sym; j++) {
         k = newdfa[i][j]-'A';   /* index of equiv. vector */
         if (k >= n)  /* equiv. class 'i' should be segmented */
            return split_equiv_class(stnt, i, k, n, n_dfa);
      }
   }

   return n;
}

void print_equiv_classes(char stnt[][STATES+1], int n)
{
   int i;

   printf("\nEQUIV. CLASS CANDIDATE ==>");
   for (i = 0; i < n; i++)
      printf(" %d:[%s]", i, stnt[i]);
   printf("\n");
}
```

```
/*
   State-minimization of DFA: 'dfa' --> 'newdfa'
   Return value: number of DFA states.
*/
int optimize_DFA(
   int dfa[][SYMBOLS], /* DFA state-transition table */
   int n_dfa,  /* number of DFA states */
   int n_sym,  /* number of input symbols */
   char *finals,   /* final states of DFA */
   char stnt[][STATES+1],  /* state name table */
   int newdfa[][SYMBOLS])  /* reduced DFA table */
{
   char nextstate[STATES+1];
   int n;  /* number of new DFA states */
   int n2; /* 'n' + <num. of state-dividing info> */

   n = init_equiv_class(stnt, n_dfa, finals);

   while (1) {
      print_equiv_classes(stnt, n);
      n2 = get_optimized_DFA(stnt, n, dfa, n_sym, newdfa);
      if (n != n2)
         n = set_new_equiv_class(stnt, n, newdfa, n_sym, n_dfa);
      else break; /* equiv. class segmentation ended!!! */
   }
```

```
    return n;   /* number of DFA states */
}

/*
    Check if 't' is a subset of 's'.
*/
int is_subset(char *s, char *t)
{
    int i;

    for (i = 0; *t; i++)
        if (!strchr(s, *t++)) return 0;
    return 1;
}

/*
    New finals states of reduced DFA.
*/
void get_NEW_finals(
    char *newfinals,    /* new DFA finals */
    char *oldfinals,    /* source DFA finals */
    char stnt[][STATES+1],  /* state name table */
    int n)  /* number of states in 'stnt' */
{
```

```
    int i;


    for (i = 0; i < n; i++)
        if (is_subset(oldfinals, stnt[i])) *newfinals++ = i+'A';
    *newfinals++ = '\0';
}


void main()
{
    load_DFA_table();
    print_dfa_table(DFAtab, N_DFA_states, N_symbols, DFA_finals);


    N_optDFA_states = optimize_DFA(DFAtab, N_DFA_states,
            N_symbols, DFA_finals, StateName, OptDFA);
    get_NEW_finals(NEW_finals, DFA_finals, StateName, N_optDFA_states);


    print_dfa_table(OptDFA, N_optDFA_states, N_symbols, NEW_finals);
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

EXPERIMENT NO 8
OPERATOR PRECEDENCE PARSER

AIM

Develop an operator precedence parser for a given language.

PROGRAM/PROCEDURE:

```
#include<stdio.h>
int main(void)

{
char stack[20],ip[20],opt[10][10][1],ter[10];
int i,j,k,n,top=0,col,row;
for(i=0;i<10;i++)
{
 stack[i]=NULL;
 ip[i]=NULL;
 for(j=0;j<10;j++)
 {
 opt[i][j][1]=NULL;
 }
}
printf("Enter the no.of terminals :\n");
scanf("%d",&n);
printf("\nEnter the terminals :\n");
scanf("%s",&ter);
printf("\nEnter the table values :\n");
for(i=0;i<n;i++)
{
 for(j=0;j<n;j++)
 {
```

```
 printf("Enter the value for %c %c:",ter[i],ter[j]);
 scanf("%s",opt[i][j]);
 }
}
printf("\n**** OPERATOR PRECEDENCE TABLE ****\n");
for(i=0;i<n;i++)
{
printf("\t%c",ter[i]);
}
printf("\n");
for(i=0;i<n;i++){printf("\n%c",ter[i]);
for(j=0;j<n;j++){printf("\t%c",opt[i][j][0]);}}
stack[top]='$';
printf("\nEnter the input string:");
scanf("%s",ip);
i=0;
printf("\nSTACK\t\t\tINPUT STRING\t\t\tACTION\n");
printf("\n%s\t\t\t%s\t\t\t",stack,ip);
while(i<=strlen(ip))
{
for(k=0;k<n;k++)
{
if(stack[top]==ter[k])
col=k;
if(ip[i]==ter[k])
```

```
row=k;
}
if((stack[top]=='$')&&(ip[i]=='$')){
printf("String is accepted\n");
break;}
else if((opt[col][row][0]=='<') ||(opt[col][row][0]=='='))
{ stack[++top]=opt[col][row][0];
stack[++top]=ip[i];
 printf("Shift %c",ip[i]);
 i++;
  }
else{
 if(opt[col][row][0]=='>')
{
while(stack[top]!='<'){--top;}
top=top-1;
printf("Reduce");
}
else
{
printf("\nString is not accepted");
break;
}
}
printf("\n");
```

```
for(k=0;k<=top;k++)
{
printf("%c",stack[k]);
}
printf("\t\t\t");
for(k=i;k<strlen(ip);k++){
printf("%c",ip[k]);
}
printf("\t\t\t");
}
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:

```
Enter the no.of terminals:4
Enter the terminals: i+*$
Enter the table values:
Enter the value for * *:>
Enter the value for * $:>
Enter the value for $ i:<
Enter the value for $ +:<
Enter the value for $ *:<
Enter the value for $ $:accept

**** OPERATOR PRECEDENCE TABLE ****
     i     +     *     $

i     e     >     >     >
```

```
+    <    >    <    >
*    <    >    >    >
$    <    <    <    a
*/
Enter the input string:
i*i

STACK          INPUT STRING          ACTION

$              i*i              Shift i
$<i             *i               Reduce
$              *i              Shift *
$<*             i               Shift i
$<*<i
String is not accepted
```

Viva questions

1. What is operatoe precedence parser
2. Which are the three operator precedence relation
3. What is precedence function
4. What is the complexity of operator precedence parser

EXPERIMENT NO 9

FIRST AND FOLLOW

AIM

To write an assembly language program to find square and square root of a number using 8051

PROGRAM/PROCEDURE

a) FIRST

```
#include<stdio.h>
#include<ctype.h>
void FIRST(char[],char );
void result(char[],char);
intnop;
char prod[10][10];
void main()
{
int i;
char choice;
char c;
char res1[20];
clrscr();
printf("How many number of productions ? :");
scanf(" %d",&nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
}
do
{
printf("\n Find the FIRST of :");
```

```
scanf(" %c",&c);
memset(res1,'0',sizeof(res));
FIRST(res1,c);
printf("\n FIRST(%c)= { ",c);
for(i=0;res1[i]!='\0';i++)
printf(" %c ",res1[i]);
printf("}\n");
printf("press 'y' to continue : ");
scanf(" %c",&choice);
}
while(choice=='y'||choice =='Y');
}



void FIRST(char res[],char c)
{
inti,j,k;
char subres[5];
int eps;
subres[0]='\0';
res[0]='\0';
memset(res,'0',sizeof(res));
memset(subres,'0',sizeof(res));
if(!(isupper(c)))
```

```
{
result(res,c);
return ;
}
for(i=0;i<nop;i++)
{
if(prod[i][0]==c)
{
if(prod[i][2]=='$')
result(res,'$');
else
{
j=2;
while(prod[i][j]!='\0')
{
eps=0;
FIRST(subres,prod[i][j]);
for(k=0;subres[k]!='\0';k++)
result(res,subres[k]);
for(k=0;subres[k]!='\0';k++)
if(subres[k]=='$')
{
eps=1;
break;
}
```

```
if(!eps)
break;
j++;
}
}
}
}
return ;
}

void result(char res[],char val)
{
int k;
for(k=0 ;res[k]!='\0';k++)
if(res[k]==val)
return;
res[k]=val;
res[k+1]='\0';
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:

```
How many number of productions ?:8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
```

Enter productions Number 2 : X=+TX

Enter productions Number 3 : X=$

Enter productions Number 4 : T=FY

Enter productions Number 5 : Y=*FY

Enter productions Number 6 : Y=$

Enter productions Number 7 : F=(E)

Enter productions Number 8 : F=i

Find the FIRST of :X

FIRST(X)= { + $ }

press 'y' to continue : Y

Find the FIRST of :F

FIRST(F)= { (i }

press 'y' to continue : Y

Find the FIRST of :Y

FIRST(Y)= { * $ }

press 'y' to continue : Y

Find the FIRST of :E

FIRST(E)= { (i }

press 'y' to continue : Y

Find the FIRST of :T

FIRST(T)= { (i }

press 'y' to continue : N

b) **FOLLOW**

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

```
#include<string.h>
intnop,m=0,p,i=0,j=0;
char prod[10][10],res[10];
void FOLLOW(char c);
void first(char c);
void result(char);
void main()
{
int i;
int choice;
charc,ch;
printf("Enter the no.of productions: ");
scanf("%d", &nop);
printf("enter the production string like E=E+T\n");
for(i=0;i<nop;i++)
{
printf("Enter productions Number %d : ",i+1);
scanf(" %s",prod[i]);
}
do
{
m=0;
memset(res,'0',sizeof(res));
printf("Find FOLLOW of -->");
scanf(" %c",&c);
```

```
FOLLOW(c);
printf("FOLLOW(%c) = { ",c);
for(i=0;i<m;i++)
printf("%c ",res[i]);
printf(" }\n");
printf("Do you want to continue(Press 1 to continue....)?");
scanf("%d%c",&choice,&ch);
}
while(choice==1);
}
void FOLLOW(char c)
{
if(prod[0][0]==c)
result('$');
for(i=0;i<nop;i++)
{
for(j=2;j<strlen(prod[i]);j++)
{
if(prod[i][j]==c)
{
if(prod[i][j+1]!='\0')
first(prod[i][j+1]);
if(prod[i][j+1]=='\0'&&c!=prod[i][0])
FOLLOW(prod[i][0]);
}
```

```
}
}
}
void first(char c)
{
int k;
if(!(isupper(c)))
result(c);
for(k=0;k<nop;k++)
{if(prod[k][0]==c)
{
if(prod[k][2]=='$')
FOLLOW(prod[i][0]);
else if(islower(prod[k][2]))
result(prod[k][2]);
else
first(prod[k][2]);
}
}
}
void result(char c)
{
int i;
for( i=0;i<=m;i++)
if(res[i]==c)
```

```
return;
res[m++]=c;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:

```
Enter the no.of productions: 8
enter the production string like E=E+T
Enter productions Number 1 : E=TX
Enter productions Number 2 : X=+TX
Enter productions Number 3 : X=$
Enter productions Number 4 : T=FY
Enter productions Number 5 : Y=*FY
Enter productions Number 6 : Y=$
Enter productions Number 7 : F=(E)
Enter productions Number 8 : F=i
Find FOLLOW of -->X
FOLLOW(X) = { $ ) }
```

Viva questions

1. What are the rules to find first of an element
2. What are the rules to find follow of an element
3. What is left recursion
4. How to eliminate left recursion

EXPERIMENT NO 10
RECURSIVE DESCENT PARSER

AIM

Construct a recursive descent parser for an expression

PROGRAM/PROCEDURE:

```
#include<stdio.h>
#include<string.h>
char input[10];
int i=0,error=0;
void E();
void T();
voidEprime();
voidTprime();
void F();
void main()
{
clrscr();
printf("Enter an arithmetic expression :\n");
gets(input);
E();
if(strlen(input)==i&&error==0)
printf("\nAccepted..!!!");
else
printf("\nRejected..!!!");
getch();
}
void E()
{
T();
```

```
Eprime();
}
voidEprime()
{
if(input[i]=='+')
{
i++;
T();
Eprime();
}
}
void T()
{
F();
Tprime();
}
voidTprime()
{
if(input[i]=='*')
{
i++;
F();
Tprime();
}
}
```

```
void F()
{
if(input[i]=='(')
{
i++;
E();
if(input[i]==')')
i++;
}
else if(isalpha(input[i]))
{
i++;
while(isalnum(input[i])||input[i]=='_')
i++;
}
else
error=1;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output

1) Enter an arithmetic expression :

sum+month*interest

Accepted..!!!

2) Enter an arithmetic expression :

sum+avg*+interest

Rejected..!!!

Viva questions

1. The grammar A → AA | (A) | e is not suitable for predictive-parsing because the grammar is?
2. Consider the grammar. E → E + n | E × n | n For a sentence n + n × n, the handles in the right-sentential form of the reduction ar
3. What are the classification of recursive descendant parser

EXPERIMENT NO 11
SHIFT REDUCE PARSER

AIM:

Construct a Shift Reduce Parser for a given language.

PROGRAM/PROCEDURE:

```
#include<stdio.h>


#include<string.h>
int k=0,z=0,i=0,j=0,c=0;
char a[16],ac[20],stk[15],act[10];
void check();
void main()
  {


    puts("GRAMMAR is E->E+E \n E->E*E \n E->(E) \n E->id");
    puts("enter input string ");
    fgets(a,16,stdin);
    c=strlen(a);
    strcpy(act,"SHIFT->");
    puts("stack \t input \t action");
    for(k=0,i=0; j<c; k++,i++,j++)
    {
      if(a[j]=='i' && a[j+1]=='d')
        {
          stk[i]=a[j];
          stk[i+1]=a[j+1];
          stk[i+2]='\0';
          a[j]=' ';
          a[j+1]=' ';
          printf("\n$%s\t%s$\t%sid",stk,a,act);
```

```
        check();
      }
     else
      {
        stk[i]=a[j];
        stk[i+1]='\0';
        a[j]=' ';
        printf("\n$%s\t%s$\t%ssymbols",stk,a,act);
        check();
      }
   }

 }
void check()
 {
  strcpy(ac,"REDUCE TO E");
  for(z=0; z<c; z++)
   if(stk[z]=='i' && stk[z+1]=='d')
    {
      stk[z]='E';
      stk[z+1]='\0';
      printf("\n$%s\t%s$\t%s",stk,a,ac);
      j++;
    }
  for(z=0; z<c; z++)
```

```
  if(stk[z]=='E' && stk[z+1]=='+' && stk[z+2]=='E')
   {
    stk[z]='E';
    stk[z+1]='\0';
    stk[z+2]='\0';
    printf("\n$%s\t%s$\t%s",stk,a,ac);
    i=i-2;
   }
 for(z=0; z<c; z++)
  if(stk[z]=='E' && stk[z+1]=='*' && stk[z+2]=='E')
   {
    stk[z]='E';
    stk[z+1]='\0';
    stk[z+1]='\0';
    printf("\n$%s\t%s$\t%s",stk,a,ac);
    i=i-2;
   }
 for(z=0; z<c; z++)
  if(stk[z]=='(' && stk[z+1]=='E' && stk[z+2]==')')
   {
    stk[z]='E';
    stk[z+1]='\0';
    stk[z+1]='\0';
    printf("\n$%s\t%s$\t%s",stk,a,ac);
    i=i-2;
```

```
    }
  }
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output

1)

SHIFT REDUCE PARSER GRAMMER

E->E+E

E->E/E

E->E*E

E->E-E

E->id

enter the input symbol: a+b*c

stack implementation table

stack input symbol action

_____ _____ _____

$ a+b*c$ --

$a +b*c$ shift a

$E +b*c$ E->a

$E+ b*c$ shift +

$E+b *c$ shift b

$E+E *c$ E->b

$E *c$ E->E+E

$E* c$ shift *

$E*c $ shift c

$E*E $ E->c

$E $ E->E*E

$E $ ACCEPT

2) SHIFT REDUCE PARSER GRAMMER

E->E+E

E->E/E

E->E*E

E->E-E

E->id

enter the input symbol: a+b*+c

stack implementation table

stack input symbol action

_____ _____ _____

$ a+b*+c$ --

$a +b*+c$ shift a

$E +b*+c$ E->a

$E+ b*+c$ shift +

$E+b *+c$ shift b

$E+E *+c$ E->b

$E *+c$ E->E+E

$E* +c$ shift *

$E*+ c$ shift +

$E*+c $ shift c

$E*+E $ E->c

$E*+E reject

Viva questions

1. What is a shift step
2. What is reduce step
3. Consider the grammar. E → E + n | E × n | n For a sentence n + n × n, the handles in the right-sentential form of the reduction ar
4. What are the classification of shift reduce parser

EXPERIMENT NO 12
INTERMEDIATE CODE GENERATION FOR

SIMPLE EXPRESSIONS

AIM

Program to perform constant propagation and folding.

PROGRAM/PROCEDURE:

```
#include"stdio.h"
#include"conio.h"
#include"string.h"
int i=1,j=0,no=0,tmpch=90;
char str[100],left[15],right[15];
void findopr();
void explore();
void fleft(int);
void fright(int);
struct exp
{
 int pos;
 char op;
}k[15];
void main()
{
 clrscr();
 printf("\t\tINTERMEDIATE CODE GENERATION\n\n");
```

```
 printf("Enter the Expression :");
 scanf("%s",str);
 printf("The intermediate code:\t\tExpression\n");
 findopr();
 explore();
 getch();
}
void findopr()
{
 for(i=0;str[i]!='\0';i++)
  if(str[i]==':')
  {
  k[j].pos=i;
  k[j++].op=':';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='/')
  {
  k[j].pos=i;
  k[j++].op='/';
  }
 for(i=0;str[i]!='\0';i++)
  if(str[i]=='*')
  {
  k[j].pos=i;
```

```
 k[j++].op='*';
 }
 for(i=0;str[i]!='\0';i++)
 if(str[i]=='+')
 {
 k[j].pos=i;
 k[j++].op='+';
 }
 for(i=0;str[i]!='\0';i++)
 if(str[i]=='-')
 {
 k[j].pos=i;
 k[j++].op='-';
 }
}
void explore()
{
 i=1;
 while(k[i].op!='\0')
 {
 fleft(k[i].pos);
 fright(k[i].pos);
 str[k[i].pos]=tmpch--;
 printf("\t%c := %s%c%s\t\t",str[k[i].pos],left,k[i].op,right);
 for(j=0;j <strlen(str);j++)
```

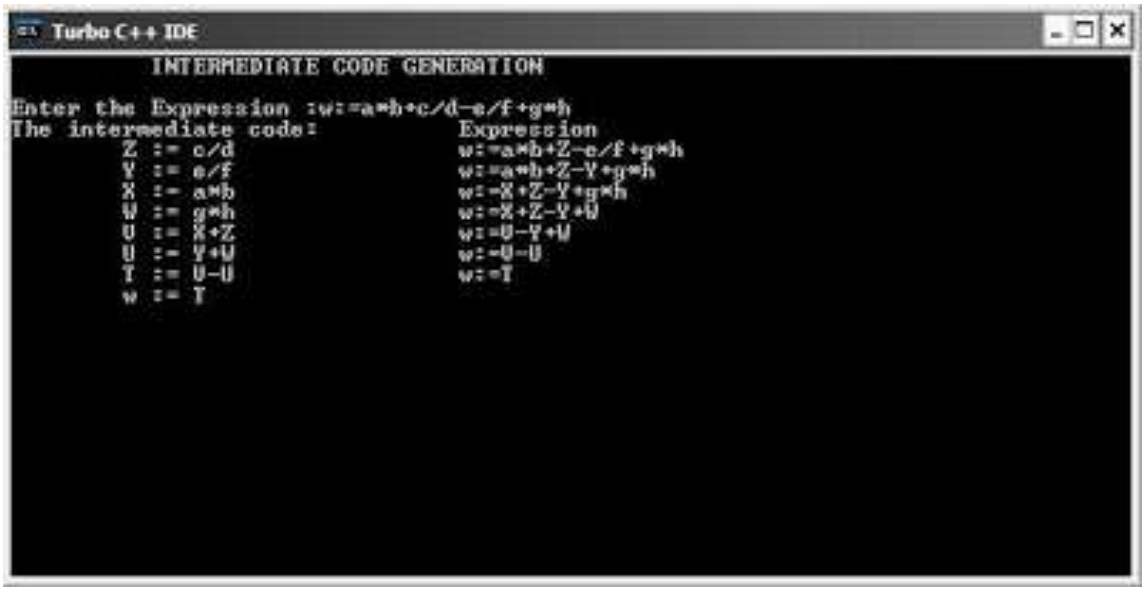
```
  if(str[j]!='$')
   printf("%c",str[j]);
 printf("\n");
 i++;
}
 fright(-1);
 if(no==0)
 {
  fleft(strlen(str));
  printf("\t%s := %s",right,left);
  getch();
  exit(0);
 }
 printf("\t%s :=  %c",right,str[k[--i].pos]);
 getch();
}
void fleft(int x)
{
 int w=0,flag=0;
 x--;
 while(x!= -1 &&str[x]!= '+' &&str[x]!='*'&&str[x]!='='&&str[x]!='\0'&&str[x]!='-
'&&str[x]!='/'&&str[x]!=':')
 {
  if(str[x]!='$'&& flag==0)
  {
```

```
 left[w++]=str[x];
 left[w]='\0';
 str[x]='$';
 flag=1;
 }
 x--;
 }
}
void fright(int x)
{
 int w=0,flag=0;
 x++;
 while(x!= -1 && str[x]!=
'+'&&str[x]!='*'&&str[x]!='\0'&&str[x]!='='&&str[x]!=':'&&str[x]!='-'&&str[x]!='/')
 {
  if(str[x]!='$'&& flag==0)
  {
  right[w++]=str[x];
  right[w]='\0';
  str[x]='$';
  flag=1;
  }
  x++;
 }
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:



Viva questions

1. How ICG helps in generating object code
2. What is 3-addres code
3. What are the type of addressing
4. What are the classification of shift reduce parser
5. What are the properties of optimization compiler

EXPERIMENT NO 13
LOOP UNROLLING

AIM:

Program to perform loop unrolling.

PROGRAM/PROCEDURE:

```
// This program does not uses loop unrolling.
#include<stdio.h>

int main(void)
{
   for (int i=0; i<5; i++)
      printf("Hello\n"); //print hello 5 times

   return 0;
}
```

```
// This program uses loop unrolling.
#include<stdio.h>

int main(void)
{
   // unrolled the for loop in program 1
   printf("Hello\n");
   printf("Hello\n");
```

```
    printf("Hello\n");
    printf("Hello\n");
    printf("Hello\n");

    return 0;
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

Output:

Hello

Hello

Hello

Hello

Hello

Viva questions

1. What is a loop unrolling
2. What are the benefits of loop unrolling in program
3. What kinds of loops can be unrolled

EXPERIMENT NO 14
CONSTANT PROPAGATION AND FOLDING

AIM:

Program to perform constant propagation and folding.

PROGRAM/PROCEDURE:

```
#include<stdio.h>
#include<string.h>
#include<ctype.h>
#include<conio.h>
void input();
void output();
void change(int p,char *res);
void constant();
struct expr
{
char op[2],op1[5],op2[5],res[5];
int flag;
}arr[10];
int n;
void main()
{
clrscr();
input();
constant();
```

```
output();
getch();
}
void input()
{
int i;
printf("\n\nEnter the maximum number of  expressions : ");
scanf("%d",&n);
printf("\nEnter the input : \n");
for(i=0;i<n;i++)
{
scanf("%s",arr[i].op);
scanf("%s",arr[i].op1);
scanf("%s",arr[i].op2);
scanf("%s",arr[i].res);
arr[i].flag=0;
}
}
void constant()
{
int i;
int op1,op2,res;
char op,res1[5];
for(i=0;i<n;i++)
{
```

```
if(isdigit(arr[i].op1[0]) && isdigit(arr[i].op2[0]) || strcmp(arr[i].op,"=")==0) /*if both digits, store them in variables*/
{
op1=atoi(arr[i].op1);
op2=atoi(arr[i].op2);
op=arr[i].op[0];
switch(op)
{
case '+':
res=op1+op2;
break;
case '-':
res=op1-op2;
break;
case '*':
res=op1*op2;
break;
case '/':
res=op1/op2;
break;
case '=':
res=op1;
break;
}
sprintf(res1,"%d",res);
```

```
arr[i].flag=1; /*eliminate expr and replace any operand below that uses result of this expr */
change(i,res1);
}
}
}
void output()
{
int i=0;
printf("\nOptimized code is : ");
for(i=0;i<n;i++)
{
if(!arr[i].flag)
{
printf("\n%s %s %s %s",arr[i].op,arr[i].op1,arr[i].op2,arr[i].res);
}
}
}
void change(int p,char *res){
int i;
for(i=p+1;i<n;i++)
{
if(strcmp(arr[p].res,arr[i].op1)==0)
strcpy(arr[i].op1,res);
else if(strcmp(arr[p].res,arr[i].op2)==0)
```

```
strcpy(arr[i].op2,res);
}
}
```

RESULT AND DISCUSSIONS

The program was executed and the output obtained successfully

INPUT:

Enter the maximum number of expressions : 4

Enter the input :

= 3 - a

+ a b t1

+ a c t2

+ t1 t2 t3

OUTPUT:

Optimized code is :

+ 3 b t1

+ 3 c t2

+ t1 t2 t3

Viva questions

1. What is a constant folding
2. Why we do folding in programs