Lab Report

BCSE307L- Compiler Design Laboratory

Slots: L43+L44

CH2023240501932

School of Computer Science and Engineering

by

Shaahid Ahmed N 21BAI1087

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

April,2024

BCSE307P –COMPILER DESIGN LABORATORY

Slots: L43+L44 CH2023240501932 (Wed- 2.00 pm to 3.45 pm)

Course faculty: Dr.R.Suganya, SCOPE, VIT

Venue: AB1-614

Course Outcomes	Content	Date	Marks
CO1	Construct lexical analyzer for the following programs a. Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, comments, operators etc.). b. Develop a lexical analyzer to construct a DFA in C c. Create a symbol table, while recognizing identifiers.	3.1.24 10.1.24 18.1.24	10
CO2	Lex Tool Implementation a. Implementation of Lexical Analyzer (any one above programs) using Lex Tool Construction of Syntax analyzer a. Implement a c program for LL(1) top down parser for any given LL(1) grammar	24.1.24 31.1.24 7.2.24	10
	CAT I – 10.2.24 – 18.2.24		
CO3	Generate YACC specification for a few syntactic categories a. Program to recognize a valid arithmetic experession that uses operator +, -, * and / b. Program to recognize a valid variable which starts with a letter followed by any number of letter or digits c. Implement an Arithmetic Calculator using LEX and YACC	21.2.24 28.2.24	10
CO4	a. Study of LLVM b. Generate three address code for a simple program using LEX and YACC	6.3.24 13.3.24 20.3.24	10
	CAT II -30.3.24 to 7.4.24		
CO5	a. Construction of DAG b. Implementation of Simple Code Optimization Techniques.	27.3.24 10.4.24	10
	Model exam	17.4.24	10
	FAT Lab	24.4.24	40

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab1 & 10-01-2024

Slot: L43+L44

Questions:

Construct lexical analyzer for the following programs

a. Develop a lexical analyzer to recognize a few patterns in C. (Ex. identifiers, constants, comments, operators etc.).

b. Develop a lexical analyzer to construct a DFA in C

c. Create a symbol table, while recognizing identifiers.

Answers:

a. Code:

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

bool checkoperator(char ch) {
   if (ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '>' || ch == '<' || ch
== '=') {
      return true;
   }
   return false;
}
```

```
bool checkdl(char ch) {
   if (ch == ' ' || ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == ',' || ch
== ';' || ch == '>' ||
       ch == '<' || ch == '=' || ch == '(' || ch == ')' || ch == '[' || ch == ']' || ch
== '{' || ch == '}') {
       return true;
   }
   return false;
}

bool checkidt(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' ||
checkdl(str[0]) == true) {
       return false;
   }
   return true;
}

bool checkkeyword(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, "void") || !strcmp(str, "struct")) {
       return true;
   }
   return false;
}

bool checkint(char* str) {
   int i, len = strlen(str);
   if (len == 0) {
       return false;
```

```
    }
    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 false;
        }
    }
    return true;
}

char* substring(char* str, int start, int end) {
    int i;
    char* substr = (char*)malloc(sizeof(char) * (end - start + 2));
    for (i = start; i <= end; i++) {
        substr[i - start] = str[i];
    }
    substr[end - start + 1] = '\0';
    return substr;
}

void lex(char* str) {
    int start = 0;
    int end = 0;
    int len = strlen(str);

    while (end <= len && start <= end) {
        if (checkdl(str[end]) == false) {
            end++;
        }
        if (checkdl(str[end]) == true && start == end) {
            if (checkoperator(str[end]) == true) {
                printf("Operator: '%c'\n", str[end]);
            }
            end++;
            start = end;
```

```
        } else if (checkdl(str[end]) == true && start != end || (end == len
&& start != end)) {
            char* substr = substring(str, start, end - 1);
            if (checkkeyword(substr) == true) {
                printf("Keyword: '%s'\n", substr);
            } else if (checkint(substr) == true) {
                printf("Integer: '%s'\n", substr);
            } else if (checkidt(substr) == true && checkdl(str[end - 1]) ==
false) {
                printf("Identifier: '%s'\n", substr);
            } else if (checkidt(substr) == false && checkdl(str[end - 1]) ==
false) {
                printf("Not Valid Identifier: '%s'\n", substr);
            }
            start = end;
        }
    }
}

int main() {
    char str[256];  // Assuming a reasonable maximum length for the input
    printf("Enter a C program statement: ");
    fgets(str, sizeof(str), stdin);
    lex(str);
    return 0;
}
```

Screenshot:

```
Enter a C program statement: "int a = b + c*e + 24; "
Identifier: '"int'
Identifier: 'a'
Operator: '='
Identifier: 'b'
Operator: '+'
Identifier: 'c'
Operator: '*'
Identifier: 'e'
Operator: '+'
Integer: '24'
Identifier: '"
'
```

b. Code:

```
#include <stdio.h>

int main() {
   // Initially, the state starts from the initial state q0
   int state = 0;
   // User input string
   char ip_str[10];

   printf("Enter a string to check whether it is valid or not:\n");
   scanf("%s", ip_str); // User entering the input string

   for (int i = 0; ip_str[i] != '\0'; i++) { // Iterating the string by checking
each character
      if (state == 0 && ip_str[i] == 'a')
         state = 1;
      else if (state == 1 && ip_str[i] == 'a')
         state = 1;
      else if (state == 2 && ip_str[i] == 'a')
         state = 3;
      else if (state == 3 && ip_str[i] == 'a')
         state = 3;
      else if (state == 0 && ip_str[i] == 'b')
         state = 3;
      else if (state == 1 && ip_str[i] == 'b')
         state = 2;
      else if (state == 2 && ip_str[i] == 'b')
         state = 2;
      else if (state == 3 && ip_str[i] == 'b')
         state = 3;
      else {
         printf("String invalid");
         return 0;
      }
   }

   // If it stops at the final state, then the string should be accepted
```

```
    if (state == 2) {
        printf("String is accepted");
    } else { // If it stops at some other state other than the final state, then it
should be rejected
        printf("String is not accepted");
    }

    return 0;
}
```

Screenshot:

```
Enter a string to check whether it is valid or not:
aab
String is accepted
```

c. Code:

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

#define MAX_TOKEN_SIZE 100

// Structure to hold symbol information
typedef struct
{
    char name[MAX_TOKEN_SIZE];
    char type[MAX_TOKEN_SIZE];
} Symbol;

// Structure to hold the symbol table
typedef struct
{
    Symbol symbols[MAX_TOKEN_SIZE];
    int count;
} SymbolTable;

// Function to check if a given token is a keyword
int isKeyword(const char *token)
```

```
{
    char keywords[][MAX_TOKEN_SIZE] = {
        "int", "float", "char", "double", "if",
        "else", "for", "while", "return"};
    int numKeywords = sizeof(keywords) / sizeof(keywords[0]);
    for (int i = 0; i < numKeywords; i++)
    {
        if (strcmp(token, keywords[i]) == 0)
        {
            return 1;
        }
    }
    return 0;
}

// Function to check if a given token is an identifier
int isIdentifier(const char *token)
{
    if (!isalpha(token[0]) && token[0] != '_')
    {
        return 0;
    }
    int length = strlen(token);
    for (int i = 1; i < length; i++)
    {
        if (!isalnum(token[i]) && token[i] != '_')
        {
            return 0;
        }
    }
    return 1;
}

// Function to construct the symbol table
void constructSymbolTable(const char *sourceCode, SymbolTable
*symbolTable)
{
```

```
    char token[MAX_TOKEN_SIZE];
    int position = 0;
    while (sourceCode[position] != '\0')
    {
        int tokenPosition = 0;
        // Skip whitespace characters
        while (isspace(sourceCode[position]))
        {
            position++;
        }
        // Check for symbols
        if (sourceCode[position] == '(' ||
            sourceCode[position] == ')' ||
            sourceCode[position] == '{' ||
            sourceCode[position] == '}' ||
            sourceCode[position] == ';' ||
            sourceCode[position] == ',')
        {
            token[tokenPosition++] = sourceCode[position++];
            token[tokenPosition] = '\0';
            // Store the symbol in the symbol table
            strcpy(symbolTable->symbols[symbolTable->count].name,
token);
            strcpy(symbolTable->symbols[symbolTable->count].type,
"symbol");
            symbolTable->count++;
            continue;
        }
        // Check for numbers
        if (isdigit(sourceCode[position]))
        {
            while (isdigit(sourceCode[position]))
            {
                token[tokenPosition++] = sourceCode[position++];
            }
            token[tokenPosition] = '\0';
            // Store the number in the symbol table
```

```
            strcpy(symbolTable->symbols[symbolTable->count].name,
token);
            strcpy(symbolTable->symbols[symbolTable->count].type,
"number");
            symbolTable->count++;
            continue;
        }
        // Check for identifiers or keywords
        if (isalpha(sourceCode[position]) || sourceCode[position] == '_')
        {
            while (isalnum(sourceCode[position]) || sourceCode[position] ==
'_')
            {
                token[tokenPosition++] = sourceCode[position++];
            }
            token[tokenPosition] = '\0';
            // Check if it is a keyword or identifier
            if (isKeyword(token))
            {
                // Store the keyword in the symbol table
                strcpy(symbolTable->symbols[symbolTable->count].name,
token);
                strcpy(symbolTable->symbols[symbolTable->count].type,
"keyword");
                symbolTable->count++;
            }
            else if (isIdentifier(token))
            {
                // Store the identifier in the symbol table
                strcpy(symbolTable->symbols[symbolTable->count].name,
token);
                strcpy(symbolTable->symbols[symbolTable->count].type,
"identifier");
                symbolTable->count++;
            }
            continue;
        }
```

```
            position++;
        }
    }

    // Function to print the symbol table
    void printSymbolTable(const SymbolTable *symbolTable)
    {
        printf("Symbol Table:\n");
        printf("-------------\n");
        printf("Token\t\tData Type\n");
        printf("-------------\n");
        for (int i = 0; i < symbolTable->count; i++)
        {
            printf("%s\t\t%s\n", symbolTable->symbols[i].name, symbolTable-
    >symbols[i].type);
        }
        printf("-------------\n");
    }

    int main()
    {
        char sourceCode[] = "int main() {\n int a = 18;\n float b = 3.14;\n
    printf(\"Hello, this is 21BAI1087!\");\n return 0;\n}";
        SymbolTable symbolTable;
        symbolTable.count = 0;
        constructSymbolTable(sourceCode, &symbolTable);
        printSymbolTable(&symbolTable);

        return 0;
    }
```

Screenshot:

```
Symbol Table:
-------------
Token           Data Type
-------------
int             keyword
main            identifier
(               symbol
)               symbol
{               symbol
int             keyword
a               identifier
18              number
;               symbol
float           keyword
b               identifier
3               number
14              number
;               symbol
printf          identifier
(               symbol
Hello           identifier
,               symbol
this            identifier
is              identifier
21              number
BAI1087         identifier
)               symbol
;               symbol
return          keyword
0               number
;               symbol
}               symbol
-------------
```

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab2 & 17-01-2024

Slot: L43+L44

Questions:

Construct a DFA for the following regular expression using Direct method manually. For girl students- a(a+b)*ab. Write a C program for a constructed DFA and check whether the given string aabab, abba are accepted or not. for boys - **(a+b)*abb**. Write a C program for a constructed DFA and check whether the given string **aabbabab, aabb** are accepted or not.

Answers:

a. **Code:**

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

bool isAccepted(char input[]) {
   char currentState = 'A'; // Starting state

   for (int i = 0; input[i] != '\0'; i++) {
      char currentInput = input[i];


      switch (currentState) {
         case 'A':
            if (currentInput == 'a') currentState = 'B';
```

```
                break;
            case 'B':
                if (currentInput == 'a') currentState = 'B';
                else if (currentInput == 'b') currentState = 'C';
                break;
            case 'C':
                if (currentInput == 'a') currentState = 'B';
                else if (currentInput == 'b') currentState = 'D';
                break;
            case 'D':
                if (currentInput == 'a') currentState = 'B';
                else if (currentInput == 'b') currentState = 'A';
                break;
            default:

                return false;
        }
    }
    return currentState == 'D';
}

int main() {
    char input[100];
    printf("Enter the input string: ");
    scanf("%s", input);

    if (isAccepted(input)) {
        printf("Accepted\n");
    } else {
        printf("Not Accepted\n");
    }

    return 0;
}
```

Screenshot:

```
Enter the input string:aabbabab
Not Accepted
```

```
Enter the input string: aabb
Accepted
```

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab3 & 31-01-2024

Slot: L43+L44

Lex Tool Implementation

1. **Implementation of Lexical Analyzer (any one above programs) using Lex Tool, Construction of Syntax analyzer .**
2. **Implement a c program for LL(1) top down parser for any given LL(1) grammar.**

Code:

1)

```
//Implementation of Lexical Analyzer using Lex tool
%{
int COMMENT=0;
%}
identifier [a-zA-Z][a-zA-Z0-9]*
%%
#.* {printf("\n%s is a preprocessor directive",yytext);}
int |
float |
char |
double |
while |
for |
struct |
typedef |
```

```
do |
if |
break |
continue |
void |
switch |
return |
else |
goto {printf("\n\t%s is a keyword",yytext);}
"/*" {COMMENT=1;}{printf("\n\t %s is a COMMENT",yytext);}
{identifier}\( {if(!COMMENT)printf("\nFUNCTION \n\t%s",yytext);}
\{  {if(!COMMENT)printf("\n BLOCK BEGINS");}
\}  {if(!COMMENT)printf("BLOCK ENDS ");}
{identifier}(\[[0-9]*\])? {if(!COMMENT) printf("\n %s
IDENTIFIER",yytext);}
\".*\" {if(!COMMENT)printf("\n\t %s is a STRING",yytext);}
[0-9]+ {if(!COMMENT) printf("\n %s is a NUMBER ",yytext);}
\)(\:)? {if(!COMMENT)printf("\n\t");ECHO;printf("\n");}
\( ECHO;
= {if(!COMMENT)printf("\n\t %s is an ASSIGNMENT
OPERATOR",yytext);}
\<= |
\>= |
\< |
== |
\> {if(!COMMENT) printf("\n\t%s is a RELATIONAL
OPERATOR",yytext);}
%%
int main(int argc, char **argv)
{
FILE *file;
file=fopen("var.c","r");
if(!file)
{
printf("could not open the file");
exit(0);
```

```
}
yyin=file;
yylex();
printf("\n");
return(0);
}
int yywrap()
{
return(1);
}
```

INPUT:

```
//var.c
#include<stdio.h>
#include<conio.h>
void main()
{
int a,b,c;
a=1;
b=2;
c=a+b;
printf("Sum:%d",c);
}
```

```
student@614:~/Downloads$ lex words.l
student@614:~/Downloads$ cc lex.yy.c
student@614:~/Downloads$ ./a.out

#include<stdio.h> is a preprocessor directive

#include<conio.h> is a preprocessor directive

        void is a keyword
FUNCTION
        main(
        )


 BLOCK BEGINS

        int is a keyword
 a IDENTIFIER,
 b IDENTIFIER,
 c IDENTIFIER;

 a IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 1 is a NUMBER ;

 b IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 2 is a NUMBER ;

 c IDENTIFIER
         = is an ASSIGNMENT OPERATOR
 a IDENTIFIER+
 b IDENTIFIER;

FUNCTION
        printf(
         "Sum:%d" is a STRING,
 c IDENTIFIER
        )
;
BLOCK ENDS
```

2)

```
#include<stdio.h>
#include<string.h>
#define TSIZE 128
// table[i][j] stores
// the index of production that must be applied on
// ith varible if the input is
// jth nonterminal
```

```
int table[100][TSIZE];
// stores all list of terminals
// the ASCII value if use to index terminals
// terminal[i] = 1 means the character with
// ASCII value is a terminal
char terminal[TSIZE];
// stores all list of terminals
// only Upper case letters from 'A' to 'Z'
// can be nonterminals
// nonterminal[i] means ith alphabet is present as
// nonterminal is the grammar
char nonterminal[26];
// structure to hold each production
// str[] stores the production
// len is the length of production
struct product {
    char str[100];
    int len;
}pro[20];
// no of productions in form A->ß
int no_pro;
char first[26][TSIZE];
char follow[26][TSIZE];
// stores first of each production in form A->ß
char first_rhs[100][TSIZE];
// check if the symbol is nonterminal
int isNT(char c) {
    return c >= 'A' && c <= 'Z';
}
// reading data from the file
void readFromFile() {
    FILE* fptr;
    fptr = fopen("text.txt", "r");
    char buffer[255];
    int i;
    int j;
```

```
    while (fgets(buffer, sizeof(buffer), fptr)) {
        printf("%s", buffer);
        j = 0;
        nonterminal[buffer[0] - 'A'] = 1;
        for (i = 0; i < strlen(buffer) - 1; ++i) {
            if (buffer[i] == '|') {
                ++no_pro;
                pro[no_pro - 1].str[j] = '\0';
                pro[no_pro - 1].len = j;
                pro[no_pro].str[0] = pro[no_pro - 1].str[0];
                pro[no_pro].str[1] = pro[no_pro - 1].str[1];
                pro[no_pro].str[2] = pro[no_pro - 1].str[2];
                j = 3;
            }
            else {
                pro[no_pro].str[j] = buffer[i];
                ++j;
                if (!isNT(buffer[i]) && buffer[i] != '-' && buffer[i] != '>') {
                    terminal[buffer[i]] = 1;
                }
            }
        }
        pro[no_pro].len = j;
        ++no_pro;
    }
}
void add_FIRST_A_to_FOLLOW_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || first[A - 'A'][i];
    }
}
void add_FOLLOW_A_to_FOLLOW_B(char A, char B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
```

```
        if (i != '^')
            follow[B - 'A'][i] = follow[B - 'A'][i] || follow[A - 'A'][i];
    }
}
void FOLLOW() {
    int t = 0;
    int i, j, k, x;
    while (t++ < no_pro) {
        for (k = 0; k < 26; ++k) {
            if (!nonterminal[k])    continue;
            char nt = k + 'A';
            for (i = 0; i < no_pro; ++i) {
                for (j = 3; j < pro[i].len; ++j) {
                    if (nt == pro[i].str[j]) {
                        for (x = j + 1; x < pro[i].len; ++x) {
                            char sc = pro[i].str[x];
                            if (isNT(sc)) {
                                add_FIRST_A_to_FOLLOW_B(sc, nt);
                                if (first[sc - 'A']['^'])
                                    continue;
                            }
                            else {
                                follow[nt - 'A'][sc] = 1;
                            }
                            break;
                        }
                        if (x == pro[i].len)
                            add_FOLLOW_A_to_FOLLOW_B(pro[i].str[0], nt);
                    }
                }
            }
        }
    }
}
void add_FIRST_A_to_FIRST_B(char A, char B) {
    int i;
```

```
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^') {
            first[B - 'A'][i] = first[A - 'A'][i] || first[B - 'A'][i];
        }
    }
}
void FIRST() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_B(sc, pro[i].str[0]);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first[pro[i].str[0] - 'A'][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first[pro[i].str[0] - 'A']['^'] = 1;
        }
        ++t;
    }
}
void add_FIRST_A_to_FIRST_RHS__B(char A, int B) {
    int i;
    for (i = 0; i < TSIZE; ++i) {
        if (i != '^')
            first_rhs[B][i] = first[A - 'A'][i] || first_rhs[B][i];
    }
}
```

```
// Calculates FIRST(ß) for each A->ß
void FIRST_RHS() {
    int i, j;
    int t = 0;
    while (t < no_pro) {
        for (i = 0; i < no_pro; ++i) {
            for (j = 3; j < pro[i].len; ++j) {
                char sc = pro[i].str[j];
                if (isNT(sc)) {
                    add_FIRST_A_to_FIRST_RHS__B(sc, i);
                    if (first[sc - 'A']['^'])
                        continue;
                }
                else {
                    first_rhs[i][sc] = 1;
                }
                break;
            }
            if (j == pro[i].len)
                first_rhs[i]['^'] = 1;
        }
        ++t;
    }
}
int main() {
    readFromFile();
    follow[pro[0].str[0] - 'A']['$'] = 1;
    FIRST();
    FOLLOW();
    FIRST_RHS();
    int i, j, k;

    // display first of each variable
    printf("\n");
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
```

```
            char c = pro[i].str[0];
            printf("FIRST OF %c: ", c);
            for (j = 0; j < TSIZE; ++j) {
                if (first[c - 'A'][j]) {
                    printf("%c ", j);
                }
            }
            printf("\n");
        }
    }

    // display follow of each variable
    printf("\n");
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
            char c = pro[i].str[0];
            printf("FOLLOW OF %c: ", c);
            for (j = 0; j < TSIZE; ++j) {
                if (follow[c - 'A'][j]) {
                    printf("%c ", j);
                }
            }
            printf("\n");
        }
    }
    // display first of each variable ß
    // in form A->ß
    printf("\n");
    for (i = 0; i < no_pro; ++i) {
        printf("FIRST OF %s: ", pro[i].str);
        for (j = 0; j < TSIZE; ++j) {
            if (first_rhs[i][j]) {
                printf("%c ", j);
            }
        }
        printf("\n");
```

```
    }

    // the parse table contains '$'
    // set terminal['$'] = 1
    // to include '$' in the parse table
    terminal['$'] = 1;

    // the parse table do not read '^'
    // as input
    // so we set terminal['^'] = 0
    // to remove '^' from terminals
    terminal['^'] = 0;

    // printing parse table
    printf("\n");
    printf("\n\t**************** LL(1) PARSING TABLE
******************\n");
    printf("\t-------------------------------------------------------\n");
    printf("%-10s", "");
    for (i = 0; i < TSIZE; ++i) {
        if (terminal[i])    printf("%-10c", i);
    }
    printf("\n");
    int p = 0;
    for (i = 0; i < no_pro; ++i) {
        if (i != 0 && (pro[i].str[0] != pro[i - 1].str[0]))
            p = p + 1;
        for (j = 0; j < TSIZE; ++j) {
            if (first_rhs[i][j] && j != '^') {
                table[p][j] = i + 1;
            }
            else if (first_rhs[i]['^']) {
                for (k = 0; k < TSIZE; ++k) {
                    if (follow[pro[i].str[0] - 'A'][k]) {
                        table[p][k] = i + 1;
                    }
```

```
                }
            }
        }
    }
    k = 0;
    for (i = 0; i < no_pro; ++i) {
        if (i == 0 || (pro[i - 1].str[0] != pro[i].str[0])) {
            printf("%-10c", pro[i].str[0]);
            for (j = 0; j < TSIZE; ++j) {
                if (table[k][j]) {
                    printf("%-10s", pro[table[k][j] - 1].str);
                }
                else if (terminal[j]) {
                    printf("%-10s", "");
                }
            }
            ++k;
            printf("\n");
        }
    }
}
```

```
E->TA
A->+TA|^
T->FB
B->*FB|^
F->t|(E)
FIRST OF E: ( t
FIRST OF A: + ^
FIRST OF T: ( t
FIRST OF B: * ^
FIRST OF F: ( t

FOLLOW OF E: $ * +
FOLLOW OF A: $ * +
FOLLOW OF T: $ * +
FOLLOW OF B: $ * +
FOLLOW OF F: $ * +

FIRST OF E->TA: ( t
FIRST OF A->+TA: +
FIRST OF A->^: ^
FIRST OF T->FB: ( t
FIRST OF B->*FB: *
FIRST OF B->^: ^
FIRST OF F->t: t
FIRST OF F->(E: (


        ***************** LL(1) PARSING TABLE ********************
        ----------------------------------------------------------
          $         (         *         +         t
E                   E->TA                         E->TA
A         A->^                A->^      A->^
T                   T->FB                         T->FB
B         B->^                B->^      B->^
F                   F->(E                         F->t
```

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab3 & 04-03-2024

Slot: L43+L44

Questions:

Generate YACC specification for a few syntactic categories

a. Program to recognize a valid arithmetic experession that uses operator +, -, * and /
b. Program to recognize a valid variable which starts with a letter followed by any number of letter or digits
c. Implement an Arithmetic Calculator using LEX and YACC

Answers:

a)

Lex:

```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z_][a-zA-Z_0-9]* return id;
[0-9]+(\.[0-9]*)? return num;
[+/*] return op;
. return yytext[0];
\n return 0;
%%
int yywrap()
{
```

```
return 1;
}
```

YACC:

```
%{
#include<stdio.h>
int valid = 1;
%}
%token num id op
%%
start:id '=' s ';'
s: id x
| num x
| '-' num x
| '('s')' x
;
x: op s
|'-' s
|
;
%%
int yyerror()
{
valid = 0;
printf("\nInvalid expression!\n");
return 0;
}
int main()
{
printf("\nEnter the expression:\n");
yyparse();
if(valid)
{
printf("\nValid Expression!\n");
}
}
```

```
student@614:~$ gedit l1.y
student@614:~$ lex l1.l
student@614:~$ yacc -d l1.y
student@614:~$ gcc lex.yy.c y.tab.c -w
student@614:~$ ./a.out

Enter the expression:
a+b-c

Invalid expression!
student@614:~$ ./a.out

Enter the expression:
a=a+b-c

Invalid expression!
student@614:~$ ./a.out

Enter the expression:
a=a-b+c;

Valid Expression!
```

b)

Lex:

```
%{
#include "y.tab.h"
%}
%%
[a-zA-Z] { return LETTER ;}
[0-9] { return DIGIT ; }
[\n] { return NL ;}
[_] { return UND; }
. { return yytext[0]; }
%%
int yywrap()
{
return 1;
}
```

YACC:

```
%token DIGIT LETTER NL UND
%%
stmt : variable NL { printf("Valid Identifiers\n"); exit(0);}
;
```

```
variable : LETTER alphanumeric
;
alphanumeric: LETTER alphanumeric
| DIGIT alphanumeric
| UND alphanumeric
| LETTER
| DIGIT
| UND
;
%%
int yyerror(char *msg)
{
printf("Invalid Expression\n");
exit(0);
}
main ()
{
printf("Enter the variable name\n");
yyparse();
}
```

```
student@614:~$ yacc -d l2.y
student@614:~$ lex l2.l
student@614:~$ gcc lex.yy.c y.tab.c -w
student@614:~$ ./a.out
Enter the variable name
abc
Valid Identifiers
student@614:~$ ./a.out
Enter the variable name
1ab
Invalid Expression
```

c)

Lex:

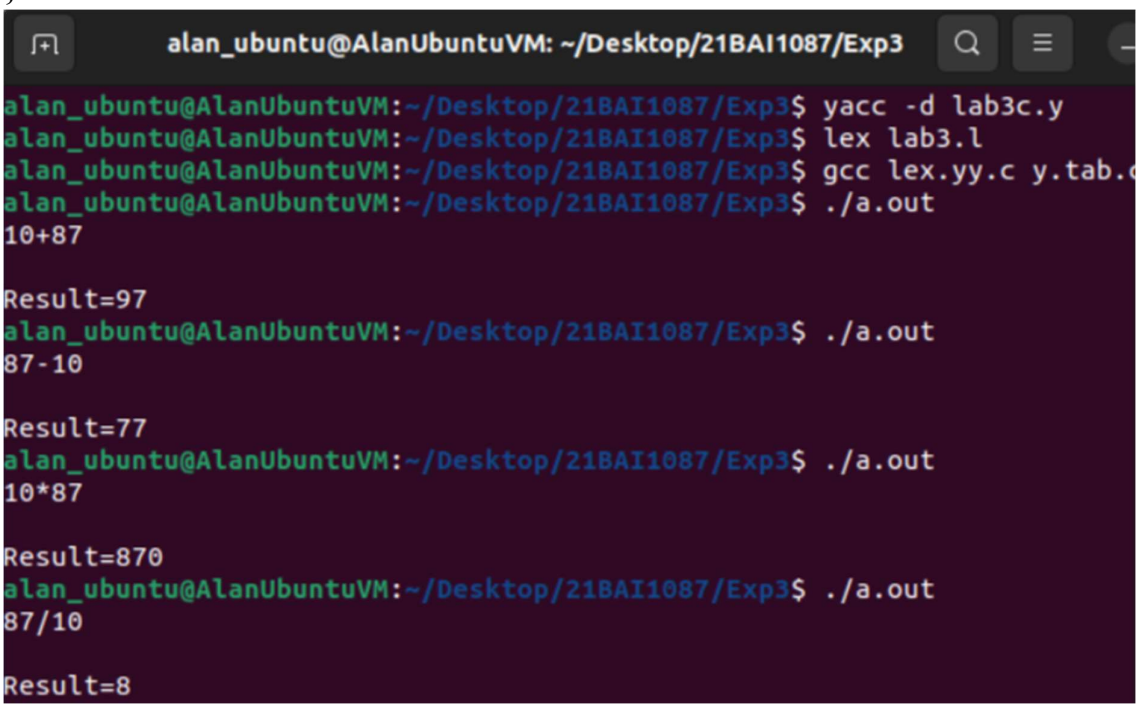
```
%{
   #include <stdio.h>
   extern int yylval;
%}

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

```
"+"        { return PLUS; }
"-"        { return MINUS; }
"*"        { return TIMES; }
"/"|"÷"    { return DIVIDE; }
\n         { return EOL; }
[ \t\r]    {}

%%
int yywrap(void){ return 1; }
```

YACC:

```
%{
   #include <stdio.h>
   int yylex();
   int yyerror();
%}

%%
%token NUMBER PLUS MINUS TIMES DIVIDE EOL;
%left PLUS MINUS;
%left TIMES DIVIDE;
%start statements;

statements  :   statement statements
          |   statement
          ;

statement   :   expression EOL            { printf("= %d\n", $1); }

expression  :   NUMBER                    { $$ = $1; printf("number: %d\n", $$); }
          |   expression TIMES expression { $$ = $1 * $3; printf("*: %d\n", $$); }
          |   expression PLUS expression  { $$ = $1 + $3; printf("+: %d\n", $$); }
          ;

%%
#include "lex.yy.c"

int main() {
```

```
  yyparse();
  return 1;
}
```



VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab4 & 19-03-2024

Slot: L43+L44

Question 1:

Write a program in Lex/ Yacc compiler to generate three address code for an expression X= a+b*c using Syntax Directed Translations.

Code:

```
#include <stdio.h>
 #include <stdlib.h>
 #include <math.h>
 #include <string.h>
 #include <ctype.h>
 char stack[100], top = -1;
 int prec(char op)
 {
 if(op == '+' || op == '-')
 {
 return 1;
 }
```

```
else if(op == '*' || op == '/')
 {
 return 2;
 }
else if(op == '^')
{
 return 3;
 }
else
 {
 return -1;
 }
 }
 char *inftopost(char* exp)
 {
 int ind = 0;
 char *post = (char *)malloc(sizeof(char));
 int length = strlen(exp);
 for(int i=0; i<length; i++)
 {
 if(isalnum(exp[i]))
 {
 post[ind++] = exp[i];
 }
else if(exp[i] == '(')
 {
 stack[++top] = exp[i];
```

```
 }
else if(exp[i] == ')')
 {
 while(top > -1 && stack[top] != '(')
 {
 post[ind++] = stack[top--];
 }
 top--;
 }
else
 {
 while(top > -1 && prec(stack[top] >= prec(exp[i])))
 {
 post[ind++] = stack[top--];
 }
 stack[++top] = exp[i];
}
 }
 while(top > - 1)
 {
 if(stack[top] != '(')
 {
 post[ind++] = stack[top--];
 }
 else
 {
 top--;
```

```
 }
 }
 post[ind] = '\0';
 return post;
 }
 int main()
 {
 char* exp = (char*)malloc(sizeof(char));
 char temp1, temp2;
 scanf("%s", exp);
char* post = (char*)malloc(sizeof(char));
 post = inftopost(exp);
 top = -1;
 char t = 'A' - 1;
for(int i = 0; i < strlen(post); i++)
 {
 if(isalnum(post[i]))
 {
 stack[++top] = post[i];
 }
else if(post[i] == '+')
 {
 printf("%c = %c + %c", ++t, stack[top--], stack[top--]);
 printf("\n");
 stack[++top] = t;
 }
else if(post[i] == '+')
```

```
 {
printf("%c = %c - %c", ++t, stack[top--], stack[top--]);
 printf("\n");
 stack[++top] = t; }
else if(post[i] == '*')
 {
 printf("%c = %c * %c", ++t, stack[top--], stack[top--]);
 printf("\n");
 stack[++top] = t;
 }
else if(post[i] == '/')
 {
 printf("%c = %c / %c", ++t, stack[top--], stack[top--]);
 printf("\n");
 stack[++top] = t;
 }
 }
 }
```

Screenshot:

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab3.c
PS D:\VIT_SEM6\Compiler Design\Lab> .\a.exe
a=b+c*d
A = b + c
B = A * d
```

Question 2:

Study of LLVM

LLVM (Low Level Virtual Machine) is a compiler infrastructure project that was originally developed at the University of Illinois at Urbana-Champaign, but is now maintained by a broad community of contributors. LLVM plays a crucial role in modern compiler design and has become a widely adopted framework

for building compilers, optimizers, and other tools for a variety of programming languages.

1. Overview of LLVM

LLVM is designed to be a highly modular and reusable compiler framework. It provides a collection of libraries and tools that can be used to build compilers, optimizers, just-in-time (JIT) compilers, static analysis tools, and other language-related tools. The core of LLVM is the LLVM Intermediate Representation (IR), which is a low-level, language-independent, and type-safe representation of code.

2. LLVM Intermediate Representation (IR)

The LLVM IR is a key component of the LLVM framework. It is a Static Single Assignment (SSA) based representation that captures the essential features of a program while abstracting away many low-level details. The LLVM IR is designed to be a common representation that can be generated from various source languages and can serve as input to various optimization passes and code generation backends.

The LLVM IR is designed to be language-agnostic, meaning that it can represent code from any programming language. This makes LLVM a powerful tool for building compilers for new languages or for providing language interoperability features.

3. LLVM Compilation Pipeline

The LLVM compilation pipeline consists of several stages:

1. Front-end: This stage is responsible for parsing the source code of a specific programming language and translating it into LLVM IR. Each programming language has its own front-end that generates LLVM IR from the source code.

2. Optimizer: Once the LLVM IR is generated, it can be optimized using various optimization passes provided by LLVM. These optimization passes can perform tasks such as dead code elimination, constant propagation, loop transformations, and many others. The optimizations are performed on the LLVM IR, which makes them language-independent.

3. Back-end: After optimization, the LLVM IR is passed to a back-end, which is responsible for generating machine code for a specific target architecture (e.g., x86, ARM, CUDA, etc.). LLVM provides a collection of back-ends for various architectures, allowing the same optimized LLVM IR to be compiled for different platforms.

4. LLVM Tools and Libraries

LLVM provides a rich set of tools and libraries that can be used for various purposes:

- Clang: A C/C++/Objective-C/Objective-C++ front-end that generates LLVM IR.

- LLVM Linker: A modular linker that can link LLVM bitcode files together.

- LLVM Libraries: A collection of libraries that provide functionality for IR generation, optimization, code generation, and more.

- LLVM Tools: Various tools for performing tasks such as disassembling, profiling, and debugging.

5. Applications of LLVM

LLVM has found widespread adoption in various domains, including:

- Compiler Development: LLVM is used as the basis for building compilers for various programming languages, such as Rust, Swift, Julia, and others.

- Just-in-Time (JIT) Compilation: LLVM can be used to implement efficient JIT compilers for dynamic languages like Python, Ruby, and JavaScript.

- Static Analysis and Program Transformation: LLVM provides a powerful infrastructure for performing static analysis, code instrumentation, and program transformation.

- Optimizations for Specific Domains: LLVM has been used to implement optimizations for domains such as graphics processing, machine learning, and high-performance computing.

6. LLVM Community and Ecosystem:

LLVM has a vibrant and active community of contributors from academia, industry, and open-source projects. The LLVM project is hosted by the LLVM Foundation, which promotes the development and adoption of LLVM and its related projects. The LLVM ecosystem includes a wide range of projects and tools built on top of LLVM, such as compiler-rt, libcxx, and various language front-ends.

In summary, LLVM is a powerful and versatile compiler infrastructure that has revolutionized the field of compiler design. Its modular architecture, language-agnostic IR, and extensive optimization capabilities make it a valuable tool for building compilers, optimizers, and other language-related tools. LLVM's widespread adoption and active community ensure its continuous development and expansion into new domains.

X----------X

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab4B & 20-03-2024

Slot: L43+L44

Question 1:

Generate the TAC for the following arithmetic expression and represent them in Quadruples, Triples and Indirect triple structures x= a+(b^c) * d-e/ (f^ (g* i%j)).

Code:

```
#include <iostream>
#include <vector>
#include <stack>
#include <string>
#include <map>

using namespace std;

// Structure to represent a quadruple
struct Quadruple {
    string op;
    string arg1;
```

```
    string arg2;
    string result;
};

// Structure to represent a triple
struct Triple {
    string op;
    string arg1;
    string arg2;
};

// Structure to represent an indirect triple
struct IndirectTriple {
    string op;
    vector<string> args;
};

// Function to generate 3-address code for the given expression
vector<Quadruple> generate3AddressCode(const string& expression) {
    vector<Quadruple> quadruples;
    stack<char> operators;
    stack<string> operands;
    map<char, int> precedence;
    precedence['+'] = precedence['-'] = 1;
    precedence['*'] = precedence['/'] = precedence['%'] = 2;
    precedence['^'] = 3;
```

```
    for (char c : expression) {
        if (c == '(') {
            operators.push(c);
        } else if (isdigit(c) || isalpha(c)) {
            operands.push(string(1, c));
        } else if (c == ')') {
            while (!operators.empty() && operators.top() != '(') {
                char op = operators.top();
                operators.pop();
                string arg2 = operands.top();
                operands.pop();
                string arg1 = operands.top();
                operands.pop();
                string result = "t" + to_string(quadruples.size() + 1);
                quadruples.push_back({string(1, op), arg1, arg2, result});
                operands.push(result);
            }
            operators.pop(); // Discard the opening parenthesis
        } else {
            while (!operators.empty() && precedence[operators.top()] >=
precedence[c]) {
                char op = operators.top();
                operators.pop();
                string arg2 = operands.top();
                operands.pop();
                string arg1 = operands.top();
                operands.pop();
                string result = "t" + to_string(quadruples.size() + 1);
```

```
            quadruples.push_back({string(1, op), arg1, arg2, result});
            operands.push(result);
        }
        operators.push(c);
    }
}

// Process remaining operators in the stack
while (!operators.empty()) {
    char op = operators.top();
    operators.pop();
    string arg2 = operands.top();
    operands.pop();
    string arg1 = operands.top();
    operands.pop();
    string result = "t" + to_string(quadruples.size() + 1);
    quadruples.push_back({string(1, op), arg1, arg2, result});
    operands.push(result);
}

return quadruples;
}

// Function to represent quadruples as triples
vector<Triple> convertToTriples(const vector<Quadruple>& quadruples) {
    vector<Triple> triples;
    for (const auto& quad : quadruples) {
```

```
        triples.push_back({quad.op, quad.arg1, quad.arg2});
    }
    return triples;
}

// Function to represent quadruples as indirect triples
vector<IndirectTriple> convertToIndirectTriples(const vector<Quadruple>&
quadruples) {
    vector<IndirectTriple> indirectTriples;
    for (const auto& quad : quadruples) {
        indirectTriples.push_back({quad.op, {quad.arg1, quad.arg2,
quad.result}});
    }
    return indirectTriples;
}

int main() {
    string expression = "x=a+(b^c)*d-e/f^(g*i%j)";
    vector<Quadruple> quadruples = generate3AddressCode(expression);

    // Display quadruples
    cout << "Quadruples:\n";
    for (const auto& quad : quadruples) {
        cout << quad.result << " = " << quad.arg1 << " " << quad.op << " " <<
quad.arg2 << endl;
    }
    cout << endl;
```

```
    // Display triples
    vector<Triple> triples = convertToTriples(quadruples);
    cout << "Triples:\n";
    for (const auto& triple : triples) {
        cout << triple.op << " " << triple.arg1 << " " << triple.arg2 << endl;
    }
    cout << endl;

    // Display indirect triples
    vector<IndirectTriple> indirectTriples =
convertToIndirectTriples(quadruples);
    cout << "Indirect Triples:\n";
    for (const auto& indirectTriple : indirectTriples) {
        cout << indirectTriple.op << " ";
        for (const auto& arg : indirectTriple.args) {
            cout << arg << " ";
        }
        cout << endl;
    }

    return 0;
}
```

Screenshot:

```
vboxuser@ubuntu:~/Desktop$ g++ l4.cpp
vboxuser@ubuntu:~/Desktop$ ./a.out
Quadruples:
t1 = b ^ c
t2 = t1 * d
t3 = a + t2
t4 = g * i
t5 = t4 % j
t6 = f ^ t5
t7 = e / t6
t8 = t3 - t7
t9 = x = t8

Triples:
^ b c
* t1 d
+ a t2
* g i
% t4 j
^ f t5
/ e t6
- t3 t7
= x t8

Indirect Triples:
^ b c t1
* t1 d t2
+ a t2 t3
* g i t4
% t4 j t5
^ f t5 t6
/ e t6 t7
- t3 t7 t8
= x t8 t9
```

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

CHENNAI

Name: Shaahid Ahmed N

RegNo. : 21BAI1087

Course & Course Code: Compiler Design Lab & BCSE307P

Lab & Date: Lab-5 & 10-04-2024

Slot: L43+L44

---

1) **Compile Time Evaluation:**
   - **Unoptimized Code:**

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

int main() {
    clock_t start, end;
    double cpu_time_used;

    start = clock();

    // Non-optimized calculation
    double A = 2 * (22.0 / 7.0) * 10.0; // Assuming r = 10.0
    printf("A = %.2f\n", A);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for non-optimized calculation: %f seconds\n",
cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
A = 62.86
Time taken for non-optimized calculation: 0.003000 seconds
```

- **Optimized Code:**

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

#define PI 3.14159265359

int main() {
    clock_t start, end;
    double cpu_time_used;

    start = clock();

    // Optimized calculation
    double A = 2 * PI * 10.0; // Assuming r = 10.0
    printf("A = %.2f\n", A);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for optimized calculation: %f seconds\n",
cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
A = 62.83
Time taken for optimized calculation: 0.002000 seconds
```

2) **Variable Propagation:**

- **Unoptimized Code:**

```
#include <stdio.h>
#include <time.h>
int main() {
    clock_t start, end;
```

```
    double cpu_time_used;
    start = clock();
    // Your code before optimization
    int a = 5;
    int b = 10;
    int c, x, d;
    c = a * b;
    x = a;
    d = x * b + 4;
    printf("c = %d\n", c);
    printf("d = %d\n", d);
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken: %f seconds\n", cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
c = 50
d = 54
Time taken(Unoptimized): 0.004000 seconds
```

- **Optimized Code:**

```
#include <stdio.h>
#include <time.h>
int main() {
    clock_t start, end;
    double cpu_time_used;
    start = clock();
    int a = 5;
    int b = 10;
    int c, x, d;
    c = a * b;
    x = a;
    d = a * b + 4; // Using 'a * b' directly instead of 'x * b'
    printf("c = %d\n", c);
    printf("d = %d\n", d);
```

```
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken: %f seconds\n", cpu_time_used);
    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
c = 50
d = 54
Time taken(Optimized): 0.001000 seconds
```

3) Constant Propagation:

- **Unoptimized Code:**

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

#define PI 3.14159265359

int main() {
    clock_t start, end;
    double cpu_time_used;

    // Unoptimized code with CPU time measurement
    start = clock();

    const double r_unopt = 5.0; // Assuming r is known at compile time
    const double A_unopt = 2 * (22.0 / 7.0) * r_unopt;
    printf("Unoptimized A = %.2f\n", A_unopt);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for unoptimized calculation: %f seconds\n",
cpu_time_used);
    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Unoptimized A = 31.43
Time taken for unoptimized calculation: 0.032000 seconds
```

- **Optimized Code:**

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

#define PI 3.14159265359

int main() {
   clock_t start, end;
   double cpu_time_used;
   // Optimized code with CPU time measurement
   start = clock();
   const double r_opt = 5.0; // Assuming r is known at compile time
   const double A_opt = 2 * PI * r_opt;
   printf("Optimized A = %.2f\n", A_opt);
   end = clock();
   cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
   printf("Time taken for optimized calculation: %f seconds\n",
cpu_time_used);
   return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Optimized A = 31.42
Time taken for optimized calculation: 0.005000 seconds
```

4) **Constant Folding:**

- **Code:**

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

#define k 5

int main() {
   clock_t start, end;
   double cpu_time_used;

   // Unoptimized code with CPU time measurement
   start = clock();
```

```
    const int x_unopt = 2 * k;
    const int y_unopt = k + 5;

    printf("Unoptimized x = %d\n", x_unopt);
    printf("Unoptimized y = %d\n", y_unopt);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for unoptimized code: %f seconds\n",
cpu_time_used);

    // Optimized code with CPU time measurement
    start = clock();

    const int x_opt = 10; // Value of x at compile time
    const int y_opt = 10; // Value of y at compile time

    printf("Optimized x = %d\n", x_opt);
    printf("Optimized y = %d\n", y_opt);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for optimized code: %f seconds\n",
cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Unoptimized x = 10
Unoptimized y = 10
Time taken for unoptimized code: 0.003000 seconds
Optimized x = 10
Optimized y = 10
Time taken for optimized code: 0.001000 seconds
```

5) **Copy Propagation:**

- **Code:**

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

int main() {
    clock_t start, end;
    double cpu_time_used;

    // Before Optimization code with CPU time measurement
    start = clock();

    int a = 5;
    int b = 10;
    int c, x, d;

    c = a * b;
    x = a;

    // Some code in between

    d = x * b + 4;

    printf("Before Optimization:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for before optimization code: %f seconds\n\n",
cpu_time_used);

    // After Optimization code with CPU time measurement
    start = clock();

    c = a * b;
    x = a;

    // Some code in between
```

```
    d = a * b + 4; // Using 'a * b' directly instead of 'x * b'

    printf("After Optimization:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for after optimization code: %f seconds\n",
cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Before Optimization:
c = 50
d = 54
Time taken for before optimization code: 0.003000 seconds

After Optimization:
c = 50
d = 54
Time taken for after optimization code: 0.001000 seconds
```

6) Common Sub Expression Elimination:

- **Code:**

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

int main() {
    clock_t start, end;
    double cpu_time_used;

    // Before Optimization code with CPU time measurement
    start = clock();

    int a = 5;
    int b = 10;
```

```
    int c, x, d;

    c = a * b;
    x = a;

    // Some code in between

    d = a * b + 4;

    printf("Before Optimization:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for before optimization code: %f seconds\n\n",
cpu_time_used);

    // After Elimination code with CPU time measurement
    start = clock();

    c = a * b;
    x = a;

    // Some code in between

    d = c + 4; // Using the value of c directly

    printf("After Elimination:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for after elimination code: %f seconds\n",
cpu_time_used);
```

```
    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Before Optimization:
c = 50
d = 54
Time taken for before optimization code: 0.004000 seconds

After Elimination:
c = 50
d = 54
Time taken for after elimination code: 0.003000 seconds
```

7) Dead Code Elimination:

- **Code:**

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

int main() {
    clock_t start, end;
    double cpu_time_used;

    // Before Optimization code with CPU time measurement
    start = clock();

    int a = 5;
    int b = 10;
    int c, x, d;

    c = a * b;
    x = a;

    // Some code in between

    d = a * b + 4;

    printf("Before Optimization:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);
```

```
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for before optimization code: %f seconds\n\n",
cpu_time_used);

    // After Elimination code with CPU time measurement
    start = clock();

    c = a * b;
    // Some code in between
    d = c + 4; // Using the value of c directly

    printf("After Elimination:\n");
    printf("c = %d\n", c);
    printf("d = %d\n", d);

    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("Time taken for after elimination code: %f seconds\n",
cpu_time_used);

    return 0;
}
```

```
PS D:\VIT_SEM6\Compiler Design\Lab> gcc lab5.c
PS D:\VIT_SEM6\Compiler Design\Lab> ./a.exe
Before Optimization:
c = 50
d = 54
Time taken for before optimization code: 0.010000 seconds

After Elimination:
c = 50
d = 54
Time taken for after elimination code: 0.000000 seconds
```

8) DAG:

- **Code:**

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

```
int main()
{
struct da
{
int ptr, left, right;
char label;
} dag[25];
int ptr, l, j, change, n = 0, i = 0, state = 1, x, y, k;
char store, input1[100], input[25], var;
for(i = 0; i < 25; i++)
{
dag[i].ptr = NULL;
dag[i].left = NULL;
dag[i].right = NULL;
dag[i].label = NULL;
}
printf("\n\nEnter the expression: ");
scanf("%s",input1);
for(i = 0; i < 25; i++)
```

```
input[i] = NULL;
l = strlen(input1);
a:
for(i = 0; input1[i] != ')'; i++);
for(j = i; input1[j] != '('; j--);
for(x = j + 1; x < i; x++)
if(isalpha(input1[x]))
input[n++] = input1[x];
else
if (input1[x] != '0')
store = input1[x];
input[n++] = store;
for(x = j; x <= i; x++)
input1[x] = '0';
if(input1[0] != '0')
goto a;
for(i = 0; i < n; i++)
{
```

```
dag[i].label = input[i];

dag[i].ptr = i;

if(!isalpha(input[i]) && !isdigit(input[i]))

{

dag[i].right = i - 1;

ptr = i;

var = input[i - 1];

if(isalpha(var))

ptr = ptr - 2;

else

{

ptr = i - 1;

b:

if(!isalpha(var) && !isdigit(var))

{

ptr = dag[ptr].left;

var = input[ptr];

goto b;

}
```

```
else

ptr = ptr - 1;

}

dag[i].left = ptr;

}

}

printf("\n Sytax Tree for given expression: \n\n");

printf("\n\n PTR \t LEFT PTR \t RIGHT PTR \t LABEL\n\n");

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

printf("\n%d\t%d\t%d\t%c\n", dag[i].ptr, dag[i].left, dag[i].right,
dag[i].label);

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

{

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

{

if((dag[i].label == dag[j].label && dag[i].left == dag[j].left) &&
dag[i].right == dag[j].right)

{

for(k = 0; k < n; k++)
```

```
{
if(dag[k].left == dag[j].ptr)
dag[k].left = dag[i].ptr;
if(dag[k].right == dag[j].ptr)
dag[k].right = dag[i].ptr;
}
dag[j].ptr = dag[i].ptr;
}
}
}
printf("\n DAG for given expression: \n\n");
printf("\n\n PTR \t LEFT PTR \t RIGHT PTR \t LABEL \n\n");
for(i = 0; i < n; i++)
printf("\n %d\t\t%d\t\t%d\t\t%c\n", dag[i].ptr, dag[i].left, dag[i].right,
dag[i].label);}
```

```
Enter the expression: (a+a*(b-c)+(b-c)*d)
 Sytax Tree for given expression:

 PTR     LEFT PTR        RIGHT PTR       LABEL

0       0       0       b
1       0       0       c
2       0       1       -
3       0       0       b
4       0       0       c
5       3       4       -
6       0       0       a
7       0       0       a
8       0       0       d
9       7       8       *
 DAG for given expression:

 PTR     LEFT PTR        RIGHT PTR       LABEL

 0              0               0               b
 1              0               0               c
 2              0               1               -
 0              0               0               b
 1              0               0               c
 2              0               1               -
 6              0               0               a
 6              0               0               a
 8              0               0               d
 9              6               8               *
```