CERTIFICATE

The term work of Compiler Design Lab (PCS-601), being submitted by Deepanshu Mishra s/o Ashok Kumar, Enrollment no PV-21610128, Roll no - 2161128 to Graphic Era Hill University, Bhimtal Campus is a bonafide work carried out by him/her. He has worked under my guidance and supervision and fulfilled the requirement for the submission of this lab file.

(…………………)	**(……………………)**
Faculty Incharge	**HOD, Dept. of CSE**

ACKNOWLEDGEMENT

I take immense pleasure in thanking **Mr. Anubhav Bewerwal** (Assistant Professor, Dept. of CSE, GEHU, Bhimtal Campus) for allowing me to carry out this lab work under his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance and useful suggestions that have helped me in developing my subject concepts as a student.

I want to extend thanks to our President, Honorable **Prof. (Dr.) Kamal Ghanshala** for providing us all infrastructure and facilities to work in need without which this work would not be possible.

(DEEPANSHU MISHRA)

deepanshumishra9427@gmail.com

STUDENT’S DECLARATION

I Deepanshu Mishra, hereby declare the work, which is being presented in the report, entitled **Term work** of **Compiler Design Lab (PCS-601)** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (Computer Science & Engineering)** in the session **2023-2024** for semester VI, is an authentic record of my own work carried out under the supervision of **Mr. Anubhav Bewerwal,** Dept. of CSE (Graphic Era Hill University, Bhimtal Campus).

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date: …………   ……………….

(Full signature of student)

5.Write a program in C or C++ language for the following functions without using string.h header file:

a: "to get the length of a string, you use the strlen() function"
b: "To concatenate (combine) two strings, you can use the strcat() function
c: "To copy the value of one string to another, you can use the strcpy()"
d: "To compare two strings, you can use the strcmp() function."
and other related functions.

```
#include <iostream>

// Function to compute the length of a string
int my_strlen(const char* str) {
    int length = 0;
    while (str[length] != '\0') {
        length++;
    }
    return length;
}

// Function to concatenate two strings
char* my_strcat(char* dest, const char* src) {
    int dest_len = my_strlen(dest);
    int i = 0;
    while (src[i] != '\0') {
        dest[dest_len + i] = src[i];
        i++;
    }
    dest[dest_len + i] = '\0';
    return dest;
}

// Function to copy one string to another
char* my_strcpy(char* dest, const char* src) {
    int i = 0;
    while (src[i] != '\0') {
        dest[i] = src[i];
        i++;
    }
    dest[i] = '\0';
    return dest;
}

// Function to compare two strings
int my_strcmp(const char* str1, const char* str2) {
    int i = 0;
    while (str1[i] != '\0' && str2[i] != '\0') {
        if (str1[i] != str2[i]) {
            return str1[i] - str2[i];
        }
        i++;
    }
    return str1[i] - str2[i];
}

int main() {
    // Testing my_strlen
    const char* test_str = "hello";
    std::cout << "Length of \"" << test_str << "\" is: " << my_strlen(test_str) << std::endl;

    // Testing my_strcat
    char dest[50] = "hello";
    const char* src = " world";
    std::cout << "Concatenation of \"" << dest << "\" and \"" << src << "\" is: " <<
my_strcat(dest, src) << std::endl;

    // Testing my_strcpy
    char dest_copy[50];
    const char* src_copy = "source";
    std::cout << "Copying \"" << src_copy << "\" results in: " << my_strcpy(dest_copy,
src_copy) << std::endl;

    // Testing my_strcmp
    const char* str1 = "apple";
    const char* str2 = "banana";
    std::cout << "Comparison of \"" << str1 << "\" and \"" << str2 << "\" results in: " <<
my_strcmp(str1, str2) << std::endl;

    return 0;
}
```

6. Write a program in C or C++ language to implement Predictive Parsing Algorithm.

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

using namespace std;

// Define the grammar rules
map<string, map<string, string>> parsingTable = {
    {"E",  {{"id", "T E'"},  {"(", "T E'"} }},
    {"E'", {{"+", "+ T E'"}, {")", ""}, {"$", ""} }},
    {"T",  {{"id", "F T'"},  {"(", "F T'"} }},
    {"T'", {{"*", "* F T'"}, {"+", ""}, {")", ""}, {"$", ""} }},
    {"F",  {{"id", "id"},    {"(", "( E )"} }}
};

// Tokenize input string
vector<string> tokenize(const string& input) {
    vector<string> tokens;
    stringstream ss(input);
    string token;
    while (ss >> token) {
        tokens.push_back(token);
    }
    tokens.push_back("$");
    return tokens;
}

// LL(1) Parser function
bool parse(const vector<string>& tokens) {
    stack<string> parseStack;
    parseStack.push("$");
    parseStack.push("E");

    int index = 0;
    while (!parseStack.empty()) {
        string top = parseStack.top();
        string currentToken = tokens[index];

        if (top == currentToken) {
            parseStack.pop();
            index++;
        } else if (parsingTable.find(top) != parsingTable.end() &&
parsingTable[top].find(currentToken) != parsingTable[top].end()) {
            parseStack.pop();
            string rule = parsingTable[top][currentToken];
            if (!rule.empty()) {
                vector<string> symbols;
                stringstream ss(rule);
                string symbol;
                while (ss >> symbol) {
                    symbols.push_back(symbol);
                }
                for (auto it = symbols.rbegin(); it != symbols.rend(); ++it) {
                    parseStack.push(*it);
                }
            }
        } else {
            return false;
        }
    }

    return index == tokens.size();
}

int main() {
    string input;
    cout << "Enter the string to parse (tokens separated by spaces): ";
    getline(cin, input);

    vector<string> tokens = tokenize(input);
    bool result = parse(tokens);

    if (result) {
        cout << "The input string is successfully parsed!" << endl;
    } else {
        cout << "The input string is rejected by the parser!" << endl;
    }

    return 0;
}
```

7. Write a program in C or C++ language to find the FIRST and FOLLOW of all the variables. Create functions for FIRST and FOLLOW.

```
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <string>
#include <cctype>

using namespace std;

map<char, vector<string>> grammar;
map<char, set<char>> firstSets;
map<char, set<char>> followSets;

void addFirst(char symbol, set<char> &firstSet);
void addFollow(char symbol, set<char> &followSet);
void computeFirst();
void computeFollow();

int main() {
    // Example grammar
    grammar['A'] = {"aBC"};
    grammar['B'] = {"b"};
    grammar['C'] = {"c"};

    // Compute FIRST sets
    computeFirst();
    cout << "FIRST sets:" << endl;
    for (const auto &pair : firstSets) {
        cout << "FIRST(" << pair.first << ") = { ";
        for (char c : pair.second) {
            cout << c << " ";
        }
        cout << "}" << endl;
    }

    // Compute FOLLOW sets
    computeFollow();

    cout << "FOLLOW sets:" << endl;
    for (const auto &pair : followSets) {
        cout << "FOLLOW(" << pair.first << ") = { ";
        for (char c : pair.second) {
            cout << c << " ";
        }
        cout << "}" << endl;
    }

    return 0;
}

void computeFirst() {
    for (const auto &pair : grammar) {
        char variable = pair.first;
        if (firstSets.find(variable) == firstSets.end()) {
            set<char> firstSet;
            addFirst(variable, firstSet);
            firstSets[variable] = firstSet;
        }
    }
}

void addFirst(char symbol, set<char> &firstSet) {
    if (islower(symbol) || symbol == 'ε') {
        firstSet.insert(symbol);
        return;
    }

    for (const string &production : grammar[symbol]) {
        for (char ch : production) {
            if (ch == symbol) break;
            if (islower(ch) || ch == 'ε') {
                firstSet.insert(ch);
                break;
            } else {
                set<char> subFirstSet;
                addFirst(ch, subFirstSet);
                firstSet.insert(subFirstSet.begin(), subFirstSet.end());
                if (subFirstSet.find('ε') == subFirstSet.end()) break;
            }
        }
    }
}

void computeFollow() {
    // Initialize follow set of start symbol with '$'
    followSets[grammar.begin()->first].insert('$');

    for (const auto &pair : grammar) {
        char variable = pair.first;
        if (followSets.find(variable) == followSets.end()) {
            set<char> followSet;
            addFollow(variable, followSet);
            followSets[variable] = followSet;
        }
    }
}

void addFollow(char symbol, set<char> &followSet) {
    for (const auto &pair : grammar) {
        char variable = pair.first;
        for (const string &production : pair.second) {
            for (size_t i = 0; i < production.length(); ++i) {
                if (production[i] == symbol) {
                    if (i + 1 < production.length()) {
                        char nextSymbol = production[i + 1];
                        if (islower(nextSymbol) || nextSymbol == 'ε') {
                            followSet.insert(nextSymbol);
                        } else {
                            set<char> firstSet = firstSets[nextSymbol];
                            followSet.insert(firstSet.begin(), firstSet.end());
                            followSet.erase('ε');

                            if (firstSet.find('ε') != firstSet.end() && variable != symbol) {
                                if (followSets.find(variable) == followSets.end()) {
                                    set<char> variableFollowSet;
                                    addFollow(variable, variableFollowSet);
                                    followSets[variable] = variableFollowSet;
                                }
                                followSet.insert(followSets[variable].begin(), followSets[varia-
ble].end());
                            }
                        }
                    } else if (variable != symbol) {
                        if (followSets.find(variable) == followSets.end()) {
                            set<char> variableFollowSet;
                            addFollow(variable, variableFollowSet);
                            followSets[variable] = variableFollowSet;
                        }
                        followSet.insert(followSets[variable].begin(), followSets[variable].end());
                    }
                }
            }
        }
    }
}
```

8. Write a program in C or C++ language to implement LR Parser.

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

using namespace std;

// Define the grammar
struct Production {
    char lhs;
    string rhs;
};

// Define the parser tables
map<pair<int, char>, string> actionTable;
map<pair<int, char>, int> gotoTable;

// Function to initialize the parser tables for the given grammar
void initializeTables() {
    // Action Table
    actionTable[{0, 'a'}] = "s3";
    actionTable[{0, 'b'}] = "s4";
    actionTable[{1, '$'}] = "acc";
    actionTable[{2, 'a'}] = "s3";
    actionTable[{2, 'b'}] = "s4";
    actionTable[{3, 'a'}] = "s3";
    actionTable[{3, 'b'}] = "s4";
    actionTable[{4, 'a'}] = "r3"; // A -> b
    actionTable[{4, 'b'}] = "r3";
    actionTable[{4, '$'}] = "r3";
    actionTable[{5, 'a'}] = "r1"; // S -> AA
    actionTable[{5, 'b'}] = "r1";
    actionTable[{5, '$'}] = "r1";
    actionTable[{6, 'a'}] = "r2"; // A -> aA
    actionTable[{6, 'b'}] = "r2";
    actionTable[{6, '$'}] = "r2";
    // Goto Table
    gotoTable[{0, 'S'}] = 1;
    gotoTable[{0, 'A'}] = 2;
    gotoTable[{2, 'A'}] = 5;
    gotoTable[{3, 'A'}] = 6;
}

// Main parser function
bool parse(const vector<char>& input) {
    stack<int> stateStack;
    stack<char> symbolStack;
    stateStack.push(0);
    int ip = 0;

    while (true) {
        int currentState = stateStack.top();
        char currentInput = input[ip];

        string action = actionTable[{currentState, currentInput}];

        if (action[0] == 's') {
            int nextState = stoi(action.substr(1));
            stateStack.push(nextState);
            symbolStack.push(currentInput);
            ip++;
        } else if (action[0] == 'r') {
            int productionNumber = stoi(action.substr(1));
            // Production rules for the new grammar
            vector<Production> productions = {
                {'S', "AA"},
                {'A', "aA"},
                {'A', "b"}
            };

            Production production = productions[productionNumber - 1];
            for (int i = 0; i < production.rhs.length(); i++) {
                stateStack.pop();
                symbolStack.pop();
            }
            symbolStack.push(production.lhs);

            int gotoState = gotoTable[{stateStack.top(), production.lhs}];
            stateStack.push(gotoState);
        } else if (action == "acc") {
            return true;
        } else {
            return false;
        }
    }
}
```

```
int main() {
    initializeTables();

    // Example input: aab$
    vector<char> input = {'a', 'a', 'b','b','$'};

    if (parse(input)) {
        cout << "Input accepted." << endl;
    } else {
        cout << "Input rejected." << endl;
    }

    return 0;
}
```

9. Write a program in C or C++ to generate the three-address code.

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

using namespace std;

// Function to check if the character is an operator
bool isOperator(char c) {
    return (c == '+' || c == '-' || c == '*' || c == '/');
}

// Function to generate three-address code
void generateThreeAddressCode(const string& expression) {
    stack<string> operands;
    stack<char> operators;
    int tempCounter = 1;

    for (char c : expression) {
        if (isalpha(c)) {
            operands.push(string(1, c)); // Convert char to string and push to stack
        } else if (isOperator(c)) {
            while (!operators.empty() && operators.top() != '(') {
                char op = operators.top();
                operators.pop();
                string operand2 = operands.top();
                operands.pop();
                string operand1 = operands.top();
                operands.pop();
                string temp = "t" + to_string(tempCounter++);
                cout << temp << " = " << operand1 << " " << op << " " << operand2 << endl;
                operands.push(temp);
            }
            operators.push(c);
        } else if (c == '(') {
            operators.push(c);
        } else if (c == ')') {
            while (!operators.empty() && operators.top() != '(') {
                char op = operators.top();
                operators.pop();
                string operand2 = operands.top();
                operands.pop();
                string operand1 = operands.top();
                operands.pop();
                string temp = "t" + to_string(tempCounter++);
                cout << temp << " = " << operand1 << " " << op << " " << operand2 << endl;
                operands.push(temp);
            }
            operators.pop(); // Pop '('
        }
    }

    while (!operators.empty()) {
        char op = operators.top();
        operators.pop();
        string operand2 = operands.top();
        operands.pop();
        string operand1 = operands.top();
        operands.pop();
        string temp = "t" + to_string(tempCounter++);
        cout << temp << " = " << operand1 << " " << op << " " << operand2 << endl;
        operands.push(temp);
    }
}

int main() {
    string expression;
    cout << "Enter the arithmetic expression: ";
    getline(cin, expression);
    cout << "Generated Three-Address Code:" << endl;
    generateThreeAddressCode(expression);

    return 0;
}
```

10. Write a program in C or C++ to generate machine code from the abstract syntax tree generated by the parser.

```
#include <iostream>
#include <stack>

using namespace std;

// Node structure for the Abstract Syntax Tree (AST)
struct Node {
    char data;
    Node* left;
    Node* right;
};

// Function to create a new node
Node* createNode(char data) {
    Node* newNode = new Node();
    newNode->data = data;
    newNode->left = newNode->right = nullptr;
    return newNode;
}

// Function to generate machine code from AST and return the result
// Function to generate machine code from AST and return the result
int generateMachineCode(Node* root) {
    stack<int> machineStack;
    if (root) {
        int leftResult = generateMachineCode(root->left);
        int rightResult = generateMachineCode(root->right);
        switch (root->data) {
            case '+':
                return leftResult + rightResult;
            case '-':
                return leftResult - rightResult;
            case '*':
                return leftResult * rightResult;
            case '/':
                return leftResult / rightResult;
            default:
                return root->data - '0'; // Convert character to integer
        }
    }
    return 0; // Return 0 if root is null
}


int main() {
    // Example AST
    Node* root = createNode('+');
    root->left = createNode('3');
    root->right = createNode('*');
    root->right->left = createNode('4');
    root->right->right = createNode('5');

    // Generate machine code and get result
    int result = generateMachineCode(root);

    // Output result
    cout << "Result: " << result << endl;

    return 0;
}
```