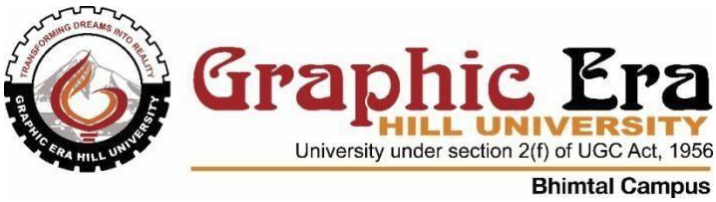

Term work

of

Compiler Design Lab (PCS-601)

Submitted in partial fulfillment of the requirement for the VI semester of

Bachelor of Technology (Computer Science & Engineering)

By

YASH JOSHI

University Roll No

2161347

Under the Guidance of

Ms. Megha Upreti

Lecturer

Department of Computer Science & Engineering

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI

DISTRICT- NAINITAL-263132

2023-2024

CERTIFICATE

The term work of **Compiler Design Lab (PCS-601),** being submitted by **Yash Joshi**, Roll no 2161347 to Graphic Era Hill University Bhimtal Campus for the award of bona fide work carried outby him. He has worked under my guidance and supervision and fulfilled the requirement for the submission of this work report.

(……)

Ms. Megha Upreti

ACKNOWLEDGEMENT

I take immense pleasure in thanking **Ms. Megha Upreti** (**Lecturer ,** CS, GEHU Bhimtal Campus) for allowing us to carry out this project work under his excellent and optimistic supervision. This has all been possible due to her 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 **"Prof. (Dr.) Kamal Ghanshala"** for providing us all infrastructure and facilities to work in need without which this work would not be possible.

YASH JOSHI

STUDENT'S DECLARATION

I, **Yash Joshi** hereby declare the work, which is being presented in the report, entitled **Term workof Compiler Design Lab (PCS-601)** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (Computer Science)** in the session **2023-2024** for **semester VI**, is an authentic record of my own work carried out under the supervision of **Ms. Megha Upreti** (Lecturer , Graphic Era Hill University, Bhimtal)

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

Date: …… …….

(Full signature of student)

Graphic Era HILL UNIVERSITY
University under section 2(f) of UGC Act, 1956
Bhimtal Campus

Department of Computer Science and Engineering
COMPILER DESIGN LAB (PCS-601)

Requirements: Windows/Linux based Computer System

Index/List of Practicals

Sl. No.	Name of Practicals	Date	Page No.	Teacher's Remark & Signature
1	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.	08-02-2024		
2	Write a program in C or C++ language to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.	15-02-2024		
3	Write a C or C++ program to convert NFA to its equivalent DFA.	29-02-2024		
4	Write a C or C++ program to convert RE to its equivalent NFA.	07-03-2024		
5	Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.	14-03-2024		
6	Write a program in C or C++ language to implement Predictive Parsing Algorithm.	21-03-2024		
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.	02-05-2024		
8	Write a program in C or C++ language to implement LR Parser.	09-05-2024		
9	Write a program in C or C++ to generate the three-address code.	16-05-2024		
10	Write a program in C or C++ to generate machine code from the abstract syntax tree generated by the parser.	30-05-2024		

1.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;
}
```

2. Write a program in C or C++ language to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.

```
#include <iostream>
#include     <string>
#include <cctype>
#include  <unordered_set>

// Set of keywords in C++
const std::unordered_set<std::string> keywords = {
      "auto", "break", "case", "char", "const", "continue",
"default", "do",
      "double",  "else",  "enum",  "extern",   "float",   "for",
"goto", "if",
      "int",  "long",  "register",  "return",  "short",  "signed",
"sizeof",
      "static", "struct", "switch", "typedef", "union",
"unsigned", "void",
      "volatile",  "while", "bool",  "catch", "class",
"const_cast",
      "delete",   "dynamic_cast",    "explicit",   "export",   "false",
"friend",
      "inline",   "mutable",   "namespace",   "new",   "operator",
"private",
      "protected",    "public",    "reinterpret_cast",     "static_cast",
"template",
      "this",   "throw",   "true",   "try",   "typeid",   "typename",
"using",
      "virtual",      "wchar_t"
};

enum TokenType {
       IDENTIFIER, KEYWORD,  NEWLINE,  TAB,  WHITESPACE,  CHARACTER
};

void  printToken(TokenType type, const std::string& token) {
      switch (type) {
            case IDENTIFIER:
                  std::cout << "Identifier: " << token << std::endl;
                  break;
            case KEYWORD:
                  std::cout << "Keyword: " << token << std::endl;
                  break;
            case NEWLINE:
                  std::cout << "Newline" << std::endl;
                  break;
```

```
            case TAB:
                std::cout << "Tab" << std::endl;
                break;
            case WHITESPACE:
                std::cout << "Whitespace" << std::endl;
                break;
            case CHARACTER:
                std::cout << "Character: " << token << std::endl;
                break;
        }
    }

    void tokenize(const std::string& line) {
        std::string token;
        for (size_t i = 0; i < line.length(); ++i) {
            char c = line[i];

            if (std::isspace(c)) {
                if (!token.empty()) {
                    if (keywords.find(token) != keywords.end()) {
                        printToken(KEYWORD, token);
                    } else {
                        printToken(IDENTIFIER, token);
                    }
                    token.clear();
                }

                if (c == '\n') {
                    printToken(NEWLINE, "\\n");
                } else if (c == '\t') {
                    printToken(TAB, "\\t");
                } else {
                    printToken(WHITESPACE, " ");
                }
            } else if (std::isalpha(c) || c == '_') {
                token += c;
            } else if (std::isdigit(c)) {
                token += c;
            } else {
                if (!token.empty()) {
                    if (keywords.find(token) != keywords.end()) {
                        printToken(KEYWORD, token);
                    } else {
                        printToken(IDENTIFIER, token);
                    }
                    token.clear();
```

```
                }
                printToken(CHARACTER,   std::string(1,   c));
            }
        }

        if (!token.empty()) {
            if (keywords.find(token) != keywords.end()) {
                printToken(KEYWORD, token);
            } else {
                printToken(IDENTIFIER,    token);
            }
        }
}

int main() {
    std::string line;
    std::cout << "Enter your code (end input with an empty
line):" << std::endl;

    while (true) {
        std::getline(std::cin, line);
        if (line.empty()) break; // End input on empty line
        tokenize(line);
        std::cout   <<   std::endl;
    }

    return    0;
}
```

3. Write a C or C++ program to convert NFA to its equivalent DFA.

```
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <queue>

using namespace std;

// Structure to represent an NFA state
struct NFAState {
   set<int> states; // Set of NFA states
};

// Function to convert NFA to DFA
void convertNFAToDFA(const vector<vector<set<int>>>& transitions, const set<int>&
nfaFinalStates, int nfaStartState) {
   // Initialize DFA start state
   set<int> dfaStartState;
   dfaStartState.insert(nfaStartState);

   // Queue for unmarked DFA states
   queue<set<int>> unmarkedStates;
   unmarkedStates.push(dfaStartState);

   // Map to store DFA states mapping to their corresponding NFA states
   map<set<int>, int> dfaStatesMap;
   int dfaStateCount = 0;
   dfaStatesMap[dfaStartState] = dfaStateCount++;

   // DFA transitions
   map<pair<set<int>, char>, set<int>> dfaTransitions;

   while (!unmarkedStates.empty()) {
      set<int> currentState = unmarkedStates.front();
      unmarkedStates.pop();

      // Iterate over input symbols
      for (char symbol = 'a'; symbol <= 'z'; symbol++) {
      set<int> nextState;

         // Compute epsilon closure of current state
         queue<int> epsilonClosureQueue;
         for (int state : currentState) {
         epsilonClosureQueue.push(state);
         }
         while (!epsilonClosureQueue.empty()) { int
         state = epsilonClosureQueue.front();
         epsilonClosureQueue.pop();
```

```
            nextState.insert(state);
            for (int next : transitions[state][symbol]) {
            if (transitions[state]['e'].count(next)) {
                  epsilonClosureQueue.push(next); }
            }
         }

         // Compute next state
         for (int state : nextState) {
            for (int next : transitions[state][symbol]) {
            nextState.insert(next);
            }
         }

         if (!nextState.empty()) {
            // Add new DFA state if not already present
            if (dfaStatesMap.find(nextState) == dfaStatesMap.end()) {
            dfaStatesMap[nextState] = dfaStateCount++;
            unmarkedStates.push(nextState);
            }

            // Add transition to DFA transitions
            dfaTransitions[{currentState, symbol}] =  nextState;
         }
      }
   }

   // Print DFA
   cout << "DFA States:\n";
   for (auto state : dfaStatesMap) {
      cout << "{ ";
      for (int s : state.first) {
      cout << s << " ";
      }
      cout << "} -> DFA State " << state.second << endl;
   }

   // Print DFA transitions
   cout << "\nDFA Transitions:\n";
   for (auto transition : dfaTransitions) {
      cout << "DFA State " << dfaStatesMap[transition.first.first] << " -- " <<
transition.first.second << " --> ";
      cout << "{ ";
      for (int s : transition.second) {
      cout << s << " ";
      }
      cout << "} (DFA State " << dfaStatesMap[transition.second] << ")" << endl;
   }
```

```
    // Print DFA start state
    cout << "\nDFA Start State: DFA State " << dfaStatesMap[dfaStartState] << endl;

    // Print DFA final states
    cout << "DFA Final States:\n";
    for (auto state : dfaStatesMap) {
        for (int s : state.first) {
            if (nfaFinalStates.count(s)) {
            cout << "{ ";
                for (int s : state.first) {
                cout << s << " ";
                }
                cout << "} (DFA State " << state.second << ")" << endl;
                break;
            }
        }
}
}

int main() {
    // Example NFA
    int numStates = 3;
    int nfaStartState = 0;
    set<int> nfaFinalStates = {2};
    vector<vector<set<int>>> transitions(numStates, vector<set<int>>(256));

    transitions[0]['a'].insert(1);
    transitions[1]['b'].insert(2);
    transitions[2]['c'].insert(0);
    transitions[2]['e'].insert(1);

    // Convert NFA to DFA
    convertNFAToDFA(transitions, nfaFinalStates, nfaStartState);

    return 0;
}
```

4. Write a C or C++ program to convert RE to its equivalent NFA.

```
#include <iostream>
#include <vector>
#include <set>
#include <map>
#include <queue>

using namespace std;

// Structure to represent an NFA state
struct NFAState {
   set<int> states; // Set of NFA states
};

// Function to convert NFA to DFA
void convertNFAToDFA(const vector<vector<set<int>>>& transitions, const set<int>&
nfaFinalStates, int nfaStartState) {
   // Initialize DFA start state
   set<int> dfaStartState;
   dfaStartState.insert(nfaStartState);

   // Queue for unmarked DFA states
   queue<set<int>> unmarkedStates;
   unmarkedStates.push(dfaStartState);

   // Map to store DFA states mapping to their corresponding NFA states
   map<set<int>, int> dfaStatesMap;
   int dfaStateCount = 0;
   dfaStatesMap[dfaStartState] = dfaStateCount++;

   // DFA transitions
   map<pair<set<int>, char>, set<int>> dfaTransitions;

   while (!unmarkedStates.empty()) {
      set<int> currentState = unmarkedStates.front();
      unmarkedStates.pop();

      // Iterate over input symbols
      for (char symbol = 'a'; symbol <= 'z'; symbol++) {
      set<int> nextState;

         // Compute epsilon closure of current state
         queue<int> epsilonClosureQueue;
         for (int state : currentState) {
         epsilonClosureQueue.push(state);
         }
         while (!epsilonClosureQueue.empty()) { int
         state = epsilonClosureQueue.front();
         epsilonClosureQueue.pop();
```

```
            nextState.insert(state);
            for (int next : transitions[state][symbol]) {
            if (transitions[state]['e'].count(next)) {
                  epsilonClosureQueue.push(next); }
            }
         }

         // Compute next state
         for (int state : nextState) {
            for (int next : transitions[state][symbol]) {
            nextState.insert(next);
            }
         }

         if (!nextState.empty()) {
            // Add new DFA state if not already present
            if (dfaStatesMap.find(nextState) == dfaStatesMap.end()) {
            dfaStatesMap[nextState] = dfaStateCount++;
            unmarkedStates.push(nextState);
            }

            // Add transition to DFA transitions
            dfaTransitions[{currentState, symbol}] =  nextState;
         }
      }
   }

   // Print DFA
   cout << "DFA States:\n";
   for (auto state : dfaStatesMap) {
      cout << "{ ";
      for (int s : state.first) {
      cout << s << " ";
      }
      cout << "} -> DFA State " << state.second << endl;
   }

   // Print DFA transitions
   cout << "\nDFA Transitions:\n";
   for (auto transition : dfaTransitions) {
      cout << "DFA State " << dfaStatesMap[transition.first.first] << " -- " <<
transition.first.second << " --> ";
      cout << "{ ";
      for (int s : transition.second) {
      cout << s << " ";
      }
      cout << "} (DFA State " << dfaStatesMap[transition.second] << ")" << endl;
   }
```

```
    // Print DFA start state
    cout << "\nDFA Start State: DFA State " << dfaStatesMap[dfaStartState] << endl;

    // Print DFA final states
    cout << "DFA Final States:\n";
    for (auto state : dfaStatesMap) {
        for (int s : state.first) {
            if (nfaFinalStates.count(s)) {
            cout << "{ ";
                for (int s : state.first) {
                cout << s << " ";
                }
                cout << "} (DFA State " << state.second << ")" << endl;
                break;
            }
        }
}
}

int main() {
    // Example NFA
    int numStates = 3;
    int nfaStartState = 0;
    set<int> nfaFinalStates = {2};
    vector<vector<set<int>>> transitions(numStates, vector<set<int>>(256));

    transitions[0]['a'].insert(1);
    transitions[1]['b'].insert(2);
    transitions[2]['c'].insert(0);
    transitions[2]['e'].insert(1);

    // Convert NFA to DFA
    convertNFAToDFA(transitions, nfaFinalStates, nfaStartState);

    return 0;
}
```

5. Write a Lex program to generate tokens as identifiers, keywords, newline, tabs, whitespaces and characters.

```
%{
#include    <stdio.h>
%}

%option noyywrap

%%
int|return|if|else|while { printf("Keyword: %s\n", yytext); }
[a-zA-Z_][a-zA-Z0-9_]*  { printf("Identifier: %s\n", yytext); }
\n                  { printf("Newline\n"); }
\t                  { printf("Tab\n"); }
[ ]+                 { printf("Whitespace\n"); }
                    {  printf("Character:  %s\n",  yytext);  }
%%

int main() {
yylex();
   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[variable].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;
}
```