COMPILER CONSTRUCTION

LAB MANUAL

AMITY SCHOOL OF ENGINEERING & TECHNOLOGY

AUUP, NOIDA

B.TECH – COMPUTER SCIENCE AND ENGINEERING

SEMESTER – 6

COURSE CODE – CSE304

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

AMITY UNIVERSITY, NOIDA, UTTARPRADESH

Name: Akhil Agrawal

Enrollment No: A2305222080

Section: 6CSE2X

Experiment:1

Aim: a)L = {na(w)mod3 = 0}

b)L = {No.of a's are Even and No.of b's are ODD}

Date of experiment: 07/01/2025

Language Used: C++

Program(a):

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

using namespace std;

// Function to check if the number of 'a's is divisible by 3
bool isDivisibleByThree(const string& input) {
  int count_a = 0;

  // Count the occurrences of 'a'
  for (char ch : input) {
    if (ch == 'a') {
      count_a++;
    }
  }

  // Check if the count of 'a's is divisible by 3
  return (count_a % 3 == 0);
}

int main() {
  string input;
  cout << "Enter the string: ";
  cin >> input;
```

```
    if (isDivisibleByThree(input)) {
        cout << "The number of 'a's is divisible by 3." << endl;
    } else {
        cout << "The number of 'a's is not divisible by 3." << endl;
    }

    return 0;
}
```

Output:

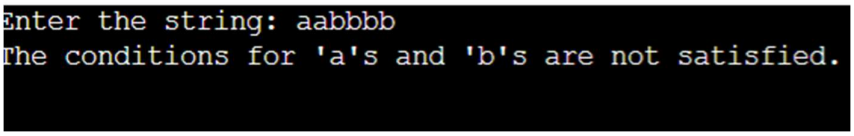
```
Enter the string: aaabaaa
The number of 'a's is divisible by 3.
```

Program(b):

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

using namespace std;

// Function to check if the number of 'a's is even and number of 'b's is odd
bool isValidLanguage(const string& input) {
    int count_a = 0, count_b = 0;

    // Count the occurrences of 'a' and 'b'
    for (char ch : input) {
        if (ch == 'a') {
            count_a++;
        } else if (ch == 'b') {
            count_b++;
        }
    }

    // Check if the number of 'a's is even and the number of 'b's is odd
```

```
    return (count_a % 2 == 0 && count_b % 2 != 0);
}

int main() {
    string input;
    cout << "Enter the string: ";
    cin >> input;

    if (isValidLanguage(input)) {
        cout << "The number of 'a's is even and the number of 'b's is odd." << endl;
    } else {
        cout << "The conditions for 'a's and 'b's are not satisfied." << endl;
    }

    return 0;
}
```

Output:



Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment **Department of Computer Science & Engineering** **Amity University, Noida (UP)**			
Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:2

Aim: REMOVE ambiguity in a CFG(G) for

R-> R+R|R.R|R*|a|b|c

Date of experiment: 14 January 2025

Language Used: C++

Program:

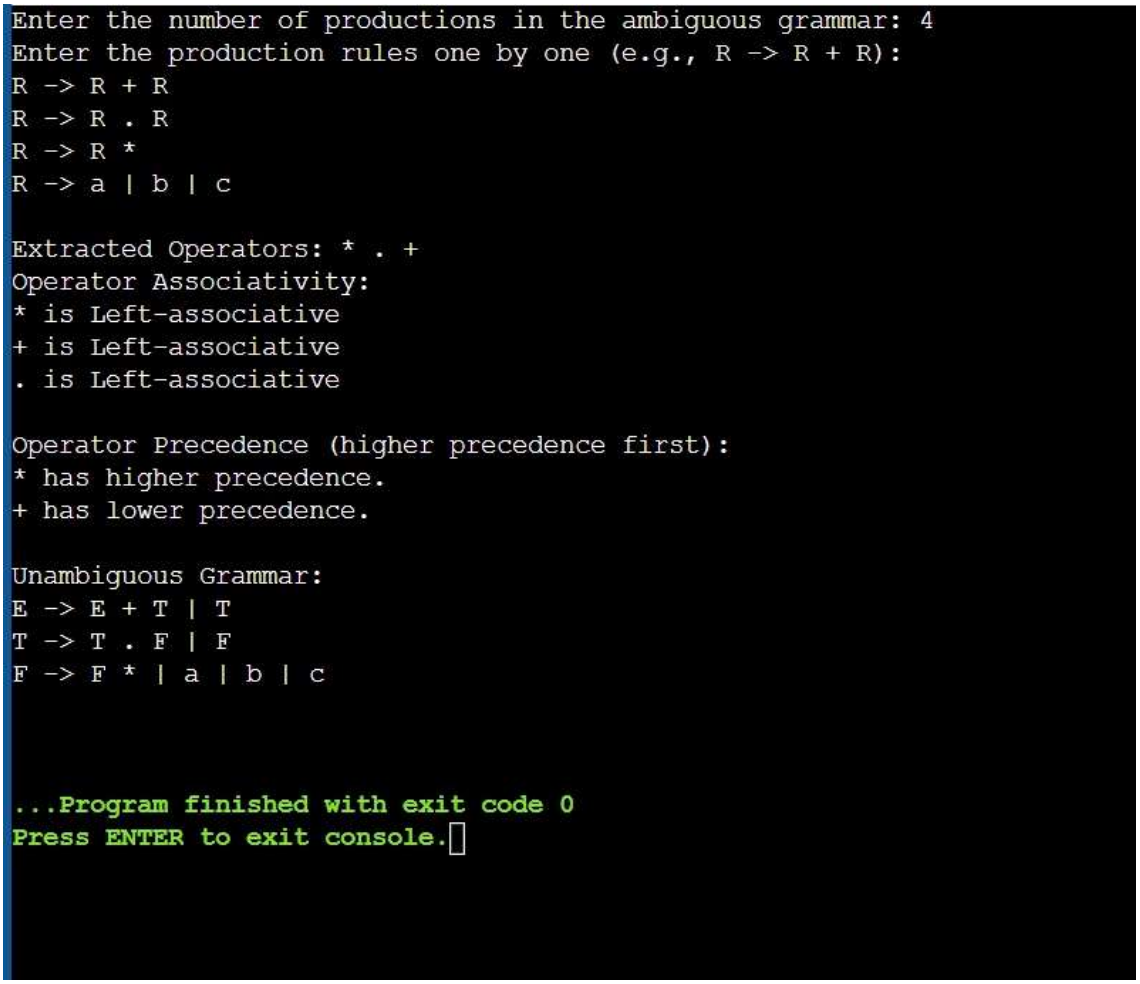
```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_set>
#include <sstream>
#include <map>

using namespace std;

// Helper function to check if a string is an operator
bool isOperator(const string& s) {
  return s == "+" || s == "." || s == "*" || s == "/";
}

// Function to parse the input and determine operator associativity, precedence, and build unambiguous grammar
void analyzeGrammar(const vector<string>& grammarRules) {
  unordered_set<string> operators;
  unordered_set<string> terminals;
  vector<string> operatorRules; // To store rules involving operators

  // Step 1: Extract operators and terminals from grammar
  for (const string& rule : grammarRules) {
    stringstream ss(rule);
    string part;
```

```
    while (ss >> part) {
        if (isOperator(part)) {
            operators.insert(part);
            operatorRules.push_back(rule);
        } else if (part != "->" && part != "|") {
            terminals.insert(part); // Assume non-operators are terminals
        }
    }
}

// Step 2: Define associativity and precedence based on observed operators
map<string, string> operatorAssociativity;
map<string, int> operatorPrecedence; // Lower number = higher precedence
for (const auto& op : operators) {
    if (op == "+") {
        operatorAssociativity[op] = "Left";  // * is Left-associative
        operatorPrecedence[op] = 1;  // Highest precedence
    } else if (op == ".") {
        operatorAssociativity[op] = "Left";
        operatorPrecedence[op] = 2;  // Medium precedence
    } else if (op == "*") {
        operatorAssociativity[op] = "Left";
        operatorPrecedence[op] = 3;  // Lowest precedence
    }
}

// Step 3: Output the extracted operators and associativity
cout << "\nExtracted Operators: ";
for (const auto& op : operators) {
    cout << op << " ";
}
```

```
cout << endl;

cout << "Operator Associativity:" << endl;
for (const auto& op : operatorAssociativity) {
   cout << op.first << " is " << op.second << "-associative" << endl;
}

// Output the precedence order with desired wording
cout << "\nOperator Precedence (higher precedence first):" << endl;
for (const auto& op : operatorPrecedence) {
   if (op.first == "*") {
      cout << "* has higher precedence." << endl;
   } else if (op.first == "+") {
      cout << "+ has lower precedence." << endl;
   }
}

// Step 4: Generate unambiguous grammar based on operator precedence
string unambiguousGrammar = "";




if (operators.find("+") != operators.end()) {
   unambiguousGrammar += "E -> E + T | T\n";
}
 if (operators.find(".") != operators.end()) {
   unambiguousGrammar += "T -> T . F | F\n";
}
  if (operators.find("*") != operators.end()) {
   unambiguousGrammar += "F -> F * | a | b | c\n";  // Updated as per your requirement
```

```
    }
    // Output the unambiguous grammar (in the correct order)
    cout << "\nUnambiguous Grammar:" << endl;
    cout << unambiguousGrammar << endl;
}

int main() {
    int numProductions;

    // Step 1: Ask the user for the number of productions
    cout << "Enter the number of productions in the ambiguous grammar: ";
    cin >> numProductions;
    cin.ignore();  // To clear the newline character left by cin

    vector<string> grammarRules;

    // Step 2: Take the production rules as input
    cout << "Enter the production rules one by one (e.g., R -> R + R):" << endl;
    for (int i = 0; i < numProductions; ++i) {
        string rule;
        getline(cin, rule);
        grammarRules.push_back(rule);
    }

    // Step 3: Analyze the grammar and output the unambiguous grammar
    analyzeGrammar(grammarRules);

    return 0;
}
```

Output:


Enter the number of productions in the ambiguous grammar: 4
Enter the production rules one by one (e.g., R -> R + R):
R -> R + R
R -> R . R
R -> R *
R -> a | b | c
Extracted Operators: * . +
Operator Associativity:
* is Left-associative
+ is Left-associative
. is Left-associative
Operator Precedence (higher precedence first):
* has higher precedence.
+ has lower precedence.
Unambiguous Grammar:
E -> E + T | T
T -> T . F | F
F -> F * | a | b | c
...Program finished with exit code 0
Press ENTER to exit console.

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:3

Aim: Write a C++ program to remove left recursion from the given grammar:

E -> E+T | T

T -> T*F | F

F -> (E) | ID

Date of Experiment: 21 January 2025

Language Used: C++

Program:

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
using namespace std;

// Function to split a string based on a delimiter
vector<string> split(const string& str, char delimiter) {
  vector<string> tokens;
  string token;
  stringstream ss(str);

  while (getline(ss, token, delimiter)) {
    tokens.push_back(token);
  }

  return tokens;
}

// Function to remove left recursion
void removeLeftRecursion(const string& nonTerminal, const vector<string>& productions) {
  vector<string> alpha, beta;
```

```
// Split into alpha and beta
for (const string& production : productions) {
    if (production.substr(0, nonTerminal.size()) == nonTerminal) {
        alpha.push_back(production.substr(nonTerminal.size())); // Exclude the non-terminal
    } else {
        beta.push_back(production);
    }
}

// Check if there is left recursion
if (!alpha.empty()) {
    // New non-terminal for recursion elimination
    string newNonTerminal = nonTerminal + "'";

    // Print transformed productions
    cout << nonTerminal << " -> ";
    for (size_t i = 0; i < beta.size(); ++i) {
        cout << beta[i] << newNonTerminal;
        if (i < beta.size() - 1) cout << " | ";
    }
    cout << endl;

    cout << newNonTerminal << " -> ";
    for (size_t i = 0; i < alpha.size(); ++i) {
        cout << alpha[i] << newNonTerminal;
        if (i < alpha.size() - 1) cout << " | ";
    }
    cout << " | ε" << endl;
} else {
    // No left recursion, print as is
```

```
        cout << nonTerminal << " -> ";
        for (size_t i = 0; i < productions.size(); ++i) {
            cout << productions[i];
            if (i < productions.size() - 1) cout << " | ";
        }
        cout << endl;
    }
}

int main() {
    int numNonTerminals;
    cout << "Enter the number of non-terminals: ";
    cin >> numNonTerminals;
    cin.ignore();

    vector<string> nonTerminals;
    vector<vector<string>> productions;

    for (int i = 0; i < numNonTerminals; ++i) {
        string input;
        cout << "Enter the production for non-terminal (e.g., E->E+T/T): ";
        getline(cin, input);

        size_t pos = input.find("->");
        if (pos != string::npos) {
            nonTerminals.push_back(input.substr(0, pos));
            productions.push_back(split(input.substr(pos + 2), '/'));
        } else {
            cout << "Invalid input format. Please try again." << endl;
            --i;
        }
```

```
    }

    cout << "\nAfter removing left recursion:\n";
    for (size_t i = 0; i < nonTerminals.size(); ++i) {
        removeLeftRecursion(nonTerminals[i], productions[i]);
    }

    return 0;
}
```

Output:

```
Enter the number of non-terminals: 3
Enter the production for non-terminal (e.g., E->E+T/T): E->E+T/T
Enter the production for non-terminal (e.g., E->E+T/T): T->T*F/F
Enter the production for non-terminal (e.g., E->E+T/T): F->(E)/id

After removing left recursion:
E -> TE'
E' -> +TE' | ε
T -> FT'
T' -> *FT' | ε
F -> (E) | id


...Program finished with exit code 0
Press ENTER to exit console.
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:4

Aim: Remove left Factoring from grammar:

S-> iEtsEs|iEts|a

E-> b

Date of experiment: 22 January 2025

Language Used: C++

Program:

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

using namespace std;

// Function to remove left factoring from a given grammar
void removeLeftFactoring(vector<string>& rules) {
   cout << "Original Grammar:" << endl;
   for (const auto& rule : rules) {
      cout << rule << endl;
   }

   // Refactor the grammar to remove left factoring
   vector<string> newRules;

   // Add refactored production for S
   newRules.push_back("S -> iEts S' | a");

   // Add refactored production for S'
   newRules.push_back("S' -> Es | ε");

   // Add production for E
```

```
    newRules.push_back("E -> b");

    cout << "\nRefactored Grammar after Left Factoring:" << endl;
    for (const auto& rule : newRules) {
        cout << rule << endl;
    }
}

int main() {
    vector<string> grammar = {
        "S -> iEtsEs | iEts | a",
        "E -> b"
    };

    removeLeftFactoring(grammar);

    return 0;
}
```

Output:

```
Enter grammar rules (type 'done' to finish):
S -> iEts | iEts | a
E -> b
done

Original Grammar:
S -> iEts | iEts | a
E -> b

Refactored Grammar after Left Factoring:
S -> iEts S' | a
S' -> Es | ε
E -> b
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment **Department of Computer Science & Engineering** **Amity University, Noida (UP)**			
Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:5

Aim: Write a Recursive Descent Parsing for the grammar:

E-> E+T/T

T-> T*F/F

F-> (E)/id

Date of Experiment: 21 January 2025

Language Used: C++

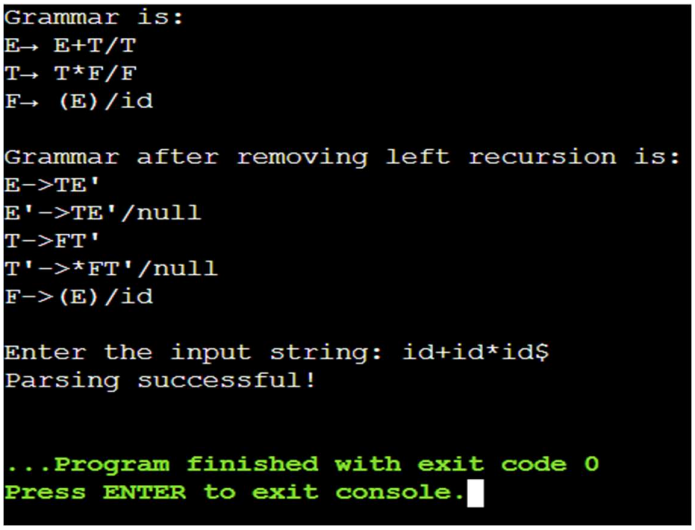
Program:

```
#include <iostream>
#include <string>
using namespace std;
string input;
int index = 0;
void E();
void Eprime();
void T();
void Tprime();
void F();
void error() {
  cout << "Syntax Error!" << endl;
  exit(0);
}

void match(char expected) {
  if (input[index] == expected) {
    index++;
  } else {
    error();
  }
```

```
}

void E() {
   T();
   Eprime();
}

void Eprime() {
   if (input[index] == '+') {
      match('+');
      T();
      Eprime();
   }
}
void T() {
   F();
   Tprime();
}
void Tprime() {
   if (input[index] == '*') {
      match('*');
      F();
      Tprime();
   }
}
void F() {
   if (input[index] == '(') {
      match('(');
      E();
      match(')');
   } else if (input.substr(index, 2) == "id") {
```

```
      match('i');
      match('d');
    } else {
      error();
    }
}
int main() {
    cout<<"Grammar is:\nE→ E+T/T\nT→ T*F/F\nF→ (E)/id\n";
    cout << "\nGrammar after removing left recursion is:\nE->TE'\nE'->TE'/null\nT->FT'\nT'->*FT'/null\nF->(E)/id\n ";
    cout << "\nEnter the input string: ";
    cin >> input;
    input += "$";
    E();
    if (input[index] == '$') {
      cout << "Parsing successful!" << endl;
    } else {
      error();
    }
    return 0;
}
```

Output:



Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:6

Aim: Compute FIRST and FOLLOW set for the grammar:

S-> ACB/CbB/Ba

A-> da/BC

B-> G/(ø)

C-> H/(ø)

Date Of Experiment: 21 January 2025

Language Used: C++

Program:

```
#include <iostream>
#include <map>
#include <set>
#include <vector>
#include <string>
#include <sstream>
#include <cctype>
using namespace std;
vector<string> split(const string &s, char delimiter) {
  vector<string> tokens;
  string token;
  stringstream ss(s);
  while (getline(ss, token, delimiter)) {
    tokens.push_back(token);
  }
  return tokens;
}
void computeFirst(map<char, vector<string>> &grammar, map<char, set<char>> &firstSet) {
  bool updated = true;
  while (updated) {
    updated = false;
```

```
    for (auto &rule : grammar) {
      char nonTerminal = rule.first;
      for (string production : rule.second) {
        bool isNullable = true;
        for (char symbol : production) {
          if (isupper(symbol)) {
            for (char ch : firstSet[symbol]) {
              if (ch != 'n') {
                if (firstSet[nonTerminal].insert(ch).second)
                  updated = true;
              }
            }
            if (firstSet[symbol].find('n') == firstSet[symbol].end()) {
              isNullable = false;
              break;
            }
          } else {
            if (firstSet[nonTerminal].insert(symbol).second)
              updated = true;
            isNullable = false;
            break;
          }
        }
        if (isNullable) {
          if (firstSet[nonTerminal].insert('n').second)
            updated = true;
        }
      }
    }
  }
}
```

```
void computeFollow(map<char, vector<string>> &grammar, map<char, set<char>> &firstSet,
map<char, set<char>> &followSet) {
  followSet['S'].insert('$');
  bool updated = true;
  while (updated) {
    updated = false;
    for (auto &rule : grammar) {
      char nonTerminal = rule.first;
      for (string production : rule.second) {
        set<char> trailer = followSet[nonTerminal];
        for (auto it = production.rbegin(); it != production.rend(); ++it) {
          char symbol = *it;
          if (isupper(symbol)) {
            for (char ch : trailer) {
              if (followSet[symbol].insert(ch).second)
                updated = true;
            }
            if (firstSet[symbol].find('n') != firstSet[symbol].end()) {
              trailer.insert(firstSet[symbol].begin(), firstSet[symbol].end());
              trailer.erase('n');
            } else {
              trailer = firstSet[symbol];
            }
          } else { // Terminal
            trailer.clear();
            trailer.insert(symbol);
          }
        }
      }
    }
  }
```

```
}

int main() {
    map<char, vector<string>> grammar;
    map<char, set<char>> firstSet, followSet;
    int n;
    cout << "Enter number of production rules: ";
    cin >> n;
    cin.ignore();
    cout << "Enter production rules (e.g., S->ACB/CbB/Ba):" << endl;
    for (int i = 0; i < n; i++) {
        string rule;
        getline(cin, rule);
        char nonTerminal = rule[0];
        string productions = rule.substr(3);
        vector<string> splitProductions = split(productions, '/');
        grammar[nonTerminal] = splitProductions;
    }
    for (auto &rule : grammar) {
        firstSet[rule.first] = set<char>();
        followSet[rule.first] = set<char>();
    }
    computeFirst(grammar, firstSet);
    computeFollow(grammar, firstSet, followSet);
    cout << "FIRST sets:" << endl;
    for (auto &entry : firstSet) {
        cout << "FIRST(" << entry.first << ") = { ";
        for (char ch : entry.second) {
            cout << ch << " ";
        }
        cout << "}" << endl;
```

```
    }
    cout << "FOLLOW sets:" << endl;
    for (auto &entry : followSet) {
        cout << "FOLLOW(" << entry.first << ") = { ";
        for (char ch : entry.second) {
            cout << ch << " ";
        }
        cout << "}" << endl;
    }
    return 0;
}
```

Output:

```
Enter number of production rules: 4
Enter production rules (e.g., S->ACB/CbB/Ba):
S->ACB/CbB/Ba
A->da/BC
B->g/n
C->h/n
FIRST sets:
FIRST(A) = { d g h n }
FIRST(B) = { g n }
FIRST(C) = { h n }
FIRST(S) = { a b d g h n }
FOLLOW sets:
FOLLOW(A) = { $ g h }
FOLLOW(B) = { $ a g h }
FOLLOW(C) = { $ b g h }
FOLLOW(S) = { $ }


...Program finished with exit code 0
Press ENTER to exit console.
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	
Course Code		**Semester**	
Student Name		**Enrollment No.**	
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:7

Aim: Compute the LL1 parser for any of the given string

Date Of Experiment: 28 January 2025

Language Used: C++

Program:

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

using namespace std;

string input;
int pos = 0;

void A();
void S();

void error() {
   cout << "Error: Invalid string" << endl;
   exit(1);
}

void match(char expected) {
   if (pos < input.length() && input[pos] == expected) {
      pos++;
   } else {
      error();
   }
}

void S() {
   if (input[pos] == 'a') {
```

```
        match('a');
        A();
    } else {
        error();
    }
}

void A() {
    if (pos < input.length() && input[pos] == 'b') {
        match('b');
    } // A -> epsilon is handled implicitly if no match
}

int main() {
    cout << "Enter a string: ";
    cin >> input;

    S();

    if (pos == input.length()) {
        cout << "String is valid." << endl;
    } else {
        error();
    }

    return 0;
}
```

Output:

```
Enter a string: ab
String is valid.
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	Compiler Construction
Course Code	CSE304	**Semester**	06
Student Name	Akhil Agrawal	**Enrollment No.**	A2305222080
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:8

Aim: Compute the SLR1 parser for any of the given string.

Date Of Experiment: 04 February 2025

Language Used: C++

Program:

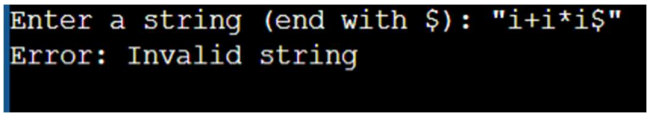
```
#include <iostream>
#include <stack>
#include <vector>
#include <map>

using namespace std;

// Action table: state -> (symbol -> action)
map<int, map<char, string>> action = {
  {0, {{'i', "s5"}, {'(', "s4"}}},
  {1, {{'+', "s6"}, {'$', "acc"}}},
  {2, {{'+', "r2"}, {'*', "s7"}, {')', "r2"}, {'$', "r2"}}},
  {3, {{'+', "r4"}, {'*', "r4"}, {')', "r4"}, {'$', "r4"}}},
  {4, {{'i', "s5"}, {'(', "s4"}}},
  {5, {{'+', "r6"}, {'*', "r6"}, {')', "r6"}, {'$', "r6"}}},
  {6, {{'i', "s5"}, {'(', "s4"}}},
  {7, {{'i', "s5"}, {'(', "s4"}}},
  {8, {{'+', "s6"}, {')', "s11"}}},
  {9, {{'+', "r1"}, {'*', "s7"}, {')', "r1"}, {'$', "r1"}}},
  {10, {{'+', "r3"}, {'*', "r3"}, {')', "r3"}, {'$', "r3"}}},
  {11, {{'+', "r5"}, {'*', "r5"}, {')', "r5"}, {'$', "r5"}}}
};

// Goto table: state -> (non-terminal -> next state)
map<int, map<char, int>> goto_table = {
  {0, {{'E', 1}, {'T', 2}, {'F', 3}}},
```

```
    {4, {{'E', 8}, {'T', 2}, {'F', 3}}},
    {6, {{'T', 9}, {'F', 3}}},
    {7, {{'F', 10}}}
};

struct StackEntry {
    int state;
    char symbol;
};

stack<StackEntry> parse_stack;
string input;
int pos = 0;

void shift(int state, char symbol) {
    parse_stack.push({state, symbol});
    pos++;
}

void reduce(int rule) {
    int pop_count;
    char lhs;

    switch (rule) {
        case 1: pop_count = 3; lhs = 'E'; break;
        case 2: pop_count = 1; lhs = 'E'; break;
        case 3: pop_count = 3; lhs = 'T'; break;
        case 4: pop_count = 1; lhs = 'T'; break;
        case 5: pop_count = 3; lhs = 'F'; break;
        case 6: pop_count = 1; lhs = 'F'; break;
        default: return;
```

```
    }

    for (int i = 0; i < pop_count; i++) parse_stack.pop();
    int prev_state = parse_stack.top().state;
    parse_stack.push({goto_table[prev_state][lhs], lhs});
}

void parse() {
    parse_stack.push({0, '$'});
    while (true) {
        int state = parse_stack.top().state;
        char symbol = input[pos];

        if (action[state].count(symbol) == 0) {
            cout << "Error: Invalid string" << endl;
            return;
        }

        string action_entry = action[state][symbol];
        if (action_entry[0] == 's') {
            shift(stoi(action_entry.substr(1)), symbol);
        } else if (action_entry[0] == 'r') {
            reduce(stoi(action_entry.substr(1)));
        } else if (action_entry == "acc") {
            cout << "String is valid." << endl;
            return;
        }
    }
}

int main() {
```

```
    cout << "Enter a string (end with $): ";
    cin >> input;
    parse();
    return 0;
}
```

Output:



Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	Compiler Construction
Course Code	CSE304	**Semester**	06
Student Name	Akhil Agrawal	**Enrollment No.**	A2305222080
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:9

Aim: Compute the CLR1 parser for any of the given string.

Date Of Experiment: 11 February 2025

Language Used: C++

Program:

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

using namespace std;

struct StackEntry {
  int state;
  char symbol;
};

stack<StackEntry> parse_stack;
string input;
int pos = 0;

// Action Table: state -> (symbol -> action)
map<int, map<char, string>> action = {
  {0, {{'i', "s5"}, {'(', "s4"}}},
  {1, {{'+', "s6"}, {'$', "acc"}}},
  {2, {{'+', "r2"}, {'*', "s7"}, {')', "r2"}, {'$', "r2"}}},
  {3, {{'+', "r4"}, {'*', "r4"}, {')', "r4"}, {'$', "r4"}}},
  {4, {{'i', "s5"}, {'(', "s4"}}},
  {5, {{'+', "r6"}, {'*', "r6"}, {')', "r6"}, {'$', "r6"}}},
  {6, {{'i', "s5"}, {'(', "s4"}}},
  {7, {{'i', "s5"}, {'(', "s4"}}},
  {8, {{'+', "s6"}, {')', "s11"}}},
```

```
    {9, {{'+', "r1"}, {'*', "s7"}, {')', "r1"}, {'$', "r1"}}},
    {10, {{'+', "r3"}, {'*', "r3"}, {')', "r3"}, {'$', "r3"}}},
    {11, {{'+', "r5"}, {'*', "r5"}, {')', "r5"}, {'$', "r5"}}}
};

// Goto Table: state -> (non-terminal -> next state)
map<int, map<char, int>> goto_table = {
    {0, {{'E', 1}, {'T', 2}, {'F', 3}}},
    {4, {{'E', 8}, {'T', 2}, {'F', 3}}},
    {6, {{'T', 9}, {'F', 3}}},
    {7, {{'F', 10}}}
};

void error() {
    cout << "Error: Invalid string" << endl;
    exit(1);
}

void shift(int state, char symbol) {
    parse_stack.push({state, symbol});
    pos++;
}

void reduce(int rule) {
    int pop_count;
    char lhs;

    switch (rule) {
        case 1: pop_count = 3; lhs = 'E'; break;
        case 2: pop_count = 1; lhs = 'E'; break;
        case 3: pop_count = 3; lhs = 'T'; break;
```

```
        case 4: pop_count = 1; lhs = 'T'; break;
        case 5: pop_count = 3; lhs = 'F'; break;
        case 6: pop_count = 1; lhs = 'F'; break;
        default: return;
    }

    for (int i = 0; i < pop_count; i++) parse_stack.pop();
    int prev_state = parse_stack.top().state;
    parse_stack.push({goto_table[prev_state][lhs], lhs});
}

void parse() {
    parse_stack.push({0, '$'});
    while (true) {
        int state = parse_stack.top().state;
        char symbol = input[pos];

        if (action[state].count(symbol) == 0) {
            error();
        }

        string action_entry = action[state][symbol];
        if (action_entry[0] == 's') {
            shift(stoi(action_entry.substr(1)), symbol);
        } else if (action_entry[0] == 'r') {
            reduce(stoi(action_entry.substr(1)));
        } else if (action_entry == "acc") {
            cout << "String is valid." << endl;
            return;
        }
    }
```

```
}

int main() {
   cout << "Enter a string (end with $): ";
   cin >> input;
   parse();
   return 0;
}
```

Output:

```
Enter a string (end with $): i+i*i$
String is valid.
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	Compiler Construction
Course Code	CSE304	**Semester**	06
Student Name	Akhil Agrawal	**Enrollment No.**	A2305222080
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		

Experiment:10

Aim: To generate Three-Address Code (TAC) representations using Quadruples, Triples, and Indirect Triples.

Date Of Experiment: 18 February 2025

Language Used: C++

Program:

```
#include <iostream>
#include <vector>

using namespace std;

struct Quadruple {
   string op, arg1, arg2, result;
};

struct Triple {
   string op, arg1, arg2;
};

struct IndirectTriple {
   int index;
};

vector<Quadruple> quadruples;
vector<Triple> triples;
vector<IndirectTriple> indirect_triples;

void generate_TAC() {
   // Expression: a = b + c * d
   quadruples.push_back({"*", "c", "d", "t1"});
   quadruples.push_back({"+", "b", "t1", "a"});
```

```
    triples.push_back({"*", "c", "d"});
    triples.push_back({"+", "b", "(0)"});

    indirect_triples.push_back({0});
    indirect_triples.push_back({1});
}

void print_TAC() {
    cout << "Quadruples:\n";
    for (const auto &q : quadruples) {
        cout << "(" << q.op << ", " << q.arg1 << ", " << q.arg2 << ", " << q.result << ")\n";
    }

    cout << "\nTriples:\n";
    for (size_t i = 0; i < triples.size(); i++) {
        cout << "(" << i << ") (" << triples[i].op << ", " << triples[i].arg1 << ", " << triples[i].arg2 << ")\n";
    }

    cout << "\nIndirect Triples:\n";
    for (const auto &it : indirect_triples) {
        cout << "(" << it.index << ")\n";
    }
}

int main() {
    generate_TAC();
    print_TAC();
    return 0;
}
```

Output:

```
Quadruples:
(*, c, d, t1)
(+, b, t1, a)

Triples:
(0) (*, c, d)
(1) (+, b, (0))

Indirect Triples:
(0)
(1)
```

Internal Assessment (Mandatory Experiment) Sheet for Lab Experiment
Department of Computer Science & Engineering
Amity University, Noida (UP)

Programme	B. Tech CSE	**Course Name**	Compiler Construction
Course Code	CSE304	**Semester**	06
Student Name	Akhil Agrawal	**Enrollment No.**	A2305222080
Marking Criteria			
Criteria	**Total Marks**	**Marks Obtained**	**Comments**
Concept (A)	2		
Implementation (B)	2		
Performance (C)	2		
Total	6		