**COMPILER CONSTRUCTION**

**LAB MANUAL**

**A logo of a university

Description automatically generated**

**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:** Aryan Nair

**Enrollment No:** A2305222105

**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;

*int* main(){

    string w;

    cout<<"Enter a string 'w' here : ";

    getline(cin, w);

*int* aCount=0;

    for(*char* c: w){

        if(c=='a') aCount++;

    }

    if(aCount%3==0) cout << "The string " << w <<" belongs to the language \u001b[32mL = {na(w)mod3 = 0}\u001b[0m ";

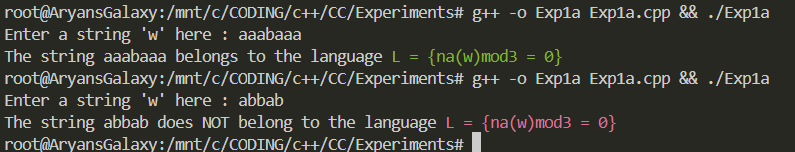
    else cout << "The string " << w <<" does NOT belong to the language \u001b[31mL = {na(w)mod3 = 0}\u001b[0m";

    cout<<endl;

    return 0;

}

**Output:**

****

**Program(b):**

#include<iostream>

#include<string>

using *namespace* std;

*int* main(){

    cout << "Checking if a string is accepted in L = {No.of a's are Even and No.of b's are ODD}\n";

    x:cout<<"Enter a string 'w' here :";

    string w;

    getline(cin,w);

*int* aCount=0,bCount=0;

    for(*char* c: w){

        if(c=='a') aCount++;

        else if(c=='b') bCount++;

        else{

            cout<< "Please enter a valid string containing only 'a' and/or 'b' \n";

            goto x;

        }

    }

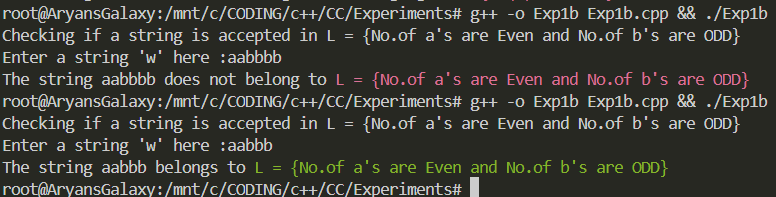
    if(aCount%2==0 && bCount%2!=0) cout << "The string "<<w<<" belongs to \u001b[32mL = {No.of a's are Even and No.of b's are ODD}\u001b[0m\n";

    else cout << "The string "<<w<<" does not belong to \u001b[31mL = {No.of a's are Even and No.of b's are ODD}\u001b[0m\n";

    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>

using *namespace* std;

*int* main(){

    vector<string> gram;

    string inputRule;

    cout << "Enter a grammar rule (type 'ok' to end):" << endl;

    while(1){

        getline(cin, inputRule);

        if(inputRule == "ok"){

            break;

        }

        gram.push\_back(inputRule);

    }

    cout << "\nOriginal Grammar:" << endl;

    for(const *auto*& rule : gram){

        cout << rule << endl;

    }

    vector<string> newRules;

    newRules.push\_back("R -> E");   // new production for R

    newRules.push\_back("E -> E + T | T");   // productions for E (handles the '+' operator)

    newRules.push\_back("T -> T . F | F");   // productions for T (handles the '.' operator)

    newRules.push\_back("F -> F \* | a | b | c");    // productions for F (handles the '\*' operator and terminals a b c )

    cout << "\nRefactored Grammar after Removing Ambiguity:" << endl;

    for(const *auto*& rule : newRules){

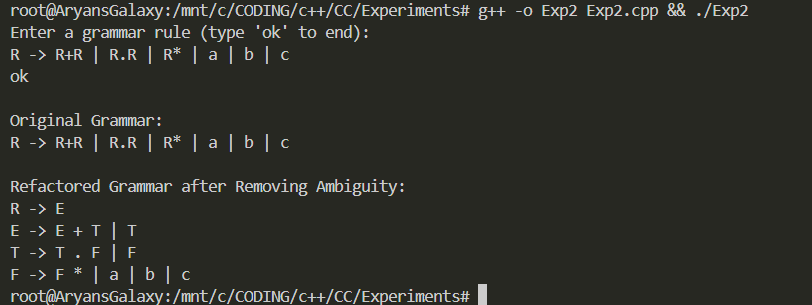
        cout << rule << endl;

    }

    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 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>

using *namespace* std;

*void* displayProductions(string& *Nt*, vector<string>& *productions*){ //non terminal (Nt)

    cout << *Nt* << " -> ";

    for(*auto* i=0; i<*productions*.size();i++){

        cout << *productions*[i];

        if(i != *productions*.size()-1)cout << " | ";

    }

    cout << endl;

}

*void* removeLeftRecursion(string& *Nt*, vector<string>& *productions*){

    vector<string> alpha; // Stores left recursive productions

    vector<string> beta;  // Stores non-left recursive productions

    for(*auto*& prod : *productions*){

        if(prod[0]==*Nt*[0]){

            alpha.push\_back(prod.substr(1)); // Remove the left recursion part

        }

        else{

            beta.push\_back(prod);

        }

    }

    if(alpha.empty()){

        displayProductions(*Nt*, *productions*);

        return;

    }

    // New productions without left recursion

    string newNonTerminal = *Nt* + "'";

    vector<string> newBeta;

    for(*auto*& b : beta){

        newBeta.push\_back(b + newNonTerminal);

    }

    vector<string> newAlpha;

    for(*auto*& a : alpha){

        newAlpha.push\_back(a + newNonTerminal);

    }

    newAlpha.push\_back("ε");

    displayProductions(*Nt*, newBeta);

    displayProductions(newNonTerminal, newAlpha);

}

*int* main(){

*int* numNT;

    cout << "Enter the number of non-terminals: ";

    cin >> numNT;

    vector<string> nonTerminals(numNT);

    vector<vector<string>> productions(numNT);

    for(*int* i=0;i<numNT;i++){

        cout << "Enter non-terminal " << i+1 << ": ";

        cin >> nonTerminals[i];

*int* numProductions;

        cout << "Enter the number of productions for " << nonTerminals[i] << ": ";

        cin >> numProductions;

        productions[i].resize(numProductions);

        cin.ignore(); // Clear newline from the input buffer

        for(*int* j=0;j<numProductions;j++){

            cout << "Enter production " << j + 1 << ": ";

            getline(cin, productions[i][j]);

        }

    }

    cout << "\nOriginal Grammar:\n";

    for(*int* i=0;i<numNT;i++){

        displayProductions(nonTerminals[i], productions[i]);

    }

    cout << "\nGrammar after removing left recursion:\n";

    for(*int* i=0;i<numNT;i++){

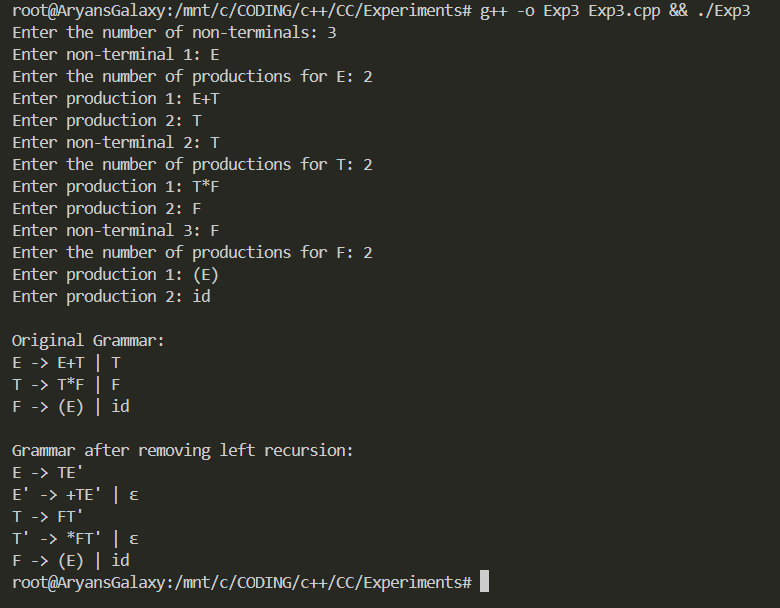
        removeLeftRecursion(nonTerminals[i], productions[i]);

    }

    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 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"

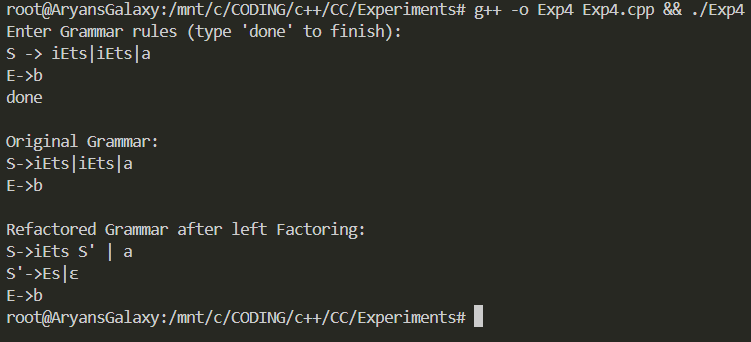
};

removeLeftFactoring(grammar);

return 0;

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **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 {

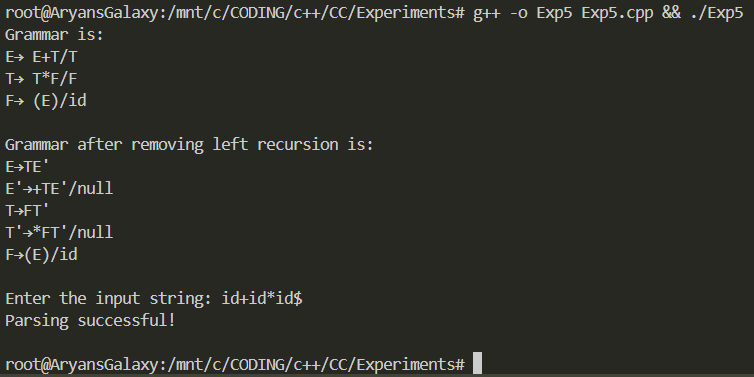
error();

}

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/(∅)

C-> H/(∅)

**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 << " ,";

        }

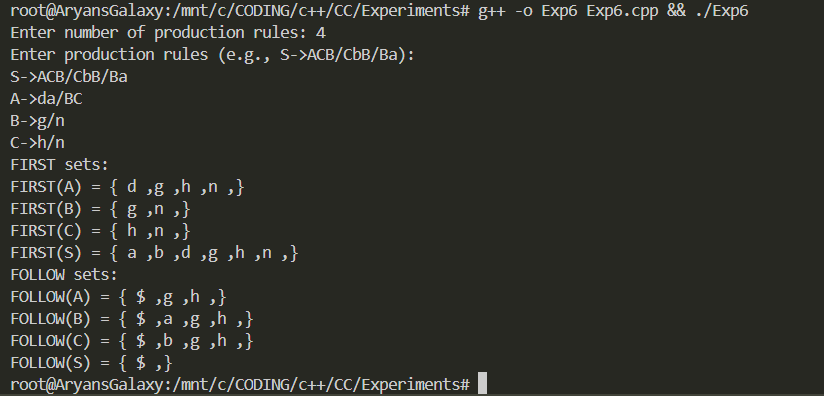
        cout << "}" << endl;

    }

    return 0;

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **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 {

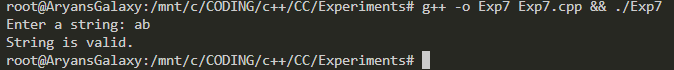
error();

}

return 0;

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **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** | Aryan Nair | **Enrollment No.** | A2305222105 |
| **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 (Use I for id and end with $): ";

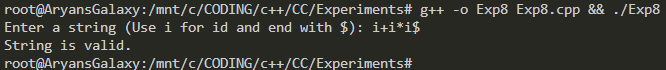
cin >> input;

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** | Aryan Nair | **Enrollment No.** | A2305222105 |
| **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 $): ";

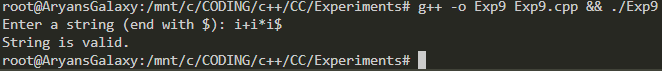
cin >> input;

parse();

return 0;

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **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** | Aryan Nair | **Enrollment No.** | A2305222105 |
| **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() {

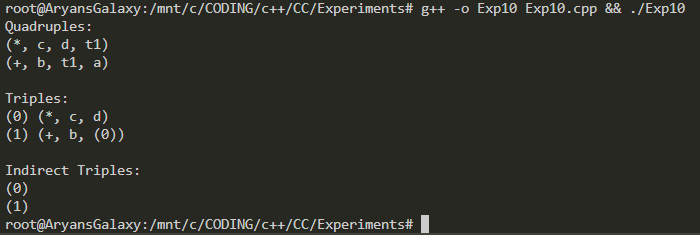
generate\_TAC();

print\_TAC();

return 0;

}

**Output:**

****

|  |  |  |  |
| --- | --- | --- | --- |
| **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** | Aryan Nair | **Enrollment No.** | A2305222105 |
| **Marking Criteria** | | | |
| **Criteria** | **Total Marks** | **Marks Obtained** | **Comments** |
| Concept (A) | 2 |  |  |
| Implementation (B) | 2 |  |  |
| Performance (C) | 2 |  |  |
| Total | 6 |  |  |

|  |
| --- |
|  |
|  |
|  |