Compiler-Design Lab File

Name: **Bushra Shahzad**

Roll No: **21BCS046**

Branch: Computer Engineering

Semester: 6th

Subject Code: CEN 692

**Submitted to: Dr. Sarfaraz Masood**

**Dr. Musheer Ahmad**

**Mr. Jawahar Lal**

Index­­­­

|  |  |  |  |
| --- | --- | --- | --- |
| SR No | Program | Date | Signature |
| **1** | Write a program to implement a Regular expression. The program should read an R.E through a file and should check whether a string given from the console is acceptable by the given R.E. or not. | 24th Jan,24 |  |
| **2** | Write a program to implement Mealy & Moore Machines. The program should read the machine from a file and should generate the corresponding output for a string given from the console. An error must be generated for the case where no valid transition is available. | 31st Jan,24  7th Feb,24 |  |
| **3** | Write a program to implement the conversion of an NFA to a DFA. The program should read an NFA through a file and should generate the corresponding tabular DFA for the Q same. | 14th FEB,24 |  |
| **4** | Write a program to implement a Regular Grammar. The program should read an R.G. through a file and should check whether a string given from the console is acceptable by the given R.G. or not. | 21st Feb,24 |  |
| **5** | Write a program to implement a Context Free Grammar. The program should read the C.F.G. through a file and should check whether a string given from the console is acceptable by the given C.F.G. or not. | 28th Feb,24 |  |
| **6** | Write a program to find out the FIRST & FOLLOW values for a given Context Free Grammar. The program should read the C.F.G. from a file | 20th Mar,24 |  |
| **7** | Write a program that verifies whether a given CFG is suitable for LL(1) parsing or not. If not then the program should convert the given CFG to a form which is suitable for the LL parsing. | 27th Mar,24 |  |
| **8** | Write a program that generates LL(1) parsing table for a given CFG and also performs LL(1) Parsing using the same table. The CFG will be given through a tile and the string to be checked will be given through the console. | 3rd Apr,24 |  |
| **9** | Write a program to find the Leaders and Basic Blocks for a Three Address Code given through a file. | 24th Apr,24 |  |
| **10** | Write a program to find the Flow Graph and the Dominator nodes in a Three Address Code given through a file | 24th Apr,24 |  |
| **11** | Write a program that evaluates GEN & KILL Values for a TAC given through a file. | 1st May,24 |  |
| **12** | Write a program to find the Natural Loops in a Three Address Code given through a file. | 1st May,24 |  |

Program 1

Code: -

#include <iostream>

#include <fstream>

#include <string>

#include <regex>

using namespace std;

int main()

{

    ifstream file("regularExpression.txt");

    if (!file.is\_open())

    {

        cout << "Error opening file" << endl;

        return 1;

    }

    string line;

    string exp;

    while (getline(file, line))

    {

        exp += line;

    }

    cout << "Regular expression is " << exp << endl;

    regex regexp(exp);

    int type;

    while (1)

    {

        cout << "Enter 1 to check for a string." << endl;

        cout << "Enter 2 to exit." << endl;

        cin >> type;

        if (type == 1)

        {

            cout << "Give input string to check with the given regular expression" << endl;

            string input;

            cin >> input;

            if (regex\_match(input, regexp))

            {

                cout << input;

                cout << " string accepted by the given regular expression" << endl;

            }

            else

            {

                cout << input;

                cout << " string not accepted by the given regular expression" << endl;

            }

        }

        else if (type == 2)

        {

            cout << "Exit" << endl;

            break;

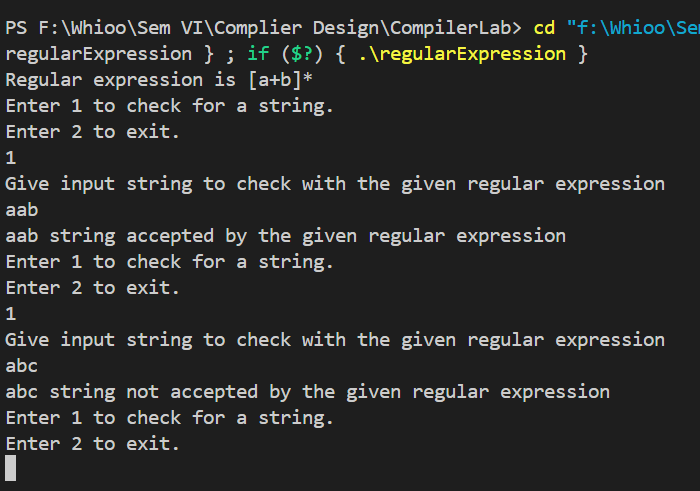
        }

    }

    return 0;

}

Output: -



Program 2

Code: -

Moore Machine

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> dfa;

int initialState;

int convertToInt(string s)

{

    int num = 0;

    for (int i = 0; i < s.size(); i++)

    {

        num = num \* 10 + (s[i] - '0');

    }

    return num;

}

vector<int> stringToVector(string &line)

{

    int i = 0;

    vector<int> v;

    while (i < line.size())

    {

        if (line[i] == '-')

        {

            int x = -1;

            v.push\_back(x);

            i++;

        }

        else if (line[i] != '-' and line[i] != ' ')

        {

            v.push\_back(line[i] - '0');

        }

        i++;

    }

    return v;

}

string isAccepted(vector<vector<int>> &dfa, string input)

{

    string ans = "";

    int currentState = initialState;

    int i = 0;

    int n = input.size();

    int size = dfa[0].size();

    for (int i = 0; i < n; i++)

    {

        if (input[i] - '0' >= size)

            return "Invalid Input";

    }

    cout << "\nTransitions: ";

    cout << "q" << currentState << " ->";

    while (i < input.size() && currentState != -1)

    {

        ans += dfa[currentState][size - 1] + '0';

        currentState = dfa[currentState][input[i] - '0'];

        if (currentState != -1)

            cout << "q" << currentState << " -> ";

        else

            cout << "Dead state";

        i++;

    }

    cout << endl;

    if (currentState == -1)

        return "Not Accepted and the output is " + ans;

    else

    {

        ans += dfa[currentState][size - 1] + '0';

        return "Accepted and the output is  " + ans;

    }

}

int main()

{

    fstream myfile("moore.txt");

    string line;

    if (!myfile.is\_open())

        cout << "Error opening file!\n";

    else

        cout << "File opened successfully!\n";

    cout << "Given DFA - \n";

    while (getline(myfile, line))

    {

        cout << line << endl;

    }

    myfile.clear();

    myfile.seekg(0);

    int currentLine = 0;

    while (getline(myfile, line))

    {

        vector<int> temp;

        if (currentLine == 0)

        {

            initialState = convertToInt(line);

        }

        else

        {

            temp = stringToVector(line);

            dfa.push\_back(temp);

        }

        currentLine++;

    }

    string input;

    cout << "\nEnter input consisting of 0's and 1's: ";

    getline(cin, input);

    string ans = isAccepted(dfa, input);

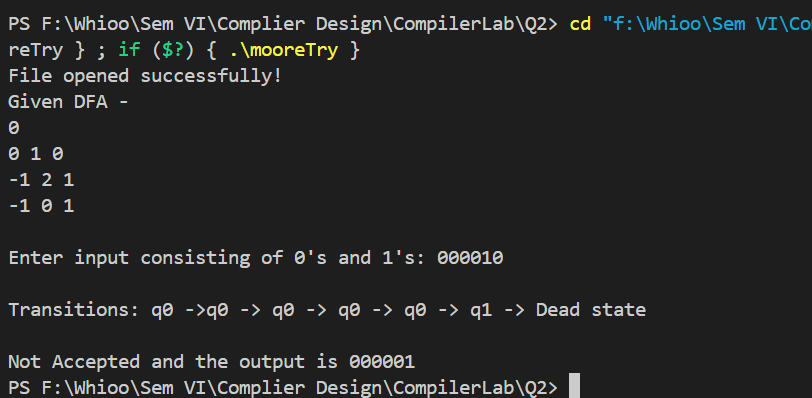
    cout << endl;

    cout << ans << endl;

    return 0;

}

Output: -



A computer screen shot of a program code

Description automatically generated

Mealy machine

#include <bits/stdc++.h>

using namespace std;

vector<vector<int>> dfa;

int initialState;

string isAccepted(vector<vector<int>> &dfa, string input)

{

    string ans = "";

    int currentState = initialState;

    int i = 0;

    int n = input.size();

    int size = dfa[0].size();

    for (int i = 0; i < n; i++)

    {

        if (input[i] - '0' >= size)

            return "Invalid Input";

    }

    cout << "\nTransitions: ";

    cout << "q" << currentState << " ->";

    while (i < input.size() && currentState != -1)

    {

        int currentInput = input[i] - '0';

        if (currentInput == 0)

        {

            if (dfa[currentState][currentInput + 1] != -1)

                ans += (dfa[currentState][currentInput + 1]) + '0'; // dfa[currentState][1]

            currentState = dfa[currentState][currentInput];         // dfa[currentState][0]

        }

        else if (currentInput == 1)

        {

            if (dfa[currentState][currentInput + 2] != -1)

                ans += (dfa[currentState][currentInput + 2]) + '0'; // dfa[currentState][3]

            currentState = dfa[currentState][currentInput + 1];     // dfa[currentState][2]

        }

        if (currentState != -1)

            cout << "q" << currentState << " -> ";

        else

            cout << "Dead state";

        i++;

    }

    cout << endl;

    if (currentState == -1)

        return "Not Accepted and the output is " + ans;

    else

    {

        return "Accepted and the output is  " + ans;

    }

}

int main()

{

    fstream myfile("mealy.txt");

    string line;

    if (!myfile.is\_open())

        cout << "Error opening file!\n";

    else

        cout << "File opened successfully!\n";

    cout << "Given DFA - \n";

    while (getline(myfile, line))

    {

        cout << line << endl;

    }

    myfile.clear();

    myfile.seekg(0);

    int currentLine = 0;

    while (getline(myfile, line))

    {

        if (currentLine == 0)

        {

            initialState = stoi(line);

        }

        else

        {

            int i = 0;

            vector<int> v;

            while (i < line.size())

            {

                if (line[i] == '-')

                {

                    v.push\_back(-1);

                    i++;

                }

                else if (line[i] != '-' and line[i] != ' ')

                {

                    v.push\_back(line[i] - '0');

                }

                i++;

            }

            dfa.push\_back(v);

        }

        currentLine++;

    }

    string input;

    cout << "\nEnter input consisting of 0's and 1's: ";

    getline(cin, input);

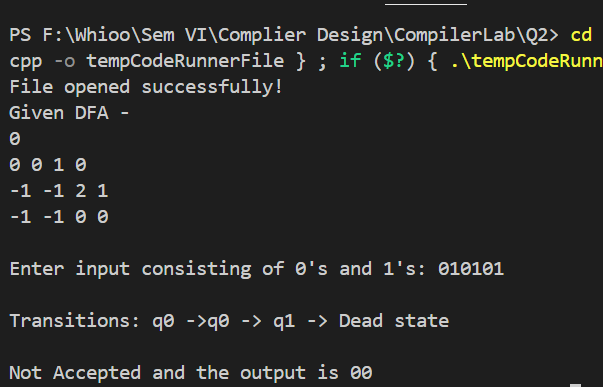
    string ans = isAccepted(dfa, input);

    cout << endl;

    cout << ans << endl;

    return 0;}

Output: -



A screen shot of a computer program

Description automatically generated

Program 3

Code: -

#include <bits/stdc++.h>

#include <fstream>

using namespace std;

int convertToInt(string &s)

{

    int i = 0;

    int ans = 0;

    while (i < s.size())

    {

        int t = s[i] - '0';

        ans = ans \* 10 + t;

        i++;

    }

    return ans;

}

vector<set<int>> stringToVectorOfSet(string &line)

{

    int i = 0;

    vector<set<int>> v;

    set<int> s;

    while (i < line.size())

    {

        if (line[i] == '-')

        {

            int x = -1;

            s.insert(x);

            v.push\_back(s);

            s.clear();

            i++;

        }

        else if (line[i] != '-' and line[i] != ' ')

        {

            string t = "";

            while (line[i] != ' ' and i < line.size())

            {

                t += line[i];

                i++;

            }

            string k = "";

            for (int j = 0; j < t.size(); j++)

            {

                if (t[j] != ',')

                {

                    k += t[j];

                }

                else

                {

                    int x = convertToInt(k);

                    k = "";

                    s.insert(x);

                }

            }

            if (k != "")

            {

                int x = convertToInt(k);

                k = "";

                s.insert(x);

            }

            v.push\_back(s);

            s.clear();

        }

        i++;

    }

    return v;

}

int main()

{

    ifstream myfile("NFA.txt", ios::in);

    int curr\_line = 0;

    string line;

    int istate = 0;

    set<int> finalState;

    vector<vector<set<int>>> nfa;

    while (getline(myfile, line))

    {

        int state;

        vector<set<int>> temp;

        if (curr\_line == 0)

        {

            istate = convertToInt(line);

        }

        else if (curr\_line == 1)

        {

            string t;

            for (int i = 0; i < line.size(); i++)

            {

                if (line[i] != ',')

                {

                    t += line[i];

                }

                else

                {

                    int x = convertToInt(t);

                    t = "";

                    finalState.insert(x);

                }

            }

            if (t != "")

            {

                int x = convertToInt(t);

                t = "";

                finalState.insert(x);

            }

        }

        else

        {

            vector<set<int>> v = stringToVectorOfSet(line);

            nfa.push\_back(v);

        }

        curr\_line++;

    }

    map<int, set<int>> convergeStates;

    map<set<int>, int> mapping;

    int n = nfa.size();

    for (int i = 0; i < nfa.size(); i++)

    {

        for (int j = 0; j < nfa[i].size(); j++)

        {

            if (nfa[i][j].size() > 1)

            {

                convergeStates[n] = nfa[i][j];

                mapping[nfa[i][j]] = n;

                n++;

            }

        }

    }

    int m = nfa[0].size();

    for (auto it : convergeStates)

    {

        set<int> s;

        vector<set<int>> nS;

        for (int j = 0; j < m; j++)

        {

            set<int> newSet;

            for (auto i : it.second)

            {

                for (auto k : nfa[i][j])

                {

                    if (k != -1)

                    {

                        newSet.insert(k);

                    }

                }

            }

            if (newSet.size() > 1)

            {

                if (mapping.find(newSet) == mapping.end())

                {

                    convergeStates[n] = newSet;

                    mapping[newSet] = n;

                    n++;

                }

            }

            if (newSet.size() == 0)

                newSet.insert(-1);

            nS.push\_back(newSet);

        }

        nfa.push\_back(nS);

    }

    for (int i = 0; i < nfa.size(); i++)

    {

        for (int j = 0; j < nfa[i].size(); j++)

        {

            for (auto it : nfa[i][j])

            {

                cout << it;

            }

            cout << " ";

        }

        cout << endl;

    }

    cout << endl;

    cout << endl;

    cout << endl;

    int count = 0;

    vector<vector<int>> dfa;

    set<int> newMergeFinal = finalState;

    for (int i = 0; i < nfa.size(); i++)

    {

        vector<int> v;

        count = 0;

        for (int j = 0; j < nfa[i].size(); j++)

        {

            if (nfa[i][j].size() == 1)

            {

                v.push\_back(\*nfa[i][j].begin());

                count++;

            }

            else

            {

                for (auto it : nfa[i][j])

                {

                    if (finalState.find(it) != finalState.end())

                    {

                        //                        finalState.insert(mapping[nfa[i][j]]);

                        newMergeFinal.insert(mapping[nfa[i][j]]);

                        break;

                    }

                }

                v.push\_back(mapping[nfa[i][j]]);

            }

        }

        dfa.push\_back(v);

    }

    ofstream fout("convertedDFA.txt", ios::out);

    fout << istate;

    fout << endl;

    for (auto it : newMergeFinal)

    {

        cout << it << " ";

        fout << it << " ";

    }

    fout << endl;

    cout << endl;

    for (int i = 0; i < dfa.size(); i++)

    {

        for (int j = 0; j < dfa[i].size(); j++)

        {

            if (finalState.find(dfa[i][j]) != finalState.end())

                cout << "[" << dfa[i][j] << "]"

                     << " ";

            else

                cout << dfa[i][j] << " ";

            fout << dfa[i][j] << " ";

        }

        fout << endl;

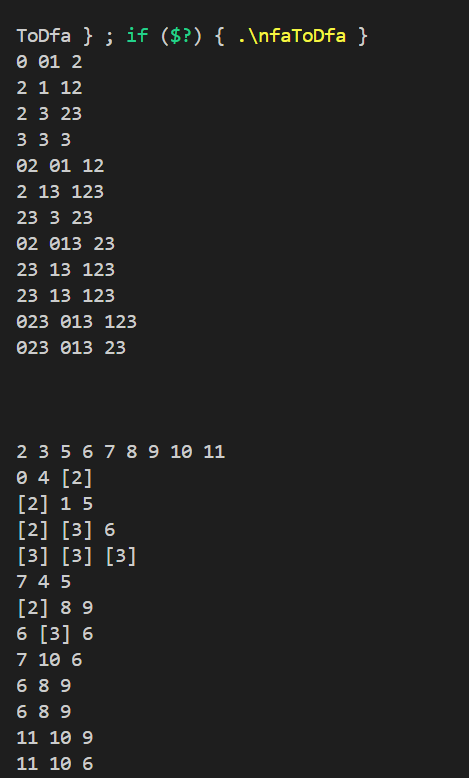
        cout << endl;

    }

    return 0;

}

Output: -



Program 4

Code: -

#include <iostream>

#include <fstream>

#include <string>

#include <set>

#include <map>

using namespace std;

map<char, set<string>> mp;

bool solve(string &s, int idx, string toCheck)

{

    if (s.substr(idx) == toCheck)

        return true;

    else if (toCheck == "")

        return false;

    for (auto t : mp[toCheck[0]])

    {

        int sz = t.size(), i = idx, j = 0;

        while (j < sz && i < s.size() && s[i] == t[j])

        {

            i++, j++;

        }

        if (j == sz && solve(s, i, toCheck.substr(1)))

            return true;

        else if (j < sz && t[j] == '#' && solve(s, i, toCheck.substr(1)))

            return true;

        else if (j < sz && t[j] >= 'A' && t[j] <= 'Z' && solve(s, i, t.substr(j) + toCheck.substr(1)))

            return true;

    }

    return false;

}

string checkGrammar(string s = "")

{

    if (solve(s, 0, "S"))

        return "Accepted\n";

    return "Not Accepted\n";

}

int main()

{

    ifstream inputF("regularGrammer.txt");

    string t;

    while (getline(inputF, t))

    {

        string x;

        for (int i = 3; i < t.size(); i++)

        {

            if (t[i] != '|')

                x.push\_back(t[i]);

            else

            {

                mp[t[0]].insert(x);

                x = "";

            }

        }

        mp[t[0]].insert(x);

    }

    for (auto ch : mp)

    {

        cout << ch.first << "-> ";

        for (auto t : mp[ch.first])

            cout << t << "| ";

        cout << endl;

    }

    while (true)

    {

        cout << "Enter a string : ";

        string s;

        cin >> s;

        if (s == "exit")

            break;

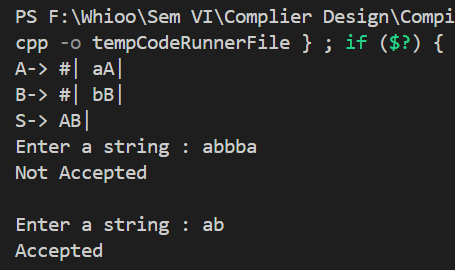
        cout << checkGrammar(s) << endl;

    }

    return 0;

}

Output: -



Program 5

Code: -

#include <iostream>

#include <fstream>

#include <string>

#include <set>

#include <map>

using namespace std;

map<char, set<string>> mp;

bool solve(string &s, int idx, string toCheck)

{

    if (s.substr(idx) == toCheck)

        return true;

    else if (toCheck == "")

        return false;

    for (auto t : mp[toCheck[0]])

    {

        int sz = t.size(), i = idx, j = 0;

        while (j < sz && i < s.size() && s[i] == t[j])

        {

            i++, j++;

        }

        if (j == sz && solve(s, i, toCheck.substr(1)))

            return true;

        else if (j < sz && t[j] == '#' && solve(s, i, toCheck.substr(1)))

            return true;

        else if (j < sz && t[j] >= 'A' && t[j] <= 'Z' && solve(s, i, t.substr(j) + toCheck.substr(1)))

            return true;

    }

    return false;

}

string checkGrammar(string s = "")

{

    if (solve(s, 0, "S"))

        return "Accepted\n";

    return "Not Accepted\n";

}

int main()

{

    ifstream inputF("cfg.txt");

    string t;

    while (getline(inputF, t))

    {

        string x;

        for (int i = 3; i < t.size(); i++)

        {

            if (t[i] != '|')

                x.push\_back(t[i]);

            else

            {

                mp[t[0]].insert(x);

                x = "";

            }

        }

        mp[t[0]].insert(x);

    }

    for (auto ch : mp)

    {

        cout << ch.first << "-> ";

        for (auto t : mp[ch.first])

            cout << t << "| ";

        cout << endl;

    }

    while (true)

    {

        cout << "Enter a string : ";

        string s;

        cin >> s;

        if (s == "exit")

            break;

        cout << checkGrammar(s) << endl;

    }

    return 0;

}

Output: -

A screenshot of a computer program

Description automatically generated

Program 6

Code: -

#include <bits/stdc++.h>

using namespace std;

void computingFollow(char c, vector<string> &cfg, map<char, string> &first, map<char, string> &follow)

{

    for (int i = 0; i < cfg.size(); i++)

    {

        for (int j = 1; j < cfg[i].size(); j++)

        {

            if (cfg[i][j] == c)

            {

                for (int k = j + 1; k <= cfg[i].size(); k++)

                {

                    bool flag = false;

                    if (k < cfg[i].size())

                    {

                        if (cfg[i][k] <= 'Z' && cfg[i][k] >= 'A')

                        {

                            for (int l = 0; l < first[cfg[i][k]].size(); l++)

                            {

                                if (first[cfg[i][k]][l] != '#')

                                {

                                    if (follow[c].find(first[cfg[i][k]][l]) >= follow[c].size())

                                        follow[c].push\_back(first[cfg[i][k]][l]);

                                }

                                else

                                    flag = true;

                            }

                        }

                        else

                        {

                            if (follow[c].find(cfg[i][k]) >= follow[c].size())

                                follow[c].push\_back(cfg[i][k]);

                            break;

                        }

                    }

                    else

                    {

                        if (cfg[i][0] == c)

                            break;

                        if (follow[cfg[i][0]].empty())

                            computingFollow(cfg[i][0], cfg, first, follow);

                        for (int l = 0; l < follow[cfg[i][0]].size(); l++)

                        {

                            if (follow[c].find(follow[cfg[i][0]][l]) >= follow[c].size())

                                follow[c].push\_back(follow[cfg[i][0]][l]);

                        }

                    }

                    if (!flag)

                        break;

                }

            }

        }

    }

}

void computingFirst(char c, vector<string> &cfg, map<char, string> &first)

{

    if (!(c <= 'Z' and c >= 'A'))

    {

        if (first[c].find(c) >= first[c].size())

            first[c].push\_back(c);

        return;

    }

    for (int i = 0; i < cfg.size(); i++)

    {

        if (cfg[i][0] == c)

        {

            for (int j = 1; j < cfg[i].size(); j++)

            {

                if (cfg[i][j] == c)

                {

                    break;

                }

                bool flag = false;

                if (first[cfg[i][j]].empty())

                    computingFirst(cfg[i][j], cfg, first);

                for (int k = 0; k < first[cfg[i][j]].size(); k++)

                {

                    if (first[cfg[i][j]][k] != '#')

                    {

                        if (first[c].find(first[cfg[i][j]][k]) >= first[c].size())

                            first[c].push\_back(first[cfg[i][j]][k]);

                    }

                    else if (j == cfg[i].size() - 1)

                    {

                        if (first[c].find('#') >= first[c].size())

                            first[c].push\_back('#');

                    }

                    else

                        flag = true;

                }

                if (!flag)

                    break;

            }

        }

    }

}

int main()

{

    ifstream myfile("cfgprac2.txt", ios::in);

    int curr\_line = 0;

    string line;

    vector<string> cfg;

    while (getline(myfile, line))

    {

        string temp = "";

        for (int i = 0; i < line.size(); i++)

        {

            if (line[i] != ' ')

            {

                temp += line[i];

            }

        }

        cfg.push\_back(temp);

    }

    char s = cfg[0][0];

    map<char, string> first;

    map<char, string> follow;

    follow[s].push\_back('$');

    for (int i = 0; i < cfg.size(); i++)

    {

        for (int j = 0; j < cfg[i].size(); j++)

        {

            computingFirst(cfg[i][0], cfg, first);

        }

    }

    for (int i = 0; i < cfg.size(); i++)

    {

        for (int j = 0; j < cfg[i].size(); j++)

        {

            computingFollow(cfg[i][0], cfg, first, follow);

        }

    }

    for (auto it : first)

    {

        if ((it.first <= 'Z' and it.first >= 'A'))

        {

            cout << "First(" << it.first << ") :  { ";

            int j = 0;

            for (auto i : it.second)

            {

                cout << "'" << i << "'";

                if (j < it.second.size() - 1)

                    cout << " , ";

                j++;

            }

            cout << " }" << endl;

        }

    }

    cout << endl

         << endl;

    for (auto it : follow)

    {

        cout << "Follow(" << it.first << ") :  { ";

        int j = 0;

        for (auto i : it.second)

        {

            cout << "'" << i << "'";

            if (j < it.second.size() - 1)

                cout << " , ";

            j++;

        }

        cout << " }" << endl;

    }

    return 0;

}

Output: -

A computer screen with white text and yellow and green letters

Description automatically generated

Program 7

Code: -

#include <iostream>

#include <fstream>

#include <vector>

#include <algorithm>

#include <string>

#include <set>

#include <map>

using namespace std;

char start;

map<char, vector<vector<char>>> readGrammar(const string &filename)

{

    ifstream fin(filename);

    int i, j;

    map<char, vector<vector<char>>> grammar;

    string line;

    bool flag = 0;

    cout << "Grammar: " << '\n';

    while (getline(fin, line))

    {

        if (flag == 0)

        {

            start = line[0], flag = 1;

        }

        cout << line << '\n';

        char nonTerminal = line[0];

        vector<char> production;

        for (int i = 3; i < line.size(); i++)

        {

            if (line[i] == '|')

            {

                grammar[nonTerminal].push\_back(production);

                production.clear();

            }

            else

            {

                production.push\_back(line[i]);

            }

        }

        grammar[nonTerminal].push\_back(production);

    }

    return grammar;

}

void printGrammar(const std::map<char, std::vector<std::vector<char>>> &grammar)

{

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

    for (const auto &entry : grammar)

    {

        char nonTerminal = entry.first;

        std::string output = "";

        output += nonTerminal;

        output += " -> ";

        bool first = true;

        for (const auto &production : entry.second)

        {

            if (!first)

            {

                output += " | ";

            }

            else

            {

                first = false;

            }

            for (const auto &symbol : production)

            {

                output += symbol;

            }

        }

        std::cout << output << std::endl;

    }

}

void eliminateLeftRecursion(map<char, vector<vector<char>>> &grammar)

{

    map<char, vector<vector<char>>> additionalRules;

    vector<int> usedNonTerminals(26, 0);

    int flag = 0;

    for (auto &entry : grammar)

    {

        char nonTerminal = entry.first;

        vector<vector<char>> nonLeftRecursiveProductions;

        vector<vector<char>> leftRecursiveProductions;

        for (auto &production : entry.second)

        {

            if (production[0] == nonTerminal)

            {

                vector<char> adjustedProduction(production.begin() + 1, production.end());

                leftRecursiveProductions.push\_back(adjustedProduction);

            }

            else

            {

                nonLeftRecursiveProductions.push\_back(production);

            }

        }

        if (!leftRecursiveProductions.empty())

        {

            flag = 1;

            cout << "Given CFG is not suitable for LL(1) parsing as is has Left Recurssion." << endl;

            char newNonTerminal = 'A';

            while (grammar.count(newNonTerminal) || usedNonTerminals[newNonTerminal - 'A'])

            {

                newNonTerminal++;

            }

            usedNonTerminals[newNonTerminal - 'A'] = 1;

            for (auto &production : nonLeftRecursiveProductions)

            {

                production.push\_back(newNonTerminal);

            }

            for (auto &production : leftRecursiveProductions)

            {

                production.push\_back(newNonTerminal);

            }

            leftRecursiveProductions.push\_back({'#'});

            grammar[nonTerminal] = nonLeftRecursiveProductions;

            additionalRules[newNonTerminal] = leftRecursiveProductions;

        }

    }

    for (auto &entry : additionalRules)

    {

        grammar[entry.first] = entry.second;

    }

    if (flag)

    {

        printGrammar(grammar);

    }

}

set<char> getFirstSet(char nonTerminal, const map<char, vector<vector<char>>> &grammar, map<char, set<char>> &firstSets)

{

    if (firstSets.count(nonTerminal))

    {

        return firstSets[nonTerminal];

    }

    set<char> firstSet;

    bool canDeriveEpsilon = false;

    for (const auto &production : grammar.at(nonTerminal))

    {

        bool canProduceEpsilon = true;

        for (const char &symbol : production)

        {

            if (symbol >= 'A' && symbol <= 'Z')

            {

                set<char> subFirstSet = getFirstSet(symbol, grammar, firstSets);

                firstSet.insert(subFirstSet.begin(), subFirstSet.end());

                firstSet.erase('#');

                if (!subFirstSet.count('#'))

                {

                    canProduceEpsilon = false;

                    break;

                }

            }

            else if (symbol == '#')

            {

                firstSet.insert(symbol);

                break;

            }

            else

            {

                firstSet.insert(symbol);

                canProduceEpsilon = false;

                break;

            }

        }

        if (canProduceEpsilon)

        {

            canDeriveEpsilon = true;

        }

    }

    if (canDeriveEpsilon)

    {

        firstSet.insert('#');

    }

    firstSets[nonTerminal] = firstSet;

    return firstSet;

}

map<char, set<char>> calculateFollowSets(const map<char, vector<vector<char>>> &grammar, char startSymbol, map<char, set<char>> &firstSets)

{

    int i;

    map<char, set<char>> followSets;

    followSets[startSymbol].insert('$');

    int iterations = 10;

    while (iterations--)

    {

        for (auto q : grammar)

        {

            for (auto r : q.second)

            {

                for (i = 0; i < r.size() - 1; i++)

                {

                    if (r[i] >= 'A' && r[i] <= 'Z')

                    {

                        if (!(r[i + 1] >= 'A' && r[i + 1] <= 'Z'))

                            followSets[r[i]].insert(r[i + 1]);

                        else

                        {

                            char temp = r[i + 1];

                            int j = i + 1;

                            while (temp >= 'A' && temp <= 'Z')

                            {

                                if (\*firstSets[temp].begin() == '#')

                                {

                                    for (auto g : firstSets[temp])

                                    {

                                        if (g == '#')

                                            continue;

                                        followSets[r[i]].insert(g);

                                    }

                                    j++;

                                    if (j < r.size())

                                    {

                                        temp = r[j];

                                        if (!(temp >= 'A' && temp <= 'Z'))

                                        {

                                            followSets[r[i]].insert(temp);

                                            break;

                                        }

                                    }

                                    else

                                    {

                                        for (auto g : followSets[q.first])

                                            followSets[r[i]].insert(g);

                                        break;

                                    }

                                }

                                else

                                {

                                    for (auto g : firstSets[temp])

                                    {

                                        followSets[r[i]].insert(g);

                                    }

                                    break;

                                }

                            }

                        }

                    }

                }

                if (r[r.size() - 1] >= 'A' && r[r.size() - 1] <= 'Z')

                {

                    for (auto g : followSets[q.first])

                        followSets[r[i]].insert(g);

                }

            }

        }

    }

    return followSets;

}

bool isLL1(const map<char, set<char>> &firstSets, const map<char, set<char>> &followSets, const map<char, vector<vector<char>>> &grammar)

{

    for (const auto &entry : grammar)

    {

        char nonTerminal = entry.first;

        set<char> firstUnion;

        for (const auto &production : entry.second)

        {

            set<char> currentFirstSet;

            for (const auto &symbol : production)

            {

                if (symbol >= 'A' && symbol <= 'Z')

                {

                    const auto &subFirstSet = firstSets.at(symbol);

                    currentFirstSet.insert(subFirstSet.begin(), subFirstSet.end());

                    if (!subFirstSet.count('#'))

                    {

                        break;

                    }

                }

                else

                {

                    currentFirstSet.insert(symbol);

                    break;

                }

            }

            set<char> intersection;

            set\_intersection(firstUnion.begin(), firstUnion.end(), currentFirstSet.begin(), currentFirstSet.end(), inserter(intersection, intersection.begin()));

            if (!intersection.empty())

            {

                return false;

            }

            firstUnion.insert(currentFirstSet.begin(), currentFirstSet.end());

        }

        if (firstUnion.count('#'))

        {

            set<char> intersection;

            set\_intersection(firstUnion.begin(), firstUnion.end(), followSets.at(nonTerminal).begin(), followSets.at(nonTerminal).end(), inserter(intersection, intersection.begin()));

            if (!intersection.empty())

            {

                return false;

            }

        }

    }

    return true;

}

int main()

{

    map<char, vector<vector<char>>> grammar = readGrammar("ll1.txt");

    eliminateLeftRecursion(grammar);

    map<char, set<char>> firstSets;

    for (const auto &entry : grammar)

    {

        getFirstSet(entry.first, grammar, firstSets);

    }

    char startSymbol = start;

    map<char, set<char>> followSets = calculateFollowSets(grammar, startSymbol, firstSets);

    bool ll1Compatible = isLL1(firstSets, followSets, grammar);

    if (ll1Compatible)

    {

        cout << "Given CFG is suitable for LL(1) parsing." << endl;

    }

    else

    {

        cout << "Given CFG is not suitable for LL(1) parsing as First & Follow have no common elements." << endl;

    }

    cout << "\nFIRST sets:" << endl;

    for (const auto &entry : firstSets)

    {

        cout << entry.first << " = { ";

        for (const auto &s : entry.second)

        {

            cout << s << " ";

        }

        cout << "}" << endl;

    }

    cout << "\nFOLLOW sets:" << endl;

    for (const auto &entry : followSets)

    {

        cout << entry.first << " = { ";

        for (const auto &s : entry.second)

        {

            cout << s << " ";

        }

        cout << "}" << endl;

    }

    return 0;

}

Output: -

A screenshot of a computer screen

Description automatically generated

Program 8

Code: -

#include <bits/stdc++.h>

#include <unordered\_map>

#include <set>

#include <vector>

#include <fstream>

using namespace std;

unordered\_map<char, set<char>> FIRST, FOLLOW;

unordered\_map<char, vector<string>> productions;

unordered\_map<char, unordered\_map<char, string>> parsingTable;

vector<char> terminals;

char startSymbol = '\0';

void readTxt();

void calculateFIRST(char);

void calculateFOLLOW(char);

void printFirstFollow();

void terminalSet();

void constructParsingTable();

void printParsingTable();

void parseInput(string &input);

int main()

{

  readTxt();

  FOLLOW[startSymbol].insert('$'); // Add $ to the FOLLOW of starting symbol

  printFirstFollow();

  terminalSet();

  constructParsingTable();

  printParsingTable();

  string input;

  cout << "\nEnter the input string to parse: ";

  cin >> input;

  parseInput(input);

  return 0;

}

void readTxt()

{

  fstream myfile;

  myfile.open("LL\_cfg2.txt", ios::in);

  string line;

  cout << "\nCFG:\n\n";

  while (getline(myfile, line))

  {

    cout << line << endl;

    int equalPos = line.find("->"); // Find the position of '='

    if (equalPos != string::npos)

    {

      char NT = line[0];

      if (startSymbol == '\0')

      {

        startSymbol = line[0]; // Set the starting symbol

      }

      string production = line.substr(equalPos + 2); // Get the substring after '->'

      int pos = 0;

      string token;

      while (production.find('|') != string::npos)

      {

        pos = production.find('|');

        token = production.substr(0, pos);

        productions[NT].push\_back(token);

        production.erase(0, pos + 1);

      }

      productions[NT].push\_back(production); // Add the last or only production

    }

  }

  cout << endl;

  myfile.close();

}

void terminalSet()

{

  for (auto prod : productions)

  {

    for (string str : prod.second)

    {

      for (int i = 0; i < str.size(); i++)

      {

        if (!isupper(str[i]))

        {

          terminals.push\_back(str[i]);

        }

      }

    }

  }

  sort(terminals.begin(), terminals.end());

  terminals.erase(unique(terminals.begin(), terminals.end()), terminals.end());

  terminals.erase(remove(terminals.begin(), terminals.end(), ' '), terminals.end());

  terminals.push\_back('$');

}

void calculateFIRST(char NT)

{

  if (!FIRST[NT].empty())

    return; // Already calculated

  int flag = 0;

  for (string &prod : productions[NT])

  {

    int len = prod.size();

    int i = 0;

    for (char symbol : prod)

    {

      if (isupper(symbol))

      { // Non-terminal

        len--;

        calculateFIRST(symbol);

        FIRST[NT].insert(FIRST[symbol].begin(), FIRST[symbol].end());

        FIRST[NT].erase('#'); // remove epsilon

        if (FIRST[symbol].find('#') == FIRST[symbol].end())

        {

          flag = 1;

          break;

        }

        if (flag == 0 && len == 0)

        {

          FIRST[NT].insert('#'); // if RHS is all epsilon then add epsilon

        }

      }

      else

      { // Terminal or Epsilon

        FIRST[NT].insert(symbol);

        break;

      }

    }

  }

}

void calculateFOLLOW(char NT)

{

  if (!FOLLOW[NT].empty())

    if (NT != startSymbol)

    {

      return; // Already calculated

    }

  for (auto &prod : productions)

  {

    for (string &str : prod.second)

    {

      for (int i = 0; i < str.size(); ++i)

      {

        if (str[i] == NT)

        {

          int j = i + 1;

          while (j < str.size())

          {

            char nextSymbol = str[j];

            if (isupper(nextSymbol))

            { // Next is a Non-terminal

              FOLLOW[NT].insert(FIRST[nextSymbol].begin(), FIRST[nextSymbol].end());

              FOLLOW[NT].erase('#'); // Remove Epsilon if it's there

              if (j == str.size() - 1 && FIRST[nextSymbol].find('#') != FIRST[nextSymbol].end())

              {

                calculateFOLLOW(prod.first);

                FOLLOW[NT].insert(FOLLOW[prod.first].begin(), FOLLOW[prod.first].end());

              }

              else if (FIRST[nextSymbol].find('#') == FIRST[nextSymbol].end())

              {

                break;

              }

            }

            else

            { // Next is a Terminal

              FOLLOW[NT].insert(nextSymbol);

            }

            j++;

          }

          if (prod.first != NT && i + 1 >= str.size())

          {

            calculateFOLLOW(prod.first);

            FOLLOW[NT].insert(FOLLOW[prod.first].begin(), FOLLOW[prod.first].end());

          }

        }

      }

    }

  }

}

void printFirstFollow()

{ // Calculate FIRST

  for (auto prod : productions)

  {

    calculateFIRST(prod.first);

  }

  // Calculate FOLLOW

  for (auto prod : productions)

  {

    calculateFOLLOW(prod.first);

  }

  cout << "FIRST OF NON-TERMINALS: " << endl

       << endl;

  for (auto &prod : productions)

  {

    cout << "FIRST(" << prod.first << ") = { ";

    for (char c : FIRST[prod.first])

    {

      cout << c << " ";

    }

    cout << "}" << endl;

  }

  cout << "\n------------------------------\n\n";

  cout << "FOLLOW OF NON-TERMINALS: " << endl

       << endl;

  for (auto &prod : productions)

  {

    cout << "FOLLOW(" << prod.first << ") = { ";

    for (char c : FOLLOW[prod.first])

    {

      cout << c << " ";

    }

    cout << "}" << endl;

  }

  cout << endl;

}

void constructParsingTable()

{

  for (auto &prod : productions)

  {

    char nonTerminal = prod.first;

    for (const string &prodStr : prod.second)

    {

      set<char> firstSet;

      for (char symbol : prodStr)

      {

        if (isupper(symbol))

        { // Non-terminal

          firstSet.insert(FIRST[symbol].begin(), FIRST[symbol].end());

          if (FIRST[symbol].find('#') == FIRST[symbol].end())

          {

            break;

          }

        }

        else

        { // Terminal or Epsilon

          firstSet.insert(symbol);

          break;

        }

      }

      for (char terminal : firstSet)

      {

        if (terminal != '#')

        {

          parsingTable[nonTerminal][terminal] = prodStr;

        }

        else

        {

          for (char follow : FOLLOW[nonTerminal])

          {

            parsingTable[nonTerminal][follow] = prodStr;

          }

        }

      }

    }

  }

}

void printParsingTable()

{

  cout << "LL(1) Parsing Table:\n\n";

  cout << "NT" << setw(15);

  for (const auto t : terminals)

  {

    if (t != '#')

    {

      cout << t << setw(17);

    }

  }

  cout << endl;

  cout << setw(0) << "--------------------------------------------------------------------------------------------------------" << endl;

  for (const auto &nonTerminalEntry : parsingTable)

  {

    char nonTerminal = nonTerminalEntry.first;

    const auto &terminalMap = nonTerminalEntry.second;

    cout << nonTerminal;

    for (const auto t : terminals)

    {

      // char terminal = terminalEntry.first;

      if (t != '#')

      {

        if (terminalMap.count(t))

        {

          const string &production = terminalMap.at(t);

          cout << setw(15) << nonTerminal << "->" << production;

        }

        else

        {

          cout << setw(15) << "-";

        }

      }

    }

    cout << endl;

  }

}

void parseInput(string &input)

{

  input += '$';

  stack<char> symbolStack;

  symbolStack.push('$');         // Push end of input marker

  symbolStack.push(startSymbol); // Push starting symbol

  cout << "\nParsing Steps:" << endl;

  int inputIndex = 0;

  char currentInput = input[inputIndex];

  cout << "\nStack: ";

  stack<char> printStack = symbolStack;

  while (!printStack.empty())

  {

    cout << printStack.top();

    printStack.pop();

  }

  cout << "\t\tInput: " << input.substr(inputIndex) << endl;

  while (!symbolStack.empty())

  {

    char stackTop = symbolStack.top();

    if (stackTop == currentInput && currentInput == '$')

    {

      cout << "\nString Parsed Successfully." << endl;

      return;

    }

    if (!isupper(stackTop))

    { // Terminal symbol

      if (stackTop == currentInput)

      {

        symbolStack.pop();

        inputIndex++;

        currentInput = input[inputIndex];

      }

      else

      {

        cout << "\nError: Mismatched terminal symbol." << endl;

        return;

      }

    }

    else

    { // Non-terminal symbol

      auto it = parsingTable.find(stackTop);

      if (it != parsingTable.end())

      {

        auto &row = it->second;

        auto colIt = row.find(currentInput);

        if (colIt != row.end())

        {

          const string &production = colIt->second;

          cout << "\nUsing production rule: " << stackTop << " -> " << production << endl;

          // Pop the non-terminal from the stack

          symbolStack.pop();

          // Push the production rule in reverse order onto the stack

          for (auto rit = production.rbegin(); rit != production.rend(); ++rit)

          {

            if (\*rit != '#')

            { // Skip epsilon

              symbolStack.push(\*rit);

            }

          }

        }

        else

        {

          cout << "\nError: No production rule found." << endl;

          break;

        }

      }

      else

      {

        cout << "\nError: No production rule found." << endl;

        break;

      }

    }

    // Print current stack and input

    cout << "Stack: ";

    stack<char> printStack = symbolStack;

    while (!printStack.empty())

    {

      cout << printStack.top();

      printStack.pop();

    }

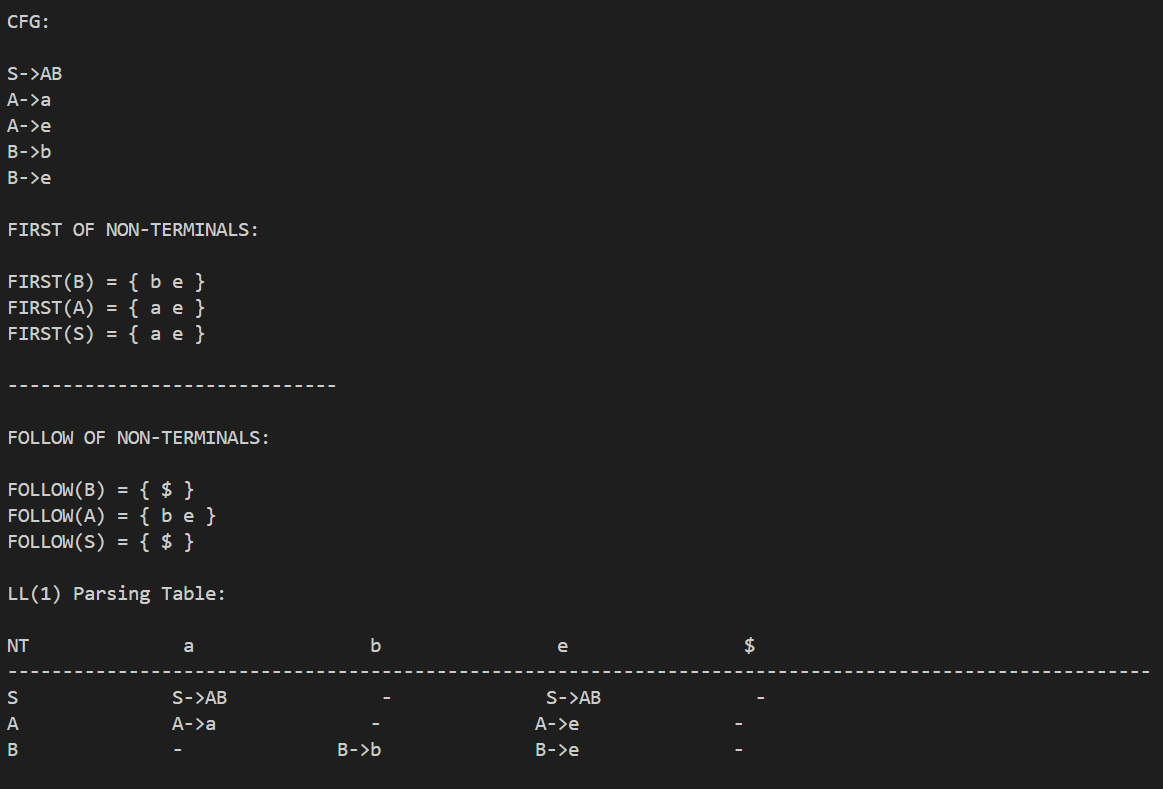
    cout << "\t\tInput: " << input.substr(inputIndex) << endl;

  }

  cout << "\nError: Parsing Failed." << endl;

}

Output: -



A screenshot of a computer program

Description automatically generated

Program 9

Code: -

#include <bits/stdc++.h>

using namespace std;

int stoi(string &num)

{

    int ans = 0;

    for (int i = 0; i < num.size(); i++)

    {

        ans = ans \* 10 + num[i] - '0';

    }

    return ans;

}

int main()

{

    ifstream myfile("TAC.txt", ios::in);

    int curr\_line = 1;

    string line;

    vector<string> tac;

    set<int> leaders;

    leaders.insert(1);

    while (getline(myfile, line))

    {

        string temp;

        int sz = line.size();

        for (int i = 0; i < sz; i++)

        {

            if (line[i] != ' ')

                temp += line[i];

        }

        tac.push\_back(temp);

        sz = temp.size();

        int pos = temp.find("goto");

        string num = "";

        if (pos != -1)

        {

            for (int i = pos + 5; temp[i] <= '9' and temp[i] >= '0'; i++)

            {

                num += temp[i];

            }

            int n = stoi(num);

            leaders.insert(n);

            leaders.insert(curr\_line + 1);

        }

        curr\_line++;

    }

    curr\_line--;

    cout << "\nLEADERS\n";

    for (auto i : leaders)

    {

        if (i == curr\_line + 1)

            leaders.erase(i);

        cout << i << " ";

    }

    cout << endl;

    vector<pair<int, int>> blocks;

    int p = -1;

    for (int i : leaders)

    {

        if (p != -1)

        {

            blocks.push\_back({p, i - 1});

        }

        p = i;

    }

    blocks.push\_back({p, curr\_line});

    cout << "B1" << endl;

    cout << "-----------------" << endl;

    int count = 1;

    for (int i = 0; i < blocks.size(); i++)

    {

        for (int j = blocks[i].first - 1; j < blocks[i].second; j++)

        {

            cout << j + 1 << " " << tac[j] << endl;

        }

        cout << "-----------------" << endl;

        count++;

        if (i < blocks.size() - 1)

        {

            cout << endl

                 << "B" << count << endl;

            cout << "-----------------" << endl;

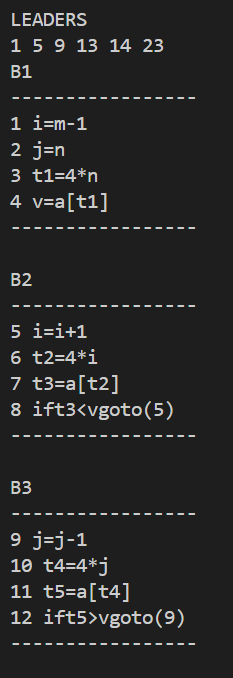
        }

    }

    return 0;

}

Output: -

 A screenshot of a computer screen

Description automatically generated

Program 10

Code: -

#include <bits/stdc++.h>

using namespace std;

int stoi(string &num)

{

    int ans = 0;

    for (int i = 0; i < num.size(); i++)

    {

        ans = ans \* 10 + num[i] - '0';

    }

    return ans;

}

int findblock(int j, vector<pair<int, int>> &blocks)

{

    int ans = 0;

    for (int i = 0; i < blocks.size(); i++)

    {

        if (blocks[i].second == j)

        {

            ans = i + 1;

            break;

        }

    }

    return ans;

}

int main()

{

    ifstream myfile("TAC.txt", ios::in);

    int curr\_line = 1;

    string line;

    vector<string> tac;

    set<int> leaders;

    leaders.insert(1);

    while (getline(myfile, line))

    {

        string temp;

        int sz = line.size();

        for (int i = 0; i < sz; i++)

        {

            if (line[i] != ' ')

                temp += line[i];

        }

        tac.push\_back(temp);

        sz = temp.size();

        int pos = temp.find("goto");

        string num = "";

        if (pos != -1)

        {

            for (int i = pos + 5; temp[i] <= '9' and temp[i] >= '0'; i++)

            {

                num += temp[i];

            }

            int n = stoi(num);

            leaders.insert(n);

            leaders.insert(curr\_line + 1);

        }

        curr\_line++;

    }

    curr\_line--;

    cout << "Leaders - \n";

    for (auto i : leaders)

    {

        if (i == curr\_line + 1)

            leaders.erase(i);

        cout << i << " ";

    }

    cout << endl;

    vector<pair<int, int>> blocks;

    int p = -1;

    for (int i : leaders)

    {

        if (p != -1)

        {

            blocks.push\_back({p, i - 1});

        }

        p = i;

    }

    blocks.push\_back({p, curr\_line});

    // cout << "B1" << endl;

    // cout << "-----------------" << endl;

    int count = 1;

    for (int i = 0; i < blocks.size(); i++)

    {

        for (int j = blocks[i].first - 1; j < blocks[i].second; j++)

        {

            // cout << j + 1 << " " << tac[j] << endl;

        }

        // cout << "-----------------" << endl;

        count++;

        if (i < blocks.size() - 1)

        {

            // cout << endl

            //  << "B" << count << endl;

            // cout << "-----------------" << endl;

        }

    }

    count--;

    vector<vector<int>> adj(count + 1, vector<int>(count + 1, 0));

    vector<vector<int>> adjWithoutBackEdges(count + 1, vector<int>(count + 1, 0));

    for (int i = 0; i < tac.size(); i++)

    {

        if (leaders.find(i + 1 + 1) != leaders.end())

        {

            int po = tac[i].find("if");

            int go = tac[i].find("goto");

            int b = findblock(i + 1, blocks);

            if (po == -1 and go == -1)

            {

                adj[b][b + 1] = 1;

            }

            else if (po != -1 and go != -1)

            {

                adj[b][b + 1] = 1;

                string num = "";

                for (int j = go + 5; tac[i][j] <= '9' and tac[i][j] >= '0'; j++)

                {

                    num += tac[i][j];

                }

                int n = stoi(num);

                int c = findblock(n - 1, blocks);

                adj[b][c + 1] = 1;

            }

            else if (go != -1 and po == -1)

            {

                string num = "";

                for (int j = go + 5; tac[i][j] <= '9' and tac[i][j] >= '0'; j++)

                {

                    num += tac[i][j];

                }

                int n = stoi(num);

                int c = findblock(n - 1, blocks);

                adj[b][c + 1] = 1;

            }

        }

    }

    cout << "Displaying control flow graph -\n";

    ofstream fout("adj.txt", ios::out);

    fout << "adj" << endl;

    for (int i = 1; i < adj.size(); i++)

    {

        for (int j = 1; j < adj[i].size(); j++)

        {

            cout << adj[i][j] << " ";

            fout << adj[i][j] << " ";

            if (j >= i && adj[i][j] == 1)

            {

                adjWithoutBackEdges[i][j] = 1;

            }

        }

        cout << endl;

        fout << endl;

    }

    cout << "\nDisplaying contol flow graph without backward loops\n";

    for (int i = 1; i <= count; i++)

    {

        for (int j = 1; j <= count; j++)

        {

            cout << adjWithoutBackEdges[i][j] << " ";

        }

        cout << endl;

    }

    // vector<vector<int>> adj2 = {

    //     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},

    //     {0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0},

    //     {0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1},

    //     {0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},

    // };

    // vector<vector<int>> adj3 = {

    //     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

    //     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

    //     {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},

    // };

    vector<int> indegree(count + 1);

    vector<pair<int, int>> edges;

    for (int i = 1; i < adjWithoutBackEdges.size(); i++)

    {

        for (int j = 1; j < adjWithoutBackEdges[i].size(); j++)

        {

            if (adjWithoutBackEdges[i][j] == 1)

            {

                edges.push\_back({i, j});

                indegree[j]++;

            }

        }

    }

    // cout << "\nDisplaying Indegree\n";

    // for (int i = 1; i <= 10; i++)

    // {

    //     cout << i << " " << indegree[i] << " \n";

    // }

    unordered\_map<int, int> parent;

    parent[1] = -1;

    for (int i = 1; i < adjWithoutBackEdges.size(); i++)

    {

        for (int j = i; j < adjWithoutBackEdges[i].size(); j++)

        {

            if (adjWithoutBackEdges[i][j] == 1)

            {

                bool flag = true;

                if (parent.find(j) == parent.end())

                {

                    parent[j] = i;

                }

                else

                {

                    if (parent[j] > i)

                    {

                        parent[j] = i;

                    }

                }

            }

        }

    }

    // cout << "\nDisplaing map\n";

    // for (auto &it : parent)

    // {

    //     cout << it.first << " " << it.second << "\n";

    // }

    unordered\_map<int, set<int>> dom;

    for (int i = 1; i <= count; i++)

    {

        int temp = i;

        // cout << i << " - " << i << " , ";

        dom[i].insert(i);

        while (temp != -1)

        {

            if ((indegree[temp] <= 1 || parent[temp] == 1) && parent[temp] != -1)

            {

                // cout << parent[temp] << " , ";

                dom[i].insert(parent[temp]);

            }

            temp = parent[temp];

        }

        // cout << endl;

    }

    cout << "\nDisplaying dominator -\n";

    for (auto a : dom)

    {

        cout << a.first << " : {";

        for (auto b : a.second)

        {

            cout << b << " , ";

        }

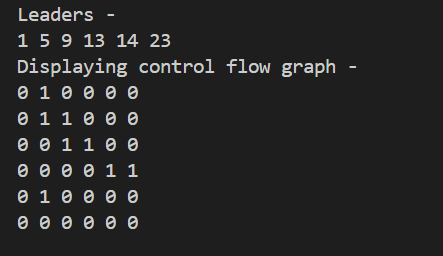
        cout << " }\n";

    }

    return 0;

}

Output: -



A computer screen shot of numbers and symbols

Description automatically generated

Program 11

Code: -

#include <iostream>

#include <fstream>

#include <vector>

#include <set>

#include <sstream>

using namespace std;

int getJumpTarget(const string &instruction, const vector<string> &tac)

{

    stringstream ss(instruction);

    string op, arg1, arg2;

    ss >> op >> arg1;

    if (op == "goto" || op == "if")

    {

        ss >> arg2;

        try

        {

            return stoi(arg2) - 1;

        }

        catch (const invalid\_argument &e)

        {

        }

    }

    return -1;

}

vector<vector<int>> findNaturalLoops(vector<string> &tac)

{

    vector<vector<int>> loops;

    set<int> visited;

    for (int i = 0; i < tac.size(); ++i)

    {

        if (visited.count(i) == 0)

        {

            visited.insert(i);

            vector<int> potentialLoop(i);

            int j = getJumpTarget(tac[i], tac);

            while (j != -1 && visited.count(j) == 0)

            {

                visited.insert(j);

                potentialLoop.push\_back(j);

                j = getJumpTarget(tac[j], tac);

            }

            if (

                j == i)

            {

                loops.push\_back(potentialLoop);

            }

        }

    }

    return loops;

}

int main()

{

    string filename = "TAC.txt";

    ifstream inputFile(filename);

    if (!inputFile.is\_open())

    {

        cerr << "Error: Could not open file " << filename << endl;

        return 1;

    }

    vector<string> tac;

    string line;

    while (getline(inputFile, line))

    {

        tac.push\_back(line);

    }

    inputFile.close();

    vector<vector<int>> loops = findNaturalLoops(tac);

    if (loops.empty())

    {

        cout << "No natural loops found in the TAC." << endl;

    }

    else

    {

        cout << "Natural loops found:" << endl;

        for (const vector<int> &loop : loops)

        {

            cout << " - Instructions: ";

            for (int instruction : loop)

            {

                cout << instruction + 1 << " ";

            }

            cout << endl;

        }

    }

    return 0;

}

Output: -

A black screen with white text

Description automatically generated

Program 12

Code: -

#include <bits/stdc++.h>

#include <fstream>

#include <vector>

#include <set>

#include <string>

#include <algorithm>

#include <unordered\_map>

using namespace std;

struct BasicBlock

{

  vector<string> instructions;

  unordered\_set<string> genSet;

  unordered\_set<string> killSet;

};

unordered\_map<int, BasicBlock> basicBlocks;

void findLeaders(vector<int> &leaders)

{

  fstream myfile;

  myfile.open("tac\_file.txt", ios::in);

  string line;

  int i = 1;

  while (getline(myfile, line))

  {

    size\_t gotoPos = line.find("goto");

    if (gotoPos != string::npos)

    {

      leaders.push\_back(stoi(line.substr(gotoPos + 6))); // Assuming the "goto" keyword is followed by line number without space

      leaders.push\_back(i + 1);

    }

    i++;

  }

  sort(leaders.begin(), leaders.end());

  auto uniqueEnd = unique(leaders.begin(), leaders.end());

  leaders.erase(uniqueEnd, leaders.end());

  sort(leaders.begin(), leaders.end());

  myfile.close();

}

void printBlocks(vector<int> &leaders)

{

  fstream myfile;

  myfile.open("tac\_file.txt", ios::in);

  string line;

  int j = 0, i = 0;

  while (getline(myfile, line))

  {

    if (find(leaders.begin(), leaders.end(), j + 1) != leaders.end())

    {

      // cout << "\nBlock " << j + 1 << ":" << endl;

      ++i;

    }

    basicBlocks[i].instructions.push\_back(line);

    // cout << j + 1 << ": " << line << endl;

    ++j;

  }

  myfile.close();

}

void computeGen()

{

  // Iterate over each basic block

  for (auto &block : basicBlocks)

  {

    BasicBlock &currentBlock = block.second;

    // Initialize GEN set for the block

    unordered\_set<string> genSet;

    // Iterate over each instruction in the block

    for (const string &instruction : currentBlock.instructions)

    {

      // Extract defined variable

      size\_t equalPos = instruction.find('=');

      if (equalPos != string::npos)

      {

        string definedVar = instruction.substr(0, equalPos);

        if (definedVar.find("if") != string::npos)

        {

          break;

        }

        // Add definedVar to GEN set

        genSet.insert(definedVar);

      }

    }

    // Update GEN set for the block

    currentBlock.genSet = genSet;

  }

}

void computeKill()

{

  // Iterate over each basic block

  for (auto &block : basicBlocks)

  {

    BasicBlock &currentBlock = block.second;

    // Iterate over each variable in the current block's GEN set

    for (const string &var : currentBlock.genSet)

    {

      // Check if the variable is defined in another block

      for (const auto &otherBlock : basicBlocks)

      {

        if (&otherBlock != &block)

        {

          const BasicBlock &other = otherBlock.second;

          if (other.genSet.find(var) != other.genSet.end())

          {

            // If the variable is defined in another block, add it to the KILL set

            currentBlock.killSet.insert(var);

            break;

          }

        }

      }

    }

  }

}

int main()

{

  vector<int> leaders = {1};

  findLeaders(leaders);

  cout << "Leaders:\n"

       << endl;

  for (int i = 0; i < leaders.size(); i++)

  {

    cout << leaders[i] << " ";

  }

  cout << endl;

  printBlocks(leaders);

  for (int i = 1; i <= basicBlocks.size(); ++i)

  {

    auto it = basicBlocks.find(i);

    if (it != basicBlocks.end())

    {

      cout << "\nBlock " << it->first << " Instructions:" << endl;

      cout << "---------------------\n";

      for (const auto &instruction : it->second.instructions)

      {

        cout << instruction << endl;

      }

    }

  }

  cout << endl;

  computeGen();

  computeKill();

  // Print GEN and KILL sets for each block

  for (int i = 1; i <= basicBlocks.size(); ++i)

  {

    auto it = basicBlocks.find(i);

    if (it != basicBlocks.end())

    {

      cout << "Block " << it->first << " GEN Set:";

      for (const string &var : it->second.genSet)

      {

        cout << " " << var;

      }

      cout << endl;

      cout << "Block " << it->first << " KILL Set:";

      for (const string &var : it->second.killSet)

      {

        cout << " " << var;

      }

      cout << endl

           << endl;

    }

  }

}

Output: -

