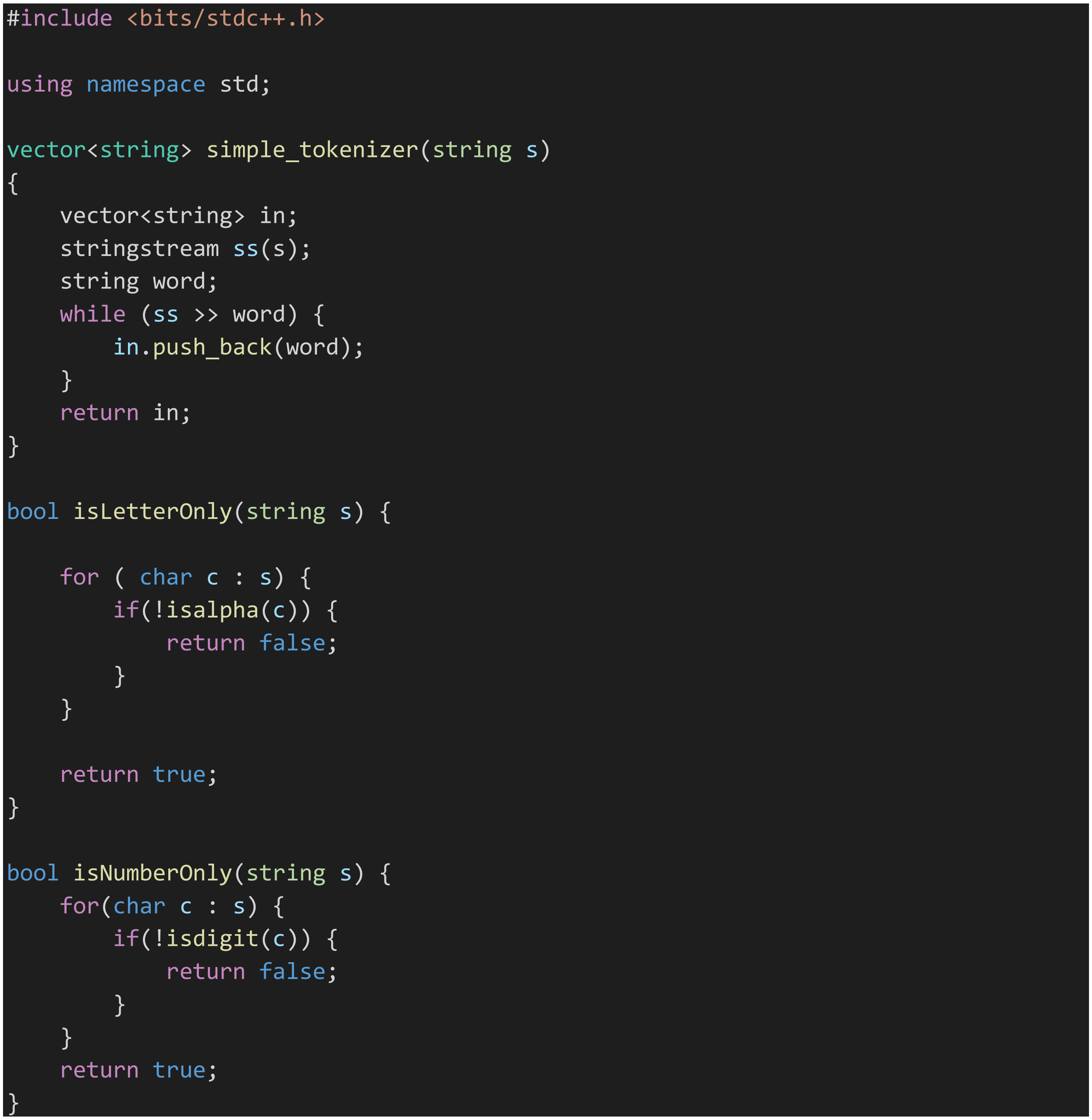
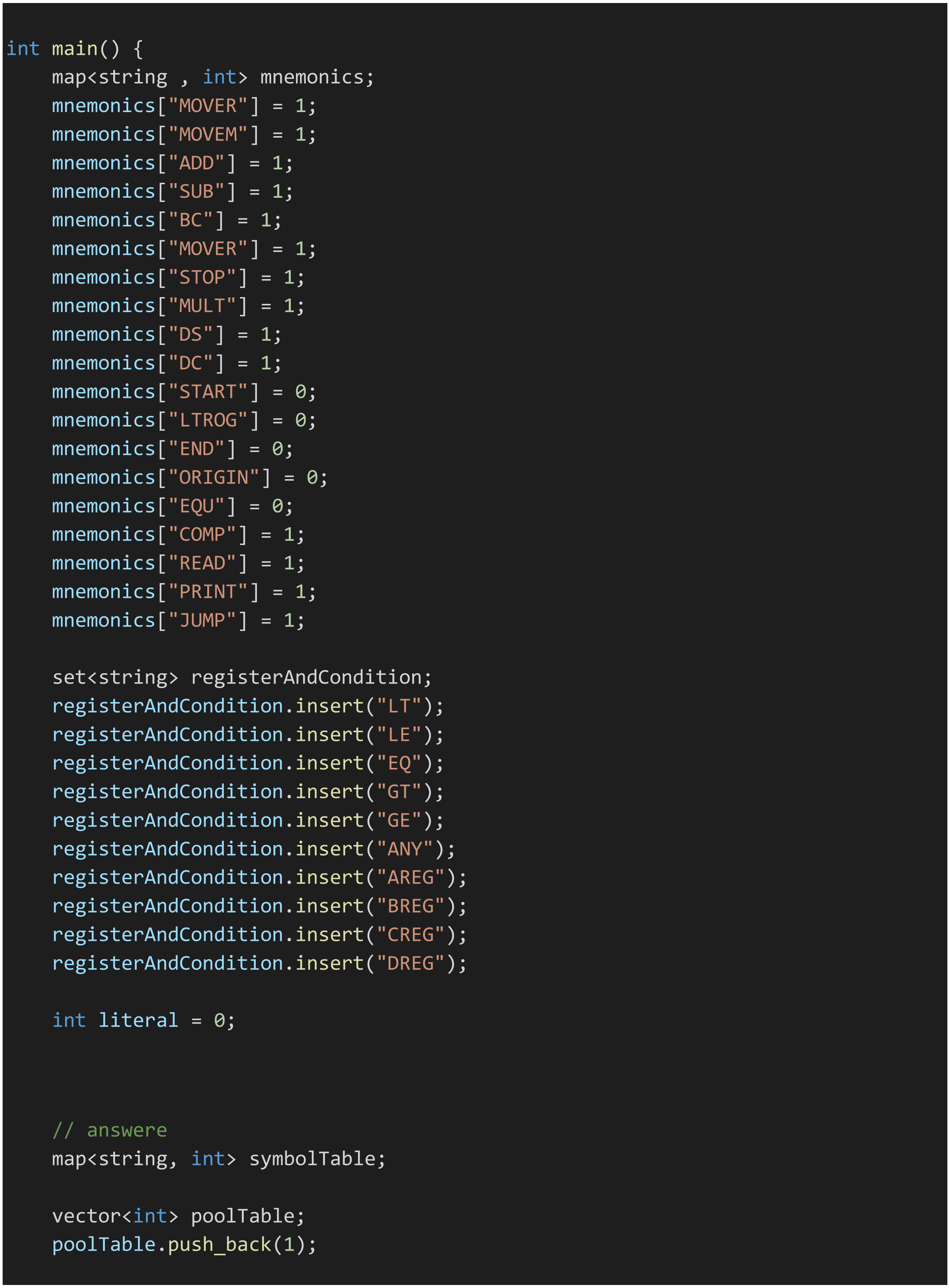
U18CO018
Shubham Shekhaliya
Assignment – 4 (SS)

Generate variant-I and variant-II representation for multiplication of two numbers.

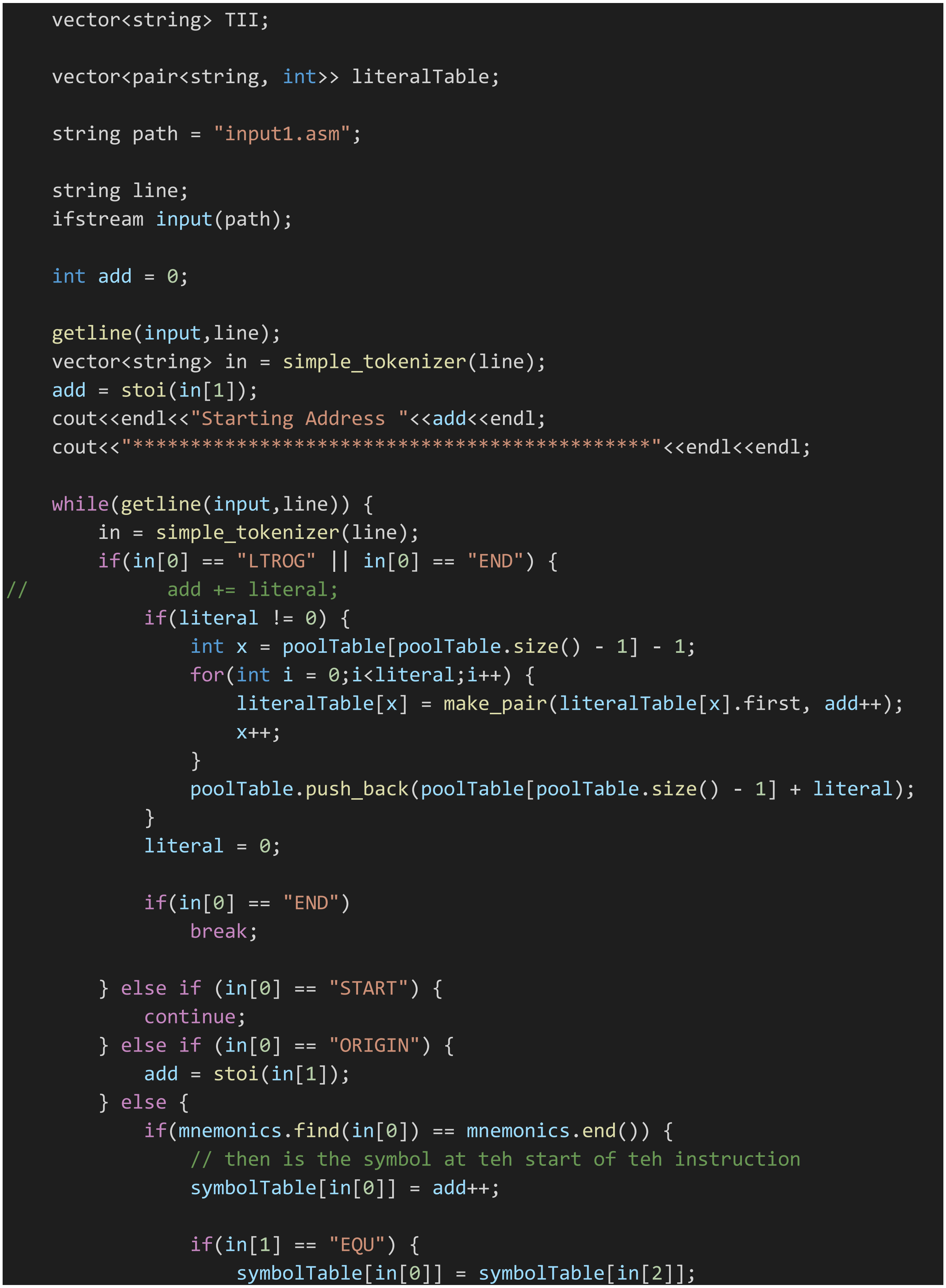
Code:-



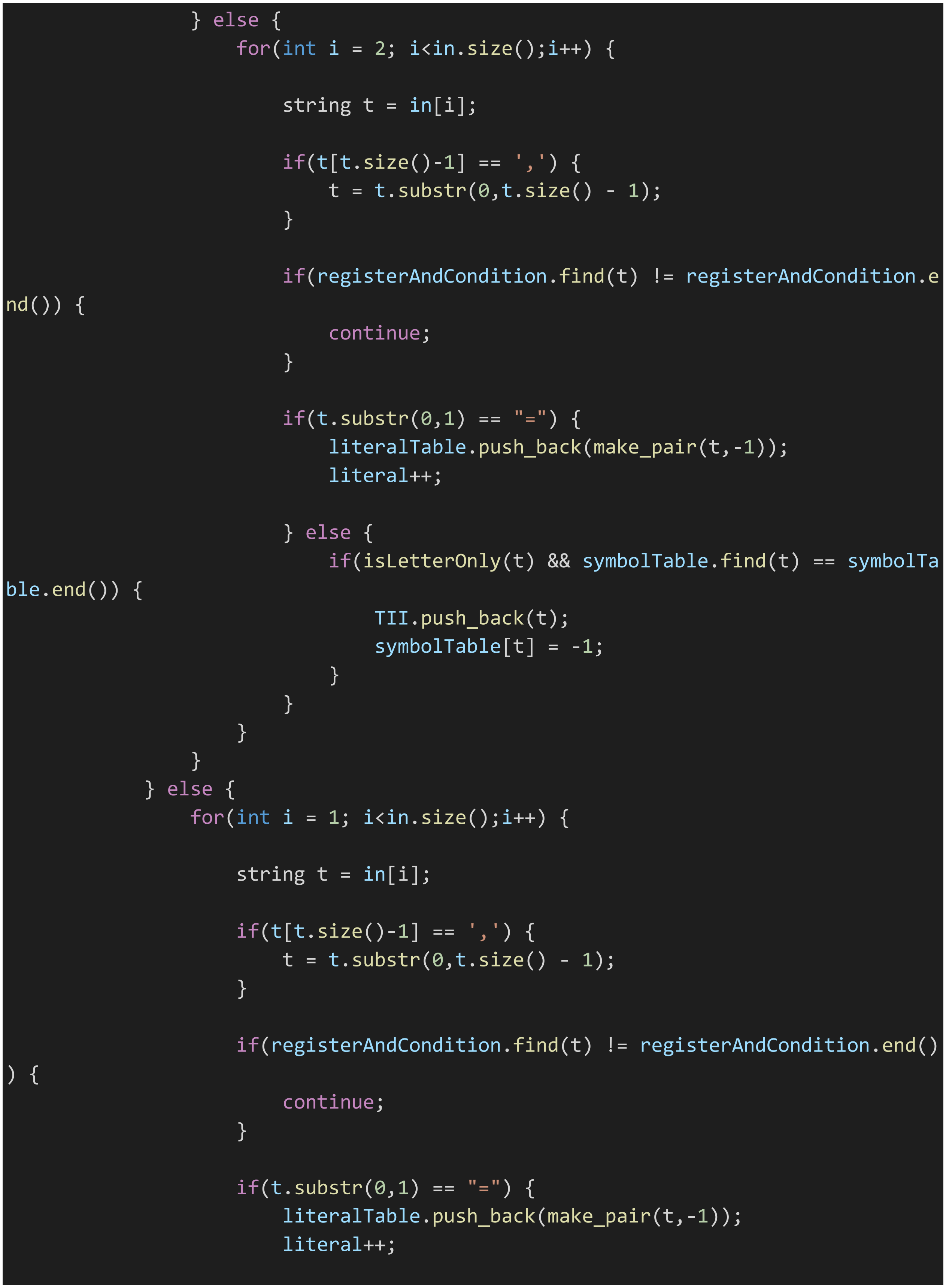
```
#include <bits/stdc++.h>

using namespace std;

vector<string> simple_tokenizer(string s)
{
    vector<string> in;
    stringstream ss(s);
    string word;
    while (ss >> word) {
        in.push_back(word);
    }
    return in;
}

bool isLetterOnly(string s) {

    for ( char c : s) {
        if(!isalpha(c)) {
            return false;
        }
    }

    return true;
}

bool isNumberOnly(string s) {
    for(char c : s) {
        if(!isdigit(c)) {
            return false;
        }
    }
    return true;
}
```



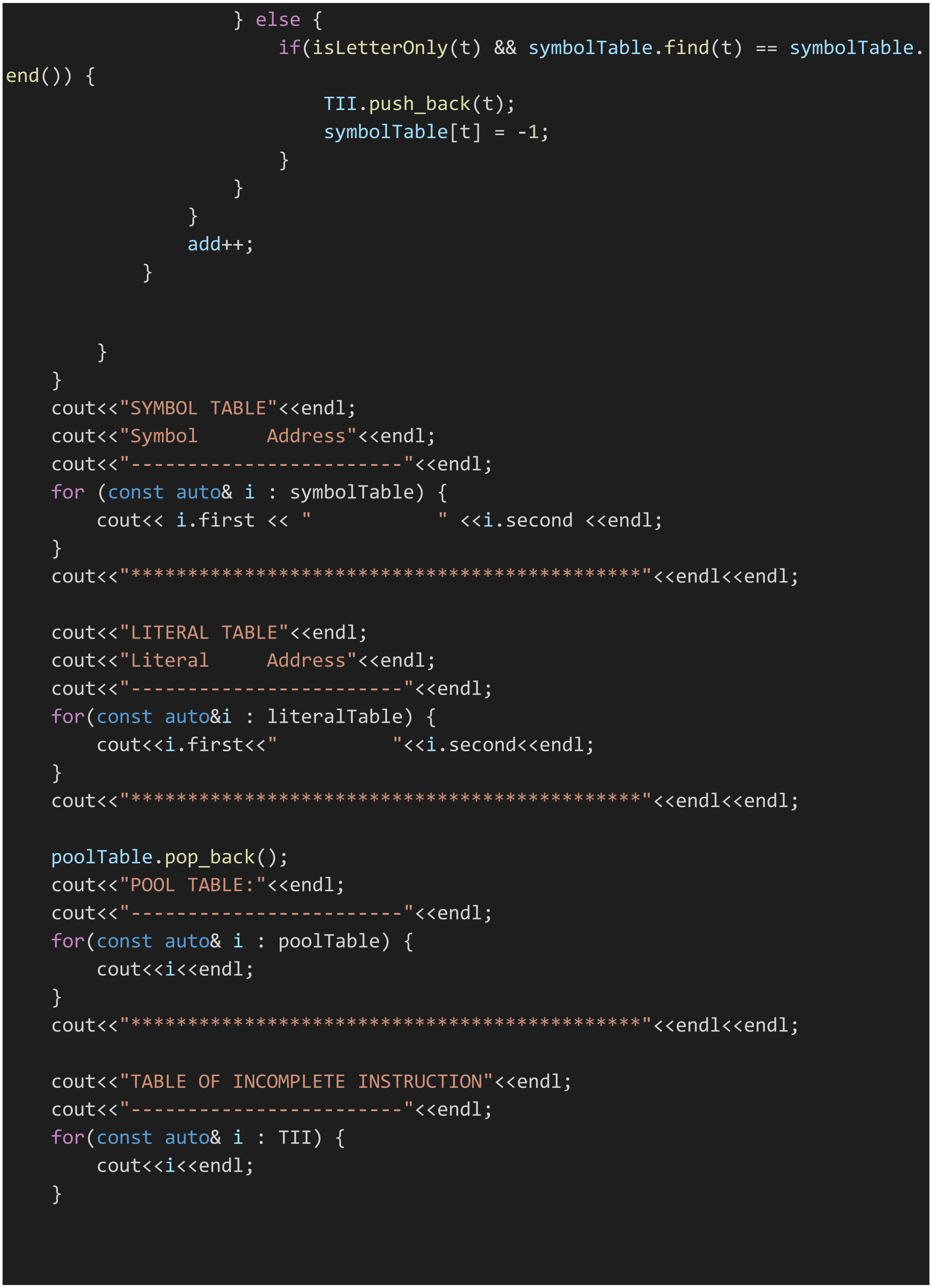
```
int main() {
    map<string , int> mnemonics;
    mnemonics["MOVER"] = 1;
    mnemonics["MOVEM"] = 1;
    mnemonics["ADD"] = 1;
    mnemonics["SUB"] = 1;
    mnemonics["BC"] = 1;
    mnemonics["MOVER"] = 1;
    mnemonics["STOP"] = 1;
    mnemonics["MULT"] = 1;
    mnemonics["DS"] = 1;
    mnemonics["DC"] = 1;
    mnemonics["START"] = 0;
    mnemonics["LTROG"] = 0;
    mnemonics["END"] = 0;
    mnemonics["ORIGIN"] = 0;
    mnemonics["EQU"] = 0;
    mnemonics["COMP"] = 1;
    mnemonics["READ"] = 1;
    mnemonics["PRINT"] = 1;
    mnemonics["JUMP"] = 1;

    set<string> registerAndCondition;
    registerAndCondition.insert("LT");
    registerAndCondition.insert("LE");
    registerAndCondition.insert("EQ");
    registerAndCondition.insert("GT");
    registerAndCondition.insert("GE");
    registerAndCondition.insert("ANY");
    registerAndCondition.insert("AREG");
    registerAndCondition.insert("BREG");
    registerAndCondition.insert("CREG");
    registerAndCondition.insert("DREG");

    int literal = 0;



    // answere
    map<string, int> symbolTable;

    vector<int> poolTable;
    poolTable.push_back(1);
```



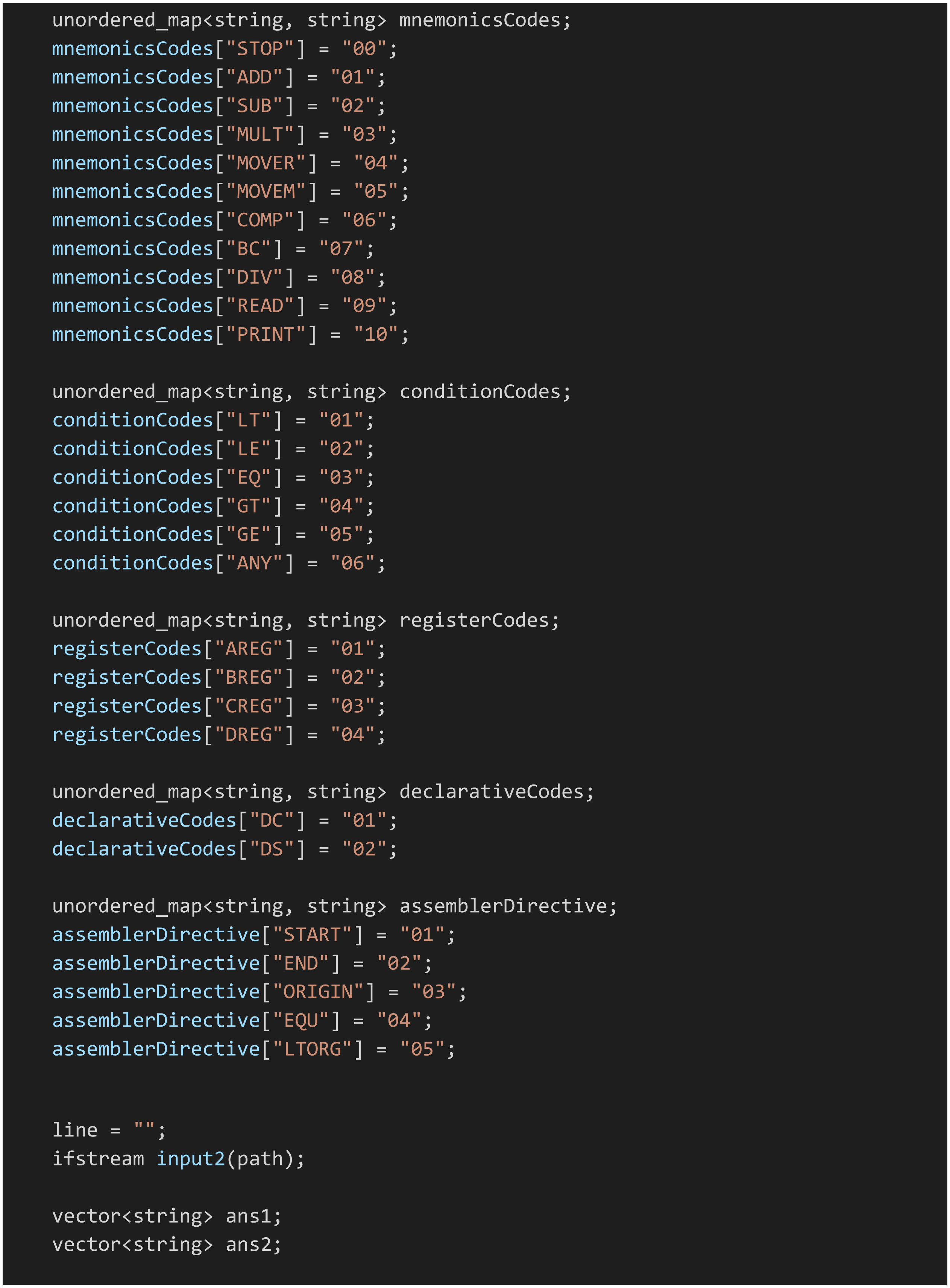
```
    vector<string> TII;

    vector<pair<string, int>> literalTable;

    string path = "input1.asm";

    string line;
    ifstream input(path);

    int add = 0;

    getline(input,line);
    vector<string> in = simple_tokenizer(line);
    add = stoi(in[1]);
    cout<<endl<<"Starting Address "<<add<<endl;
    cout<<"**********************************************"<<endl<<endl;

    while(getline(input,line)) {
        in = simple_tokenizer(line);
        if(in[0] == "LTROG" || in[0] == "END") {
//            add += literal;
            if(literal != 0) {
                int x = poolTable[poolTable.size() - 1] - 1;
                for(int i = 0;i<literal;i++) {
                    literalTable[x] = make_pair(literalTable[x].first, add++);
                    x++;
                }
                poolTable.push_back(poolTable[poolTable.size() - 1] + literal);
            }
            literal = 0;

            if(in[0] == "END")
                break;

        } else if (in[0] == "START") {
            continue;
        } else if (in[0] == "ORIGIN") {
            add = stoi(in[1]);
        } else {
            if(mnemonics.find(in[0]) == mnemonics.end()) {
                // then is the symbol at teh start of teh instruction
                symbolTable[in[0]] = add++;

                if(in[1] == "EQU") {
                    symbolTable[in[0]] = symbolTable[in[2]];
```



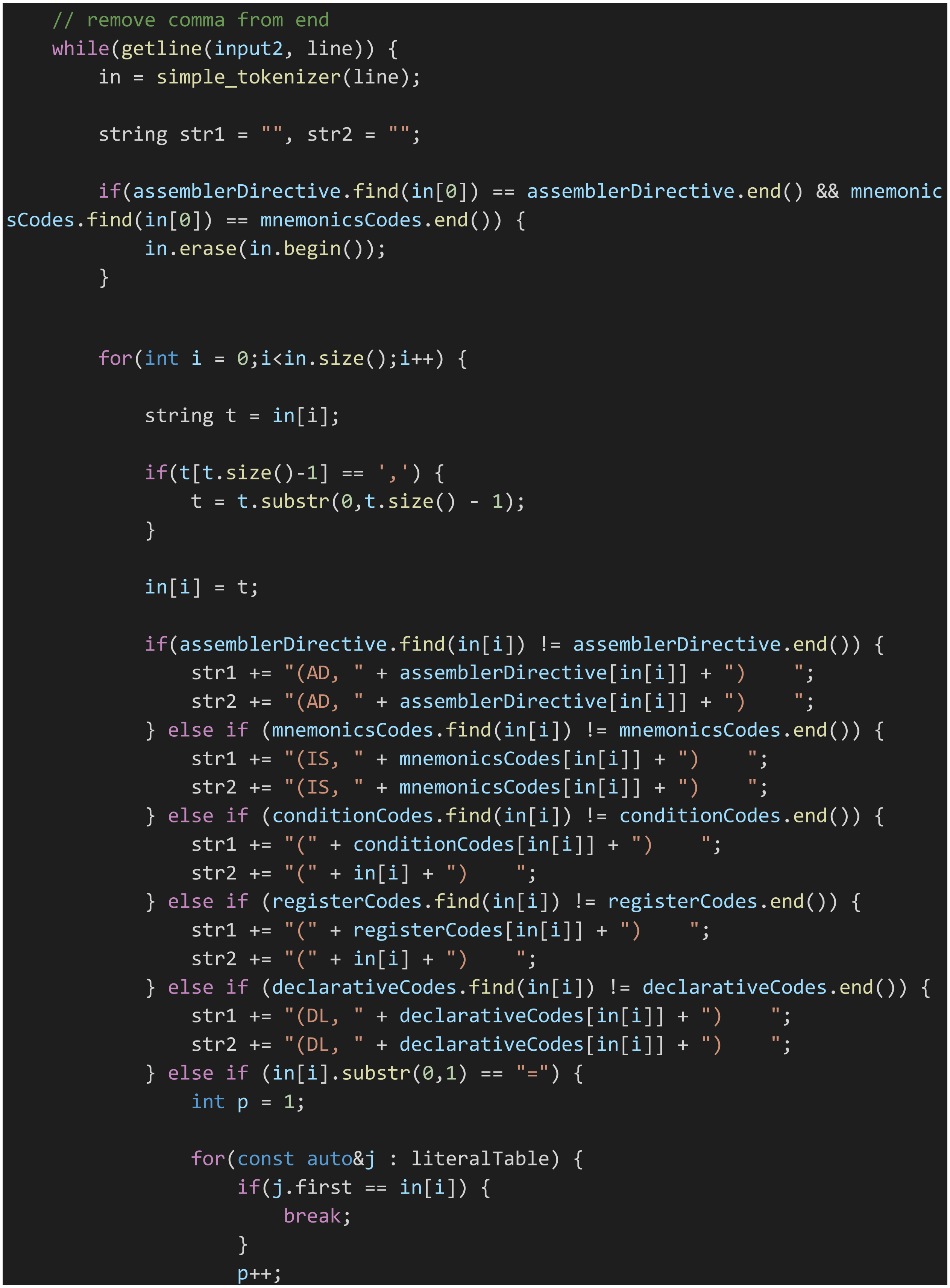
```
                } else {
                    for(int i = 2; i<in.size();i++) {

                        string t = in[i];

                        if(t[t.size()-1] == ',') {
                            t = t.substr(0,t.size() - 1);
                        }

                        if(registerAndCondition.find(t) != registerAndCondition.e
nd()) {

                            continue;
                        }

                        if(t.substr(0,1) == "=") {
                            literalTable.push_back(make_pair(t,-1));
                            literal++;

                        } else {
                            if(isLetterOnly(t) && symbolTable.find(t) == symbolTa
ble.end()) {

                                TII.push_back(t);
                                symbolTable[t] = -1;
                            }
                        }
                    }
                }
            } else {
                for(int i = 1; i<in.size();i++) {

                    string t = in[i];

                    if(t[t.size()-1] == ',') {
                        t = t.substr(0,t.size() - 1);
                    }

                    if(registerAndCondition.find(t) != registerAndCondition.end()
) {

                        continue;
                    }

                    if(t.substr(0,1) == "=") {
                        literalTable.push_back(make_pair(t,-1));
                        literal++;
```



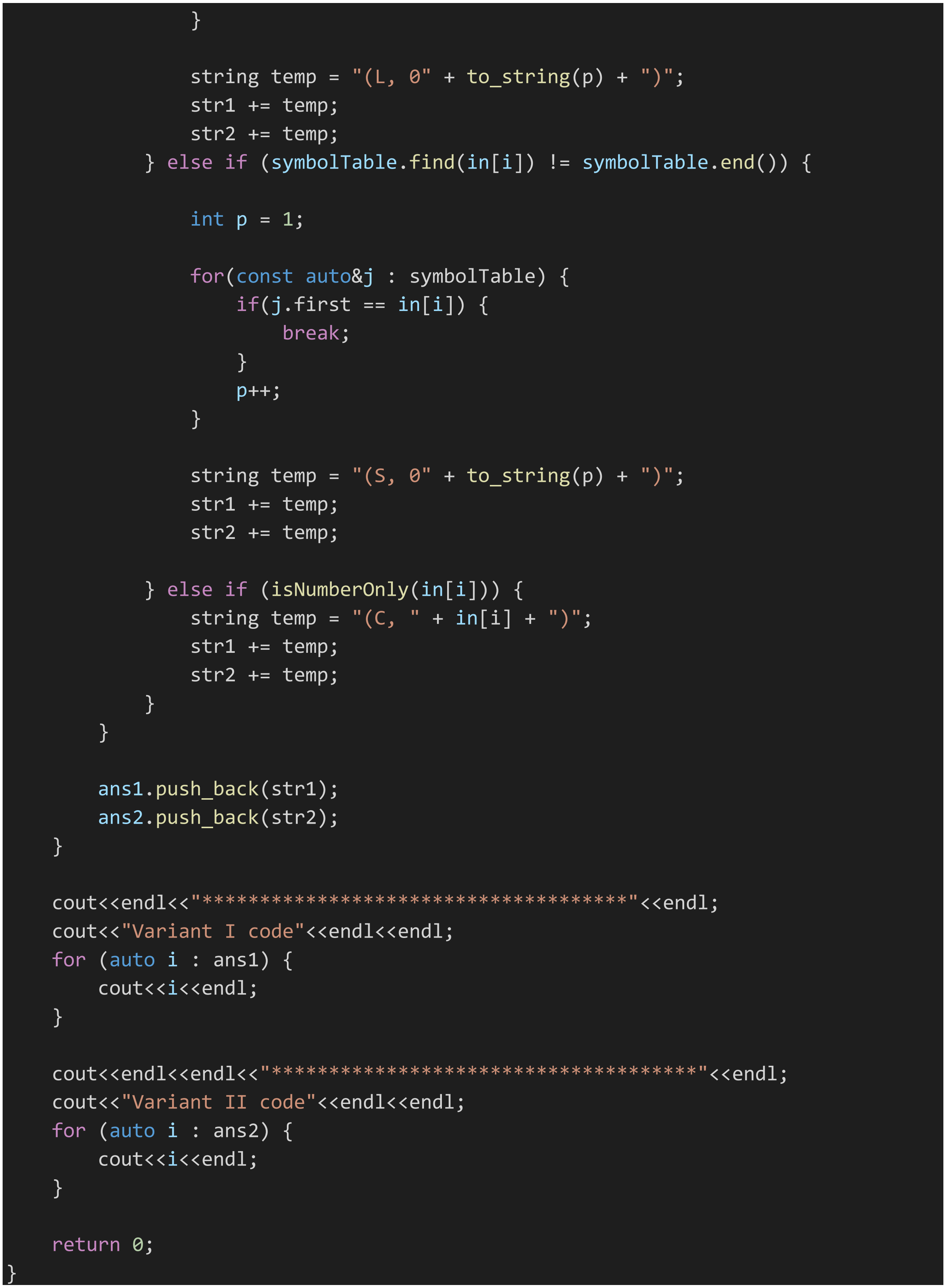
```
                    } else {
                        if(isLetterOnly(t) && symbolTable.find(t) == symbolTable.
end()) {
                            TII.push_back(t);
                            symbolTable[t] = -1;
                        }
                    }
                }
                add++;
            }


        }
    }
    cout<<"SYMBOL TABLE"<<endl;
    cout<<"Symbol      Address"<<endl;
    cout<<"------------------------"<<endl;
    for (const auto& i : symbolTable) {
        cout<< i.first << "           " <<i.second <<endl;
    }
    cout<<"**********************************************"<<endl<<endl;

    cout<<"LITERAL TABLE"<<endl;
    cout<<"Literal     Address"<<endl;
    cout<<"------------------------"<<endl;
    for(const auto&i : literalTable) {
        cout<<i.first<<"          "<<i.second<<endl;
    }
    cout<<"**********************************************"<<endl<<endl;

    poolTable.pop_back();
    cout<<"POOL TABLE:"<<endl;
    cout<<"------------------------"<<endl;
    for(const auto& i : poolTable) {
        cout<<i<<endl;
    }
    cout<<"**********************************************"<<endl<<endl;

    cout<<"TABLE OF INCOMPLETE INSTRUCTION"<<endl;
    cout<<"------------------------"<<endl;
    for(const auto& i : TII) {
        cout<<i<<endl;
    }
```



```
    unordered_map<string, string> mnemonicsCodes;
    mnemonicsCodes["STOP"] = "00";
    mnemonicsCodes["ADD"] = "01";
    mnemonicsCodes["SUB"] = "02";
    mnemonicsCodes["MULT"] = "03";
    mnemonicsCodes["MOVER"] = "04";
    mnemonicsCodes["MOVEM"] = "05";
    mnemonicsCodes["COMP"] = "06";
    mnemonicsCodes["BC"] = "07";
    mnemonicsCodes["DIV"] = "08";
    mnemonicsCodes["READ"] = "09";
    mnemonicsCodes["PRINT"] = "10";

    unordered_map<string, string> conditionCodes;
    conditionCodes["LT"] = "01";
    conditionCodes["LE"] = "02";
    conditionCodes["EQ"] = "03";
    conditionCodes["GT"] = "04";
    conditionCodes["GE"] = "05";
    conditionCodes["ANY"] = "06";

    unordered_map<string, string> registerCodes;
    registerCodes["AREG"] = "01";
    registerCodes["BREG"] = "02";
    registerCodes["CREG"] = "03";
    registerCodes["DREG"] = "04";

    unordered_map<string, string> declarativeCodes;
    declarativeCodes["DC"] = "01";
    declarativeCodes["DS"] = "02";

    unordered_map<string, string> assemblerDirective;
    assemblerDirective["START"] = "01";
    assemblerDirective["END"] = "02";
    assemblerDirective["ORIGIN"] = "03";
    assemblerDirective["EQU"] = "04";
    assemblerDirective["LTORG"] = "05";


    line = "";
    ifstream input2(path);

    vector<string> ans1;
    vector<string> ans2;
```

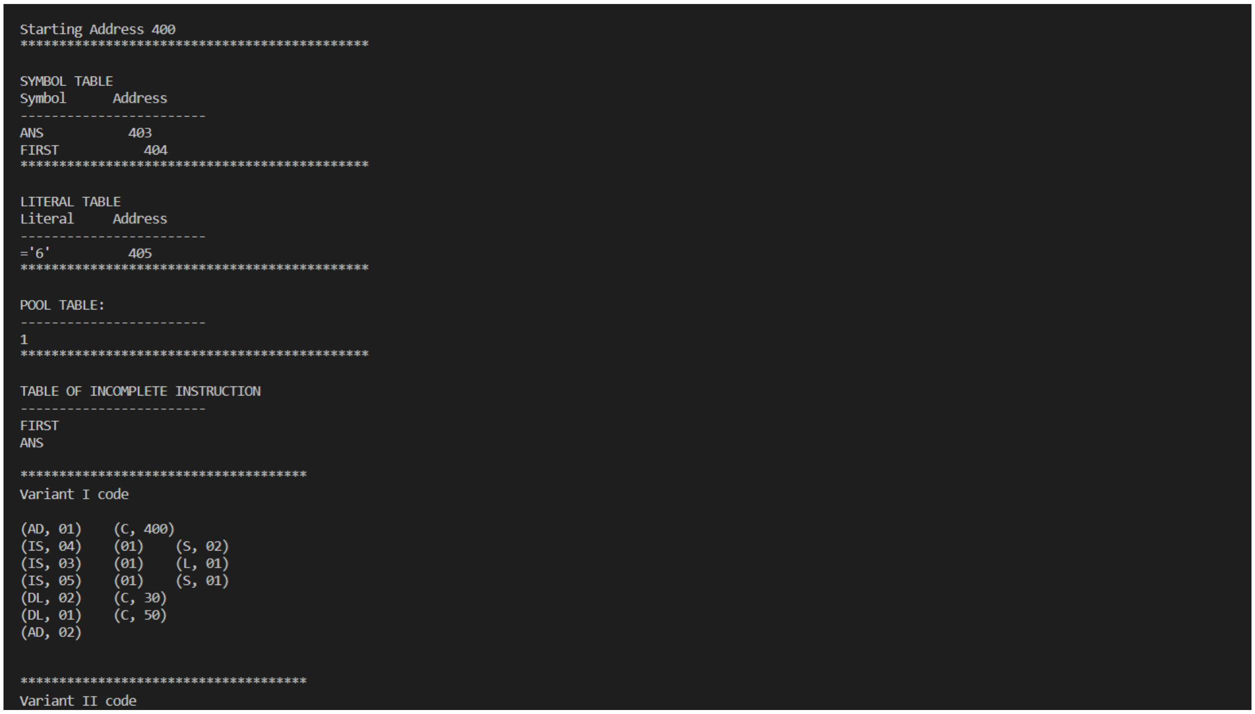


```
    // remove comma from end
    while(getline(input2, line)) {
        in = simple_tokenizer(line);

        string str1 = "", str2 = "";

        if(assemblerDirective.find(in[0]) == assemblerDirective.end() && mnemonic
sCodes.find(in[0]) == mnemonicsCodes.end()) {
            in.erase(in.begin());
        }


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

            string t = in[i];

            if(t[t.size()-1] == ',') {
                t = t.substr(0,t.size() - 1);
            }

            in[i] = t;

            if(assemblerDirective.find(in[i]) != assemblerDirective.end()) {
                str1 += "(AD, " + assemblerDirective[in[i]] + ")    ";
                str2 += "(AD, " + assemblerDirective[in[i]] + ")    ";
            } else if (mnemonicsCodes.find(in[i]) != mnemonicsCodes.end()) {
                str1 += "(IS, " + mnemonicsCodes[in[i]] + ")    ";
                str2 += "(IS, " + mnemonicsCodes[in[i]] + ")    ";
            } else if (conditionCodes.find(in[i]) != conditionCodes.end()) {
                str1 += "(" + conditionCodes[in[i]] + ")    ";
                str2 += "(" + in[i] + ")    ";
            } else if (registerCodes.find(in[i]) != registerCodes.end()) {
                str1 += "(" + registerCodes[in[i]] + ")    ";
                str2 += "(" + in[i] + ")    ";
            } else if (declarativeCodes.find(in[i]) != declarativeCodes.end()) {
                str1 += "(DL, " + declarativeCodes[in[i]] + ")    ";
                str2 += "(DL, " + declarativeCodes[in[i]] + ")    ";
            } else if (in[i].substr(0,1) == "=") {
                int p = 1;

                for(const auto&j : literalTable) {
                    if(j.first == in[i]) {
                        break;
                    }
                    p++;
```

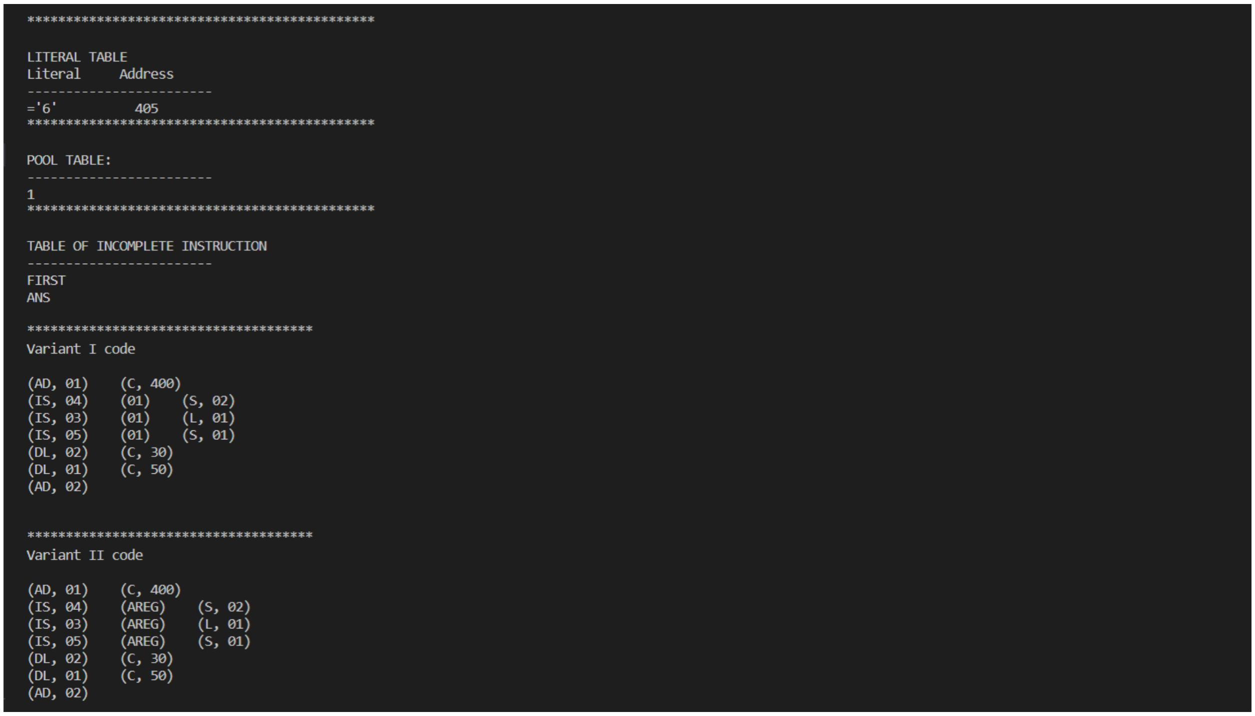


```
                }

                string temp = "(L, 0" + to_string(p) + ")";
                str1 += temp;
                str2 += temp;
            } else if (symbolTable.find(in[i]) != symbolTable.end()) {

                int p = 1;

                for(const auto&j : symbolTable) {
                    if(j.first == in[i]) {
                        break;
                    }
                    p++;
                }

                string temp = "(S, 0" + to_string(p) + ")";
                str1 += temp;
                str2 += temp;

            } else if (isNumberOnly(in[i])) {
                string temp = "(C, " + in[i] + ")";
                str1 += temp;
                str2 += temp;
            }
        }

        ans1.push_back(str1);
        ans2.push_back(str2);
    }

    cout<<endl<<"*************************************"<<endl;
    cout<<"Variant I code"<<endl<<endl;
    for (auto i : ans1) {
        cout<<i<<endl;
    }

    cout<<endl<<endl<<"*************************************"<<endl;
    cout<<"Variant II code"<<endl<<endl;
    for (auto i : ans2) {
        cout<<i<<endl;
    }

    return 0;
}
```

Output:-



```
Starting Address 400
*********************************************

SYMBOL TABLE
Symbol      Address
------------------------
ANS           403
FIRST           404
*********************************************

LITERAL TABLE
Literal     Address
------------------------
='6'          405
*********************************************

POOL TABLE:
------------------------
1
*********************************************

TABLE OF INCOMPLETE INSTRUCTION
------------------------
FIRST
ANS

*************************************
Variant I code

(AD, 01)    (C, 400)
(IS, 04)    (01)    (S, 02)
(IS, 03)    (01)    (L, 01)
(IS, 05)    (01)    (S, 01)
(DL, 02)    (C, 30)
(DL, 01)    (C, 50)
(AD, 02)


*************************************
Variant II code
```



```
*********************************************

LITERAL TABLE
Literal     Address
------------------------
='6'          405
*********************************************

POOL TABLE:
------------------------
1
*********************************************

TABLE OF INCOMPLETE INSTRUCTION
------------------------
FIRST
ANS

*************************************
Variant I code

(AD, 01)    (C, 400)
(IS, 04)    (01)    (S, 02)
(IS, 03)    (01)    (L, 01)
(IS, 05)    (01)    (S, 01)
(DL, 02)    (C, 30)
(DL, 01)    (C, 50)
(AD, 02)


*************************************
Variant II code

(AD, 01)    (C, 400)
(IS, 04)    (AREG)    (S, 02)
(IS, 03)    (AREG)    (L, 01)
(IS, 05)    (AREG)    (S, 01)
(DL, 02)    (C, 30)
(DL, 01)    (C, 50)
(AD, 02)
```