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

const int MAX_SYMBOLS = 100;
const int MAX_LITERALS = 100;
const int MAX_CODE_LINES = 100;

// Symbol and Literal tables
struct Entry {
  int index;
  string name;
  int address;
};

Entry ST[MAX_SYMBOLS], LT[MAX_LITERALS];
int ST_index = 0, LT_index = 0, address = 0;

string MOT_names[] = {"STOP", "ADD", "SUB", "MULT", "MOVER", "MOVEM", "COMP", "BC", "DIV",
"READ", "PRINT", "START", "END", "ORIGIN", "LTORG", "DS", "DC", "AREG", "BREG", "EQ"};
string MOT_codes[] = {"00", "01", "02", "03", "04", "05", "06", "07", "08", "09", "10", "01", "02",
"03","04" "05", "01", "02", "01", "02", "01"};
int MOT_size = 20;

// Code array for intermediate storage
struct Code {
  string type;
  int index;
} code[MAX_CODE_LINES][3];
int codeLine = 0;

string getClass(string word, int &opcode) {
  // Strip trailing comma if present
  if (word.back() == ',') {
    word.pop_back();
  }

  for (int i = 0; i < MOT_size; i++) {
    if (word == MOT_names[i]) {
      opcode = stoi(MOT_codes[i]);
      if (i < 10) return "IS";
      else if (i < 14) return "AD";
      else if (i < 16) return "DL";
      else if (i < 18) return "RG";
      else return "CC";
    }
  }
  return "None";
}

void addEntry(Entry table[], int &tableIndex, string name, int addr) {
  for (int i = 0; i < tableIndex; i++)
    if (table[i].name == name) return;
  table[tableIndex++] = {tableIndex, name, addr};
}

int main() {
```

```
string lines[] = {
    "START 100",
    "MOVER AREG, ='5'",
    "ADD BREG, ='10'",
    "SUB C",
    "END"
};
int lineCount = 5;

// Process START directive
if (lines[0].substr(0, 5) == "START") address = stoi(lines[0].substr(6));

for (int i = 1; i < lineCount; i++) {
    string line = lines[i], word = "";
    int entryIndex = 0;
    int opcode = -1; // Variable to hold opcode

    for (int j = 0; j <= line.size(); j++) {
        if (j < line.size() && line[j] != ' ') word += line[j];
        else {
            string type = getClass(word, opcode);
            if (type != "None") {
                code[codeLine][entryIndex++] = {type, opcode};
            } else if (word[0] == '=') {
                addEntry(LT, LT_index, word, -1);  // -1 as placeholder for address
                code[codeLine][entryIndex++] = {"L", LT_index - 1};
            } else {
                addEntry(ST, ST_index, word, address);
                code[codeLine][entryIndex++] = {"S", ST_index - 1};
            }
            word = "";
        }
    }
    address++;
    codeLine++;
}
for (int i = 0; i < LT_index; i++) {
    if (LT[i].address == -1) { // Check if address is not assigned
        LT[i].address = address;
        address++; // Increment for the next literal
    }
}

// Display results
cout << "Literal Table:\n";
for (int i = 0; i < LT_index; i++) {
    cout << LT[i].index << " " << LT[i].name << " " << LT[i].address << endl;
}

cout << "\nSymbol Table:\n";
for (int i = 0; i < ST_index; i++) {
    cout << ST[i].index << " " << ST[i].name << " " << ST[i].address << endl;
}

cout << "\nIntermediate Code:\n";
for (int i = 0; i < codeLine; i++) {
    for (int j = 0; j < 3 && code[i][j].type != ""; j++) {
            cout << "(" << code[i][j].type << ", " << code[i][j].index << ") ";
        }
        cout << endl;
    }

    return 0;
}
```