#include <iostream>

#include <fstream>

#include <sstream>

#include <unordered\_map>

#include <vector>

#include <string>

#include <iomanip>

// Structure to store symbol table entries

struct SymbolTableEntry {

std::string symbol;

int address;

};

// Structure for storing opcode details

struct Opcode {

std::string mnemonic;

int size; // Size of the instruction

int opcode; // Machine code

};

// Pass-One Assembler class

class PassOneAssembler {

private:

std::unordered\_map<std::string, SymbolTableEntry> symbolTable;

std::unordered\_map<std::string, Opcode> opcodeTable;

int locationCounter;

public:

PassOneAssembler() : locationCounter(0) {

initializeOpcodes();

}

// Initialize opcode table with some sample instructions and sizes

void initializeOpcodes() {

opcodeTable["LDA"] = {"LDA", 3, 0x00};

opcodeTable["STA"] = {"STA", 3, 0x0C};

opcodeTable["ADD"] = {"ADD", 3, 0x18};

opcodeTable["SUB"] = {"SUB", 3, 0x1C};

opcodeTable["MUL"] = {"MUL", 3, 0x20};

}

// Method to parse each line and build the symbol table

void parseLine(const std::string& line) {

std::istringstream stream(line);

std::string label, opcode, operand;

// Check if there is a label

if (line[0] != ' ') {

stream >> label >> opcode >> operand;

addSymbol(label, locationCounter);

} else {

stream >> opcode >> operand;

}

if (opcode == "START") {

locationCounter = std::stoi(operand);

std::cout << "START at " << locationCounter << "\n";

} else if (opcodeTable.find(opcode) != opcodeTable.end()) {

locationCounter += opcodeTable[opcode].size;

} else if (opcode == "RESW") {

locationCounter += 3 \* std::stoi(operand); // Reserve word (3 bytes per word)

} else if (opcode == "RESB") {

locationCounter += std::stoi(operand); // Reserve bytes

} else if (opcode == "WORD") {

locationCounter += 3; // Single word (3 bytes)

} else if (opcode == "BYTE") {

locationCounter += 1; // Single byte

} else if (opcode == "END") {

std::cout << "END of program\n";

}

}

// Method to add symbol to symbol table

void addSymbol(const std::string& symbol, int address) {

if (symbolTable.find(symbol) == symbolTable.end()) {

symbolTable[symbol] = {symbol, address};

} else {

std::cerr << "Error: Symbol " << symbol << " already defined.\n";

}

}

// Method to display the symbol table

void displaySymbolTable() const {

std::cout << "\nSymbol Table:\n";

std::cout << std::setw(10) << "Symbol" << std::setw(10) << "Address" << "\n";

for (const auto& entry : symbolTable) {

std::cout << std::setw(10) << entry.second.symbol << std::setw(10) << entry.second.address << "\n";

}

}

};

int main() {

PassOneAssembler assembler;

// Simulated input for the assembler

std::vector<std::string> sourceCode = {

"COPY START 1000",

"FIRST LDA ALPHA",

" ADD BETA",

" STA GAMMA",

"ALPHA RESW 1",

"BETA RESB 1",

"GAMMA WORD 1",

" END"

};

for (const auto& line : sourceCode) {

assembler.parseLine(line);

}

assembler.displaySymbolTable();

return 0;

}