```cpp
// TINY COMPUTER ASSEMBLY INTERPRETER
// Allison Smith | Spring 2025
// The University of Findlay - Computer Organization

// This program reads a tiny computer assembly program from the file "tiny.txt",
// converts it into machine language, and then interprets and executes it.
// ------------------------------------------------------------
// format of data file, for every line:
// 1-4  : optional label
// 5    : colon if label exists, otherwise a space
// 6-8  : instruction, three characters in length
// 10-13: first argument or LOC value
// 15-18: second argument, if any
// end of program is signified with a line containing "ZZZ"
// ------------------------------------------------------------

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

const int MAX_SYMBOLS = 990;
const int R1 = 991;
const int R2 = 992;
const int MEMORY_SIZE = 993;

struct Symbol {
	string name;
	int location;
};

Symbol symbolTable[MAX_SYMBOLS];

int symSize = 0;
int PC = 0;
int memory[MEMORY_SIZE];

void readInputP1(const string& inputFile);
void getLabels(string line);

void readInputP2(const string& inputFile);
int getAddress(string arg);
void convertToML(string line);

void interpreter();

// ------------------------------------------------------------

int main() {

	string inputFile = "tiny.txt";
	readInputP1(inputFile);
	readInputP2(inputFile);

	interpreter();

	return 0;
}
```

```
// ------------------------------------------------------------------
// PASS ONE FUNCTIONS

void readInputP1(const string& inputFile) {
	ifstream file(inputFile);
	string line;

	PC = 0;

	while (getline(file, line) && line != "ZZZ") {
		getLabels(line);
	}
	file.close();
}



void getLabels(string line) {
	string name = line.substr(0, 4); // check for label
	string command = line.substr(5, 3); // check for command

	if (command == "ZZZ") return;

	// if there is a label, record the name and location
	if (name != "    ") {
		symbolTable[symSize].name = name;
		symbolTable[symSize].location = PC;
		symSize++;
	}

	PC++;
}

// ------------------------------------------------------------------
// PASS TWO FUNCTIONS

void readInputP2(const string& inputFile) {
	ifstream file(inputFile);
	string line;

	PC = 0;

	while (getline(file, line) && line != "ZZZ") {
		convertToML(line);
	}
	file.close();
}



int getAddress(string arg) {
	// check if pre-defined registers
	if (arg == "R1  ") return R1;
	if (arg == "R2  ") return R2;
	if (arg == "    ") return 0;

	// look up in symbol table
```

```
	for (int i = 0; i < symSize; i++) {
		if (symbolTable[i].name == arg) {
			return symbolTable[i].location;
		}
	}

	return 0;
}



void convertToML(string line) {

	string command = line.substr(5, 3);
	string arg1 = line.substr(9, 4);
	string arg2 = line.substr(14, 4);

	int opCode;

	// convert command to opCode
	if (command == "INP") opCode = 1;
	else if (command == "OUT") opCode = 2;
	else if (command == "COP") opCode = 3;
	else if (command == "ADD") opCode = 4;
	else if (command == "SUB") opCode = 5;
	else if (command == "MUL") opCode = 6;
	else if (command == "DIV") opCode = 7;
	else if (command == "JMP") opCode = 8;
	else if (command == "JNG") opCode = 9;
	else if (command == "HLT") opCode = 0;

	else if (command == "ZZZ") return;
	else if (command == "LOC") {
		int value = stoi(arg1);
		memory[PC] = value;
		PC++;
		return;
	}


	// get the locations of arguments
	int arg1Add = getAddress(arg1);
	int arg2Add = getAddress(arg2);

	int instruction = (opCode * 1000000) + (arg1Add * 1000) + (arg2Add);

	memory[PC] = instruction;
	PC++;

}

// ------------------------------------------------------------
// INTERPRETER FUNCTION

void interpreter() {
	PC = 0;
	// set opCode so while loop will begin
	int opCode = -1;
```

```
	while (opCode != 0) {
		int instruction = memory[PC];
		PC++;

		int opCode = (instruction / 1000000);
		int arg1 = (instruction / 1000) % 1000;
		int arg2 = (instruction % 1000);

		switch (opCode) {
		case 0: // HALT
			return;
		case 1: // INPUT
			cout << "Input ? ";
			cin >> memory[R1];
			break;
		case 2: // OUTPUT
			cout << "Output = " << memory[R1] << endl;
			break;
		case 3: // COPY
			memory[arg2] = memory[arg1];
			break;
		case 4: // ADD
			memory[R2] += memory[arg1];
			break;
		case 5: // SUBTRACT
			memory[R2] -= memory[arg1];
			break;
		case 6: // MULTIPLY
			memory[R2] *= memory[arg1];
			break;
		case 7: // DIVIDE
			memory[R2] /= memory[arg1];
			break;
		case 8: // JUMP
			PC = arg1;
			break;
		case 9: // CONDITIONAL JUMP
			if (memory[R2] < 0)
				PC = arg1;
			break;
		default:
			return;
		}
	}
}
```

```
// ------------------------------------------------------------
// Input File: tiny.txt
// ------------------------------------------------------------
     INP
     COP R1  ,larg
     COP ONE ,i
For :COP FOUR,R2
     SUB i
     COP R2  ,R1
     JNG EndF
     INP
     COP R1  ,num
     COP num ,R2
     SUB larg
     COP R2  ,R1
     JNG EndI
     COP num ,larg
EndI:COP i   ,R2
     ADD ONE
     COP R2  ,i
     JMP For
EndF:COP larg,R1
     OUT
     HLT
larg:LOC 0
num :LOC 0
FOUR:LOC 4
ONE :LOC 1
i   :LOC 0
     ZZZ
// ------------------------------------------------------------
```