|  |
| --- |
| Experiment No.4 |
| Design & Implementation of Pass 2 of Two Pass Macro Processor |
| Date of Performance: 7/2/2025 |
| Date of Submission: 14/2/2025 |

**Aim:**  Design & Implementation of Pass 2 of Two Pass Macro Processor.

**Objective:** To study and implement Pass 2 of two pass Macro Processor for IBM 360 Machine.

**Theory:**

**Macro:** A macro is a unit of specification for program generation through expansion. A macro instruction is a notational convenience for the programmer. It allows the programmer to write shorthand version of a program (module programming).

**Macro Processor:** The macro processor replaces each macro invocation with the corresponding sequence of statements.

**Pass 2: Expand all macro invocation statements.**

* Monitors the values of expansion time variables & sequencing symbols specified in a Macro
* Handles expansion time control flow & performs expansion of model statements.

**Pass 2 Databases of Macro Processor**

1. Copy of source program statements

2. Output expanded source listings for input to assembler

3. MNT and MDT generated in pass 1

4. Macro Definition Table Pointer (MDTP) directs to the next statement to be used during expansion

5. An array called Parameter List Array (PLA) for substituting parameters of macro call in the source macro of the macro call in the stored macro definitions for the index correspondingly.

**Pass 2 flowchart:**

**MACRO name found?**

**Pass 2**

**Supply expanded source file to assembler processing**

**Write into expanded source card file**

**Read next source card (copied by pass 1)**

**MDTP <- MDTP + 1**

**Set up argument list array**

**MDTP <- MDT index from MNT entry**

**Get line from MDT**

**Substitute arguments from macro call**

**END pseudo-op?**

**MEND pseudo-op?**

**Write expanded source card**

**Yes**

**Yes**

**Yes**

**No**

**No**

**No**

**Search MNT for match with operation code**

**Code:**

**#include <iostream>**

**#include <fstream>**

**#include <sstream>**

**#include <vector>**

**#include <unordered\_map>**

**#include <string>**

**#include <algorithm>**

**using namespace std;**

**struct Macro {**

**string name;**

**vector<string> parameters;**

**vector<string> body;**

**};**

**string trim(const string& str) {**

**size\_t first = str.find\_first\_not\_of(' ');**

**if (first == string::npos) return "";**

**size\_t last = str.find\_last\_not\_of(' ');**

**return str.substr(first, (last - first + 1));**

**}**

**vector<string> split(const string& str, char delimiter) {**

**vector<string> tokens;**

**string token;**

**istringstream tokenStream(str);**

**while (getline(tokenStream, token, delimiter)) {**

**tokens.push\_back(trim(token));**

**}**

**return tokens;**

**}**

**int main() {**

**ifstream intermediateFile("intermediate.asm");**

**ifstream mntFile("mnt.txt");**

**ifstream mdtFile("mdt.txt");**

**ofstream outputFile("expanded.asm");**

**if (!intermediateFile.is\_open() || !mntFile.is\_open() || !mdtFile.is\_open() || !outputFile.is\_open()) {**

**cerr << "Error opening file." << endl;**

**return 1;**

**}**

**unordered\_map<string, Macro> macroTable;**

**string line;**

**while (getline(mntFile, line)) {**

**istringstream iss(line);**

**string macroName;**

**int paramCount;**

**iss >> macroName >> paramCount;**

**Macro macro;**

**macro.name = macroName;**

**while (getline(mdtFile, line)) {**

**if (line == "MEND") break;**

**istringstream mdtIss(line);**

**string word;**

**mdtIss >> word;**

**if (word == macroName) {**

**for (int i = 0; i < paramCount; ++i) {**

**string param;**

**mdtIss >> param;**

**macro.parameters.push\_back(param);**

**}**

**} else {**

**macro.body.push\_back(line);**

**}**

**}**

**macroTable[macroName] = macro;**

**}**

**while (getline(intermediateFile, line)) {**

**istringstream iss(line);**

**string label, opcode;**

**iss >> label >> opcode;**

**if (macroTable.find(label) != macroTable.end()) {**

**Macro macro = macroTable[label];**

**vector<string> arguments;**

**string arg;**

**while (iss >> arg) {**

**arguments.push\_back(arg);**

**}**

**unordered\_map<string, string> paramMap;**

**for (size\_t i = 0; i < macro.parameters.size(); ++i) {**

**if (i < arguments.size()) {**

**paramMap[macro.parameters[i]] = arguments[i];**

**} else {**

**paramMap[macro.parameters[i]] = "";**

**}**

**}**

**for (const auto& bodyLine : macro.body) {**

**string expandedLine = bodyLine;**

**for (const auto& param : paramMap) {**

**size\_t pos;**

**while ((pos = expandedLine.find(param.first)) != string::npos) {**

**expandedLine.replace(pos, param.first.length(), param.second);**

**}**

**}**

**outputFile << expandedLine << endl;**

**}**

**} else {**

**outputFile << line << endl;**

**}**

**}**

**intermediateFile.close();**

**mntFile.close();**

**mdtFile.close();**

**outputFile.close();**

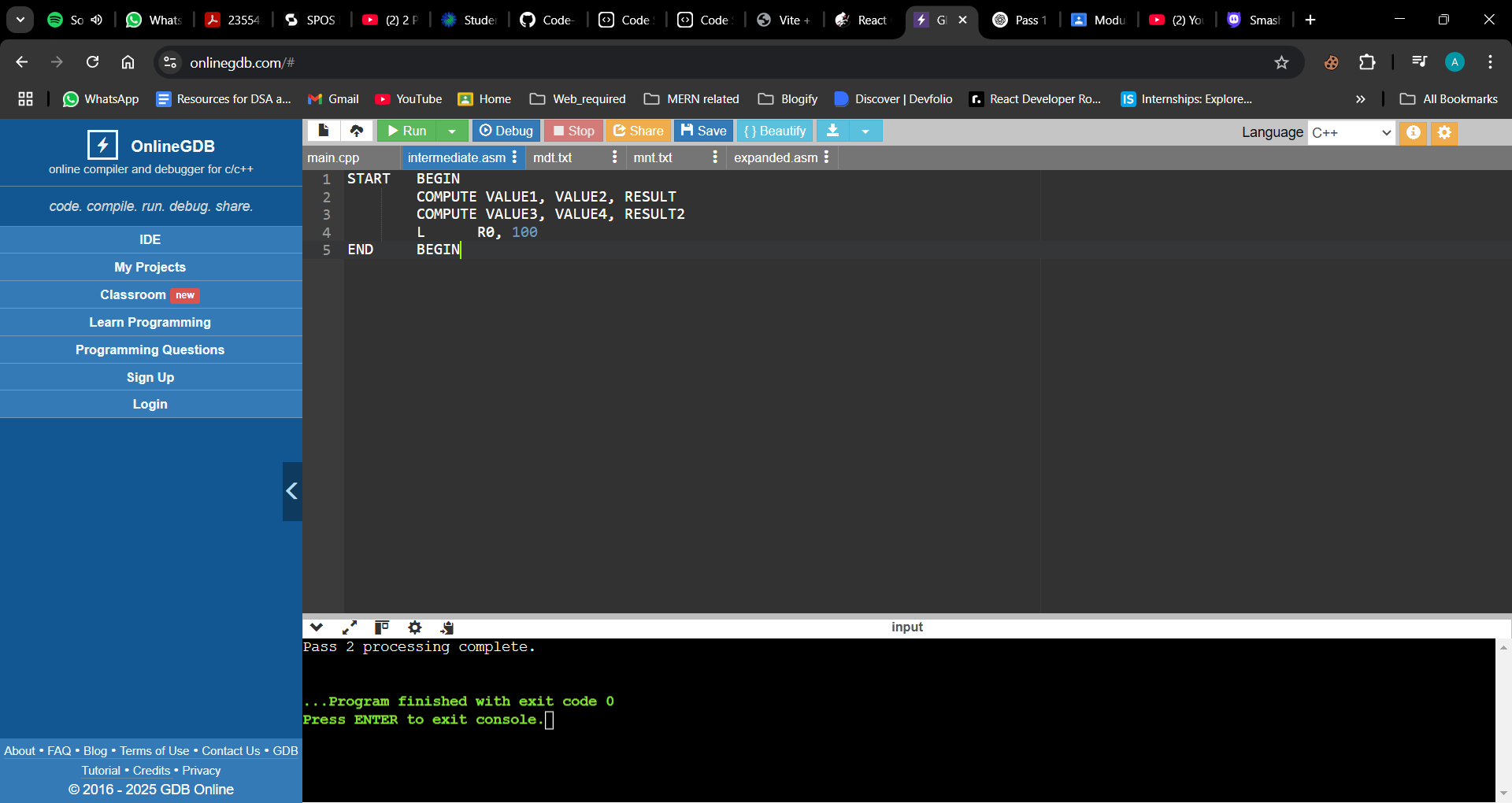
**cout << "Pass 2 processing complete." << endl;**

**return 0;**

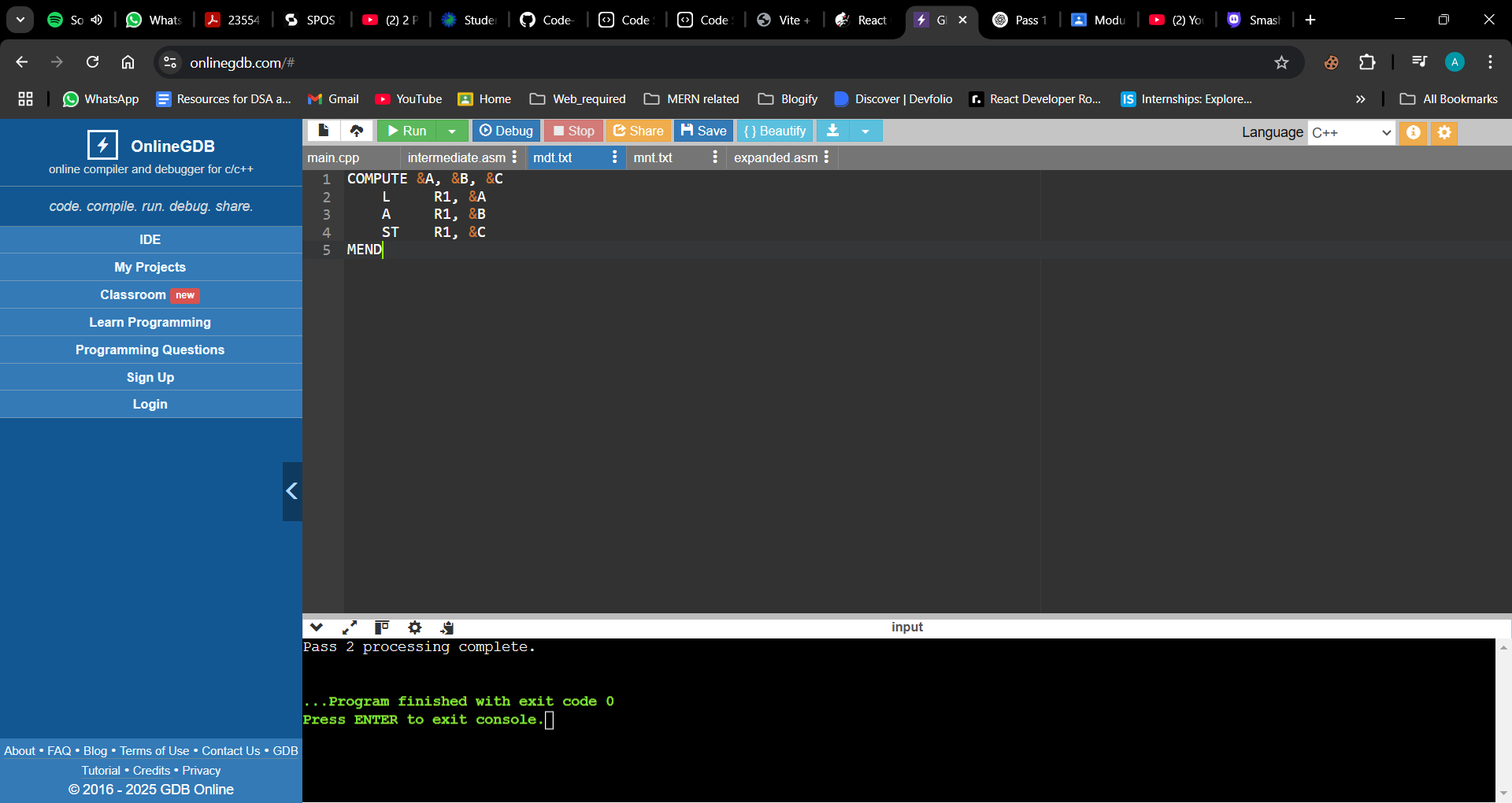
**}**

**Input:**

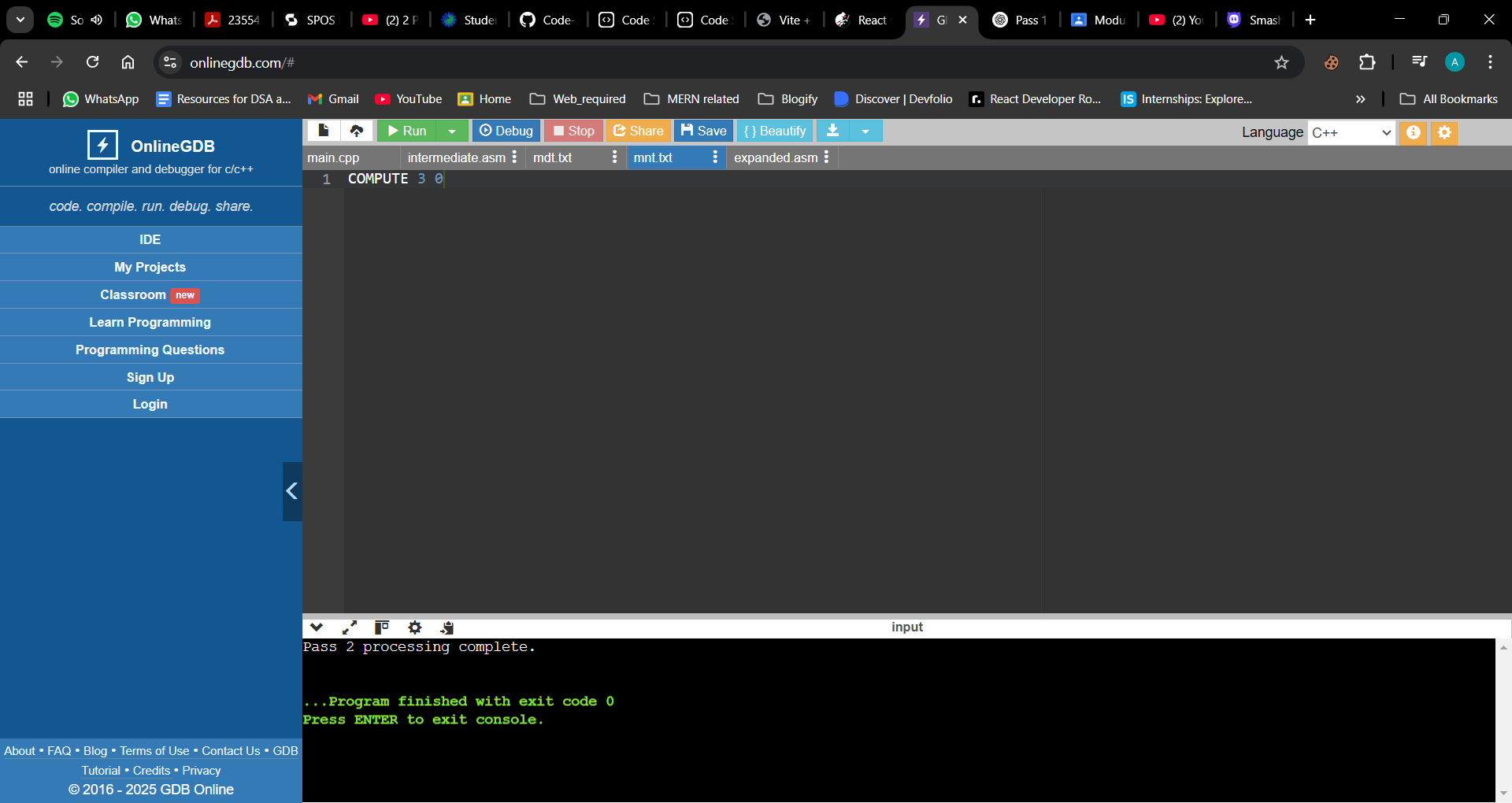
* **Intermediate File -** intermediate.asm: Contains the source code with macro invocations to be expanded.

****

* **Macro Definition Table File -** mdt.txt: Holds the macro definitions, including their bodies.

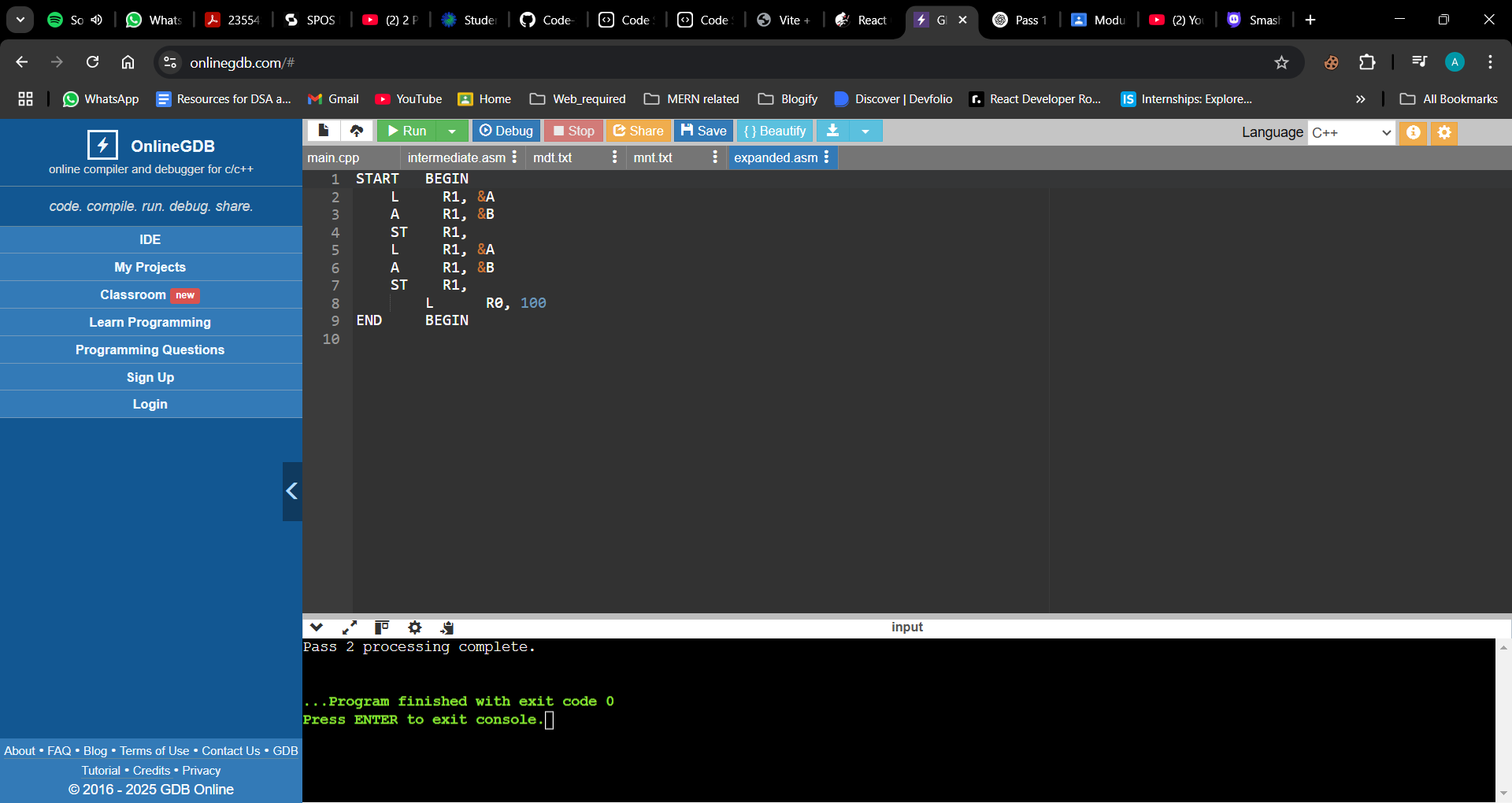
****

* **Macro Name Table File -** mnt.txt: Lists macro names and their parameter counts.

****

**Output:**

**Output File** - expanded.asm: The final expanded source code with all macros replaced by their corresponding instructions.

****

**Conclusion:**

1) **Program Functionality with Input Source Program:**

The implemented Pass 2 of the two-pass macro processor effectively expands macro invocations in the intermediate assembly source file. Utilizing the Macro Name Table (MNT) and Macro Definition Table (MDT) generated during Pass 1, the program replaces each macro call with its corresponding sequence of assembly instructions. For instance, given an input source program with a macro definition for COMPUTE and subsequent invocations, the processor accurately expands these calls into the appropriate assembly code.

2) **Output Generated During Macro Invocation Processing:**

The output of Pass 2 is a fully expanded assembly source file (expanded.asm), where all macro invocations have been replaced by their respective instruction sequences. For example, an invocation like COMPUTE VALUE1, VALUE2, RESULT is expanded into:

L R1,VALUE1

A R1,VALUE2

ST R1,RESULT

This ensures that the final assembly code is complete and ready for assembly, with all macros properly expanded.