|  |
| --- |
| Experiment No.3 |
| Design & Implementation of Two Pass Macro Processor |
| Date of Performance: 31/1/2025 |
| Date of Submission: 7/2/2025 |

**Aim:**  Design & Implementation of Pass 1 of Two Pass Macro Processor.

**Objective:** To study and implement Pass 1 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.

You may design a two-pass macro processor.

**Pass 1: Process all macro definitions.**

* Identifies macro definitions and calls in the program
* Determine the formal parameters & their values

**Pass 1 Databases of Macro Processor**

1. Copy of source program statements

2. Output macro source listings for use by Pass-2

3. MDT to store body of macro definition

4. MDTC used to points the next entry towards in MDT

5. NMT to store names of macro defines in the program

6. MNTC used to points towards next entry into MNT.

7. An array called Parameter List Array (PLA) to manage an index for formal parameters (arguments)

**Data structure**



**PASS-I of Macro Processor- Processing Macro Definitions**

1. Initialize MDTC and MNTC

2. Read the next source statement of the program.

3. If the statement contains MACRO pseudo-op. go to step 6.

4. Output the instruction of the statement.

5. If the statement contains END pseudo-op, go to PASS-II else go to step 2

6. Read the next source statement of the program.

7. Make an entry of the macro name and MTDC in MNT at location MNTC and increment MNTC by 1.

8. Prepare the parameter (arguments) list array.

9. Enter macro name into MDT and increment MTDC by 1.

10. Read the next card and substitute index for the parameters (arguments).

11. Enter line into MDT and increment MDT by 1.

12. If MEND pseudo-op is found, go to step 2 else go to step 10

**Pass 1 flowchart:**

**MACRO pseudo-op?**

**Pass 1**

**MDTC<-1**

**MNTC<-1**

**Enter line into MDT**

**Substitute index notation for arguments**

**Go to Pass 2**

**Write copy of source card**

**Read next source card**

**MNTC <- MNTC + 1**

**Enter macro name and current value of MDTC in MNT entry number MNTC**

**Read next source card**

**Prepare argument list array**

**Enter macro name card into MDT**

**MDTC <- MDTC + 1**

**END pseudo-op?**

**MEND pseudo-op?**

**Read next source card**

**MDTC <- MDTC + 1**

**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));

}

int main() {

ifstream inputFile("source.asm");

ofstream outputFile("intermediate.asm");

ofstream mntFile("mnt.txt");

ofstream mdtFile("mdt.txt");

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

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

return 1;

}

unordered\_map<string, Macro> macroTable;

string line;

bool inMacroDefinition = false;

Macro currentMacro;

while (getline(inputFile, line)) {

istringstream iss(line);

string firstWord;

iss >> firstWord;

if (firstWord == "MACRO") {

inMacroDefinition = true;

currentMacro = Macro();

}

else if (firstWord == "MEND") {

inMacroDefinition = false;

macroTable[currentMacro.name] = currentMacro;

mntFile << currentMacro.name << " " << currentMacro.parameters.size() << endl;

mdtFile << currentMacro.name;

for (const auto& param : currentMacro.parameters) {

mdtFile << " " << param;

}

mdtFile << endl;

for (const auto& bodyLine : currentMacro.body) {

mdtFile << bodyLine << endl;

}

mdtFile << "MEND" << endl;

}

else if (inMacroDefinition) {

if (currentMacro.name.empty()) {

iss >> currentMacro.name;

string param;

while (iss >> param) {

currentMacro.parameters.push\_back(param);

}

}

else {

string bodyLine = trim(line);

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

string param = currentMacro.parameters[i];

size\_t pos;

while ((pos = bodyLine.find(param)) != string::npos) {

bodyLine.replace(pos, param.length(), "#" + to\_string(i + 1));

}

}

currentMacro.body.push\_back(bodyLine);

}

} else {

outputFile << line << endl;

}

}

inputFile.close();

outputFile.close();

mntFile.close();

mdtFile.close();

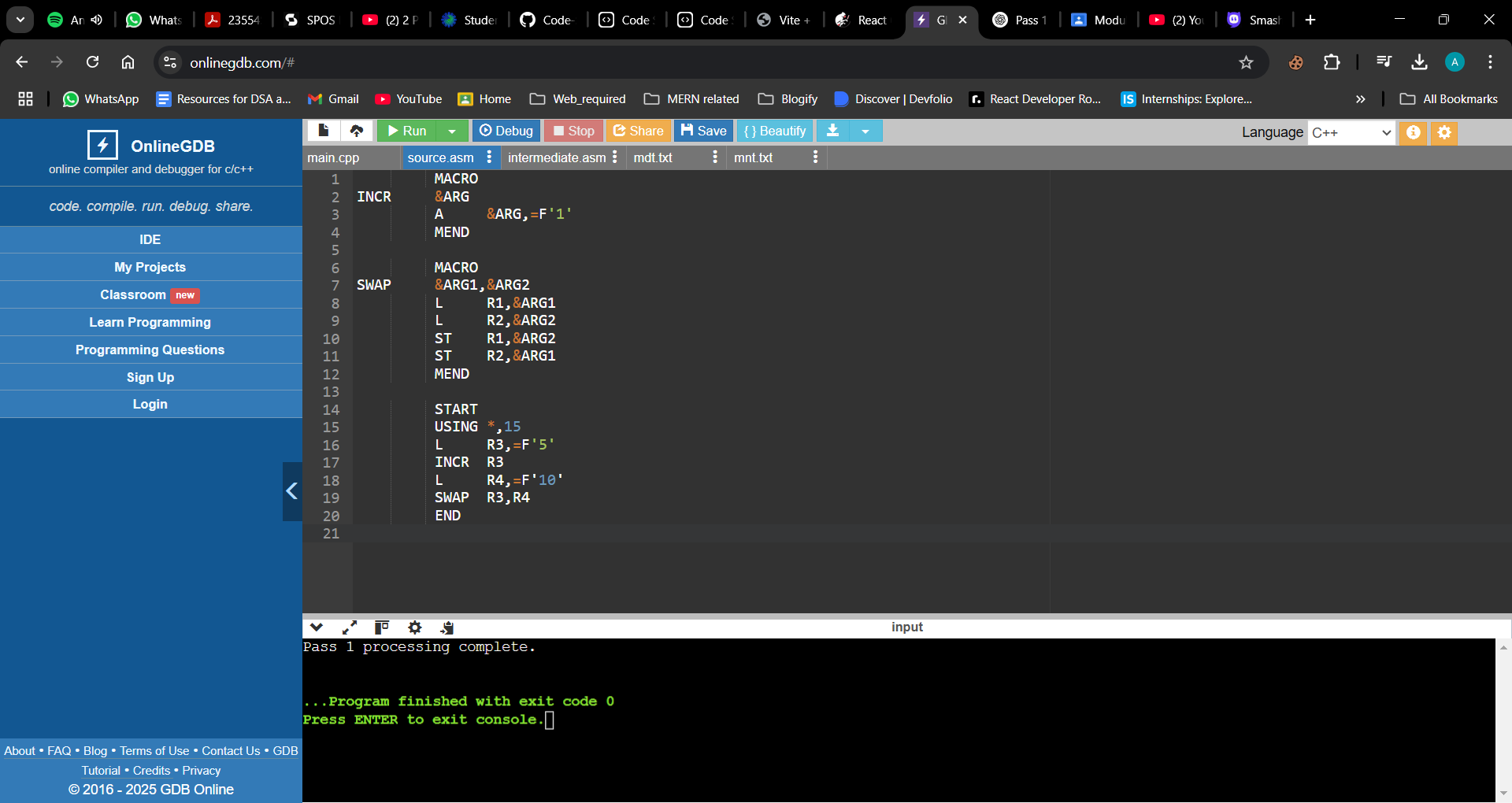
cout << "Pass 1 processing complete." << endl;

return 0;

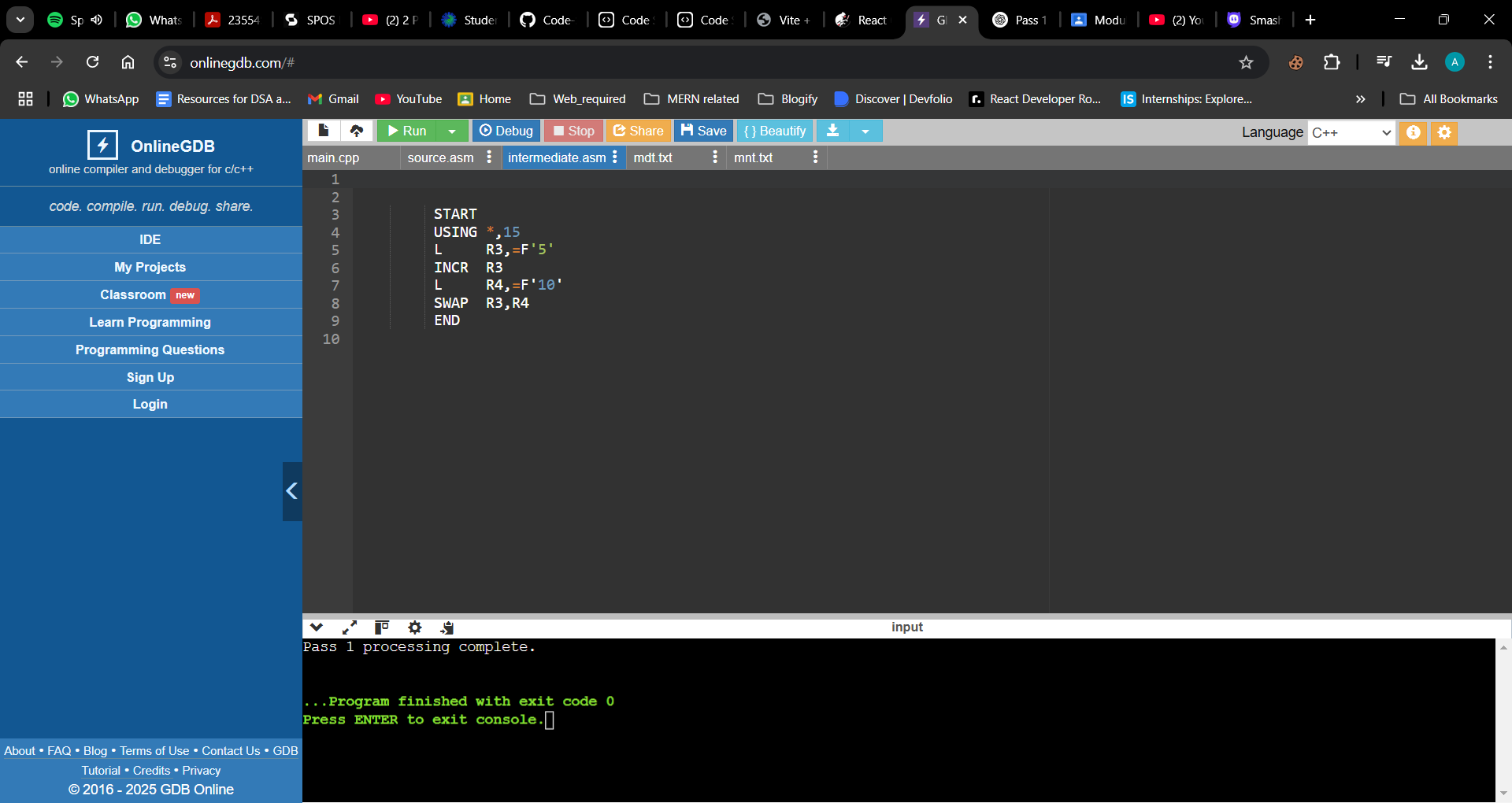
}

**Output:**

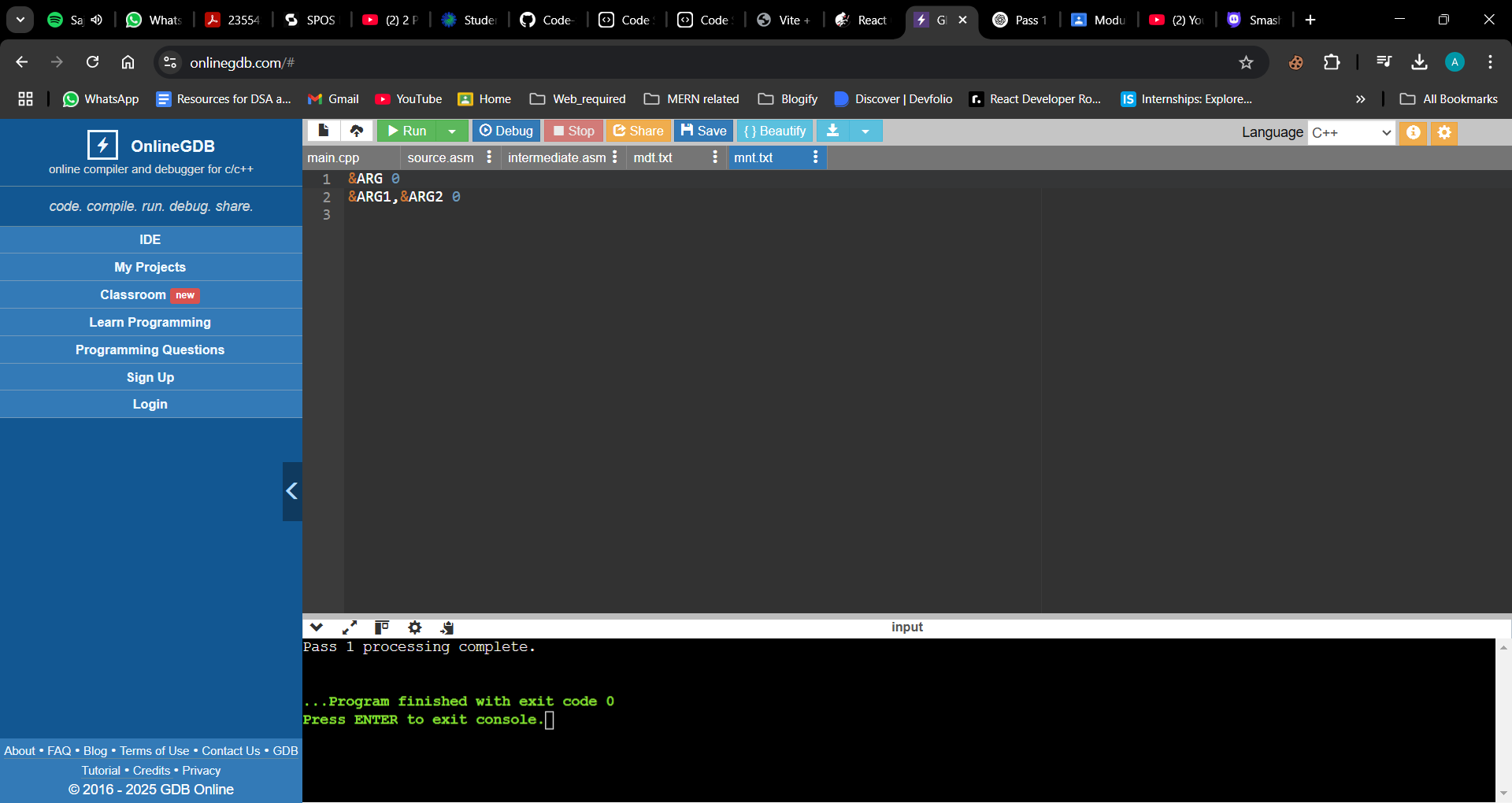
* **Input File** - source.asm**:** The source assembly program containing macro definitions and code.

****

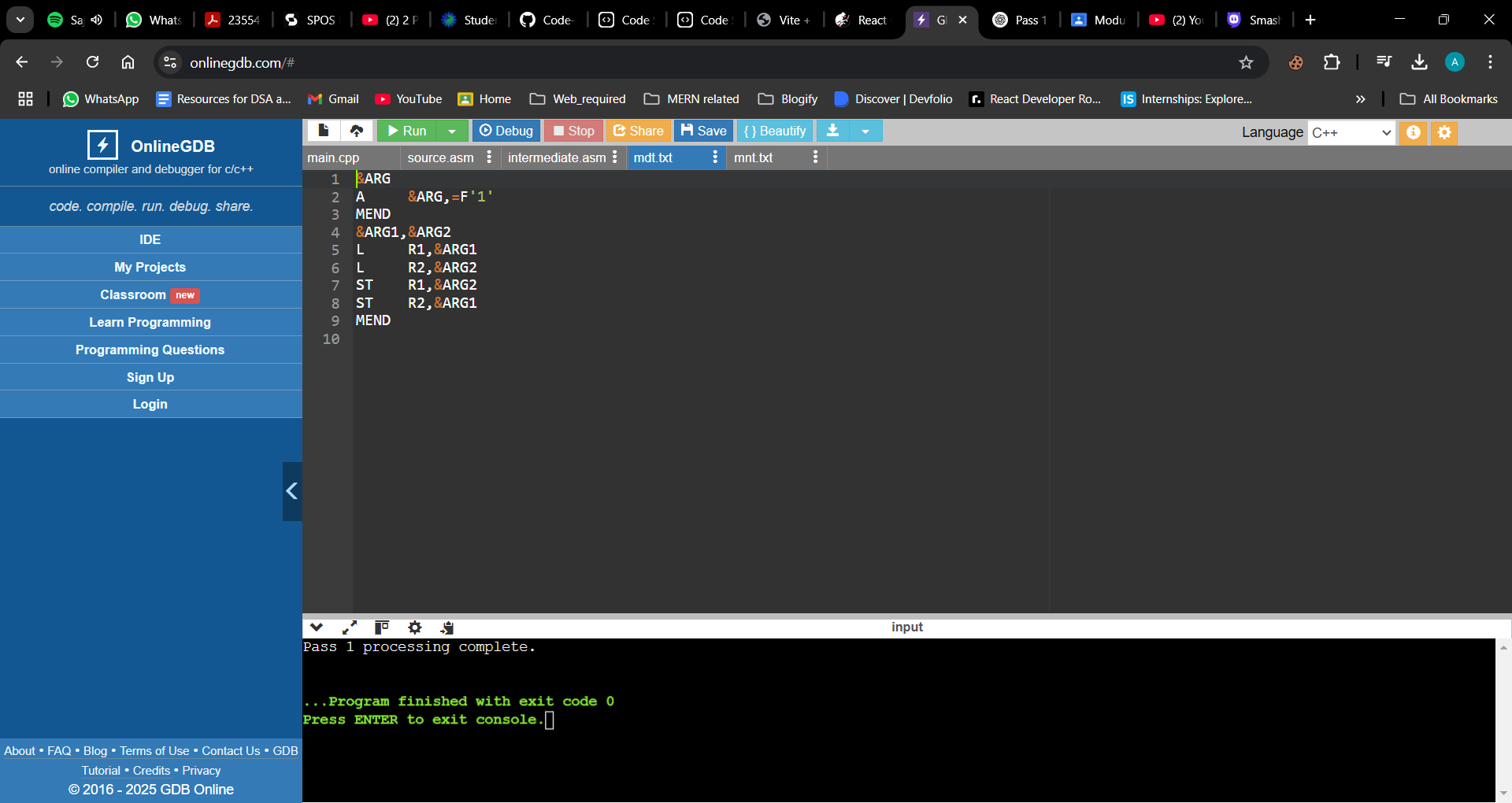
* **Output File -** intermediate.asm: The intermediate file generated after Pass 1, which will be used in Pass 2.



* **Macro Name Table File -** mnt.txt: Stores macro names and the number of parameters.



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



**Conclusion:**

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

The implemented Pass 1 of the two-pass macro processor effectively processes macro definitions in the source assembly program. It identifies macro definitions and stores them in the Macro Name Table (MNT) and Macro Definition Table (MDT). For instance, given a source program with a macro definition for COMPUTE and its subsequent invocations, the processor extracts the macro details and prepares the intermediate file for Pass 2.

2) **Generated Databases During Macro Definition Processing:**

* **Macro Name Table (MNT):** This table records each macro's name and a pointer to its definition in the MDT, facilitating quick lookup during macro expansion.
* **Macro Definition Table (MDT):** This table stores the actual macro definitions, including the sequence of instructions that constitute each macro's body.
* **Intermediate Source File:** This file contains the original source code with macro definitions removed but retains macro invocation statements. This setup allows Pass 2 to expand macros appropriately.