1.Implement Pass-I of two-pass assembler for given input assembly source program file. Use suitable Data structures MOT(Mnemonic Opcode Table),POT(Pseudo Opcode Table),DL(Declarative statements),REG(if required),Condition Code(if required) The Output of Pass1 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code.

```
START 101
READ N
READ P
MOVER BREG, ONE
MOVEM BREG, RESULT
PRINT RESULT
STOP
N DS 1
P DS 1
RESULT DS 1
ONE DC '1'
END
```

Ans

```
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <sstream>
#include <iomanip>
using namespace std;
```

```
struct Symbol {

    string name;

    int address;

    int size;

};


int main() {

    // Mnemonic Opcode Table (MOT)

    map<string, pair<string, string>> MOT = {

        {"STOP", {"IS", "00"}},

        {"ADD", {"IS", "01"}},

        {"SUB", {"IS", "02"}},

        {"MULT", {"IS", "03"}},

        {"MOVER", {"IS", "04"}},

        {"MOVEM", {"IS", "05"}},

        {"COMP", {"IS", "06"}},

        {"BC", {"IS", "07"}},

        {"DIV", {"IS", "08"}},

        {"READ", {"IS", "09"}},

        {"PRINT", {"IS", "10"}}

    };


    // Pseudo Opcode Table (POT)

    map<string, string> POT = {

        {"START", "AD"},

        {"END", "AD"},

        {"ORIGIN", "AD"},

        {"EQU", "AD"}
```

```
};

// Declarative statements (DL)
map<string, string> DL = {
    {"DC", "01"},
    {"DS", "02"}
};

// Register Table
map<string, string> REG = {
    {"AREG", "01"},
    {"BREG", "02"},
    {"CREG", "03"},
    {"DREG", "04"}
};

vector<Symbol> SYMTAB;
vector<string> INTERMEDIATE;
int LC = 0;
string line;

ifstream fin("input.asm"); // input file containing assembly program
if (!fin) {
    cout << "Error opening input file!\n";
    return 0;
}

while (getline(fin, line)) {
        if (line.empty()) continue;

        stringstream ss(line);

        string label, opcode, operand1, operand2;

        ss >> label;


        // Check if first word is opcode or label

        if (MOT.find(label) != MOT.end() || POT.find(label) != POT.end() || DL.find(label) != DL.end()) {

            opcode = label;

        } else {

            // It’s a label

            Symbol sym = {label, LC, 1};

            SYMTAB.push_back(sym);

            ss >> opcode;

        }


        // Process opcode

        if (opcode == "START") {

            ss >> operand1;

            LC = stoi(operand1);

            INTERMEDIATE.push_back("(AD,01)\t(C," + operand1 + ")");

        }

        else if (MOT.find(opcode) != MOT.end()) {

            string code = "(" + MOT[opcode].first + "," + MOT[opcode].second + ")";

            string ic = to_string(LC) + "\t" + code + "\t";

            ss >> operand1;


            if (REG.find(operand1) != REG.end()) {
```

```
            ss >> operand2;
            ic += "(" + REG[operand1] + ")\t";
            ic += operand2;
        } else {
            ic += operand1;
        }
        INTERMEDIATE.push_back(ic);
        LC++;
    }
    else if (DL.find(opcode) != DL.end()) {
        string code = "(" + string("DL,") + DL[opcode] + ")";
        ss >> operand1;
        INTERMEDIATE.push_back(to_string(LC) + "\t" + code + "\t(C," + operand1 + ")");

        // Add symbol entry (for DS/DC)
        SYMTAB.back().address = LC;
        if (opcode == "DS") SYMTAB.back().size = stoi(operand1);
        LC++;
    }
    else if (opcode == "END") {
        INTERMEDIATE.push_back("(AD,02)");
        break;
    }
}
fin.close();

// Print Symbol Table
cout << "\nSYMBOL TABLE:\n";
```

```
    cout << "--------------------------------\n";
    cout << setw(10) << "Symbol" << setw(10) << "Address" << setw(10) << "Size\n";
    cout << "--------------------------------\n";
    for (auto &s : SYMTAB)
        cout << setw(10) << s.name << setw(10) << s.address << setw(10) << s.size << "\n";

    // Print Intermediate Code
    cout << "\nINTERMEDIATE CODE:\n";
    cout << "--------------------------------\n";
    for (auto &i : INTERMEDIATE)
        cout << i << endl;

    return 0;
}
```

2.Implement Pass-I of two-pass assembler for given input assembly source program file. Use suitable Data structures MOT(Mnemonic Opcode Table),POT(Pseudo Opcode Table),DL(Declarative statements),REG(if required),Condition Code(if required) The Output of Pass1 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code.

```
START 1000
READ P
READ Q
MOVEM CREG, TERM
MOVEM BREG, RESULT
PRINT RESULT
STOP
P DS 1
```

```
Q DS 1
RESULT DS 1
TERM DS 1
END
```

Ans

```
#include <iostream>
#include <fstream>
#include <map>
#include <vector>
#include <sstream>
#include <iomanip>
using namespace std;

// Structure for Symbol Table entry
struct Symbol {
   string name;
   int address;
   int size;
};

// Structure for Literal Table entry
struct Literal {
   string literal;
   int address;
};

int main() {
```

```
// Mnemonic Opcode Table (MOT)
map<string, pair<string, string>> MOT = {
    {"STOP", {"IS", "00"}},
    {"ADD", {"IS", "01"}},
    {"SUB", {"IS", "02"}},
    {"MULT", {"IS", "03"}},
    {"MOVER", {"IS", "04"}},
    {"MOVEM", {"IS", "05"}},
    {"COMP", {"IS", "06"}},
    {"BC", {"IS", "07"}},
    {"DIV", {"IS", "08"}},
    {"READ", {"IS", "09"}},
    {"PRINT", {"IS", "10"}}
};

// Pseudo Opcode Table (POT)
map<string, string> POT = {
    {"START", "AD"},
    {"END", "AD"},
    {"ORIGIN", "AD"},
    {"EQU", "AD"}
};

// Declarative statements (DL)
map<string, string> DL = {
    {"DC", "01"},
    {"DS", "02"}
};
```

```
// Register Table
map<string, string> REG = {
    {"AREG", "01"},
    {"BREG", "02"},
    {"CREG", "03"},
    {"DREG", "04"}
};

vector<Symbol> SYMTAB;
vector<Literal> LITTAB;
vector<string> INTERMEDIATE;

int LC = 0;
string line;

ifstream fin("input.asm"); // Assembly file input
if (!fin) {
    cout << "Error opening input file!\n";
    return 0;
}

while (getline(fin, line)) {
    if (line.empty()) continue;
    stringstream ss(line);
    string label, opcode, operand1, operand2;
    ss >> label;
    // If label is opcode, shift accordingly
    if (MOT.find(label) != MOT.end() || POT.find(label) != POT.end() || DL.find(label) != DL.end()) {
        opcode = label;
    } else {
        // label is a symbol
        Symbol sym = {label, LC, 1};
        SYMTAB.push_back(sym);
        ss >> opcode;
    }

    // Process Opcodes
    if (opcode == "START") {
        ss >> operand1;
        LC = stoi(operand1);
        INTERMEDIATE.push_back("(AD,01)\t(C," + operand1 + ")");
    }
    else if (MOT.find(opcode) != MOT.end()) {
        string code = "(" + MOT[opcode].first + "," + MOT[opcode].second + ")";
        string ic = to_string(LC) + "\t" + code + "\t";

        ss >> operand1;
        if (REG.find(operand1) != REG.end()) {
            ss >> operand2;
            ic += "(" + REG[operand1] + ")\t" + operand2;
        } else {
            ic += operand1;
        }
        INTERMEDIATE.push_back(ic);
        LC++;
    }
    else if (DL.find(opcode) != DL.end()) {
        string code = "(DL," + DL[opcode] + ")";
        ss >> operand1;

        INTERMEDIATE.push_back(to_string(LC) + "\t" + code + "\t(C," + operand1 + ")");
        SYMTAB.back().address = LC;
        if (opcode == "DS") SYMTAB.back().size = stoi(operand1);
        LC++;
    }
    else if (opcode == "END") {
        INTERMEDIATE.push_back("(AD,02)");
        break;
    }
}
fin.close();

// --- Output Section ---
cout << "\n================ SYMBOL TABLE ================\n";
cout << setw(10) << "Symbol" << setw(10) << "Address" << setw(10) << "Size\n";
cout << "---------------------------------------------\n";
for (auto &s : SYMTAB)
    cout << setw(10) << s.name << setw(10) << s.address << setw(10) << s.size << "\n";

cout << "\n================ LITERAL TABLE ================\n";
if (LITTAB.empty()) cout << "(No Literals Found)\n";
```

```
    else {
        cout << setw(10) << "Literal" << setw(10) << "Address\n";
        for (auto &l : LITTAB)
            cout << setw(10) << l.literal << setw(10) << l.address << "\n";
    }

    cout << "\n================ INTERMEDIATE CODE ================\n";
    for (auto &i : INTERMEDIATE)
        cout << i << endl;

    return 0;
}
```

3. Implement Pass-I of two-pass assembler for given input assembly source program file. Use suitable Data structures MOT(Mnemonic Opcode Table),POT(Pseudo Opcode Table),DL(Declarative statements),REG(if required),Condition Code(if required) The Output of Pass1 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code

START 100

READ N

READ P

MOVER BREG, ONE

MOVEM BREG, TERM

ADD CREG, ONE

MOVEM CREG, TERM

MOVEM BREG, RESULT

PRINT RESULT

STOP

N DS 1

```
P DS 1

RESULT DS 1

ONE DC '1'

TERM DS 1

END
```

Ans

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <vector>
#include <iomanip>
using namespace std;

// Symbol structure
struct Symbol {
    string name;
    int address;
    int size;
};

// Literal structure
struct Literal {
    string literal;
    int address;
};
```

```
int main() {
    // Mnemonic Opcode Table (MOT)
    map<string, pair<string, string>> MOT = {
        {"STOP", {"IS", "00"}},
        {"ADD", {"IS", "01"}},
        {"SUB", {"IS", "02"}},
        {"MULT", {"IS", "03"}},
        {"MOVER", {"IS", "04"}},
        {"MOVEM", {"IS", "05"}},
        {"COMP", {"IS", "06"}},
        {"BC", {"IS", "07"}},
        {"DIV", {"IS", "08"}},
        {"READ", {"IS", "09"}},
        {"PRINT", {"IS", "10"}}
    };

    // Pseudo Opcode Table (POT)
    map<string, string> POT = {
        {"START", "AD"},
        {"END", "AD"},
        {"ORIGIN", "AD"},
        {"EQU", "AD"}
    };

    // Declarative Statements
    map<string, string> DL = {
        {"DC", "01"},
        {"DS", "02"}
```

```
};

// Register Table
map<string, string> REG = {
    {"AREG", "01"},
    {"BREG", "02"},
    {"CREG", "03"},
    {"DREG", "04"}
};

vector<Symbol> SYMTAB;
vector<Literal> LITTAB;
vector<string> INTERMEDIATE;

int LC = 0;
string line;

ifstream fin("input.asm");
if (!fin) {
    cout << "Error opening input file!" << endl;
    return 0;
}

while (getline(fin, line)) {
    if (line.empty()) continue;

    stringstream ss(line);
    string label, opcode, op1, op2;
    ss >> label;

    // Check if label is an opcode or a symbol
    if (MOT.find(label) != MOT.end() || POT.find(label) != POT.end() || DL.find(label) !=
DL.end()) {
        opcode = label;
    } else {
        // It's a label/symbol
        Symbol sym = {label, LC, 1};
        SYMTAB.push_back(sym);
        ss >> opcode;
    }

    // --- START ---
    if (opcode == "START") {
        ss >> op1;
        LC = stoi(op1);
        INTERMEDIATE.push_back("(AD,01)\t(C," + op1 + ")");
    }

    // --- Imperative Statements (IS) ---
    else if (MOT.find(opcode) != MOT.end()) {
        string code = "(" + MOT[opcode].first + "," + MOT[opcode].second + ")";
        string ic = to_string(LC) + "\t" + code + "\t";

        ss >> op1;
        if (REG.find(op1) != REG.end()) {
            ss >> op2;
            ic += "(" + REG[op1] + ")\t" + op2;
        } else {
            ic += op1;
        }
        INTERMEDIATE.push_back(ic);
        LC++;
    }

    // --- Declarative Statements (DL) ---
    else if (DL.find(opcode) != DL.end()) {
        string code = "(DL," + DL[opcode] + ")";
        ss >> op1;
        INTERMEDIATE.push_back(to_string(LC) + "\t" + code + "\t(C," + op1 + ")");
        SYMTAB.back().address = LC;
        if (opcode == "DS") SYMTAB.back().size = stoi(op1);
        LC++;
    }

    // --- END ---
    else if (opcode == "END") {
        INTERMEDIATE.push_back("(AD,02)");
        break;
    }
}

fin.close();

// --- OUTPUT ---
```

```
    cout << "\n============= SYMBOL TABLE =============\n";
    cout << setw(10) << "Symbol" << setw(10) << "Address" << setw(10) << "Size\n";
    cout << "----------------------------------------\n";
    for (auto &s : SYMTAB)
        cout << setw(10) << s.name << setw(10) << s.address << setw(10) << s.size << "\n";

    cout << "\n============= LITERAL TABLE =============\n";
    if (LITTAB.empty())
        cout << "(No Literals Found)\n";
    else {
        cout << setw(10) << "Literal" << setw(10) << "Address\n";
        for (auto &l : LITTAB)
            cout << setw(10) << l.literal << setw(10) << l.address << "\n";
    }

    cout << "\n============= INTERMEDIATE CODE =============\n";
    for (auto &i : INTERMEDIATE)
        cout << i << endl;

    return 0;
}
```

4. Implement Pass-I of two-pass assembler for given input assembly source program file. Use suitable Data structures MOT(Mnemonic Opcode Table),POT(Pseudo Opcode Table),DL(Declarative statements),REG(if required),Condition Code(if required) The Output of Pass1 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code.

START 100

READ N

READ P

```
MOVER BREG, ONE
MOVEM BREG, TERM
ADD CREG, ONE
MOVEM CREG, TERM
MOVEM BREG, RESULT
PRINT RESULT
STOP
N DS 1
P DS 1
RESULT DS 1
ONE DC '1'
TERM DS 1
END
```

Ans

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <vector>
#include <iomanip>
using namespace std;

// ---------- Structure Definitions ----------
struct Symbol {
    string name;
    int address;
    int size;
```

```
};

struct Literal {
    string literal;
    int address;
};

// ---------- Main Program ----------
int main() {
    // Mnemonic Opcode Table (MOT)
    map<string, pair<string, string>> MOT = {
        {"STOP", {"IS", "00"}},
        {"ADD", {"IS", "01"}},
        {"SUB", {"IS", "02"}},
        {"MULT", {"IS", "03"}},
        {"MOVER", {"IS", "04"}},
        {"MOVEM", {"IS", "05"}},
        {"COMP", {"IS", "06"}},
        {"BC", {"IS", "07"}},
        {"DIV", {"IS", "08"}},
        {"READ", {"IS", "09"}},
        {"PRINT", {"IS", "10"}}
    };

    // Pseudo Opcode Table (POT)
    map<string, string> POT = {
        {"START", "AD"},
        {"END", "AD"},
        {"ORIGIN", "AD"},
        {"EQU", "AD"}
    };

    // Declarative statements (DL)
    map<string, string> DL = {
        {"DC", "01"},
        {"DS", "02"}
    };

    // Register Table
    map<string, string> REG = {
        {"AREG", "01"},
        {"BREG", "02"},
        {"CREG", "03"},
        {"DREG", "04"}
    };

    vector<Symbol> SYMTAB;
    vector<Literal> LITTAB;
    vector<string> INTERMEDIATE;

    int LC = 0;
    string line;

    ifstream fin("input.asm"); // assembly program input file
    if (!fin) {
        cout << "Error opening input file!" << endl;
```

```
        return 0;
    }


    // ---------- PASS-I Processing ----------
    while (getline(fin, line)) {
        if (line.empty()) continue;


        stringstream ss(line);
        string label, opcode, op1, op2;
        ss >> label;


        // Check if the first word is opcode or a label
        if (MOT.find(label) != MOT.end() || POT.find(label) != POT.end() || DL.find(label) !=
DL.end()) {
            opcode = label;
        } else {
            // It's a label, store it in Symbol Table
            Symbol sym = {label, LC, 1};
            SYMTAB.push_back(sym);
            ss >> opcode;
        }


        // START
        if (opcode == "START") {
            ss >> op1;
            LC = stoi(op1);
            INTERMEDIATE.push_back("(AD,01)\t(C," + op1 + ")");
        }
```

```
// Imperative Statements (IS)
else if (MOT.find(opcode) != MOT.end()) {
    string code = "(" + MOT[opcode].first + "," + MOT[opcode].second + ")";
    string ic = to_string(LC) + "\t" + code + "\t";
    ss >> op1;

    if (REG.find(op1) != REG.end()) {
        ss >> op2;
        ic += "(" + REG[op1] + ")\t" + op2;
    } else {
        ic += op1;
    }

    INTERMEDIATE.push_back(ic);
    LC++;
}

// Declarative Statements (DL)
else if (DL.find(opcode) != DL.end()) {
    string code = "(DL," + DL[opcode] + ")";
    ss >> op1;
    INTERMEDIATE.push_back(to_string(LC) + "\t" + code + "\t(C," + op1 + ")");
    SYMTAB.back().address = LC;
    if (opcode == "DS") SYMTAB.back().size = stoi(op1);
    LC++;
}
```

```
    // END
    else if (opcode == "END") {
        INTERMEDIATE.push_back("(AD,02)");
        break;
    }
}

fin.close();

// ---------- OUTPUTS ----------
cout << "\n============= SYMBOL TABLE =============\n";
cout << setw(10) << "Symbol" << setw(10) << "Address" << setw(10) << "Size\n";
cout << "----------------------------------------\n";
for (auto &s : SYMTAB)
    cout << setw(10) << s.name << setw(10) << s.address << setw(10) << s.size << "\n";

cout << "\n============= LITERAL TABLE =============\n";
if (LITTAB.empty())
    cout << "(No Literals Found)\n";
else {
    cout << setw(10) << "Literal" << setw(10) << "Address\n";
    for (auto &l : LITTAB)
        cout << setw(10) << l.literal << setw(10) << l.address << "\n";
}

cout << "\n============= INTERMEDIATE CODE =============\n";
for (auto &i : INTERMEDIATE)
    cout << i << endl;
    return 0;

}
```

5. Implement Pass-I of two-pass assembler for given input assembly source program file. Use suitable Data structures MOT(Mnemonic Opcode Table),POT(Pseudo Opcode Table),DL(Declarative statements),REG(if required),Condition Code(if required) The Output of Pass1 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code.

```
START 1000

READ N

READ P

PRINT C

MOVER BREG, ONE

MOVEM BREG, TERM

STOP

N DS 1

P DS 1

C DS 1

ONE DC '1'

TERM DS 1

END
```

Ans

```
#include <iostream>

#include <fstream>

#include <sstream>

#include <map>

#include <vector>

#include <iomanip>
```

```
using namespace std;


struct Symbol {
    string symbol;
    int address;
};


struct Literal {
    string literal;
    int address;
};


int main() {
    // ----- MOT -----
    map<string, string> MOT = {
        {"STOP", "00"},
        {"ADD", "01"},
        {"SUB", "02"},
        {"MULT", "03"},
        {"MOVER", "04"},
        {"MOVEM", "05"},
        {"COMP", "06"},
        {"BC", "07"},
        {"DIV", "08"},
        {"READ", "09"},
        {"PRINT", "10"}
    };
```

```
// ----- POT -----
map<string, string> POT = {
    {"START", "01"},
    {"END", "02"},
    {"LTORG", "03"},
    {"ORIGIN", "04"},
    {"EQU", "05"}
};

// ----- Declarative Statements -----
map<string, string> DL = {
    {"DC", "01"},
    {"DS", "02"}
};

// ----- Registers -----
map<string, string> REG = {
    {"AREG", "01"},
    {"BREG", "02"},
    {"CREG", "03"},
    {"DREG", "04"}
};

// Tables
vector<Symbol> symtab;
vector<Literal> littab;
vector<string> interCode;
```

```
string line, label, opcode, operand1, operand2;
int LC = 0;

// Input Assembly Program
vector<string> program = {
    "START 1000",
    "READ N",
    "READ P",
    "PRINT C",
    "MOVER BREG, ONE",
    "MOVEM BREG, TERM",
    "STOP",
    "N DS 1",
    "P DS 1",
    "C DS 1",
    "ONE DC '1'",
    "TERM DS 1",
    "END"
};

cout << "\n----- PASS 1 OUTPUT -----\n";

for (string line : program) {
    label = opcode = operand1 = operand2 = "";

    stringstream ss(line);
    ss >> opcode;
```

```
if (opcode == "START") {
    ss >> LC;
    interCode.push_back("(AD,01) (C," + to_string(LC) + ")");
    continue;
}

if (opcode == "END") {
    interCode.push_back("(AD,02)");
    break;
}

if (MOT.find(opcode) != MOT.end()) {
    string code = "(IS," + MOT[opcode] + ")";
    if (opcode == "STOP") {
        interCode.push_back(code);
        LC++;
        continue;
    }

    ss >> operand1;

    if (operand1.find(",") != string::npos) {
        operand2 = operand1.substr(operand1.find(",") + 1);
        operand1 = operand1.substr(0, operand1.find(","));
    } else {
        ss >> operand2;
    }
```

```
string ic = code;

if (REG.find(operand1) != REG.end())
    ic += " (" + REG[operand1] + ")";
else if (!operand1.empty())
    ic += " (S," + operand1 + ")";

if (!operand2.empty()) {
    ic += " (S," + operand2 + ")";
}

interCode.push_back(ic);

// Add operands to symbol table if not already there
if (!operand1.empty() && REG.find(operand1) == REG.end()) {
    bool found = false;
    for (auto &s : symtab)
        if (s.symbol == operand1) found = true;
    if (!found)
        symtab.push_back({operand1, -1});
}

if (!operand2.empty()) {
    bool found = false;
    for (auto &s : symtab)
        if (s.symbol == operand2) found = true;
    if (!found)
        symtab.push_back({operand2, -1});
    }

    LC++;
}
else if (DL.find(opcode) != DL.end()) {
    string name = "";
    int size = 1;
    ss >> name >> size;
}
else if (POT.find(opcode) != POT.end()) {
    interCode.push_back("(AD," + POT[opcode] + ")");
}
else {
    // It's a symbol definition line
    label = opcode;
    ss >> opcode;

    if (opcode == "DS" || opcode == "DC") {
        string value;
        ss >> value;

        bool found = false;
        for (auto &s : symtab)
            if (s.symbol == label) {
                s.address = LC;
                found = true;
            }
        if (!found)
```

```
                symtab.push_back({label, LC});

            if (opcode == "DS")
                interCode.push_back("(DL,02) (C," + value + ")");
            else if (opcode == "DC")
                interCode.push_back("(DL,01) (C," + value.substr(1, value.size() - 2) + ")");
            LC++;
        }
    }
}

// Display Intermediate Code
cout << "\n--- INTERMEDIATE CODE ---\n";
for (auto &x : interCode)
    cout << x << endl;

// Assign addresses to undefined symbols
int address = 1000;
for (auto &s : symtab) {
    if (s.address == -1)
        s.address = LC++;
}

// Display Symbol Table
cout << "\n--- SYMBOL TABLE ---\n";
cout << left << setw(10) << "Symbol" << setw(10) << "Address" << endl;
for (auto &s : symtab)
    cout << left << setw(10) << s.symbol << setw(10) << s.address << endl;
    // Display Literal Table (empty in this case)
    cout << "\n--- LITERAL TABLE ---\n";
    cout << "No literals in this program.\n";

    return 0;
}
```

6. Implement Pass-II of two-pass assembler for given Intermediate File. Use Symbol Table and Intermediate Code generated in Pass I. The Output of Pass 2 should contain Symbol Table (ST),Literal Table(LT) and Intermediate Code.

```
(AD,1) (C,101)
101    (IS,9) (00) (S,00)
102    (IS,9) (00) (S,01)
103    (IS,10) (00) (S,02)
104    (IS,4) (02) (S,03)
105    (IS,5) (02) (S,04)
106    (IS,7) (CC,02) (S,05)
107    (IS,5) (02) (S,06)
108    (IS,10) (00) (S,06)
109    (IS,0)
110    (DL,2) (C,1)
111    (DL,2) (C,1)
112    (DL,2) (C,1)
113    (AD,2)
```

Ans

```
#include <iostream>
#include <iomanip>
```

```
#include <sstream>
#include <vector>
#include <string>
#include <map>
using namespace std;

// Symbol Table Entry
struct Symbol {
    string name;
    int address;
};

// Literal Table Entry
struct Literal {
    string literal;
    int address;
};

int main() {
    // Example Symbol Table (from Pass-I)
    map<int, string> symIndex = {
        {0, "N"}, {1, "P"}, {2, "C"}, {3, "ONE"}, {4, "TERM"}, {5, "RES1"}, {6, "RES2"}
    };
    map<string, int> symAddr = {
        {"N", 200}, {"P", 201}, {"C", 202}, {"ONE", 203},
        {"TERM", 204}, {"RES1", 205}, {"RES2", 206}
    };
    // Intermediate Code Lines
    vector<string> IC = {
        "(AD,1) (C,101)",
        "101 (IS,9) (00) (S,00)",
        "102 (IS,9) (00) (S,01)",
        "103 (IS,10) (00) (S,02)",
        "104 (IS,4) (02) (S,03)",
        "105 (IS,5) (02) (S,04)",
        "106 (IS,7) (CC,02) (S,05)",
        "107 (IS,5) (02) (S,06)",
        "108 (IS,10) (00) (S,06)",
        "109 (IS,0)",
        "110 (DL,2) (C,1)",
        "111 (DL,2) (C,1)",
        "112 (DL,2) (C,1)",
        "113 (AD,2)"
    };

    cout << "\n---------- PASS-II OUTPUT ----------\n";

    cout << left << setw(10) << "LC" << setw(20) << "Intermediate Code" << setw(20) << "Machine Code" << endl;
    cout << "------------------------------------------------------------\n";

    for (string line : IC) {
        string lc, token, opcode, reg, sym, type;
        stringstream ss(line);
        ss >> lc >> token;
        if (token.find("(AD") != string::npos) {
            continue; // AD (Assembler Directives) don't generate code
        }

        if (token.find("(IS") != string::npos) {
            string op = token.substr(4, token.find(")") - 4);
            string machineCode = op;

            ss >> token;
            if (token.find("(00") != string::npos || token.find("(02") != string::npos)
                machineCode += " " + token.substr(1, 2);
            else
                machineCode += " 00";

            ss >> token;
            if (token.find("(S") != string::npos) {
                int index = stoi(token.substr(3, 2));
                machineCode += " " + to_string(symAddr[symIndex[index]]);
            }
            else if (token.find("(C") != string::npos) {
                string val = token.substr(3, token.find(")") - 3);
                machineCode += " " + val;
            }

            cout << left << setw(10) << lc << setw(20) << line << setw(20) << machineCode <<
endl;
        }
        else if (token.find("(DL") != string::npos) {
            string val;
            ss >> token;
            val = token.substr(3, token.find(")") - 3);
            cout << left << setw(10) << lc << setw(20) << line << setw(20) << val << endl;
        }
    }

    // Display Symbol Table
    cout << "\n--- SYMBOL TABLE ---\n";
    cout << left << setw(10) << "Index" << setw(15) << "Symbol" << setw(10) << "Address" << endl;
    for (auto &s : symIndex)
        cout << left << setw(10) << s.first << setw(15) << s.second << setw(10) << symAddr[s.second] << endl;

    // Literal Table (if any)
    cout << "\n--- LITERAL TABLE ---\n";
    cout << "No literals in this program.\n";

    return 0;
}
```

7. Design suitable data structures and implement Pass- I of a two-pass macroprocessor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

MACRO Mymacro

ADD AREG,A

ADD AREG,B

```
MEND
START 100
READ A
READ B
Mymacro
PRINT A
PRINT B
END
```

Ans

```
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include <map>
using namespace std;

int main() {
    vector<string> program = {
        "MACRO Mymacro",
        "ADD AREG,A",
        "ADD AREG,B",
        "MEND",
        "START 100",
        "READ A",
        "READ B",
        "Mymacro",
        "PRINT A",
        "PRINT B",
        "END"
    };

    map<string, int> MNT; // Macro Name Table: MacroName -> MDT Index
    vector<string> MDT;   // Macro Definition Table
    vector<string> IC;    // Intermediate Code

    bool inMacro = false;
    string macroName;

    cout << "\n----- PASS-I MACROPROCESSOR OUTPUT -----\n";

    for (int i = 0; i < program.size(); i++) {
        string line = program[i];
        stringstream ss(line);
        string word1, word2;
        ss >> word1 >> word2;

        if (word1 == "MACRO") {
            inMacro = true;
            macroName = word2;
            MNT[macroName] = MDT.size(); // store MDT start index
            continue;
        }

        if (inMacro) {
            if (word1 == "MEND") {
                MDT.push_back("MEND");
                inMacro = false;
            } else {
                MDT.push_back(line);
            }
        } else {
            IC.push_back(line);
        }
    }

    // ---------------- OUTPUT ----------------
    cout << "\n--- MACRO NAME TABLE (MNT) ---\n";
    cout << "MacroName\tMDT_Index\n";
    for (auto &m : MNT)
        cout << m.first << "\t\t" << m.second << endl;

    cout << "\n--- MACRO DEFINITION TABLE (MDT) ---\n";
    for (int i = 0; i < MDT.size(); i++)
        cout << i << "\t" << MDT[i] << endl;

    cout << "\n--- INTERMEDIATE CODE (Without Macros) ---\n";
    for (auto &line : IC)
        cout << line << endl;

    return 0;
}
```

8. Design suitable data structures and implement Pass- I and Pass-II of a two-pass macroprocessor. The output of Pass-I (MNT, MDT and intermediate code file without any macro definitions) should be input for Pass-II.

MACRO Mymacro

ADD AREG,X

ADD AREG,Y

MEND

START 100

READ X

READ Y

Mymacro

PRINT X

PRINT Y

END

Ans

Pass I code:

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <sstream>
using namespace std;

// ---------- PASS 1 FUNCTION ----------
void pass1_macroprocessor(vector<string> &input_lines,
                    map<string, int> &MNT,
                    map<int, string> &MDT,
                    vector<string> &IC) {
int mdtp = 1;
bool in_macro = false;
string macro_name = "";

for (string line : input_lines) {
    // Trim line
    if (line.empty()) continue;

    stringstream ss(line);
    vector<string> words;
    string word;
    while (ss >> word)
        words.push_back(word);

    if (words.empty())
        continue;

    if (words[0] == "MACRO") {
        in_macro = true;
        continue;
    }

    else if (in_macro) {
        if (words[0] == "MEND") {
            MDT[mdtp] = "MEND";
            in_macro = false;
            macro_name = "";
```

```
                mdtp++;
                continue;
            } else {
                if (macro_name == "") {
                    macro_name = words[0];
                    MNT[macro_name] = mdtp;
                    continue;
                } else {
                    MDT[mdtp] = line;
                    mdtp++;
                }
            }
        } else {
            IC.push_back(line);
        }
    }
}

// ---------- MAIN ----------
int main() {
    vector<string> input_code = {
        "MACRO Mymacro",
        "ADD AREG,X",
        "ADD AREG,Y",
        "MEND",
        "START 100",
        "READ X",
        "READ Y",
        "Mymacro",
        "PRINT X",
        "PRINT Y",
        "END"};

    map<string, int> MNT;  // Macro Name Table
    map<int, string> MDT;  // Macro Definition Table
    vector<string> IC;     // Intermediate Code

    // Run Pass 1
    pass1_macroprocessor(input_code, MNT, MDT, IC);

    // ---------- OUTPUT ----------
    cout << "---- PASS 1 OUTPUT ----\n";

    cout << "\nMNT (Macro Name Table):\n";
    for (auto &entry : MNT)
        cout << entry.first << " -> MDT[" << entry.second << "]\n";

    cout << "\nMDT (Macro Definition Table):\n";
    for (auto &entry : MDT)
        cout << entry.first << ": " << entry.second << "\n";

    cout << "\nIntermediate Code (Without Macro Definitions):\n";
    for (auto &line : IC)
        cout << line << "\n";
    return 0;
}
```

pass 2 code:

```
#include <iostream>
#include <map>
#include <vector>
#include <string>
#include <sstream>
using namespace std;

// Function to simulate Pass-II of Macroprocessor
vector<string> pass2_macroprocessor(vector<string> &IC,
                                    map<string, int> &MNT,
                                    map<int, string> &MDT) {
    vector<string> expanded_code;

    for (auto &line : IC) {
        stringstream ss(line);
        string word;
        ss >> word;

        // Check if first word is a macro call
        if (MNT.find(word) != MNT.end()) {
            int ptr = MNT[word];
            while (MDT[ptr] != "MEND") {
                expanded_code.push_back(MDT[ptr]);
                ptr++;
            }
        } else {
            expanded_code.push_back(line);
        }
    }

    return expanded_code;
}

// Example usage
int main() {
    // Example MNT (Macro Name Table)
    map<string, int> MNT = {{"Mymacro", 1}};

    // Example MDT (Macro Definition Table)
    map<int, string> MDT = {
        {1, "ADD AREG,X"},
        {2, "ADD AREG,Y"},
        {3, "MEND"}};

    // Example Intermediate Code (IC)
    vector<string> IC = {
        "START 100",
        "READ X",
        "READ Y",
        "Mymacro",
        "PRINT X",
        "PRINT Y",
        "END"};
    // Call Pass-II
    vector<string> expanded_code = pass2_macroprocessor(IC, MNT, MDT);

    // Display Pass-II Output
    cout << "\n---- PASS 2 OUTPUT ----\n";
    cout << "Expanded Code:\n";
    for (auto &line : expanded_code)
        cout << line << endl;

    return 0;
}
```

pass I and pass ll in one program

```
#include <iostream>
#include <fstream>
#include <sstream>
#include <map>
#include <vector>
#include <string>
using namespace std;

// ---------- PASS 1 ----------
void pass1_macroprocessor(string input_file,
                          map<string, int> &MNT,
                          map<int, string> &MDT,
                          vector<string> &IC) {
    ifstream fin(input_file);
    if (!fin) {
```

```
    cout << " Error: Cannot open input file.\n";
    return;
}

string line;
bool inMacro = false;
string macroName = "";
int mdtp = 1;

while (getline(fin, line)) {
    if (line.empty()) continue;
    stringstream ss(line);
    vector<string> words;
    string word;
    while (ss >> word)
        words.push_back(word);

    if (words.empty()) continue;

    if (words[0] == "MACRO") {
        inMacro = true;
        continue;
    } else if (inMacro) {
        if (words[0] == "MEND") {
            MDT[mdtp++] = "MEND";
            inMacro = false;
            macroName = "";
            continue;
        } else {
            if (macroName == "") {
                macroName = words[0];
                MNT[macroName] = mdtp;
            } else {
                MDT[mdtp++] = line;
            }
        }
    } else {
        IC.push_back(line);
    }
}

fin.close();

// ----- Write MNT -----
ofstream mntf("MNT.txt");
mntf << "MNT (Macro Name Table)\n";
for (auto &x : MNT)
    mntf << x.first << "\tMDT[" << x.second << "]\n";
mntf.close();

// ----- Write MDT -----
ofstream mdtf("MDT.txt");
mdtf << "MDT (Macro Definition Table)\n";
for (auto &x : MDT)
    mdtf << x.first << "\t" << x.second << "\n";
mdtf.close();
```

```
    // ----- Write Intermediate Code -----
    ofstream icf("pass1_output.txt");
    icf << "Intermediate Code (Without Macro Definitions)\n";
    for (auto &x : IC)
        icf << x << "\n";
    icf.close();

    cout << " PASS 1 COMPLETED\n";
    cout << "Files generated: MNT.txt, MDT.txt, pass1_output.txt\n";
}

// ---------- PASS 2 ----------
void pass2_macroprocessor(map<string, int> &MNT,
                          map<int, string> &MDT,
                          vector<string> &IC) {
    vector<string> expanded_code;

    for (auto &line : IC) {
        stringstream ss(line);
        string word;
        ss >> word;

        if (MNT.find(word) != MNT.end()) {
            int ptr = MNT[word];
            while (MDT[ptr] != "MEND") {
                expanded_code.push_back(MDT[ptr]);
                ptr++;
            }
        } else {
            expanded_code.push_back(line);
        }
    }

    ofstream fout("pass2_output.txt");
    fout << "Final Expanded Code (Pass 2 Output)\n";
    for (auto &x : expanded_code)
        fout << x << "\n";
    fout.close();

    cout << " PASS 2 COMPLETED\n";
    cout << "File generated: pass2_output.txt\n";
}

// ---------- MAIN ----------
int main() {
    string input_file = "input.txt";
    map<string, int> MNT;
    map<int, string> MDT;
    vector<string> IC;

    pass1_macroprocessor(input_file, MNT, MDT, IC);
    pass2_macroprocessor(MNT, MDT, IC);

    return 0;
}
```

9. Write a program to simulate CPU Scheduling Algorithms: FCFS, SJF (Non-Preemptive) PID Burst Time Arrival Time

PID	Burst Time	Arrival Time
P1	3	0
P2	5	2
P3	4	4

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;

struct Process {
    string pid;
    int at, bt;
    int ct, tat, wt;
    bool completed;
};

// ---------- FCFS ----------
void FCFS(vector<Process> procs) {
    cout << "\n===== FCFS (First Come First Serve) =====\n";
    sort(procs.begin(), procs.end(), [](Process a, Process b) {
        return a.at < b.at;
    });

    int time = 0;
    float total_tat = 0, total_wt = 0;

    for (auto &p : procs) {
        if (time < p.at)
            time = p.at;
        time += p.bt;
        p.ct = time;
        p.tat = p.ct - p.at;
        p.wt = p.tat - p.bt;

        total_tat += p.tat;
        total_wt += p.wt;
    }

    cout << "PID\tAT\tBT\tCT\tTAT\tWT\n";
    for (auto &p : procs) {
        cout << p.pid << "\t" << p.at << "\t" << p.bt << "\t"
             << p.ct << "\t" << p.tat << "\t" << p.wt << endl;
    }

    cout << fixed << setprecision(2);
    cout << "Average TAT = " << total_tat / procs.size() << endl;
    cout << "Average WT = " << total_wt / procs.size() << endl;
}
```

```
// ---------- SJF (Non-Preemptive) ----------
void SJF_NonPreemptive(vector<Process> procs) {
    cout << "\n===== SJF (Shortest Job First - Non Preemptive) =====\n";
    int n = procs.size();
    int completed = 0, time = 0;
    float total_tat = 0, total_wt = 0;

    for (auto &p : procs) p.completed = false;

    while (completed < n) {
        vector<int> ready;
        for (int i = 0; i < n; i++) {
            if (procs[i].at <= time && !procs[i].completed)
                ready.push_back(i);
        }

        if (ready.empty()) {
            time++;
            continue;
        }

        // Find process with minimum burst time
        int idx = ready[0];
        for (int i : ready) {
            if (procs[i].bt < procs[idx].bt)
                idx = i;
        }
        time += procs[idx].bt;
        procs[idx].ct = time;
        procs[idx].tat = procs[idx].ct - procs[idx].at;
        procs[idx].wt = procs[idx].tat - procs[idx].bt;
        procs[idx].completed = true;

        total_tat += procs[idx].tat;
        total_wt += procs[idx].wt;
        completed++;
    }

    cout << "PID\tAT\tBT\tCT\tTAT\tWT\n";
    for (auto &p : procs) {
        cout << p.pid << "\t" << p.at << "\t" << p.bt << "\t"
             << p.ct << "\t" << p.tat << "\t" << p.wt << endl;
    }

    cout << fixed << setprecision(2);
    cout << "Average TAT = " << total_tat / n << endl;
    cout << "Average WT = " << total_wt / n << endl;
}

// ---------- MAIN ----------
int main() {
    vector<Process> processes = {
        {"P1", 0, 3, 0, 0, 0, false},
        {"P2", 2, 5, 0, 0, 0, false},
        {"P3", 4, 4, 0, 0, 0, false}
    };


    FCFS(processes);
    SJF_NonPreemptive(processes);


    return 0;
}
```

10. Write a program to simulate CPU Scheduling Algorithms: FCFS, Priority (Non-Preemptive)

PID	Arrival Time	Burst Time	Priority
P1	0	2	1
P2	1	3	2
P3	2	1	3

Ans

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;


struct Process {
    string pid;
    int at, bt, pr;
    int ct, tat, wt;
    bool completed;
};
```

```
// ---------- FCFS ----------
void FCFS(vector<Process> procs) {
    cout << "\n===== FCFS (First Come First Serve) =====\n";
    sort(procs.begin(), procs.end(), [](Process a, Process b) {
        return a.at < b.at;
    });

    int time = 0;
    float total_tat = 0, total_wt = 0;

    for (auto &p : procs) {
        if (time < p.at)
            time = p.at;
        time += p.bt;
        p.ct = time;
        p.tat = p.ct - p.at;
        p.wt = p.tat - p.bt;

        total_tat += p.tat;
        total_wt += p.wt;
    }

    cout << "PID\tAT\tBT\tCT\tTAT\tWT\n";
    for (auto &p : procs)
        cout << p.pid << "\t" << p.at << "\t" << p.bt << "\t"
             << p.ct << "\t" << p.tat << "\t" << p.wt << endl;
    cout << fixed << setprecision(2);
    cout << "Average TAT = " << total_tat / procs.size() << endl;
    cout << "Average WT = " << total_wt / procs.size() << endl;
}

// ---------- Priority (Non-Preemptive) ----------
void Priority_NonPreemptive(vector<Process> procs) {
    cout << "\n===== PRIORITY SCHEDULING (Non Preemptive) =====\n";
    int n = procs.size();
    int completed = 0, time = 0;
    float total_tat = 0, total_wt = 0;

    for (auto &p : procs) p.completed = false;

    while (completed < n) {
        vector<int> ready;
        for (int i = 0; i < n; i++) {
            if (procs[i].at <= time && !procs[i].completed)
                ready.push_back(i);
        }

        if (ready.empty()) {
            time++;
            continue;
        }

        // Lower priority number = higher priority
        int idx = ready[0];
        for (int i : ready)
            if (procs[i].pr < procs[idx].pr)
                idx = i;

        time += procs[idx].bt;
        procs[idx].ct = time;
        procs[idx].tat = procs[idx].ct - procs[idx].at;
        procs[idx].wt = procs[idx].tat - procs[idx].bt;
        procs[idx].completed = true;

        total_tat += procs[idx].tat;
        total_wt += procs[idx].wt;
        completed++;
    }

    cout << "PID\tAT\tBT\tPR\tCT\tTAT\tWT\n";
    for (auto &p : procs)
        cout << p.pid << "\t" << p.at << "\t" << p.bt << "\t" << p.pr << "\t"
             << p.ct << "\t" << p.tat << "\t" << p.wt << endl;

    cout << fixed << setprecision(2);
    cout << "Average TAT = " << total_tat / n << endl;
    cout << "Average WT = " << total_wt / n << endl;
}

// ---------- MAIN ----------
int main() {
    vector<Process> processes = {
        {"P1", 0, 2, 1, 0, 0, 0, false},
        {"P2", 1, 3, 2, 0, 0, 0, false},
        {"P3", 2, 1, 3, 0, 0, 0, false}
    };

    FCFS(processes);
    Priority_NonPreemptive(processes);

    return 0;
}
```

11. Write a program to simulate CPU Scheduling Algorithms: SJF (Preemptive)

PID	**Burst Time**	**Arrival Time**
P1	**6**	**0**
P2	**3**	**1**
P3	**7**	**2**

Ans

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

struct Process {
    int pid;
    int bt;
    int at;
    int rt;   // Remaining time
    int ct;   // Completion time
    int tat;  // Turnaround time
    int wt;   // Waiting time
    bool completed;
};

int main() {
    int n = 3;
    Process p[n] = {
        {1, 6, 0, 6, 0, 0, 0, false},
        {2, 3, 1, 3, 0, 0, 0, false},
        {3, 7, 2, 7, 0, 0, 0, false}
    };

    int completed = 0, currentTime = 0;
    float totalTAT = 0, totalWT = 0;

    cout << "\n--- SJF (Preemptive) Scheduling Simulation ---\n";

    while (completed != n) {
        int idx = -1;
        int minRT = 1e9;

        // Find process with shortest remaining time among arrived processes
        for (int i = 0; i < n; i++) {
            if (p[i].at <= currentTime && !p[i].completed && p[i].rt < minRT) {
                minRT = p[i].rt;
                idx = i;
            }
        }

        if (idx != -1) {
            p[idx].rt--;
            currentTime++;

            // If process completes
            if (p[idx].rt == 0) {
                p[idx].completed = true;
                p[idx].ct = currentTime;
                p[idx].tat = p[idx].ct - p[idx].at;
                p[idx].wt = p[idx].tat - p[idx].bt;
                totalTAT += p[idx].tat;
                totalWT += p[idx].wt;
                completed++;
            }
        } else {
            // If no process has arrived yet
            currentTime++;
        }
    }

    // Output table
    cout << "\nPID\tAT\tBT\tCT\tTAT\tWT\n";
    for (int i = 0; i < n; i++) {
        cout << "P" << p[i].pid << "\t"
             << p[i].at << "\t"
             << p[i].bt << "\t"
```

```
            << p[i].ct << "\t"
            << p[i].tat << "\t"
            << p[i].wt << endl;
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Turnaround Time = " << totalTAT / n;
    cout << "\nAverage Waiting Time = " << totalWT / n << endl;

    return 0;
}
```

12. Write a program to simulate CPU Scheduling Algorithms: Priority (Preemptive)

PID	**Arrival Time**	**Burst Time**	**Priority**
P1	**0**	**4**	**2**
P2	**1**	**3**	**3**
P3	**2**	**1**	**4**

Ans

```
#include <iostream>
#include <iomanip>
#include <vector>
#include <algorithm>
using namespace std;

struct Process {
    string pid;
    int arrival, burst, priority;
    int remainingTime, completion, waiting, turnaround;
};

int main() {
    int n = 3;
    vector<Process> p = {
        {"P1", 0, 4, 2},
        {"P2", 1, 3, 3},
        {"P3", 2, 1, 4}
    };

    int completed = 0, time = 0;
    float totalWT = 0, totalTAT = 0;

    // Initialize remaining burst time
    for (auto &pr : p) pr.remainingTime = pr.burst;

    cout << "\n--- Preemptive Priority Scheduling ---\n";

    while (completed != n) {
        int idx = -1, highest = -1;

        // Find process with highest priority that has arrived
        for (int i = 0; i < n; i++) {
            if (p[i].arrival <= time && p[i].remainingTime > 0) {
                if (p[i].priority > highest) {
                    highest = p[i].priority;
                    idx = i;
                }
            }
        }

        if (idx != -1) {
            p[idx].remainingTime--;
            time++;

            if (p[idx].remainingTime == 0) {
                p[idx].completion = time;
                p[idx].turnaround = p[idx].completion - p[idx].arrival;
                p[idx].waiting = p[idx].turnaround - p[idx].burst;
                totalWT += p[idx].waiting;
                totalTAT += p[idx].turnaround;
                completed++;
            }
        } else {
            time++; // No process ready, CPU idle
        }
    }

    cout << "\nPID\tAT\tBT\tPR\tCT\tTAT\tWT";
    for (auto &pr : p) {
        cout << "\n" << pr.pid << "\t" << pr.arrival << "\t" << pr.burst
             << "\t" << pr.priority << "\t" << pr.completion
             << "\t" << pr.turnaround << "\t" << pr.waiting;
    }
```

```
    cout << fixed << setprecision(2);
    cout << "\n\nAverage Turnaround Time: " << totalTAT / n;
    cout << "\nAverage Waiting Time: " << totalWT / n << endl;

    return 0;
}
```

13. Write a program to simulate CPU Scheduling Algorithms:Round Robin (Preemptive) Time Quantum:2ms

Process	**Burst Time**	**Arrival Time**
P1	**4**	**0**
P2	**5**	**0**
P3	**3**	**0**

ans

```
#include <iostream>
#include <queue>
#include <iomanip>
using namespace std;

struct Process {
    string pid;
    int arrival, burst, remaining, completion, waiting, turnaround;
};

int main() {
    int n = 3;
    int timeQuantum = 2;
    vector<Process> p = {
```

```
    {"P1", 0, 4, 4, 0, 0, 0},
    {"P2", 0, 5, 5, 0, 0, 0},
    {"P3", 0, 3, 3, 0, 0, 0}
};

queue<int> q;
int time = 0, completed = 0;
vector<bool> inQueue(n, false);

// Initially push all processes that arrive at time 0
for (int i = 0; i < n; i++) {
    if (p[i].arrival == 0) {
        q.push(i);
        inQueue[i] = true;
    }
}

cout << "\n--- Round Robin Scheduling (TQ = " << timeQuantum << " ms) ---\n";
cout << "\nGantt Chart:\n";

while (!q.empty()) {
    int idx = q.front();
    q.pop();

    cout << "| " << p[idx].pid << " ";

    int execTime = min(timeQuantum, p[idx].remaining);
    time += execTime;
    p[idx].remaining -= execTime;

    // Add newly arrived processes during this time to the queue
    for (int i = 0; i < n; i++) {
        if (p[i].arrival <= time && !inQueue[i] && p[i].remaining > 0) {
            q.push(i);
            inQueue[i] = true;
        }
    }

    // If current process not finished, push it again
    if (p[idx].remaining > 0) {
        q.push(idx);
    } else {
        p[idx].completion = time;
        completed++;
    }
}
cout << "|\n";

float totalWT = 0, totalTAT = 0;
for (int i = 0; i < n; i++) {
    p[i].turnaround = p[i].completion - p[i].arrival;
    p[i].waiting = p[i].turnaround - p[i].burst;
    totalWT += p[i].waiting;
    totalTAT += p[i].turnaround;
}
```

```
    cout << "\nPID\tAT\tBT\tCT\tTAT\tWT\n";
    for (auto &pr : p) {
        cout << pr.pid << "\t" << pr.arrival << "\t" << pr.burst << "\t"
             << pr.completion << "\t" << pr.turnaround << "\t" << pr.waiting << endl;
    }

    cout << fixed << setprecision(2);
    cout << "\nAverage Turnaround Time: " << totalTAT / n;
    cout << "\nAverage Waiting Time: " << totalWT / n << endl;

    return 0;
}
```

14. Write a program to simulate Page replacement algorithm : FIFO Reference String : 1,2,3,4,2,5,3,4

Page Frame Size:3

Ans

```
#include <iostream>
#include <vector>
#include <queue>
#include <iomanip>
using namespace std;

int main() {
    vector<int> reference = {1, 2, 3, 4, 2, 5, 3, 4};
    int frameSize = 3;

    queue<int> q;          // To maintain FIFO order
```

```
vector<int> frames;    // Current pages in frames
int pageFaults = 0;

cout << "--- FIFO Page Replacement Algorithm ---\n";
cout << "\nReference String: ";
for (int r : reference) cout << r << " ";
cout << "\nPage Frame Size: " << frameSize << "\n\n";

cout << left << setw(10) << "Ref" << setw(25) << "Frames" << "Status\n";
cout << "------------------------------------------\n";

for (int page : reference) {
    bool hit = false;

    // Check if page is already in frame
    for (int f : frames) {
        if (f == page) {
            hit = true;
            break;
        }
    }

    // If not in frame, replace using FIFO
    if (!hit) {
        pageFaults++;

        if (frames.size() < frameSize) {
            frames.push_back(page);
            q.push(page);
        } else {
            int removePage = q.front();
            q.pop();

            // Replace oldest page
            for (int i = 0; i < frameSize; i++) {
                if (frames[i] == removePage) {
                    frames[i] = page;
                    break;
                }
            }
            q.push(page);
        }
    }

    // Display current state
    cout << left << setw(10) << page;
    for (int f : frames) cout << f << " ";
    int spaces = (frameSize - frames.size()) * 2;
    for (int i = 0; i < spaces; i++) cout << " ";
    cout << "\t" << (hit ? "HIT" : "MISS") << endl;
}

cout << "\nTotal Page Faults: " << pageFaults << endl;
cout << "Total Hits: " << reference.size() - pageFaults << endl;

return 0;
```

```
}
```

15. Write a program to simulate Page replacement algorithm : LRU Reference String : 1,2,3,4,2,5,3,4

Page Frame Size:3

Ans

```
#include <iostream>
#include <vector>
#include <unordered_map>
#include <iomanip>
using namespace std;

int main() {
    vector<int> reference = {1, 2, 3, 4, 2, 5, 3, 4};
    int frameSize = 3;

    vector<int> frames;              // current frames
    unordered_map<int, int> lastUsed;   // page -> last used time
    int time = 0, pageFaults = 0;

    cout << "--- LRU Page Replacement Algorithm ---\n";
    cout << "\nReference String: ";
    for (int r : reference) cout << r << " ";
    cout << "\nPage Frame Size: " << frameSize << "\n\n";

    cout << left << setw(10) << "Ref" << setw(25) << "Frames" << "Status\n";
    cout << "---------------------------------------------\n";
```

```
for (int page : reference) {
    time++;
    bool hit = false;

    // Check if page already in frame
    for (int f : frames) {
        if (f == page) {
            hit = true;
            lastUsed[page] = time;
            break;
        }
    }

    if (!hit) {
        pageFaults++;

        if (frames.size() < frameSize) {
            frames.push_back(page);
        } else {
            // Find least recently used page
            int lruPage = frames[0];
            int minTime = lastUsed[lruPage];

            for (int f : frames) {
                if (lastUsed[f] < minTime) {
                    minTime = lastUsed[f];
                    lruPage = f;
                    }
                }

                // Replace LRU page
                for (int i = 0; i < frameSize; i++) {
                    if (frames[i] == lruPage) {
                        frames[i] = page;
                        break;
                    }
                }
            }

            lastUsed[page] = time;
        }

        // Display current state
        cout << left << setw(10) << page;
        for (int f : frames) cout << f << " ";
        int spaces = (frameSize - frames.size()) * 2;
        for (int i = 0; i < spaces; i++) cout << " ";
        cout << "\t" << (hit ? "HIT" : "MISS") << endl;
    }

    cout << "\nTotal Page Faults: " << pageFaults << endl;
    cout << "Total Hits: " << reference.size() - pageFaults << endl;

    return 0;
}
```

16. Write a program to simulate Page replacement algorithm : OPTIMAL Reference String : 1,2,3,4,2,5,3,4

Page Frame Size:3

Ans

```
#include <iostream>

#include <vector>

#include <iomanip>

#include <algorithm>

using namespace std;


int main() {

    vector<int> reference = {1, 2, 3, 4, 2, 5, 3, 4};

    int frameSize = 3;


    vector<int> frames; // current pages in frames

    int pageFaults = 0;


    cout << "--- Optimal Page Replacement Algorithm ---\n";

    cout << "\nReference String: ";

    for (int r : reference) cout << r << " ";

    cout << "\nPage Frame Size: " << frameSize << "\n\n";


    cout << left << setw(10) << "Ref" << setw(25) << "Frames" << "Status\n";

    cout << "----------------------------------------\n";
```

```
for (int i = 0; i < reference.size(); i++) {
    int page = reference[i];
    bool hit = false;

    // Check if page already exists in frame
    for (int f : frames) {
        if (f == page) {
            hit = true;
            break;
        }
    }

    if (!hit) {
        pageFaults++;

        if (frames.size() < frameSize) {
            frames.push_back(page);
        } else {
            // Predict which page will not be used for the longest time
            int farthest = i + 1;
            int replaceIndex = -1;
            int maxFutureIndex = -1;

            for (int j = 0; j < frames.size(); j++) {
                int k;
                for (k = i + 1; k < reference.size(); k++) {
                    if (frames[j] == reference[k]) break;
                }
                // If page never used again, replace it immediately
                if (k == reference.size()) {
                    replaceIndex = j;
                    break;
                }

                if (k > maxFutureIndex) {
                    maxFutureIndex = k;
                    replaceIndex = j;
                }
            }

            frames[replaceIndex] = page;
        }
    }

    // Display current frame status
    cout << left << setw(10) << page;
    for (int f : frames) cout << f << " ";
    int spaces = (frameSize - frames.size()) * 2;
    for (int s = 0; s < spaces; s++) cout << " ";
    cout << "\t" << (hit ? "HIT" : "MISS") << endl;
}

cout << "\nTotal Page Faults: " << pageFaults << endl;
cout << "Total Hits: " << reference.size() - pageFaults << endl;
```

```
    return 0;
}
```