#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

// Structure for memory blocks, with size, id, and allocation status

struct MemoryBlock {

int size;

int id;

bool allocated;

};

// Structure for processes, with size and id of allocated block (-1 if not allocated)

struct Process {

int size;

int blockId; // -1 if not allocated

};

// Function for Best Fit allocation strategy

void bestFit(vector<MemoryBlock>& blocks, vector<Process>& processes) {

for (auto& process : processes) {

int bestIdx = -1;

for (int i = 0; i < blocks.size(); i++) {

if (!blocks[i].allocated && blocks[i].size >= process.size) {

if (bestIdx == -1 || blocks[i].size < blocks[bestIdx].size) {

bestIdx = i;

}

}

}

if (bestIdx != -1) {

process.blockId = blocks[bestIdx].id;

blocks[bestIdx].allocated = true;

}

}

}

// Function for First Fit allocation strategy

void firstFit(vector<MemoryBlock>& blocks, vector<Process>& processes) {

for (auto& process : processes) {

for (auto& block : blocks) {

if (!block.allocated && block.size >= process.size) {

process.blockId = block.id;

block.allocated = true;

break;

}

}

}

}

// Function for Next Fit allocation strategy

void nextFit(vector<MemoryBlock>& blocks, vector<Process>& processes) {

int lastAllocatedIndex = 0;

for (auto& process : processes) {

int start = lastAllocatedIndex;

int allocated = false;

for (int i = 0; i < blocks.size(); i++) {

int idx = (start + i) % blocks.size();

if (!blocks[idx].allocated && blocks[idx].size >= process.size) {

process.blockId = blocks[idx].id;

blocks[idx].allocated = true;

lastAllocatedIndex = idx;

allocated = true;

break;

}

}

if (!allocated) lastAllocatedIndex = 0;

}

}

// Function for Worst Fit allocation strategy

void worstFit(vector<MemoryBlock>& blocks, vector<Process>& processes) {

for (auto& process : processes) {

int worstIdx = -1;

for (int i = 0; i < blocks.size(); i++) {

if (!blocks[i].allocated && blocks[i].size >= process.size) {

if (worstIdx == -1 || blocks[i].size > blocks[worstIdx].size) {

worstIdx = i;

}

}

}

if (worstIdx != -1) {

process.blockId = blocks[worstIdx].id;

blocks[worstIdx].allocated = true;

}

}

}

// Function to reset all memory blocks to unallocated

void resetMemory(vector<MemoryBlock>& blocks) {

for (auto& block : blocks) {

block.allocated = false;

}

}

// Function to display the allocation results

void displayAllocation(const vector<Process>& processes) {

cout << "\nProcess Allocation:\n";

for (const auto& process : processes) {

if (process.blockId != -1)

cout << "Process of size " << process.size << " -> Block " << process.blockId << endl;

else

cout << "Process of size " << process.size << " -> Not Allocated\n";

}

}

int main() {

int numBlocks, numProcesses;

// Get memory block sizes

cout << "Enter number of memory blocks: ";

cin >> numBlocks;

vector<MemoryBlock> blocks(numBlocks);

for (int i = 0; i < numBlocks; i++) {

cout << "Enter size of block " << i + 1 << ": ";

cin >> blocks[i].size;

blocks[i].id = i + 1;

blocks[i].allocated = false;

}

// Get process sizes

cout << "\nEnter number of processes: ";

cin >> numProcesses;

vector<Process> processes(numProcesses);

for (int i = 0; i < numProcesses; i++) {

cout << "Enter size of process " << i + 1 << ": ";

cin >> processes[i].size;

processes[i].blockId = -1;

}

int choice;

do {

cout << "\nMemory Placement Strategies:\n";

cout << "1. Best Fit\n2. First Fit\n3. Next Fit\n4. Worst Fit\n5. Exit\n";

cout << "Enter your choice: ";

cin >> choice;

// Reset blocks and processes for each choice

resetMemory(blocks);

for (auto& process : processes) process.blockId = -1;

// Perform allocation based on chosen strategy

switch (choice) {

case 1:

bestFit(blocks, processes);

cout << "\nBest Fit Allocation:";

displayAllocation(processes);

break;

case 2:

firstFit(blocks, processes);

cout << "\nFirst Fit Allocation:";

displayAllocation(processes);

break;

case 3:

nextFit(blocks, processes);

cout << "\nNext Fit Allocation:";

displayAllocation(processes);

break;

case 4:

worstFit(blocks, processes);

cout << "\nWorst Fit Allocation:";

displayAllocation(processes);

break;

case 5:

cout << "Exiting...\n";

break;

default:

cout << "Invalid choice! Please choose again.\n";

}

} while (choice != 5);

return 0;

}