Finalexam
COEN 275-OOAD

Pujitha Kallu
ID : W1653660
pkallu@scu.edu

1.) Design a Dependency Injection Framework using the Factory pattern in C++. The framework

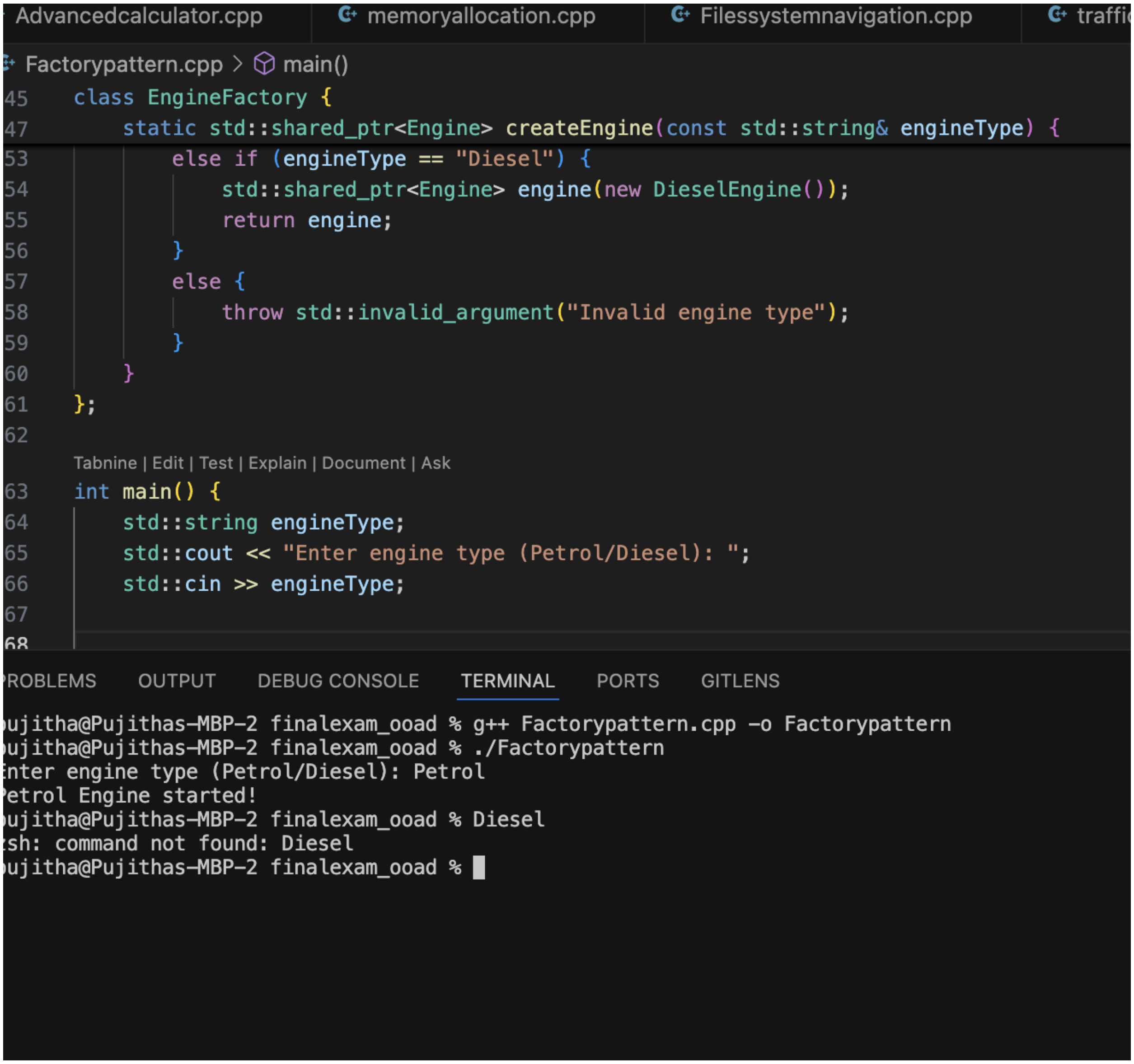
should allow injecting dependencies into objects dynamically at runtime.

• Implement a Car manufacturing system where engines (PetrolEngine, DieselEngine) are

injected into Car objects based on runtime configuration.

• Provide code and explanations.

Sol:


Advancedcalculator.cpp
memoryallocation.cpp
Filessystemnavigation.cpp
traffi
Factorypattern.cpp > main()
Tabnine | Edit | Test | Explain | Document | Ask
ROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS

```
class EngineFactory {
    static std::shared_ptr<Engine> createEngine(const std::string& engineType) {
        else if (engineType == "Diesel") {
            std::shared_ptr<Engine> engine(new DieselEngine());
            return engine;
        }
        else {
            throw std::invalid_argument("Invalid engine type");
        }
    }
};

int main() {
    std::string engineType;
    std::cout << "Enter engine type (Petrol/Diesel): ";
    std::cin >> engineType;

```

```
ujitha@Pujithas-MBP-2 finalexam_ooad % g++ Factorypattern.cpp -o Factorypattern
ujitha@Pujithas-MBP-2 finalexam_ooad % ./Factorypattern
nter engine type (Petrol/Diesel): Petrol
etrol Engine started!
ujitha@Pujithas-MBP-2 finalexam_ooad % Diesel
sh: command not found: Diesel
ujitha@Pujithas-MBP-2 finalexam_ooad % 
```

Explanation:

- Engine is the abstract base class.
- PetrolEngine and DieselEngine are concrete implementations of the Engine class.
- Car accepts a dependency of type Engine through its constructor.
- EngineFactory is a static factory class that creates the appropriate engine (PetrolEngine or DieselEngine) based on runtime input.

- The main() function gets the engine type from the user, creates the appropriate engine, injects it into the Car object, and starts the car.

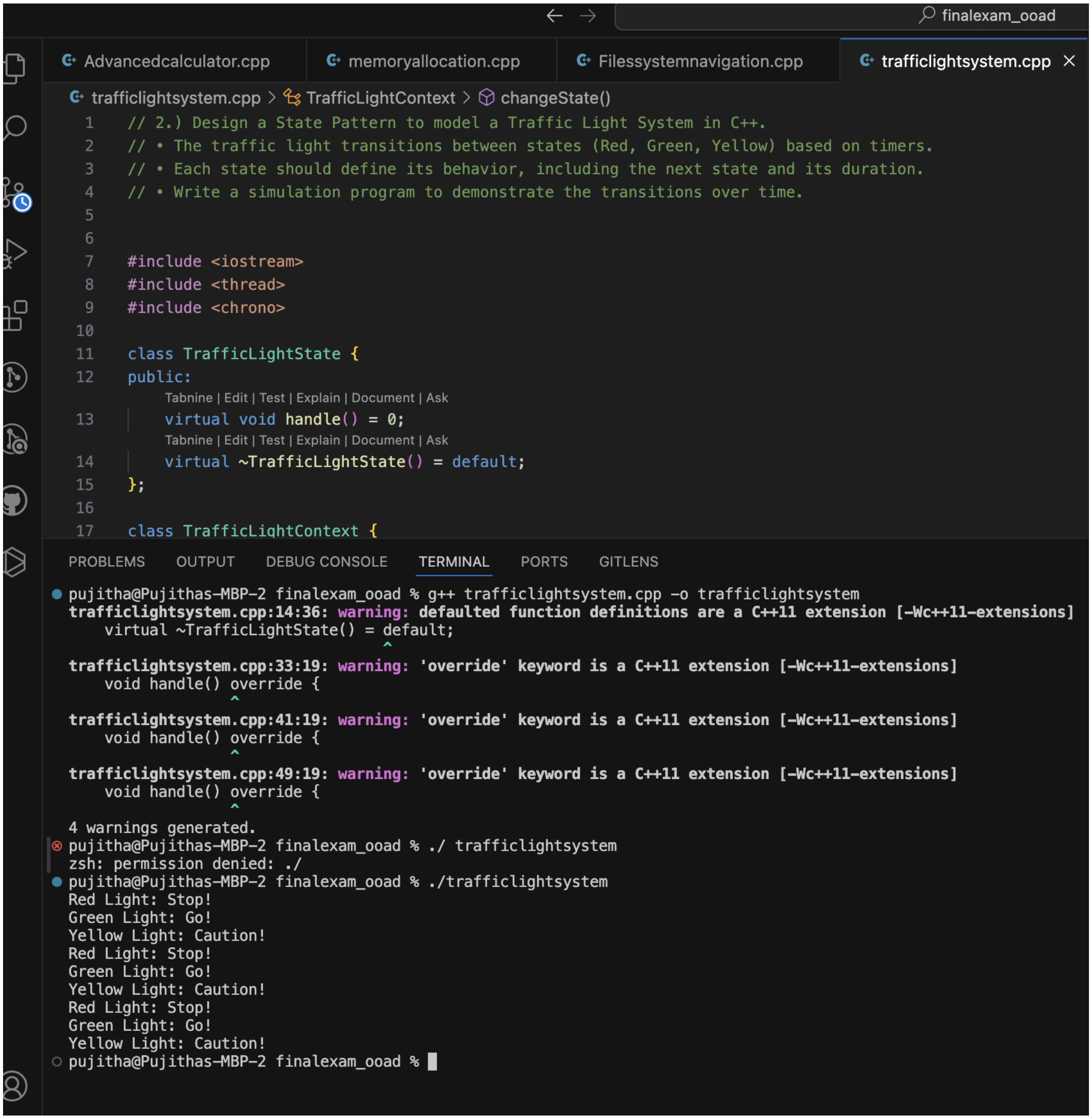
2.) Design a State Pattern to model a Traffic Light System in C++.

• The traffic light transitions between states (Red, Green, Yellow) based on timers.

• Each state should define its behavior, including the next state and its duration.

• Write a simulation program to demonstrate the transitions over time.

Sol:


finalexam_ooad
Advancedcalculator.cpp
memoryallocation.cpp
Filessystemnavigation.cpp
trafficlightsystem.cpp
trafficlightsystem.cpp > TrafficLightContext > changeState()
Tabnine | Edit | Test | Explain | Document | Ask
Tabnine | Edit | Test | Explain | Document | Ask
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS

```cpp
// 2.) Design a State Pattern to model a Traffic Light System in C++.
// • The traffic light transitions between states (Red, Green, Yellow) based on timers.
// • Each state should define its behavior, including the next state and its duration.
// • Write a simulation program to demonstrate the transitions over time.


#include <iostream>
#include <thread>
#include <chrono>

class TrafficLightState {
public:
    virtual void handle() = 0;
    virtual ~TrafficLightState() = default;
};

class TrafficLightContext {
```

```
pujitha@Pujithas-MBP-2 finalexam_ooad % g++ trafficlightsystem.cpp -o trafficlightsystem
trafficlightsystem.cpp:14:36: warning: defaulted function definitions are a C++11 extension [-Wc++11-extensions]
    virtual ~TrafficLightState() = default;
                                   ^
trafficlightsystem.cpp:33:19: warning: 'override' keyword is a C++11 extension [-Wc++11-extensions]
    void handle() override {
                  ^
trafficlightsystem.cpp:41:19: warning: 'override' keyword is a C++11 extension [-Wc++11-extensions]
    void handle() override {
                  ^
trafficlightsystem.cpp:49:19: warning: 'override' keyword is a C++11 extension [-Wc++11-extensions]
    void handle() override {
                  ^
4 warnings generated.
pujitha@Pujithas-MBP-2 finalexam_ooad % ./ trafficlightsystem
zsh: permission denied: ./
pujitha@Pujithas-MBP-2 finalexam_ooad % ./trafficlightsystem
Red Light: Stop!
Green Light: Go!
Yellow Light: Caution!
Red Light: Stop!
Green Light: Go!
Yellow Light: Caution!
Red Light: Stop!
Green Light: Go!
Yellow Light: Caution!
pujitha@Pujithas-MBP-2 finalexam_ooad %
```

Explanation:

- **TrafficLightState** is the abstract state class.
- **RedState**, **GreenState**, and **YellowState** are concrete state classes that define behavior specific to each traffic light color.
- **TrafficLightContext** is the context class that holds the current state and allows state transitions.
- In `main()`, we simulate the cycle of the traffic light by changing the state between

3.) Composite Pattern for Filesystem Navigation Design a Filesystem Navigation System using

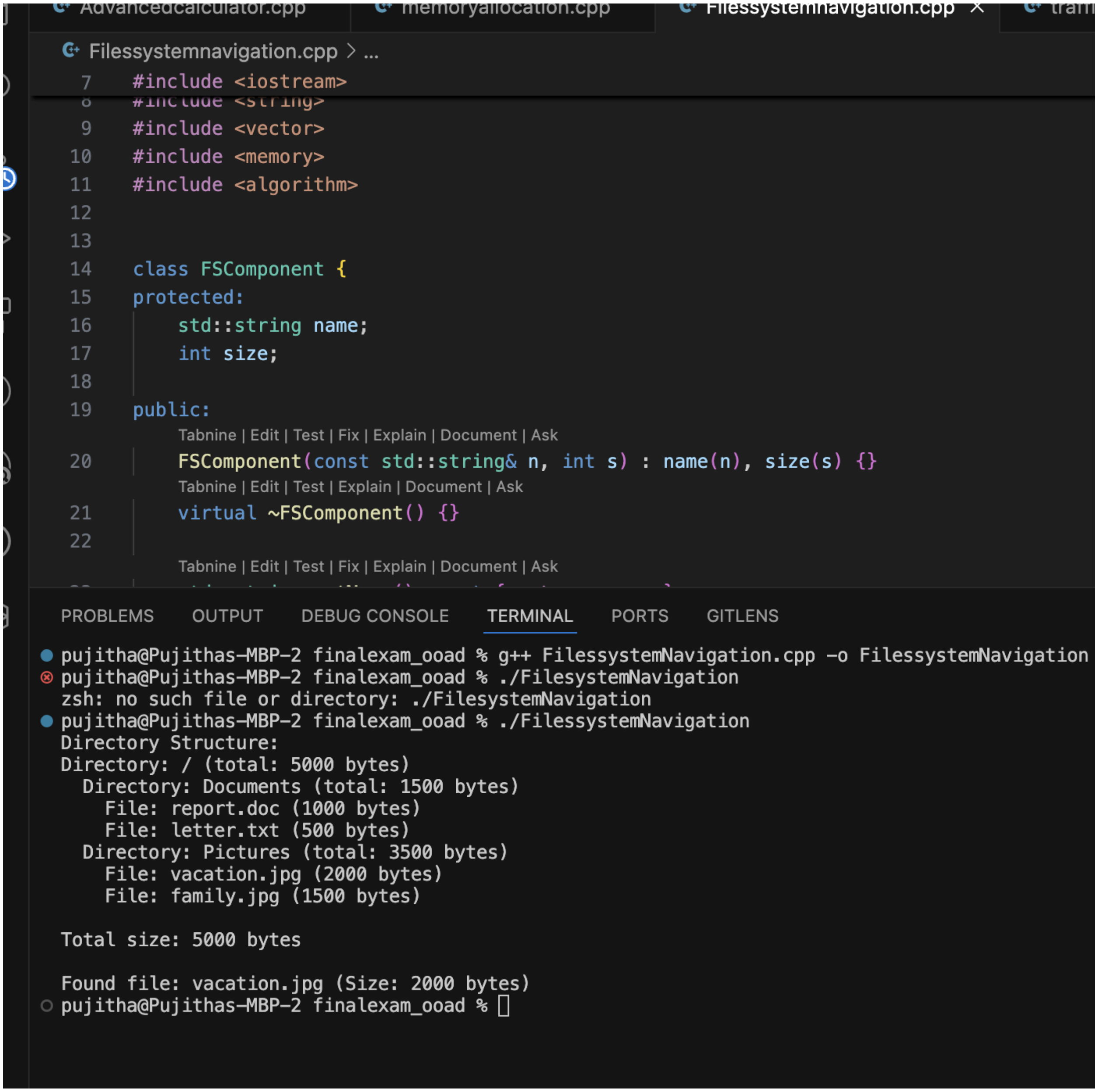
the Composite pattern in C++.

• Implement components like File and Directory, where Directory can contain File objects or

other Directory objects.

• Write methods to calculate the total size of a directory and list all files recursively.

Sol:



AdvancedCalculator.cpp
memoryallocation.cpp
Filessystemnavigation.cpp
tram
Filessystemnavigation.cpp > ...
Tabnine | Edit | Test | Fix | Explain | Document | Ask
Tabnine | Edit | Test | Explain | Document | Ask
Tabnine | Edit | Test | Fix | Explain | Document | Ask
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS

```cpp
#include <iostream>
#include <string>
#include <vector>
#include <memory>
#include <algorithm>


class FSComponent {
protected:
    std::string name;
    int size;

public:
    FSComponent(const std::string& n, int s) : name(n), size(s) {}
    virtual ~FSComponent() {}

```

```
pujitha@Pujithas-MBP-2 finalexam_ooad % g++ FilessystemNavigation.cpp -o FilessystemNavigation
pujitha@Pujithas-MBP-2 finalexam_ooad % ./FilesystemNavigation
zsh: no such file or directory: ./FilesystemNavigation
pujitha@Pujithas-MBP-2 finalexam_ooad % ./FilessystemNavigation
Directory Structure:
Directory: / (total: 5000 bytes)
  Directory: Documents (total: 1500 bytes)
    File: report.doc (1000 bytes)
    File: letter.txt (500 bytes)
  Directory: Pictures (total: 3500 bytes)
    File: vacation.jpg (2000 bytes)
    File: family.jpg (1500 bytes)

Total size: 5000 bytes

Found file: vacation.jpg (Size: 2000 bytes)
pujitha@Pujithas-MBP-2 finalexam_ooad % 
```

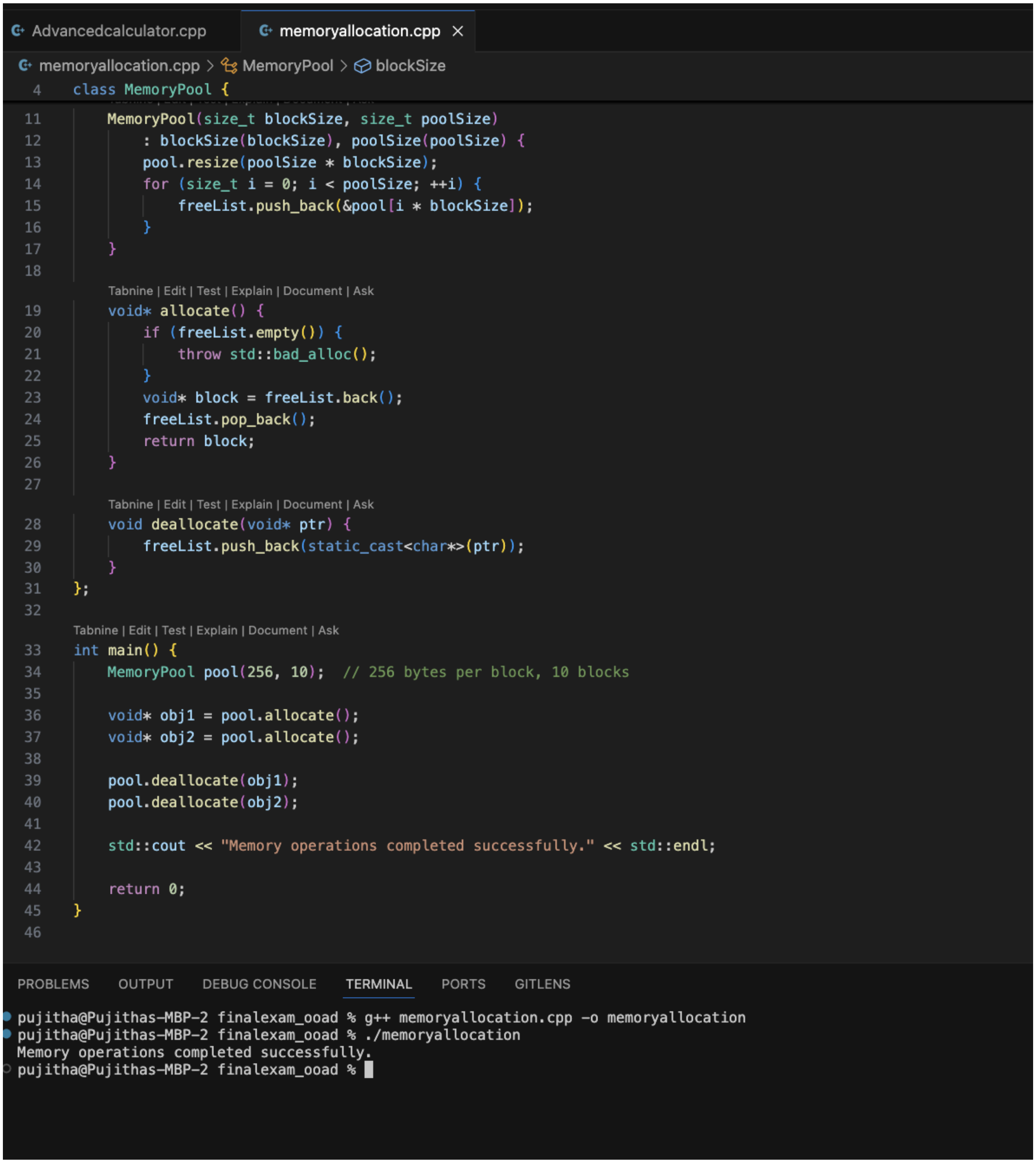
Explanation:

- **Component** is the base class for both `File` and `Directory`.
- **File** represents individual files, and **Directory** represents directories that can contain files or other directories.
- The `getSize()` method calculates the total size of the directory, including nested directories.
- **Main()** demonstrates the creation of a directory tree and lists its contents, as well as the total size.

4.) Write a custom memory allocator in C++ that: Allocates memory in chunks from a large pre-allocated pool.

- **Tracks allocated and freed memory to reuse blocks efficiently.**
- **Supports allocating and freeing memory of various sizes.**
- **Demonstrate its usage by creating and freeing several objects dynamically.**

Sol:


Advancedcalculator.cpp
memoryallocation.cpp
memoryallocation.cpp > MemoryPool > blockSize
Tabnine | Edit | Test | Explain | Document | Ask
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS

```cpp
class MemoryPool {
    MemoryPool(size_t blockSize, size_t poolSize)
        : blockSize(blockSize), poolSize(poolSize) {
        pool.resize(poolSize * blockSize);
        for (size_t i = 0; i < poolSize; ++i) {
            freeList.push_back(&pool[i * blockSize]);
        }
    }

    void* allocate() {
        if (freeList.empty()) {
            throw std::bad_alloc();
        }
        void* block = freeList.back();
        freeList.pop_back();
        return block;
    }

    void deallocate(void* ptr) {
        freeList.push_back(static_cast<char*>(ptr));
    }
};

int main() {
    MemoryPool pool(256, 10);  // 256 bytes per block, 10 blocks

    void* obj1 = pool.allocate();
    void* obj2 = pool.allocate();

    pool.deallocate(obj1);
    pool.deallocate(obj2);

    std::cout << "Memory operations completed successfully." << std::endl;

    return 0;
}
```

```
pujitha@Pujithas-MBP-2 finalexam_ooad % g++ memoryallocation.cpp -o memoryallocation
pujitha@Pujithas-MBP-2 finalexam_ooad % ./memoryallocation
Memory operations completed successfully.
pujitha@Pujithas-MBP-2 finalexam_ooad % 
```

Explanation:

- **MemoryPool** is a custom memory allocator that allocates memory from a pre-allocated pool.
- The pool is created with a block size and a number of blocks.

- **allocate()** provides a block of memory, and **deallocate()** returns the memory to the free list.

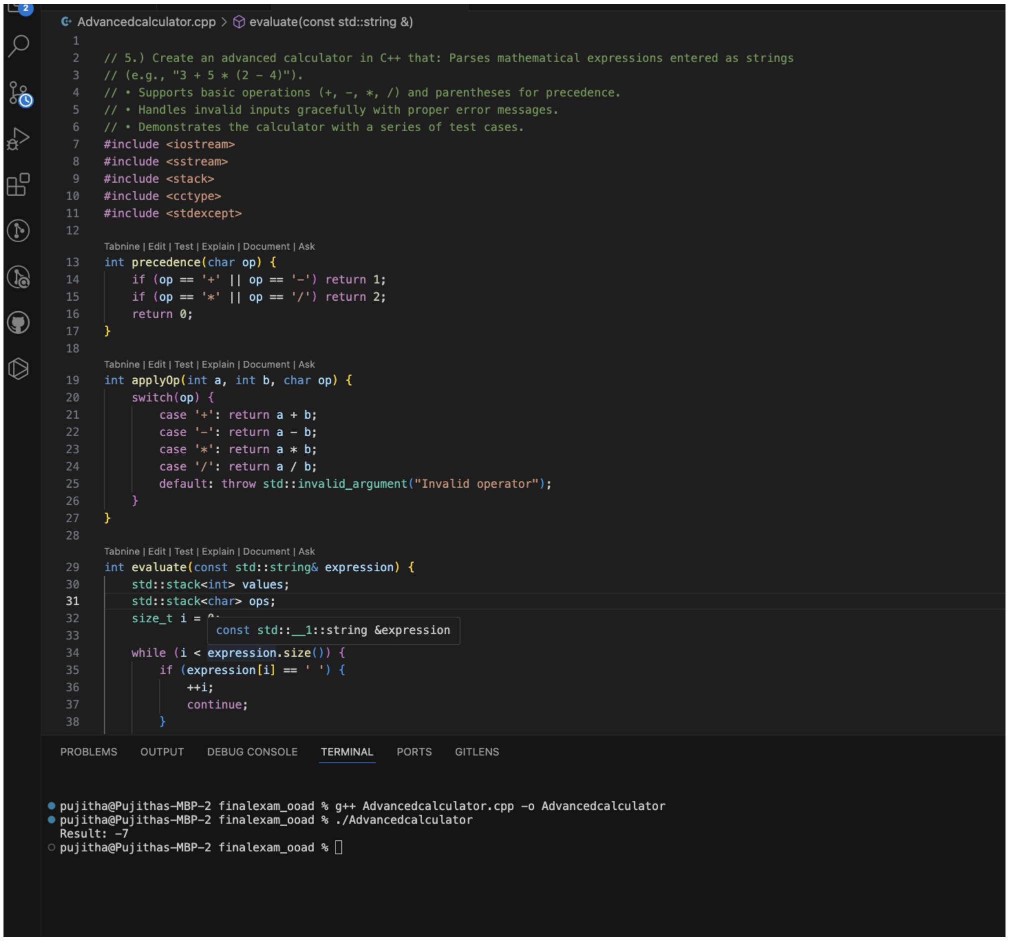
5.) Create an advanced calculator in C++ that: Parses mathematical expressions entered as strings (e.g., "3 + 5 * (2 - 4)").

• Supports basic operations (+, -, *, /) and parentheses for precedence.

• Handles invalid inputs gracefully with proper error messages.

• Demonstrates the calculator with a series of test cases.

Sol:



Advancedcalculator.cpp > evaluate(const std::string &)
Tabnine | Edit | Test | Explain | Document | Ask
const std::__1::string &expression
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
GITLENS

```cpp

// 5.) Create an advanced calculator in C++ that: Parses mathematical expressions entered as strings
// (e.g., "3 + 5 * (2 - 4)").
// • Supports basic operations (+, -, *, /) and parentheses for precedence.
// • Handles invalid inputs gracefully with proper error messages.
// • Demonstrates the calculator with a series of test cases.
#include <iostream>
#include <sstream>
#include <stack>
#include <cctype>
#include <stdexcept>

int precedence(char op) {
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    return 0;
}

int applyOp(int a, int b, char op) {
    switch(op) {
        case '+': return a + b;
        case '-': return a - b;
        case '*': return a * b;
        case '/': return a / b;
        default: throw std::invalid_argument("Invalid operator");
    }
}

int evaluate(const std::string& expression) {
    std::stack<int> values;
    std::stack<char> ops;
    size_t i = 0;

    while (i < expression.size()) {
        if (expression[i] == ' ') {
            ++i;
            continue;
        }
```

```
pujitha@Pujithas-MBP-2 finalexam_ooad % g++ Advancedcalculator.cpp -o Advancedcalculator
pujitha@Pujithas-MBP-2 finalexam_ooad % ./Advancedcalculator
Result: -7
pujitha@Pujithas-MBP-2 finalexam_ooad % 
```

Explanation:

- The calculator supports the basic operators (**+**, **-**, *****, **/**) and parentheses.
- It uses two stacks to manage the operands and operators.
- **precedence()** determines the precedence of operators, and **applyOp()** applies the operator.
- The expression is parsed and evaluated step by step, handling parentheses and operator precedence.