Trắc nghiệm TASK 7

```
|-----------------------------------------------------|
|    | 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  |
|-----------------------------------------------------|
| 0  | _  | A  | A  | A  | A  | C  | A  | C  | A  | A  |
| 1  | A  | A  | A  | A  | A  | A  | A  | A  | A  | A  |
| 2  | A  | A  | A  | A  | A  | A  | A  | A  | A  | A  |
|-----------------------------------------------------|
```

Phần đọc code:

Phần đọc code TASK7

Câu 1: Kết quả và giải thích

```
Constructor called!
Constructor called!
Destructor called!
End of program
Destructor called!
```

```
// GIẢI THÍCH
Demo obj1; // Khởi tạo đối tượng obj1, gọi đến constructor không tham số
Demo* obj2 = new Demo(); // Khởi tạo đối tượng obj2, gọi đến constructor không
tham số
delete obj2 // gọi hàm huỷ của obj2
cout << "End of program" << endl; // End of program
// Kết thúc hàm main, obj1 hết tầm vực, được gọi hàm huỷ
```

Câu 2: Kết quả và giải thích

```
Access using object: 10
Value of x: 10:
Access using pointer: 20
Value of x: 20
```

```
// GIẢI THÍCH
Sample obj(10); // Gọi constructor tham số, gán obj.x = 10
Sample* ptr = new Sample(20); // Gọi constructor tham số, gán ptr->x = 20
// In giá trị lần lượt
```

Câu 3: Kết quả và giải thích

```
Default
Single
Two
x = 0, y = 0
x = 10, y = 0
x = 5, y = 15
```

```
// GIẢI THÍCH
Sample obj1; // Gọi constructor mặc định in ra Default x = 0, y = 0
Sample obj2(10); // Gọi constructor một tham số in ra Single x = 10, y = 0
Sample obj3(5, 15); // Gọi constructor 2 tham số in ra Two x = 5, y = 15
// In lần lượt bằng hàm display
```

Câu 4: Kết quả và giải thích

```
Before modification:
obj1 = 10
obj2 = 10
After modification:
obj1 = 99
obj2 = 99
free(): double free detected in tcache 2
```

```
// GIẢI THÍCH
ShallowCopy obj1(10); // Gọi constructor không tham số, cấp phát động cho data
ShallowCopy obj2 = obj1; // Gọi copy constructor, biến obj2.data được gán bằng
obj1.data, obj1.data chứa địa chỉ biến được cấp phát động khi gán bằng cả hai biến
đều trỏ tới một vùng nhớ trên heap.
// Vì cùng trỏ vào một vùng nhớ nên khi truy xuất bằng dấu . và dùng * để lấy giá
trị thì tất nhiên sẽ có giá trị giống nhau
// Cập nhật giá trị của vùng nhớ đó lên 99
// Tương tự in ra 2 lần 99
// Hết tầm vực, hàm huỷ được gọi, vì cả hai cùng trỏ vào một vùng nhớ nên khi huỷ
lần lượt obj1 và obj2, vùng nhớ bị delete 2 lần
// Xảy ra lỗi double free
```

> Câu 5: Kết quả và giải thích

```
Before modification:
obj1 = 20
obj2 = 20
After modification:
```

```
obj1 = 20
obj2 = 88
```

```
// GIẢI THÍCH
DeepCopy obj1(20); // Gọi constructor không tham số, cấp phát động cho data
DeepCopy obj2 = obj1; // Gọi copy constructor, biến obj2.data được cấp phát mới,
trỏ vào vùng nhớ khác và có giá trị bằng với biến được trỏ bởi obj1.data.
// Vì 2 vùng nhớ được cấp phát trên heap có cùng giá trị nên in ra sẽ giống nhau
// Thay đổi *obj2.data = 88 không ảnh hưởng tới *obj1.data
// Kết quả là *obj2.data bị thay đổi nhưng *obj1.data không thay đổi
// Vì data của 2 obj trỏ các vùng nhớ khác nhau nên khi huỷ lần lượt do thoát khỏi
tầm vực không ảnh hưởng đến nhau nên không bị lỗi double free giống Shadow copy
```

Câu 6: Kết quả và giải thích

```
Value of x: 0
```

```
// GIẢI THÍCH
Test obj(10); // Gọi constructor tham số, vì trùng tên biến nên nếu muốn truy xuất
biến của đối tượng thì phải thêm this->x thay vì x.
// Khi viết x = x, trình biên dịch hiểu là biến cục bộ x (tham số của hàm) đang
gán giá trị của chính nó.
// Biến thành viên x không bị thay đổi và giữ giá trị mặc định (với kiểu int, giá
trị mặc định là 0 nếu không được khởi tạo).
```

Câu 7: Kết quả và giải thích

```
Constructor called with value: 10
Constructor called with value: 0
Value: 10
Value: 0
```

```
// GIẢI THÍCH
Number num1(10); // Gọi constructor có sử dụng khởi tạo danh sách thành viên (VD:
a(a), b(b), c(c)) kết hợp với toán tử 3 ngôi để gán giá trị (expression) ? (result
1) : (result 2), ở đây nếu v là âm thì gán value là 0 còn v dương thì gán trực
tiếp cho value.
Number num2(-5); // num2.value = 0 do v = -5 ban đầu là âm
```

Câu 8: Kết quả và giải thích

```
Array contents: 10 20 30 0 0
Element at index 1: 20
```

```
// GIẢI THÍCH
int* data; // Mảng cấp phát động
int size; // Số lượng phần tử hiện tại

Array arr(5); // Gọi constructor tham số, khởi tạo mảng động data gồm 5 phần tử.
arr.setValue(0, 10); // Cập nhật giá trị cho phần tử đầu tiên
arr.setValue(1, 20); // Cập nhật giá trị cho phần tử thứ hai
arr.setValue(2, 30); // Cập nhật giá trị cho phần tử thứ ba

arr.display(); // Lặp qua vòng lặp in ra giá trị các phần tử có trong mảng
cout << "Element at index 1: " << arr.getValue(1) << endl; // In phần tử thứ hai
// Hàm huỷ tự động được gọi khi hết tầm vực của object
```

Câu 9: Kết quả và giải thích

```
Matrix contents:
1 0 0
0 5 0
0 0 9
Element at (1,1): 5
```

```
// GIẢI THÍCH
Matrix mat(3, 3); // Khởi tạo ma trận cấp phát động 3 x 3 với các giá trị 0

mat.setValue(0, 0, 1); // Gán giá trị tại cột 1 hàng 1 là 1
mat.setValue(1, 1, 5); // Gán giá trị tại cột 2 hàng 2 là 5
mat.setValue(2, 2, 9); // Gán giá trị tại cột 3 hàng 3 là 9

mat.display(); // Duyệt cột, hàng để in ma trận
cout << "Element at (1,1): " << mat.getValue(1, 1) << endl; // In giá trị tại cột
2 hàng 2
```

Câu 11: Kết quả và giải thích

```
Using returnThis():
Value: 15
Using returnByValue():
Value: 35
Final value of obj:
Value: 35
```

```
// GIẢI THÍCH
Sample obj(5); // Tạo đối tượng, khởi tạo obj.value = 5;
obj.returnThis()->display(); // Hàm returnThis trả về con trỏ this trỏ tới đối tượng obj, dùng display() thông qua toán tử -> in ra giá trị value của obj sau khi cập nhật là 15.
obj.returnByValue().display(); // Hàm returnThis trả về đối tượng bản sao giống với obj, dùng display() thông qua toán tử . in ra giá trị value của bản sao chứ không phải obj là 35
obj.display() // In ra giá trị value của obj
```

Bài tập

Câu 1:

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

class Array
{
private:
    int *data;
    int size;

public:
    // Constructor 1: Khởi tạo mảng có kích thước size
    Array(int size) : data(new int[size]()) {
    }

    // Constructor 2: Khởi tạo mảng từ dữ liệu có sẵn
    Array(int *arr, int size) : data(new int[size]()), size(size) {
        for (int i = 0; i < size; i++) data[i] = arr[i];
    }

    // Constructor 3: Constructor sao chép
    Array(const Array &other) : size(other.size), data(new int[other.size]()) {
        int val;
        for (int i = 0; i < size; i++) data[i] = other.data[i];
    }

    // Destructor: Giải phóng bộ nhớ
    ~Array() {
        delete[] data;
        size = 0;
    }

    // Lấy giá trị tại index
    int index(int i) {
        return data[i];
    }
```

```
    // Đảo ngược mảng
    void reverse() {
        int temp;
        for (int i = 0; i < size / 2; i++)
            temp = data[i],
            data[i] = data[size - 1 - i],
            data[size - 1 - i] = temp;
    }

    // Tìm phần tử lớn nhất trong mảng
    int maxElement() {
        int maxV = INT_MIN;
        for (int i = 0; i < size; i++)
            maxV = max(maxV, data[i]);
        return maxV;
    }

    // Tìm tổng lớn nhất của ba số liên tiếp
    int maxThreeSum() {
        if (size <= 2) return -1;
        int maxV = INT_MIN;
        for (int i = 0; i < size - 2; i++)
            maxV = max(maxV, data[i] + data[i + 1] + data[i + 2]);
        return maxV;
    }

    // Hiển thị mảng
    void display() {
        for (int i = 0; i < size; i++){
            cout << data[i] << " ";
        }
        cout << endl;
    }
};

int main() {
    int arr[] = {1, 2, 3, 4, 5, 6};
    Array array1(6);
    Array array2(arr, 6);  // Khởi tạo từ mảng có sẵn
    Array array3 = array2; // Constructor sao chép

    cout << "Original array: ";
    array2.display();

    array2.reverse();
    cout << "Reversed array: ";
    array2.display();

    cout << "Max element: " << array2.maxElement() << endl;
    cout << "Max sum of 3 consecutive elements: " << array2.maxThreeSum() << endl;

    return 0;
}
```

Câu 2:

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

class Matrix
{
private:
    int **data;
    int rows, cols;

public:
    // Constructor 1: Khởi tạo ma trận với giá trị mặc định 0
    Matrix(int r, int c) : rows(r), cols(c) {
        data = new int *[rows];
        for (int i = 0; i < rows; i++)
            data[i] = new int[cols]();
    }

    // Constructor 2: Khởi tạo từ dữ liệu có sẵn
    Matrix(int **arr, int r, int c) : rows(r), cols(c){
        data = new int *[rows];
        for (int i = 0; i < rows; i++){
            data[i] = new int[cols];
            for (int j = 0; j < cols; j++)
                data[i][j] = arr[i][j];
        }
    }

    // Constructor 3: Constructor sao chép
    Matrix(const Matrix &other) : rows(other.rows), cols(other.cols)
    {
        data = new int *[rows];
        for (int i = 0; i < rows; i++){
            data[i] = new int[cols];
            for (int j = 0; j < cols; j++)
                data[i][j] = other.data[i][j];
        }
    }

    // Destructor: Giải phóng bộ nhớ
    ~Matrix()
    {
        for (int i = 0; i < rows; i++)
            delete[] data[i];
        delete[] data;
    }

    // Phép cộng hai ma trận (phải cùng kích thước)
    Matrix add(const Matrix &other)
    {
```

```
        if (rows != other.rows || cols != other.cols) {
            cout << "Error: must be teh same size\n";
            return Matrix(0, 0);
        }

        Matrix result(rows, cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < cols; j++)
                result.data[i][j] = data[i][j] + other.data[i][j];
        return result;
    }

    // Phép nhân hai ma trận (số cột của ma trận 1 phải bằng số hàng của ma trận
2)
    Matrix multiply(const Matrix &other)
    {
        if (cols != other.rows) {
            cerr << "Error: cols must be equal to other rows" << endl;
            return Matrix(0, 0);
        }

        Matrix res(rows, other.cols);
        for (int i = 0; i < rows; i++)
            for (int j = 0; j < other.cols; j++){
                res.data[i][j] = 0;
                for (int k = 0; k < cols; k++)
                    res.data[i][j] += data[i][k] * other.data[k][j];
            }
        return res;
    }

    // Hiển thị ma trận
    void display()
    {
        for (int i = 0; i < rows; i++){
            for (int j = 0; j < cols; j++)
                cout << data[i][j] << " ";
            cout << endl;
        }
    }
};

int main()
{
    int r1 = 2, c1 = 3;
    int r2 = 3, c2 = 2;

    int **arr1 = new int *[r1];
    int **arr2 = new int *[r2];

    for (int i = 0; i < r1; i++) {
        arr1[i] = new int[c1]{i + 1, i + 2, i + 3};
    }
```

```
    for (int i = 0; i < r2; i++) {
        arr2[i] = new int[c2]{i + 1, i + 2};
    }

    Matrix mat1(arr1, r1, c1);
    Matrix mat2(arr2, r2, c2);

    cout << "Matrix 1:" << endl;
    mat1.display();

    cout << "Matrix 2:" << endl;
    mat2.display();

    Matrix product = mat1.multiply(mat2);
    cout << "Product of Matrices:" << endl;
    product.display();

    // Giải phóng bộ nhớ cho mảng dữ liệu
    for (int i = 0; i < r1; i++) delete[] arr1[i];
    delete[] arr1;

    for (int i = 0; i < r2; i++) delete[] arr2[i];
    delete[] arr2;

    return 0;
}
```

Câu 3:

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

enum GateType { AND_G, OR_G, XOR_G, NAND_G, NOR_G, XNOR_G, NOT_G };

class LogicGate {
private:
    GateType type;

public:
    // Constructor
    LogicGate(GateType gateType) : type(gateType) {}

    // Hàm tính toán kết quả đầu ra
    bool evaluate(bool a, bool b = false) {
        switch (type) {
            case AND_G:  return a & b;
            case OR_G:   return a | b;
            case XOR_G:  return a ^ b;
            case NAND_G: return !(a & b);
            case NOR_G:  return !(a | b);
            case XNOR_G: return !(a ^ b);
```

```
            case NOT_G:  return !a;
            default:     return false;
        }
    }

    // Hàm hiển thị kết quả
    void display(bool a, bool b = false) {
        int gateID = static_cast<int>(type);
        cout << "Gate: " << gateID << " | Input: " << a;
        if (type != NOT_G) cout << " " << b;
        cout << " | Output: " << evaluate(a, b) << endl;
    }
};

int main() {
    LogicGate andGate(AND_G);
    LogicGate orGate(OR_G);
    LogicGate notGate(NOT_G);

    andGate.display(1, 0);
    orGate.display(1, 0);
    notGate.display(1);

    return 0;
}
```

Câu 4

```
\\ Circuit.hpp
#include "wire.hpp"

class Circuit
{
private:
    std::vector<LogicGate *> LogicGates;
    std::vector<Wire *> wires;

public:
    ~Circuit()
    {
        for (LogicGate *gate : LogicGates)
            delete gate;

        for (Wire *wire : wires)
            delete wire;
    }

    LogicGate *addGate(GateType type)
    {
        // TODO: implement
        LogicGate* gae = new LogicGate(type);
```

```
        LogicGates.push_back(gae);
        return gae;
    }

    void connect(LogicGate *from, LogicGate *to, size_t toInput)
    {
        Wire *wire = new Wire(from, to, toInput);
        // TODO: implement
        wires.push_back(wire);
    }

    void update()
    {
        // TODO: implement
        for (Wire* wie : wires) wie->update();
    }
};
// ------------------------------------------------------------
```

```
\\ fullAdder.hpp
#include "circuit.hpp"

class FullAdder
{
private:
    Circuit circuit;
    LogicGate *inputA;
    LogicGate *inputB;
    LogicGate *inputCin;
    LogicGate *sumOutput;
    LogicGate *carryOutput;

public:
    FullAdder()
    {
        // TODO Create input LogicGates
        inputA = circuit.addGate(GateType::INPUT);
        inputB = circuit.addGate(GateType::INPUT);
        inputCin = circuit.addGate(GateType::INPUT);

        //  A ⊕ B
        LogicGate* AxorB = circuit.addGate(GateType::XOR);
        circuit.connect(inputA, AxorB, 0);
        circuit.connect(inputB, AxorB, 1);

        //  (A ⊕ B) ⊕ Cin
        sumOutput = circuit.addGate(GateType::XOR);
        circuit.connect(AxorB, sumOutput, 0);
        circuit.connect(inputCin, sumOutput, 1);

        //  A ∧ B
```

```
        LogicGate* AnB = circuit.addGate(GateType::AND);
        circuit.connect(inputA, AnB, 0);
        circuit.connect(inputB, AnB, 1);

        // TODO Create internal gates
        LogicGate* AcB = circuit.addGate(GateType::OR);
        circuit.connect(inputA, AcB, 0);
        circuit.connect(inputB, AcB, 1);

        //  Cin ∧ (A ⊕ B)
        LogicGate* Cx_AcB_ = circuit.addGate(GateType::AND);
        circuit.connect(AcB, Cx_AcB_, 0);
        circuit.connect(inputCin, Cx_AcB_, 1);

        //  (A ∧ B) ∨ (Cin ∧ (A ⊕ B))
        carryOutput = circuit.addGate(GateType::OR);
        circuit.connect(AnB, carryOutput, 0);
        circuit.connect(Cx_AcB_, carryOutput, 1);
        // TODO Connect gates

        // TODO Set outputs
        circuit.update();
    }

    void setInputs(bool a, bool b, bool cin)
    {
        inputA->setInput(0, a);
        inputB->setInput(0, b);
        inputCin->setInput(0, cin);
        circuit.update();
    }

    bool getSum() const { return sumOutput->getOutput(); }
    bool getCarryOut() const { return carryOutput->getOutput(); }
};
// ----------------------------------------------------------------
```

```
// logicGate.hpp
#include <iostream>
#include <vector>

enum class GateType
{
    AND,
    OR,
    XOR,
    NAND,
    NOR,
    XNOR,
    NOT,
    INPUT
```

```
};

class LogicGate
{
private:
    GateType type;
    std::vector<bool> inputs;
    bool output;

    bool calculateOutput()
    {
        switch (type)
        {
        case GateType::AND:
            return inputs[0] && inputs[1];
        case GateType::OR:
            return inputs[0] || inputs[1];
        case GateType::XOR:
            return inputs[0] != inputs[1];
        case GateType::NAND:
            return !(inputs[0] && inputs[1]);
        case GateType::NOR:
            return !(inputs[0] || inputs[1]);
        case GateType::XNOR:
            return inputs[0] == inputs[1];
        case GateType::NOT:
            return !inputs[0];
        case GateType::INPUT:
            return inputs[0];
        default:
            return false;
        }
    }

public:
    LogicGate(GateType gateType) : type(gateType)
    {
        inputs.resize(gateType == GateType::NOT || gateType == GateType::INPUT ? 1
: 2);
        output = false;
    }

    bool getOutput() const { return output; }

    void setInput(size_t index, bool value)
    {
        if (index >= inputs.size())
            return;
        // TODO: implement
        inputs[index] = value;
        output = calculateOutput();
    }

    size_t getInputCount() const { return inputs.size(); }
```

```
};
// ------------------------------------------------------------
```

```
// Wire.hpp
#include "logicGate.hpp"

class Wire
{
private:
    LogicGate *source;
    LogicGate *target;
    size_t targetInput;

public:
    Wire(LogicGate *src, LogicGate *tgt, size_t tgtInput)
        : source(src), target(tgt), targetInput(tgtInput)
    {
        update();
    }

    void update()
    {
        // TODO: implement
        if (source && target) // Kiemer tra nullptr
            target->setInput(targetInput, source->getOutput());
    }
};
```

```
Main.cpp
#include "fullAdder.hpp"

void testFullAdder()
{
    FullAdder fa;

    struct TestCase
    {
        bool a, b, cin, expectedSum, expectedCout;
    };

    TestCase tests[] = {
        {0, 0, 0, 0, 0},
        {0, 0, 1, 1, 0},
        {0, 1, 0, 1, 0},
        {0, 1, 1, 0, 1},
        {1, 0, 0, 1, 0},
        {1, 0, 1, 0, 1},
        {1, 1, 0, 0, 1},
        {1, 1, 1, 1, 1}};
```

```
    std::cout << "Running full adder tests...\n";
    for (int i = 0; i < 8; i++)
    {
        TestCase &test = tests[i];
        fa.setInputs(test.a, test.b, test.cin);

        bool sum = fa.getSum();
        bool cout = fa.getCarryOut();

        std::cout << "Test " << i + 1 << ": "
                  << "A=" << test.a << " B=" << test.b << " Cin=" << test.cin
                  << " | Sum=" << sum << " Cout=" << cout
                  << " | Expected Sum=" << test.expectedSum
                  << " Cout=" << test.expectedCout;

        if (sum == test.expectedSum && cout == test.expectedCout)
        {
            std::cout << " ✓ PASS\n";
        }
        else
        {
            std::cout << " ✗ FAIL\n";
        }
    }
}

int main()
{
    testFullAdder();
    return 0;
}
```