VÕ TIẾN
Thảo luận kiến thức CNTT trường BK về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

Kỹ Thuật Lập Trình (OOP trong c++)

KTLT2 - HK242

TASK 7 Nền móng OOP - Xây dựng thế giới đối tượng

Thảo luận kiến thức CNTT trường BK
về KHMT(CScience), KTMT(CEngineering)
https://www.facebook.com/groups/khmt.ktmt.cse.bku

TP. HỒ CHÍ MINH, THÁNG 3/2025

1 Trắc Nghiệm

1. Class trong C++ là gì?

   a) Một khuôn mẫu để tạo đối tượng.
   b) Một biến lưu trữ dữ liệu.
   c) Một kiểu dữ liệu nguyên thủy.
   d) Một từ khóa trong C++.

2. Đối tượng (Object) trong C++ là gì?

   a) Một thực thể cụ thể của class.
   b) Một kiểu dữ liệu nguyên thủy.
   c) Một biến tĩnh trong class.
   d) Một thư viện trong C++.

3. Tại sao sử dụng class trong OOP?

   a) Để đóng gói dữ liệu.
   b) Để giảm số dòng code.
   c) Để tăng tốc độ thực thi.
   d) Để làm chương trình ngắn hơn.

4. Một class trong C++ được khai báo bằng từ khóa nào?

   a) class
   b) struct
   c) object
   d) define

5. Cách nào sau đây KHÔNG đúng khi tạo một đối tượng từ class?

   a) MyClass obj;
   b) MyClass *ptr = new MyClass();
   c) MyClass obj();
   d) MyClass obj = MyClass();

6. Trong C++, một class có thể chứa gì?

   a) Thuộc tính và phương thức.
   b) Chỉ thuộc tính.
   c) Chỉ phương thức.
   d) Chỉ biến toàn cục.

7. Từ khóa nào trong C++ dùng để kế thừa một class?

   a) extends
   b) inherit
   c) public/private/protected
   d) override

8. Khi nào một object của class được tạo?

   a) Khi gọi constructor.
   b) Khi khai báo biến toàn cục.
   c) Khi biên dịch chương trình.
   d) Khi chương trình chạy xong.

9. Một class có thể có bao nhiêu đối tượng?

   a) Không giới hạn.
   b) Chỉ một đối tượng.
   c) Tối đa 100 đối tượng.
   d) Tùy thuộc vào trình biên dịch.

10. Class và struct khác nhau như thế nào trong C++?

    a) Class mặc định private, struct mặc định public.
    b) Không có sự khác biệt.
    c) Struct không thể có phương thức.
    d) Class không thể chứa biến tĩnh.

11. Thuộc tính (Attribute) là gì trong C++?

    a) Biến thành viên của class.
    b) Một kiểu dữ liệu đặc biệt.
    c) Một hàm thành viên của class.
    d) Một đối tượng trong C++.

12. Phương thức (Method) trong C++ là gì?

    a) Hàm thành viên của class.
    b) Một biến trong class.
    c) Một kiểu dữ liệu mới.
    d) Một thư viện trong C++.

13. Cách nào đúng để truy cập một thuộc tính trong class?

a) Sử dụng dấu chấm '.'.
b) Sử dụng dấu ':'.
c) Sử dụng 'new' để truy cập.
d) Sử dụng toán tử '->' cho biến không phải con trỏ.

14. Phạm vi truy cập nào giúp thuộc tính chỉ có thể được truy cập từ trong class?

a) private
b) public
c) protected
d) global

15. Có thể gọi phương thức của class như thế nào?

a) Sử dụng dấu chấm '.' hoặc toán tử '->'.
b) Sử dụng 'include'.
c) Dùng 'define' để khai báo phương thức.
d) Dùng 'virtual' để gọi phương thức.

16. Khi nào nên sử dụng phương thức 'const' trong class?

a) Khi phương thức không thay đổi giá trị của thuộc tính.
b) Khi phương thức thay đổi giá trị của thuộc tính.
c) Khi muốn tăng tốc độ thực thi.
d) Khi muốn tạo đối tượng mới từ class.

17. Con trỏ 'this' trong C++ là gì?

a) Con trỏ trỏ đến đối tượng hiện tại.
b) Một kiểu dữ liệu trong C++.
c) Một hàm trong C++.
d) Một từ khóa dùng để khai báo biến.

18. Khi nào nên sử dụng con trỏ 'this' trong phương thức?

a) Khi muốn trả về chính đối tượng hiện tại.
b) Khi muốn gọi phương thức tĩnh.
c) Khi muốn truy cập biến toàn cục.
d) Khi muốn cấp phát động bộ nhớ.

19. Hàm tạo (Constructor) trong C++ là gì?

a) Một phương thức đặc biệt được gọi khi tạo đối tượng.
b) Một phương thức hủy đối tượng.
c) Một phương thức có kiểu trả về 'void'.
d) Một biến thành viên của class.

20. Hàm hủy (Destructor) trong C++ là gì?

a) Một phương thức đặc biệt được gọi khi đối tượng bị hủy.
b) Một phương thức tạo đối tượng.
c) Một kiểu dữ liệu mới trong C++.
d) Một toán tử trong C++.

21. Hàm tạo có thể có tham số không?

a) Có, để khởi tạo giá trị ban đầu cho đối tượng.
b) Không, hàm tạo luôn không có tham số.
c) Chỉ có khi sử dụng 'static'.
d) Chỉ khi dùng với 'virtual'.

22. Khi nào hàm hủy được gọi?

a) Khi đối tượng đi ra khỏi phạm vi sử dụng.
b) Khi đối tượng được tạo ra.
c) Khi gọi phương thức tĩnh.
d) Khi sử dụng 'new' để cấp phát bộ nhớ.

23. Hàm tạo sao chép (Copy Constructor) dùng để làm gì?

a) Sao chép giá trị từ một đối tượng khác cùng class.
b) Xóa đối tượng khỏi bộ nhớ.
c) Gán giá trị 'nullptr' cho con trỏ.
d) Kiểm tra kiểu dữ liệu của đối tượng.

24. Hàm hủy có thể có tham số không?

a) Không, hàm hủy không có tham số.
b) Có, nếu sử dụng từ khóa 'virtual'.
c) Có, nếu sử dụng 'static'.
d) Chỉ trong trường hợp sử dụng kế thừa.

25. Điều gì xảy ra nếu không định nghĩa hàm hủy?

a) Trình biên dịch sẽ tự động tạo một hàm hủy mặc định.
b) Chương trình sẽ không thể chạy.
c) Biến toàn cục sẽ bị mất giá trị.
d) Các phương thức của class sẽ không thể gọi được.

26. Tại sao cần thiết kế đối tượng trước khi lập trình?

a) Giúp tổ chức và quản lý mã nguồn tốt hơn.
b) Giảm thời gian biên dịch chương trình.
c) Tăng tốc độ chạy của chương trình.
d) Giúp chương trình chạy mà không cần biên dịch.

27. Khi thiết kế đối tượng, cần xác định những yếu tố nào?

a) Thuộc tính và phương thức của đối tượng.
b) Chỉ thuộc tính, không cần phương thức.
c) Chỉ phương thức, không cần thuộc tính.
d) Không cần xác định trước, có thể bổ sung sau.

28. Đối tượng nào sau đây là một ví dụ tốt về thiết kế OOP?

a) Lớp ‘Car‘ với các thuộc tính ‘brand‘, ‘speed‘ và phương thức ‘drive()‘.
b) Một biến nguyên ‘int‘ để lưu trữ số xe.
c) Một mảng chứa danh sách số xe.
d) Một hằng số ‘define MAX_SPEED 200‘.

29. Điều gì xảy ra nếu một lớp không có constructor?

a) Trình biên dịch tự động tạo một constructor mặc định.
b) Lớp sẽ không thể tạo đối tượng.
c) Chương trình sẽ báo lỗi khi biên dịch.
d) Phải định nghĩa constructor bằng tay.

2 Đọc Code

Câu 1. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Demo {
public:
    Demo() {
        cout << "Constructor called!" <<
        ↪ endl;
    }

    ~Demo() {
        cout << "Destructor called!" << endl;
    }
};

Demo obj1;  // Đối tượng trên stack
Demo* obj2 = new Demo();  // Đối tượng trên
↪ heap

delete obj2;  // Giải phóng bộ nhớ
cout << "End of program" << endl;
```

Câu 2. Kết quả của chương trình

Kết quả và giải thích: ...

-
-

```cpp
class Sample {
public:
    int x;

    Sample(int val) {
        x = val;
    }

    void display() {
        cout << "Value of x: " << x << endl;
    }
};

Sample obj(10);  // Tạo đối tượng trên stack
Sample* ptr = new Sample(20);  // Tạo đối
↪ tượng trên heap

cout << "Access using object: " << obj.x <<
↪ endl;  // Truy cập thông thường
obj.display();

cout << "Access using pointer: " << ptr->x <<
↪ endl;  // Truy cập qua con trỏ
ptr->display();

delete ptr;  // Giải phóng bộ nhớ
```

Câu 3. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Sample {
public:
    int x, y;

    // Constructor không tham số
    Sample() {
        x = 0; y = 0;
        cout << "Default" << endl;
    }

    // Constructor có một tham số
    Sample(int val) {
        x = val; y = 0;
        cout << "Single" << endl;
    }

    // Constructor có hai tham số
    Sample(int val1, int val2) {
        x = val1; y = val2;
        cout << "Two" << endl;
    }

    void display() {
        cout << "x = " << x << ", y = " << y
        ↪ << endl;
    }
};

Sample obj1;
Sample obj2(10);
Sample obj3(5, 15);

obj1.display();
obj2.display();
obj3.display();
```

Câu 4. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

```cpp
class ShallowCopy {
public:
    int* data;

    ShallowCopy(int val) {
        data = new int(val);
    }

    // Constructor sao chép cạn (chỉ sao chép địa chỉ)
    ShallowCopy(const ShallowCopy& other) {
        data = other.data;
    }

    ~ShallowCopy() {
        delete data;  // Giải phóng bộ nhớ
    }
};

ShallowCopy obj1(10);
ShallowCopy obj2 = obj1;  // Sao chép cạn

cout << "Before modification:" << endl;
cout << "obj1 = " << *obj1.data << endl;
cout << "obj2 = " << *obj2.data << endl;

*obj2.data = 99;

cout << "After modification:" << endl;
cout << "obj1 = " << *obj1.data << endl;
cout << "obj2 = " << *obj2.data << endl;
```

Câu 5. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

```cpp
class DeepCopy {
public:
    int* data;

    DeepCopy(int val) {
        data = new int(val);
    }

    // Constructor sao chép sâu (cấp phát bộ nhớ mới)
    DeepCopy(const DeepCopy& other) {
        data = new int(*other.data);
    }

    ~DeepCopy() {
        delete data;  // Giải phóng bộ nhớ
    }
};

DeepCopy obj1(20);
DeepCopy obj2 = obj1;  // Sao chép sâu

cout << "Before modification:" << endl;
cout << "obj1 = " << *obj1.data << endl;
cout << "obj2 = " << *obj2.data << endl;

*obj2.data = 88;

cout << "After modification:" << endl;
cout << "obj1 = " << *obj1.data << endl;
cout << "obj2 = " << *obj2.data << endl;
```

Câu 6. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

```cpp
class Test {
    int x; // Biến thành viên

public:
    Test(int x) { // Tham số trùng tên biến thành viên
        x = x;
    }

    void display() {
        cout << "Value of x: " << x << endl;
    }
};

Test obj(10);
obj.display();
```

Câu 7. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

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

class Number {
    int value;

public:
    Number(int v) : value(v < 0 ? 0 : v) {
        cout << "Constructor called with value: " << value << endl;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

Number num1(10);
Number num2(-5);

num1.display();
num2.display();
```

Câu 8. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

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

public:
    // Constructor khởi tạo mảng động
    Array(int n) : size(n) {
        data = new int[size]();
    }

    // Destructor giải phóng bộ nhớ
    ~Array() {
        delete[] data;
    }

    // Gán giá trị cho phần tử mảng
    void setValue(int index, int value) {
        if (index >= 0 && index < size) {
            data[index] = value;
        }
    }

    // Lấy giá trị của phần tử mảng
    int getValue(int index) const {
        return (index >= 0 && index < size) ?
        ↪ data[index] : -1;
    }

    // Hiển thị toàn bộ mảng
    void display() const {
        for (int i = 0; i < size; i++) {
            cout << data[i] << " ";
        }
        cout << endl;
    }
};
Array arr(5);  // Khởi tạo mảng có 5 phần tử
arr.setValue(0, 10);
arr.setValue(1, 20);
arr.setValue(2, 30);

cout << "Array contents: ";
arr.display();
cout << "Element at index 1: " << arr.getValue(1)
↪ << endl;
```

Câu 9. Kết quả của chương trình

Kết quả và giải thích: ...

- ...
- ...

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

public:
    // Constructor khởi tạo ma trận động
    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]();
        }
    }
    // Destructor giải phóng bộ nhớ
    ~Matrix() {
        for (int i = 0; i < rows; i++) {
            delete[] data[i];
        }
        delete[] data;
    }
    // Gán giá trị vào ma trận
    void setValue(int r, int c, int value) {
        if (r >= 0 && r < rows && c >= 0 && c < cols) {
            data[r][c] = value;
        }
    }
    // Lấy giá trị từ ma trận
    int getValue(int r, int c) const {
        return (r >= 0 && r < rows && c >= 0 && c < cols) ? data[r][c] : -1;
    }
    // Hiển thị toàn bộ ma trận
    void display() const {
        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }
};

Matrix mat(3, 3);  // Khởi tạo ma trận 3x3

mat.setValue(0, 0, 1);
mat.setValue(1, 1, 5);
mat.setValue(2, 2, 9);

cout << "Matrix contents:" << endl;
mat.display();

cout << "Element at (1,1): " << mat.getValue(1, 1) << endl;
```

Câu 11. Kết quả của chương trình

```
class Sample {
private:
    int value;

public:
    Sample(int v) : value(v) {}

    // Trả về con trỏ đến đối tượng hiện tại
    Sample* returnThis() {
        value += 10;
        return this;
    }

    // Trả về một bản sao của đối tượng
    Sample returnByValue() {
        value += 20;
        return *this;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

Sample obj(5);

cout << "Using returnThis(): " << endl;
obj.returnThis()->display();

cout << "Using returnByValue(): " << endl;
obj.returnByValue().display();

cout << "Final value of obj: " << endl;
obj.display();
```

Kết quả và giải thích: ...

- ...
- ...

3 Bài Tập

Câu 1: Thiết kế lớp Array với nhiều chức năng

Đề bài:

Thiết kế một lớp Array trong C++ với các chức năng sau:

- **Constructor**:
  - Array(int size): Khởi tạo mảng với kích thước size.
  - Array(int* data, int size): Khởi tạo mảng từ dữ liệu có sẵn.
  - Array(const Array& other): Constructor sao chép từ một đối tượng Array khác.
- **Phương thức**:
  - int index(int i): Trả về phần tử tại vị trí i.
  - void reverse(): Đảo ngược mảng.
  - int maxElement(): Tìm phần tử lớn nhất trong mảng.
  - int maxThreeSum(): Tìm tổng lớn nhất của ba phần tử liên tiếp.

```cpp
#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) {
        // TODO
    }

    // Constructor 2: Khởi tạo mảng từ dữ liệu có sẵn
    Array(int* arr, int size) {
        // TODO
    }

    // Constructor 3: Constructor sao chép
    Array(const Array& other) : size(other.size) {
        // TODO
    }

    // Destructor: Giải phóng bộ nhớ
    ~Array() {
        // TODO
    }

    // Lấy giá trị tại index
    int index(int i) {
        // TODO
    }

    // Đảo ngược mảng
```

```
36      void reverse() {
37          // TODO
38      }
39
40      // Tìm phần tử lớn nhất trong mảng
41      int maxElement() {
42          // TODO
43      }
44
45      // Tìm tổng lớn nhất của ba số liên tiếp
46      int maxThreeSum() {
47          // TODO
48      }
49
50      // Hiển thị mảng
51      void display() {
52          for (int i = 0; i < size; i++) {
53              cout << data[i] << " ";
54          }
55          cout << endl;
56      }
57  };
58
59  int main() {
60      int arr[] = {1, 2, 3, 4, 5, 6};
61      Array array1(6);
62      Array array2(arr, 6);  // Khởi tạo từ mảng có sẵn
63      Array array3 = array2; // Constructor sao chép
64
65      cout << "Original array: ";
66      array2.display();
67
68      array2.reverse();
69      cout << "Reversed array: ";
70      array2.display();
71
72      cout << "Max element: " << array2.maxElement() << endl;
73      cout << "Max sum of 3 consecutive elements: " << array2.maxThreeSum() << endl;
74
75      return 0;
76  }
```

Test case

Input	Function	Output
{1, 2, 3, 4, 5, 6}	display()	1 2 3 4 5 6
{1, 2, 3, 4, 5, 6}	reverse()	6 5 4 3 2 1
{1, 2, 3, 4, 5, 6}	maxElement()	6
{1, 2, 3, 4, 5, 6}	maxThreeSum()	15

Câu 4: Thiết kế lớp Matrix với phép cộng và nhân ma trận

Đề bài:

Thiết kế một lớp Matrix trong C++ để thực hiện các thao tác trên ma trận kích thước rows x cols. Lớp này cần hỗ trợ:

- **Constructor**:
  - Matrix(int rows, int cols): Khởi tạo ma trận có rows hàng và cols cột với giá trị mặc định là 0.
  - Matrix(int** data, int rows, int cols): Khởi tạo ma trận từ dữ liệu có sẵn.
  - Matrix(const Matrix& other): Constructor sao chép từ một đối tượng Matrix khác.
- **Phương thức**:
  - Matrix add(const Matrix& other): Cộng hai ma trận cùng kích thước.
  - Matrix multiply(const Matrix& other): Nhân hai ma trận có kích thước hợp lệ.
  - void display(): Hiển thị ma trận.

```cpp
#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) {
        // TODO
    }

    // Constructor 2: Khởi tạo từ dữ liệu có sẵn
    Matrix(int** arr, int r, int c) {
        // TODO
    }

    // Constructor 3: Constructor sao chép
    Matrix(const Matrix& other) {
        // TODO
    }

    // 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) {
        // TODO
    }

    // Phép nhân hai ma trận (số cột của ma trận 1 phải bằng số hàng của ma trận 2)
```

```
39      Matrix multiply(const Matrix& other) {
40          // TODO
41      }
42
43      // Hiển thị ma trận
44      void display() {
45          // TODO
46      }
47  };
48
49  int main() {
50      int r1 = 2, c1 = 3;
51      int r2 = 3, c2 = 2;
52
53      int** arr1 = new int*[r1];
54      int** arr2 = new int*[r2];
55
56      for (int i = 0; i < r1; i++) {
57          arr1[i] = new int[c1]{i + 1, i + 2, i + 3};
58      }
59
60      for (int i = 0; i < r2; i++) {
61          arr2[i] = new int[c2]{i + 1, i + 2};
62      }
63
64      Matrix mat1(arr1, r1, c1);
65      Matrix mat2(arr2, r2, c2);
66
67      cout << "Matrix 1:" << endl;
68      mat1.display();
69
70      cout << "Matrix 2:" << endl;
71      mat2.display();
72
73      Matrix product = mat1.multiply(mat2);
74      cout << "Product of Matrices:" << endl;
75      product.display();
76
77      return 0;
78  }
```

Test case

Matrix 1	Matrix 2	Product
$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}$	$\begin{bmatrix} 22 & 28 \\ 49 & 64 \end{bmatrix}$

Câu 3: Thiết kế hệ thống quản lý cổng logic

Đề bài:

Thiết kế một lớp LogicGate để mô phỏng các cổng logic cơ bản (AND, OR, XOR, NAND, NOR, XNOR, NOT). Không sử dụng kế thừa, chỉ dùng enum để quản lý loại cổng logic.

- **Enum GateType**:
  - AND_G, OR_G, XOR_G, NAND_G, NOR_G, XNOR_G, NOT_G.
- **Lớp LogicGate**:
  - LogicGate(GateType type): Khởi tạo cổng logic với loại tương ứng.
  - bool evaluate(bool a, bool b = false): Tính toán đầu ra của cổng logic (cổng NOT chỉ sử dụng a).
  - void display(bool a, bool b = false): Hiển thị kết quả đầu ra.

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

class LogicGate {
    // TODO
};
```

Test Case

```
LogicGate andGate(AND_G);
LogicGate orGate(OR_G);
LogicGate notGate(NOT_G);

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

Kết quả

```
Gate: 0 | Input: 1 0 | Output: 0
Gate: 1 | Input: 1 0 | Output: 1
Gate: 6 | Input: 1 0 | Output: 0
```

Câu 4: Thiết kế hệ thống mô phỏng vi mạch

Đề bài:

Thiết kế một hệ thống ‘Circuit‘ mô phỏng hoạt động của các cổng logic cơ bản (‘AND‘, ‘OR‘, ‘XOR‘, ‘NAND‘, ‘NOR‘, ‘XNOR‘, ‘NOT‘). Mỗi cổng logic có thể nhận tín hiệu từ nhiều cổng khác và tạo ra đầu ra kết nối với cổng khác.

1. Enum `GateType`
   - Xác định loại cổng logic: `AND`, `OR`, `XOR`, `NAND`, `NOR`, `XNOR`, `NOT`, `INPUT`.
2. Lớp `LogicGate`

   Thuộc tính:
   - `type`: Loại cổng logic.
   - `inputs`: Danh sách đầu vào (1 input cho `NOT` và `INPUT`, 2 input cho các cổng khác).
   - `output`: Giá trị đầu ra.

   Hàm `calculateOutput()` (private)
   - Dựa vào `type`, tính giá trị đầu ra của cổng logic.

   Hàm khởi tạo `LogicGate(GateType gateType)`
   - Xác định số lượng đầu vào dựa trên `type`.

   Hàm `getOutput()`
   - Lấy giá trị đầu ra.

   Hàm `setInput(size_t index, bool value)`
   - Cập nhật đầu vào và tính lại đầu ra.

   Hàm `getInputCount()`
   - Trả về số lượng đầu vào của cổng logic.

```cpp
LogicGate gate(GateType::AND);
gate.setInput(0, true);
gate.setInput(1, false);
std::cout << "Output: " << gate.getOutput() << std::endl; // Output: 0
```

3. Lớp `Wire`

   Lớp `Wire` mô phỏng một dây kết nối giữa hai cổng logic (`LogicGate`), giúp truyền tín hiệu từ cổng nguồn (`source`) đến một đầu vào cụ thể của cổng đích (`target`).

   Các thuộc tính
   - `source`: Con trỏ đến cổng logic nguồn.
   - `target`: Con trỏ đến cổng logic đích.
   - `targetInput`: Vị trí đầu vào của cổng đích cần kết nối.

   Hàm khởi tạo `Wire(LogicGate *src, LogicGate *tgt, size_t tgtInput)`
   - Nhận tham chiếu đến cổng nguồn (`src`), cổng đích (`tgt`), và vị trí đầu vào cần cập nhật (`tgtInput`).
   - Gọi `update()` ngay lập tức để đồng bộ giá trị từ nguồn đến đích.

   Hàm `update()`

- Lấy giá trị đầu ra của `source`.
- Gán giá trị đó vào vị trí `targetInput` của `target`.

```
LogicGate gate1(GateType::INPUT);
LogicGate gate2(GateType::NOT);

gate1.setInput(0, true); // Đặt giá trị đầu vào cho gate1

Wire connection(&gate1, &gate2, 0); // Nối đầu ra của gate1 với đầu vào của gate2

std::cout << "Output of gate2: " << gate2.getOutput() << std::endl; // Output: 0 (NOT true)
```

4. lớp `Circuit`

   Lớp `Circuit` quản lý một tập hợp các cổng logic (`LogicGate`) và các dây nối (`Wire`). Lớp này hỗ trợ việc thêm cổng, kết nối chúng và cập nhật trạng thái của toàn bộ mạch.

   Các thuộc tính

   - `LogicGates`: Danh sách các cổng logic trong mạch.
   - `wires`: Danh sách các dây kết nối giữa các cổng logic.

   Destructor `~Circuit()`

   - **Giải phóng bộ nhớ cấp phát động cho tất cả các cổng logic và dây nối.**

   Hàm `addGate()` `LogicGate *addGate(GateType type)`

   - Tạo một cổng logic mới với kiểu `type`.
   - Thêm vào danh sách `LogicGates`.
   - Trả về con trỏ đến cổng logic mới tạo.

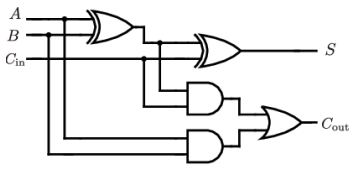
   Hàm `connect()` `void connect(LogicGate *from, LogicGate *to, size_t toInput)`

   - Tạo một dây nối từ cổng `from` đến cổng `to` với đầu vào `toInput`.
   - Thêm dây nối vào danh sách `wires`.

   Hàm `update()` `void update()`

   - Cập nhật tất cả các dây nối trong danh sách bằng cách gọi phương thức `update()` của từng `Wire`.

```
Circuit circuit;

LogicGate *inputGate = circuit.addGate(GateType::INPUT);
LogicGate *notGate = circuit.addGate(GateType::NOT);

inputGate->setInput(0, true); // Đặt giá trị đầu vào của inputGate

circuit.connect(inputGate, notGate, 0); // Kết nối đầu ra của inputGate với NOT

circuit.update(); // Cập nhật mạch

std::cout << "Output of NOT gate: " << notGate->getOutput() << std::endl; // Output: 0 (NOT true)
```



Inputs			Outputs	
A	B	C_{in}	S	C_{out}
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

Test Case full adder

```
class FullAdder
{
private:
    Circuit circuit; // Một mạch logic chứa các cổng cần thiết để thực hiện phép
     ↪  cộng.
    LogicGate *inputA, *inputB, *inputCin; //  Ba cổng đầu vào của bộ cộng đầy đủ.
    LogicGate *sumOutput; // Cổng đầu ra biểu diễn tổng (Sum).
    LogicGate *carryOutput; // Cổng đầu ra biểu diễn bit nhớ (CarryOut).

public:
    FullAdder()
    {
        // TODO Hàm khởi tạo sẽ tạo và kết nối các cổng logic để mô phỏng hoạt động
         ↪  của bộ cộng đầy đủ.
        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(); }
};
```

SRC drive của lý thuyết