Trạng thái	Đã xong
Bắt đầu vào lúc	Thứ Tư, 7 tháng 5 2025, 3:08 AM
Kết thúc lúc	Thứ Tư, 7 tháng 5 2025, 3:34 AM
Thời gian thực hiện	25 phút 53 giây

Câu hỏi 1

Đúng

Cho định nghĩa class:

```
class clockType
{
   public:
      void setTime(int, int, int);
      bool equalTime(const clockType&) const;
      clockType(int, int, int);
      clockType();
```

```
   private:
      int hr;
      int min;
      int sec;
};
```

Hãy xem xét câu lệnh:

```
if (myClock.equalTime(yourClock))
cout << "Both times are equal." << endl;
else
cout << "The two times are not equal." << endl;
```

Biểu thức kiểm tra điều kiện if-else so sánh myClock và yourClock. Nếu thời gian của 2 clock gồm (hr, min, sec) là giống nhau thì trả về **true**, ngược lại sẽ trả về **false**.

Yêu cầu: SV hiện thực phương thức **equalTime** để thực hiện như mô tả trên.

[English]

Given the class definition:

```
class clockType
{
   public:
      void setTime(int, int, int);
      bool equalTime(const clockType&) const;
      clockType(int, int, int);
      clockType();
```

```
   private:
      int hr;
      int min;
      int sec;
};
```

Consider the commands:

```
if (myClock.equalTime(yourClock))
cout << "Both times are equal." << endl;
else
cout << "The two times are not equal." << endl;
```

The if-else conditional expression compares myClock and yourClock object. If the time of 2 clocks including (hr, min, sec) is the same, return **true**, otherwise, return **false**.

Requirement: Student implements the ***equalTime*** method for it to do as described above.

For example:

Test	Result

ClockType yourClock;
ClockType myClock(5,2,30);
if (myClock.equalTime(yourClock))
cout << "Both times are equal."<< endl;
else
cout << "The two times are not equal." << endl;
	The two times are not equal.

Answer: (penalty regime: 0 %)

Reset answer

```
class ClockType
{
public:
    void setTime(int, int, int);
    bool equalTime(const ClockType&) const;
    ClockType(int, int, int);
    ClockType();

private:
    int hr;
    int min;
    int sec;
};

// TODO
bool ClockType::equalTime(const ClockType& otherClock) const
{
    // So sánh các biến thành viên hr, min, sec của đối tượng hiện tại
    // với các biến tương ứng của đối tượng otherClock
    return (hr == otherClock.hr &&
            min == otherClock.min &&
            sec == otherClock.sec);
}
```

	Test	Expected	Got	

✓	ClockType yourClock;
	ClockType myClock(5,2,30);
	if (myClock.equalTime(yourClock))
	cout << "Both times are equal."<< endl;
	else
	cout << "The two times are not equal." << endl;
		The two times are not equal.	The two times are not equal.	✓

✓	ClockType yourClock;
	ClockType myClock(5,2,30);
	yourClock.setTime(5,2,30);
	if (myClock.equalTime(yourClock))
	cout << "Both times are equal."<< endl;
	else
	cout << "The two times are not equal." << endl;
		Both times are equal.	Both times are equal.	✓

Passed all tests! ✓

▸ Show/hide question author's solution (Cpp)

Câu hỏi **2**

Đúng

Cho class ClockType như sau:

```
class ClockType
{
   public:
      void printTime() const;
      void incrementSeconds();
      void incrementMinutes();
      void incrementHours();
      clockType(int, int, int);
      clockType();
```

```
   private:
      int hr;
      int min;
      int sec;
};
```

Hãy xem xét câu lệnh:

```
myClock.incrementSeconds();
```

Câu lệnh này tăng biến sec thêm 1 đơn vị.

Hãy hiện thực 3 phương thức

- incrementHours: tăng hr thêm 1 đơn vị. Sau khi tăng, nếu hr = 24 thì ta đặt lại hr = 0.
- incrementMinutes: tăng min thêm 1 đơn vị. Sau khi tăng, nếu min = 60 thì ta đặt lại min = 0 và tăng hr thêm 1 đơn vị bằng cách phù hợp.
- incrementSeconds: tăng sec thêm 1 đơn vị. Sau khi tăng, nếu sec = 60 thì ta đặt lại sec = 0 và tăng min thêm 1 đơn vị bằng cách phù hợp.

[English]
Given the class ClockType as following:

```
class ClockType
{
   public:
      void printTime() const;
      void incrementSeconds();
      void incrementMinutes();
      void incrementHours();
      clockType(int, int, int);
      clockType();
```

```
   private:
      int hr;
      int min;
      int sec;
};
```

Consider the statement:

```
myClock.incrementSeconds();
```

This statement increments the variable sec by 1 unit.

Let's implement 3 methods

- incrementHours: increase hr by 1 unit. After incrementing, if hr = 24 then we reset hr = 0.
- incrementMinutes: increase min by 1 unit. After increment, if min = 60 then we reset min = 0 and increase hr by 1 unit accordingly.
- incrementSeconds: Increases sec by 1 unit. After increment, if sec = 60 then we reset sec = 0 and increase min by 1 unit accordingly.

For example:

Test	Result

ClockType myClock(5,2,30);
myClock.incrementSeconds();
myClock.printTime();
	05:02:31

Answer: (penalty regime: 0 %)

Reset answer

```
class ClockType
{
   public:
      void setTime(int, int, int);
      void printTime() const;
      void incrementSeconds();
      void incrementMinutes();
      void incrementHours();
      ClockType(int, int, int);
      ClockType();

   private:
      int hr;
      int min;
      int sec;
};
void ClockType::printTime() const
{
   if (hr < 10)
      cout << "0";
   cout << hr << ":";
   if (min < 10)
      cout << "0";
   cout << min << ":";
   if (sec < 10)
      cout << "0";
   cout << sec;
}
// TODO
void ClockType::incrementHours() {
    hr++;
    if (hr == 24)
        hr = 0;
}
void ClockType::incrementMinutes() {
    min++;
    if (min == 60) {
        min = 0;
        incrementHours(); // Call incrementHours to handle the hour update properly
    }
}
void ClockType::incrementSeconds() {
    sec++;
    if (sec == 60) {
        sec = 0;
        incrementMinutes(); // Call incrementMinutes to handle the minute update properly
    }
}

```

	Test	Expected	Got	

✓	ClockType myClock(5,2,30);
	myClock.incrementSeconds();
	myClock.printTime();
		05:02:31	05:02:31	✓

	Test	Expected	Got	

✓	`ClockType myClock(5,59,59);`
	`myClock.incrementSeconds();`
	`myClock.printTime();`
		`06:00:00`	`06:00:00`	✓

Passed all tests! ✓

Câu hỏi **3**

Đúng

Cho class như sau:

```
class Book {
public:
    Book(const char*);
    ~Book();
    void display();
private:
    char* name;
};
```

- Hiện thực **constructor** và **destructor** của Book sao cho chương trình sau không bị lỗi khi chạy:

```
Book * pBook = new Book("To Kill a Mockingbird");
pBook->display();
delete pBook;
```

- Kết quả in ra:

```
Book: To Kill a Mockingbird
```

[English]

Given the class definition:

```
class Book {
public:
     Book(const char*);
     ~Book();
     void display();
private:
     char* name;
};
```

- Implement Book's **constructor** and **destructor** so that the following program does not crash at runtime:

```
Book * pBook = new Book("To Kill a Mockingbird");
pBook->display();
delete pBook;
```

- Output:

```
Book: To Kill a Mockingbird
```

For example:

Test	Input	Result
1	To Kill a Mockingbird	Book: To Kill a Mockingbird Deleted

Answer: (penalty regime: 0 %)

Reset answer

```
1  // string.h is included
2
3  class Book {
```

```
    char* name;
public:
    Book(const char*);
    ~Book();

    void display();
    char* getName();
};

void Book::display() {
    cout << "Book: " << this->name << endl;
}

char* Book::getName() {
    return this->name;
}


// TODO: Book and ~Book
Book::Book(const char* inputName) {
    // Cấp phát bộ nhớ để lưu chuỗi name
    name = new char[strlen(inputName) + 1]; // +1 để chứa ký tự null '\0'
    strcpy(name, inputName); // Sao chép chuỗi vào vùng nhớ
}

Book::~Book() {
    delete[] name; // Giải phóng bộ nhớ đã cấp phát
}

```

	Test	Input	Expected	Got	
✓	1	To Kill a Mockingbird	Book: To Kill a Mockingbird Deleted	Book: To Kill a Mockingbird Deleted	✓

Passed all tests! ✓

Câu hỏi **4**

Đúng

Thực hiện các yêu cầu sau:

- Xây dựng một class **Integer** gồm một thành phần private tên là **val** là một số nguyên kiểu int.
- Hiện thực constructor: **Integer(int)**.
- Hiện thực constructor: **Integer(Integer*)**.
- Quá tải toán tử **operator +** để phép tính **Integer(2) + Integer(3)** trả về **Integer(5)**.
- Quá tải toán tử **operator +** để phép tính **Integer(3) + 2** trả về **Integer(5)**.

Chú ý: *khai báo phương thức print() trong định nghĩa của class và không cần hiện thực.*

[English]

Implement following requirements:

- Define class **Integer** that contains one attribute named **val** of type **int**.
- Implement constructor: **Integer(int)**: assign value of parameter to attribute val.
- Implement constructor: **Integer(Integer*)**.
- Overload **operator +** so that: **Integer(2) + Integer(3)** returns **Integer(5)**.
- Overload **operator +** so that: **Integer(3) + 2** returns **Integer(5)**.

Note*: declare method* ***print()*** *in class's definition and* ***do not*** *provide its implementation.*

For example:

Test	Input	Result
1	1 2 3	1 2 10

Answer: (penalty regime: 0 %)

Reset answer

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

// TODO
class Integer {
private:
    int val;

public:
    // Constructor nhận int
    Integer(int v) : val(v) {}

    // Constructor nhận con trỏ Integer*
    Integer(Integer* other) : val(other->val) {}

    // Quá tải toán tử + để Integer + Integer
    Integer operator+(const Integer& other) const {
        return Integer(this->val + other.val);
    }

    // Quá tải toán tử + để Integer + int
    Integer operator+(int x) const {
        return Integer(this->val + x);
    }

    // Khai báo print
    void print();
};

void Integer::print() {
    cout << this->val << endl;
}

int main() {
    int x, y, z;
    cin >> x >> y >> z;
```

```
    Integer a(x);
    Integer b(y);
    Integer* t = new Integer(z);
    Integer c(t);

    a.print(); b.print();
    (a + b + c + 4).print();

    delete t;
    return 0;
}
```

	Test	Input	Expected	Got	
✓	1	1 2 3	1 2 10	1 2 10	✓

Passed all tests! ✓

Câu hỏi **5**

Đúng

Cho class template Array như bên dưới dùng để chứa mảng 1 chiều được khai báo trong vùng nhớ Heap.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
```

```
    Array(const Array<T>& other);
private:
    int size;
    T * p;
};
```

Trong class Array có khai báo các thuộc tính sau:

- Thuộc tính **p** là con trỏ trỏ đến vùng nhớ Heap được cấp phát.
- Thuộc tính **size** của Array chứa số lượng phần tử của mảng.

Yêu cầu: SV hiện thực phương thức Copy Constructor (được tô màu đỏ), phương thức này khởi tạo một đối tương Array mới dựa trên một đối tượng Array khác.

[English]

Given class template **Array** as below that contains an array allocated in Heap memory.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
```

```
    Array(const Array<T>& other);
private:
    int size;
    T * p;
};
```

There are 2 attributes in class Array:

- **p**: a pointer contains the address of first element of allocated memory.
- **size**: number of elements of array.

Requirement:

- Implements the Copy Constructor method (highlighted in red), which initializes a new Array object based on other Array object.

For example:

Test	Result

`Array<int> * a3;`
`Array<int> * a4 = new Array<int>(10, 3);`
`a3 = new Array<int>(*a4);`
`delete a4;`
`a3->print();`
`delete a3;`
	3 3 3 3 3 3 3 3 3 3

Answer: (penalty regime: 0 %)

Reset answer

```
1 template <typename T>
2 class Array {
```

```
public:
    Array(int size, T initValue);
    ~Array();
    void print();

    Array(const Array<T> & other); // Copy Constructor

private:
    int size;
    T * p;
};

template<typename T>
void Array<T>::print() {
    for (int i = 0; i < this->size; ++i) {
        cout << (i > 0 ? " " : "") << this->p[i];
    }
    cout << endl;
}

// TODO
template<typename T>
Array<T>::Array(const Array<T>& other) {
    this->size = other.size;                    // Sao chép kích thước mảng
    this->p = new T[this->size];                // Cấp phát vùng nhớ mới trên heap

    for (int i = 0; i < this->size; ++i) {
        this->p[i] = other.p[i];                // Sao chép từng phần tử
    }
}
```

	Test	Expected	Got	

✓	`Array<int> * a3;`
	`Array<int> * a4 = new Array<int>(10, 3);`
	`a3 = new Array<int>(*a4);`
	`delete a4;`
	`a3->print();`
	`delete a3;`
		3 3 3 3 3 3 3 3 3 3	3 3 3 3 3 3 3 3 3 3	✓

✓	`Array<int> * a3 = new Array<int>(10, 3);`
	`Array<int> * a4 = new Array<int>(10, 4);`
	`a3 = new Array<int>(*a4);`
	`delete a4;`
	`a3->print();`
	`delete a3;`
		4 4 4 4 4 4 4 4 4 4	4 4 4 4 4 4 4 4 4 4	✓

Passed all tests! ✓

▸ Show/hide question author's solution (Cpp)

Câu hỏi **6**

Đúng

Cho class template Array như bên dưới dùng để chứa một mảng (1 chiều) được khai báo trong vùng nhớ Heap. Trong class Array có khai báo một số phương thức (hàm) để thao tác với Array.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
```

```
    Array(const Array<T> & other); // Copy constructor
    Array<T> & operator=(const Array<T> & other); // Copy assignment operator
private:
    int size;
    T * p;
};
```

Trong class Array có khai báo các thuộc tính sau:

- Thuộc tính **p** là con trỏ trỏ đến vùng nhớ Heap được cấp phát.
- Thuộc tính **size** của Array chứa số lượng phần tử của mảng.

Yêu cầu: hiện thực hàm Copy Constructor và Copy Assignment operator:

- Hàm Array(const Array<T> & other): copy constructor, khởi tạo đối tượng mới dựa trên dữ liệu của đối tượng được cung cấp (other). Đồng thời, in ra thông báo: "Call copy constructor"
- Hàm operator=: copy assignment operator, gán giá trị của đối tượng hiện tại bằng giá trị của đối tượng được cung cấp. Đồng thời, in ra thông báo: "Call assignment operator"

[English]

Given class template **Array** as below that contains an array allocated in Heap memory.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
```

```
    Array(const Array<T> & other); // Copy constructor
    Array<T> & operator=(const Array<T> & other); // Copy assignment operator
private:
    int size;
    T * p;
};
```

There are 2 attributes in class Array:

- **p**: a pointer contains the address of first element of allocated memory.
- **size**: number of elements of array.

Requirement: Implement following method:

- Method Array(const Array<T> & other): copy constructor, initialize new object based on data of provided object. Also, print out the message: "Call copy constructor"
- Method operator=: copy assignment operator, assigns the value of the current object's attributes to the value of the provided object. Print out the message: "Call assignment operator"

For example:

Test	Result

`// Test copy constructor`
`Array<int> * a3;`
`Array<int> * a4 = new Array<int>(10, 3);`
`a3 = new Array<int>(*a4);`
`delete a4;`
`a3->print();`
`delete a3;`
	`Call copy constructor`
	`3 3 3 3 3 3 3 3 3 3`

Answer: (penalty regime: 0 %)

Reset answer

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

template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
    Array(const Array<T> & other); // Copy constructor
    Array<T> & operator=(const Array<T> & other); // Copy assignment operator
    void print();
private:
    int size;
    T * p;
};
template<typename T>
void Array<T>::print() {
    for (int i = 0; i < this->size; ++i) {
        cout << (i > 0 ? " " : "")
             << this->p[i];
    }
    cout << endl;
}
// Copy Constructor
template<typename T>
Array<T>::Array(const Array<T> & other) {
    cout << "Call copy constructor" << endl;
    this->size = other.size;
    this->p = new T[this->size];
    for (int i = 0; i < this->size; ++i) {
        this->p[i] = other.p[i];
    }
}
// Copy Assignment Operator
template<typename T>
Array<T> & Array<T>::operator=(const Array<T> & other) {
    cout << "Call assignment operator" << endl;
    if (this == &other) {
        // Tránh tự gán
        return *this;
    }
    // Giải phóng bộ nhớ cũ
    delete[] this->p;
    // Cấp phát lại và sao chép dữ liệu
    this->size = other.size;
    this->p = new T[this->size];
    for (int i = 0; i < this->size; ++i) {
        this->p[i] = other.p[i];
    }
    return *this;
}

```

	Test	Expected	Got	

✓	`// Test copy constructor`
	`Array<int> * a3;`
	`Array<int> * a4 = new Array<int>(10, 3);`
	`a3 = new Array<int>(*a4);`
	`delete a4;`
	`a3->print();`
	`delete a3;`
		`Call copy constructor`
		`3 3 3 3 3 3 3 3 3 3`
			`Call copy constructor`
			`3 3 3 3 3 3 3 3 3 3`
				✓

✓	`// // Test copy assignment operator`
	`Array<int> * a5 = new Array<int>(1, 2);`
	`Array<int> * a6 = new Array<int>(12, 5);`
	`*a5 = *a6;`
	`delete a6;`
	`a5->print();`
	`delete a5;`
		`Call assignment operator`
		`5 5 5 5 5 5 5 5 5 5 5 5`
			`Call assignment operator`
			`5 5 5 5 5 5 5 5 5 5 5 5`
				✓

Passed all tests! ✓

Câu hỏi **7**

Đúng

Cho class template Array như bên dưới dùng để chứa mảng 1 chiều được khai báo trong vùng nhớ Heap.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
private:
    int size;
    T * p;
};
```

Trong class Array có khai báo các thuộc tính sau:

- Thuộc tính **p** là con trỏ trỏ đến vùng nhớ Heap được cấp phát.
- Thuộc tính **size** của Array chứa số lượng phần tử của mảng.

Yêu cầu:

- SV hiện thực 2 phương thức được mô tả như sau:

1. Hàm Array(int size, T initValue): hàm khởi tạo(constructor), gán size vào số lượng phần tử của mảng; khởi tạo mảng 1 chiều có kích thước là size trong vùng nhớ Heap và lưu địa chỉ phần tử đầu tiên của mảng vào biến p.
2. Hàm ~Array(): hàm huỷ, thu hồi vùng nhớ Heap đã cấp phát.

- SV thực hiện việc khai báo phương thức print (không định nghĩa) cho class Array.

[English]

Given class template **Array** as below that contains an array allocated in Heap memory.

```
template <typename T>
class Array {
public:
    Array(int size, T initValue);
    ~Array();
private:
    int size;
    T * p;
};
```

There are 2 attributes in class Array:

- **p**: a pointer contains the address of first element of allocated memory.
- **size**: number of elements of array.

Requirement: Implement following 2 method:

- Method Array(int size, T initValue): constructor, assigns size to the number of elements of the array; initializes a 1-dimensional array in the heap and stores the address of the first element of the array in the variable p.
- Method ~Array(): destructor, recovers the allocated Heap memory.
- Also, declare the print() method for the class Array (don't define the method).

For example:

Test	Result
`Array<int> a1(5, 0);` `a1.print();`	`0 0 0 0 0`

Answer: (penalty regime: 0 %)

Reset answer

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

```

```
4  template <typename T>
5  class Array {
6  public:
7      // Constructor: khởi tạo mảng và gán giá trị khởi đầu
8      Array(int size, T initValue);
9
10     // Destructor: thu hồi vùng nhớ Heap
11     ~Array();
12
13     // Khai báo hàm print (chưa định nghĩa)
14     void print();
15
16 private:
17     int size;
18     T* p;
19 };
20
21 // Constructor
22 template <typename T>
23 Array<T>::Array(int size, T initValue) {
24     this->size = size;
25     p = new T[size]; // cấp phát mảng trên heap
26
27     for (int i = 0; i < size; ++i) {
28         p[i] = initValue; // khởi tạo giá trị cho từng phần tử
29     }
30 }
31
32 // Destructor
33 template <typename T>
34 Array<T>::~Array() {
35     delete[] p; // thu hồi vùng nhớ Heap
36 }
37
38
```

	Test	Expected	Got	
✓	`Array<int> a1(5, 0);` `a1.print();`	`0 0 0 0 0`	`0 0 0 0 0`	✓
✓	`Array<int> a1(10, 3);` `a1.print();`	`3 3 3 3 3 3 3 3 3 3`	`3 3 3 3 3 3 3 3 3 3`	✓

Passed all tests! ✓

Câu hỏi **8**

Đúng

Hãy xem xét định nghĩa sau:

```
template < class T >
class Cell {
       protected:
           T info;
       public:
           void set(T x){ info = x; }
           T get() { return info; }
   };
enum Color {White, Yellow, Black, Red, Blue};
```

Định nghĩa một **lớp con** ColoredCell của Cell với:

1. Với một biến color.

2. Hàm setColor(Color) để cài đặt màu cho ô

3. Hàm getColor() để lấy màu của ô (trả về kiểu dữ liệu của màu).

4. Cập nhật method get để trả về con trỏ đến info nếu ô này không trắng, ngược lại trả về NULL.

[English]

Given the following declaration:

```
template < class T >
class Cell {
       protected:
           T info;
       public:
           void set(T x){ info = x; }
           T get() { return info; }
   };
enum Color {White, Yellow, Black, Red, Blue};
```

Define a class ColoredCell as a child of class Cell with:

1. A variable color.

2. Function setColor(Color) to set a cell's color.

3. Function getColor() to return a cell's color.

4. Overload function get() to return the pointer to info if the cell's color is not white, else return NULL.

For example:

Test	Input	Result
1	1 2	2

Answer: (penalty regime: 0 %)

Reset answer

```
#include <iostream>
#include <string.h>

using namespace std;

template<class T> class Cell {
protected:
    T infor;
public:
    void set(T _infor) { this->infor = _infor; };
    T get() { return this->infor; };
};

```

```cpp
enum Color {White, Yellow, Black, Red, Blue};

// TODO
template<class T>
class ColoredCell : public Cell<T> {
private:
    Color color;
public:
    void setColor(Color c) { color = c; }
    Color getColor() { return color; }

    T* get() {
        if (color == White) return NULL;
        return &(this->infor); // trả về con trỏ đến biến infor
    }
};


int main () {
    int test, c;
    cin >> test >> c;
    ColoredCell<int>* a = new ColoredCell<int>();
    switch (test)
    {
    case 1:
        a->setColor(Color(c));
        cout << int(a->getColor());
        break;
    case 2:
        a->setColor(Color(c));
        a->set(10);
        if (a->get() == NULL) cout << "null" << endl;
        else cout << *(a->get());
    default:
        break;
    }
    return 0;
}
```

	Test	Input	Expected	Got	
✓	1	1 2	2	2	✓
✓	2	1 -1	-1	-1	✓

Passed all tests! ✓