# PRELAB

## [Q 1.2] Integer

Hãy thiết kế một class **Integer** với một biến private **val** dạng ***int***, Class gồm các phương thức sau:

* **constructor** nhận biến **val** ban đầu.
* **setValue()** để thay đổi giá trị biến **val**.
* **getValue()** để lấy giá trị biến **val**.

Define a class Integer with one private variable **val** type int with:  
\_ A constructor with one input for **val**.  
\_ A function setValue() to change the value of **val**.  
\_ A function getValue() to return the value of **val**.

For example:

| **Test** | **Result** |
| --- | --- |
| cout<<"Constructor test";  Integer i(10);  Integer i2(-10);  Integer \*i3=new Integer(20);  delete i3; | Constructor test |

## Answer q1.2

class *Integer*{

private:

    int val;

public:

    Integer(int *val*){

*this*->val = *val*;

    }

    void setValue(int *val*){

*this*->val = *val*;

    }

    int getValue(){ return *this*->val; }

};

## [Q 1.3] Room

Hãy thiết kế một class Room với constructor gồm 3 biến length, breadth, height (theo thứ tự). Hiện thực các phương thức sau cho class Room:

* **Constructor:** đầu vào lần lượt là ***length, breadth, height***
* **calculateArea:** tính diện tích của căn phòng.
* **calculateVolume:** tính thể tích của căn phòng

Define class Room with three variable length, breadth and height and:  
\_ A constructor withthree input for length, breadth and height.  
\_ Function calculateArea to calculate and return the room area.  
\_ Function calculateVolume to calculate and return the room volume.

For example:

| **Test** | **Result** |
| --- | --- |
| cout<<"Constructor test";  Room r(20,3,4);  Room \*r2=new Room(10.5,5.5,5.4);  delete r2; | Constructor test |

## Answer q1.3

class *Room*{

private:

    double length;

    double breadth;

    double height;

public:

    Room(double *length*, double *breadth*, double *height*){

*this*->length = *length*;

*this*-> breadth = *breadth*;

*this*->height = *height*;

    }

    double calculateArea(){

        return length \* breadth;

    }

    double calculateVolume(){

        return length \* breadth \* height;

    }

};

## [Q 1.4] ClockType: Constructor

Hãy xem xét câu lệnh:

ClockType myClock (5, 12, 40);

Câu lệnh này khai báo một đối tượng myClock thuộc class ClockType. Ở đây, chúng ta đang truyền ba giá trị kiểu ***int***, giá trị này khớp với kiểu của các tham số chính thức của hàm tạo với một tham số.

* Xem xét đoạn code cho trước trong phần trả lời, chú ý đến hàm khởi tạo có 3 tham số. Hãy hiện thực hàm này để sau khi gọi câu lệnh khai báo trên, 3 biến thành viên được lần lượt đặt thành 5, 12, 40.
* Hiện thực hàm khởi tạo với không tham số, hàm sẽ gán giá trị **0** cho 3 biến thành viên.

**Lưu ý**: hr, min, sec cần thoả mãn các điều kiện sau. Nếu tham số đầu vào không thoả mãn điều kiện bên dưới thì ta gán giá trị 0 cho biến thành viên tương ứng.

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

[English]

Consider the following statement:

clockType myClock (5, 12, 40);

This statement declares a variable name myClock belongs to type ClockType with 3 input variables.

* Complete the constructor with 3 parameters in the class given in the answer box so that the time can be printed correctly.
* Complete the constructor with 0 parameters that set the value of three attributes to 0.

**Note:** hr, min, sec need to satisfy the following conditions. If the input parameter does not satisfy the condition below, we assign the value 0 to the corresponding member variable:

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

For example:

| **Test** | **Result** |
| --- | --- |
| ClockType myClock;  myClock.printTime(); | 00:00:00 |

class ClockType

{

public:

ClockType(int, int, int); //constructor with parameters

ClockType(); //default constructor

void printTime() const;

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

## Answer q1.4

class *ClockType*

{

   public:

      ClockType(int *hr*, int *min*, int *sec*){

*this*->hr = checkhr(*hr*);

*this*->min = checkmin(*min*);

*this*->sec = checksec(*sec*);

      } //constructor with parameters

      ClockType(){

        hr=0;

        min=0;

        sec=0;

      } //default constructor

      int checkhr(int *hr*){

        if(*hr*<0 || *hr*>=24) return 0;

        return *hr*;

      }

      int checkmin(int *min*){

        if(*min*<0 || *min*>=60) return 0;

        return *min*;

      }

      int checksec(int *sec*){

        if(*sec*<0 || *sec*>=60) return 0;

        return *sec*;

      }

      void printTime() const;

   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;

}

## [Q 1.5] ClockType: setTime

Cho định nghĩa class:

class ClockType  
{  
   public:  
      ClockType();  
      void setTime(int, int, int);  
      void printTime() const;  
   private:  
      int hr;  
      int min;  
      int sec;  
};

Cho câu lệnh:

myClock.setTime(5, 2, 30);

Trong câu lệnh ***myClock.setTime (5, 2, 30);*** phương thức setTime được thực thi. Các giá trị 5, 2 và 30 được chuyển dưới dạng tham số cho hàm setTime và hàm sử dụng các giá trị này để đặt giá trị của ba biến thành viên hr, min và sec của đối tượng myClock thành 5, 2 và 30, tương ứng.

**Yêu cầu**: SV hiện thực phương thức ***setTime*** để nó thực hiện được mô tả trên.

**Lưu ý**: hr, min, sec cần thoả mãn các điều kiện sau. Nếu tham số đầu vào không thoả mãn điều kiện bên dưới thì ta gán giá trị 0 cho biến thành viên tương ứng.

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

[English]

Given the class definition:

class ClockType  
{  
    public:  
       ClockType();  
       void setTime(int, int, int);  
       void printTime() const;  
    private:  
       int hr;  
       int min;  
       int sec;  
};

And the following command:

myClock.setTime(5, 2, 30);

In the statement ***myClock.setTime(5, 2, 30);*** the method setTime is executed. The values 5, 2, and 30 are passed as parameters to the function setTime, and the function uses these values to set the values of the three member variables hr, min, and sec of object myClock to 5, 2, and 30, respectively.

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

**Note:**hr, min, sec need to satisfy the following conditions. If the input parameter does not satisfy the condition below, we assign the value 0 to the corresponding member variable:

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

For example:

| **Test** | **Result** |
| --- | --- |
| ClockType myClock;  myClock.setTime(5, 4, 30);  myClock.printTime(); | 05:04:30 |

class ClockType

{

public:

ClockType();

void setTime(int, int, int);

void printTime() const;

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

## Answer q1.5

class *ClockType*

{

public:

    ClockType();

    void setTime(int *hr*, int *min*, int *sec*){

*this*->hr = checkhr(*hr*);

*this*->min = checkmin(*min*);

*this*->sec = checksec(*sec*);

    }

    void printTime() const;

    int checkhr(int *hr*){

        if(*hr*<0 || *hr*>=24) return 0;

        return *hr*;

      }

      int checkmin(int *min*){

        if(*min*<0 || *min*>=60) return 0;

        return *min*;

      }

      int checksec(int *sec*){

        if(*sec*<0 || *sec*>=60) return 0;

        return *sec*;

      }

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

## [Q 1.13] class Array operator []

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();  
    void print();

   void setAt(int idx, const T & value);  
   T getAt(int idx);  
 T& operator[](int idx);

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 được mô tả như sau:

* **operator[](int idx):** quá tải toán tử [] để thực hiện được cả 2 việc: đọc (lấy) giá trị của phần tử ở vị trí idx và ghi (gán) một giá trị mới vào phần tử ở vị trí idx. Ví dụ: đọc: x = a[2], ghi: a[2] = 5 với a là một đối tượng của class Array. Nếu idx có giá trị không hợp lệ (idx < 0 hoặc idx >= size) thì **throw -1;**

[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();  
    void print();

   void setAt(int idx, const T & value);  
   T getAt(int idx);  
 T& operator[](int idx);

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:

* **operator[](int idx):** overload operator [] to do: read (get) the value of element at position idx, and write (set) a new value to the element at position idx. For instance: read: x = a[2], write: a[2] = 5, for a is an object of class Array. if idx has an invalid value (idx < 0 or idx >= size), then throw -1;

Test

Array<int> a9(1000, 7);

a9.setAt(0, 99);

cout << "Subscript Array["

<< 0 << "]:"

<< a9[0] << endl;

a9[0] = -99;

a9.print();

Result

Subscript Array[0]:99

-99 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 7 …

## **The const int& is same as int const&**. So this refers to a constant integer. The integer cannot be modified through the reference.

class Array {

public:

    Array(int *size*, *T* *initValue*);

    ~Array();

    void setAt(int *idx*, const *T* & *value*);

*T* getAt(int *idx*);

*T*& operator[](int *idx*);

    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;

}

// TODO

## Answer q1.13

template <typename *T*>

class Array {

public:

    Array(int *size*, *T* *initValue*);

    ~Array();

    void setAt(int *idx*, const *T* & *value*);

*T* getAt(int *idx*);

*T*& operator[](int *idx*){

        if(*idx*<0 || *idx*>=size){

          throw -1;

        }

        else{

*T*\* temp=p;

          for(int i=0; i<*idx*; i++){

            temp=temp+1;

        }

        return \*temp;

    }

    }

    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;

}

// TODO

## [Q 1.6] ClockType: getTime

Cho định nghĩa class:

class ClockType  
{  
   public:  
      void setTime(int, int, int);  
      void getTime(int&, int&, int&) const;  
      void printTime() const;  
      clockType(int, int, int); //constructor with parameters  
      clockType(); //default constructor

 private:  
      int hr;  
      int min;  
      int sec;

};

Cho đoạn code sau:

clockType myClock;   
int hours;   
int minutes;   
int seconds;  
myClock.getTime(hours, minutes, seconds);   
cout << "hours = " << hours << ", minutes = " << minutes << ", seconds = " << seconds << endl;

Trong câu lệnh myClock.getTime(hours, minutes, seconds); hàm thành viên getTime được thực thi. Các giá trị hr, min và sec của myClock được hàm sử dụng để cài đặt giá trị của ba biến hours, minutes, seconds tương ứng.

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

**Lưu ý**: hr, min, sec cần thoả mãn các điều kiện sau. Nếu tham số đầu vào không thoả mãn điều kiện bên dưới thì ta gán giá trị 0 cho biến thành viên tương ứng.

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

[English]

Given the class definition:

class ClockType  
{  
   public:  
      void setTime(int, int, int);  
      void getTime(int&, int&, int&) const;  
      void printTime() const;  
      clockType(int, int, int); //constructor with parameters  
      clockType(); //default constructor  
   private:  
      int hr;  
      int min;  
      int sec;  
};

And the following code fragment:

clockType myClock;   
int hours;   
int minutes;   
int seconds;  
myClock.getTime(hours, minutes, seconds);   
cout << "hours = " << hours << ", minutes = " << minutes << ", seconds = " << seconds << endl;

In the statement myClock.getTime(hours, minutes, seconds); getTime member function is executed. The hr, min, and sec values of myClock object are used by the function to set the values of the three variables hours, minutes, and seconds respectively.

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

**Note:**hr, min, sec need to satisfy the following conditions. If the input parameter does not satisfy the condition below, we assign the value 0 to the corresponding member variable:

* 0 <= hr < 24
* 0 <= min < 60
* 0 <= sec < 60

For example:

| **Test** | **Result** |
| --- | --- |
| ClockType myClock;  myClock.setTime(5, 4, 30);  int hours,minutes,seconds;  myClock.getTime(hours, minutes, seconds);  cout << "Hours = " << hours << ", minutes = " << minutes << ", seconds = " << seconds << endl; | Hours = 5, minutes = 4, seconds = 30 |

class ClockType

{

public:

ClockType(int, int, int);

ClockType();

void printTime() const;

void setTime(int, int, int);

void getTime(int&, int&, int&) const;

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

## Answer q1.16

class *ClockType*

{

public:

    ClockType(int, int, int);

    ClockType();

    void printTime() const;

    void setTime(int, int, int);

    void getTime(int& *hour*, int& *minute*, int& *second*) const{

*hour*=*this*->hr;

*minute*=*this*->min;

*second*=*this*->sec;

    }

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

## [Q 1.17] class Course

Trong thân hàm main(), hãy viết chương trình chính tạo ra một mảng các đối tượng thuộc **class Course** có **n** phần tử (n được nhập từ người sử dụng – sử dụng kỹ thuật tạo mảng động bằng lệnh **new**). Sau đó nhập và hiển thị dữ liệu cho n đối tượng này bằng hai phương thức **getinfo**() và **disinfo**() đã được hiện thực sẵn.

[English]

In **main**, write a program that creates an array of Course objects with **n** elements (n is input from user). The array is created using dynamic allocation with operator **new**.

Then input and display data for each element of array using two implemented methods: **getinfo** and **disinfo**.

For example:

| **Test** | **Input** | **Result** |
| --- | --- | --- |
| 1 | 2  1 2 3 4  1 3 4 5 | ID: Number of Students: Number of Teachers: Number of TAs: ID: Number of Students: Number of Teachers: Number of TAs:  CourseID = 1  Number of student = 2  Number of teacher = 3  Number of TA = 4  CourseID = 1  Number of student = 3  Number of teacher = 4  Number of TA = 5 |

#include<iostream>

using namespace std;

class Course {

private:

int ID;

int numOfStudent;

int numOfTeacher;

int numOfTA;

public:

void getinfo();

void disinfo();

};

void Course::getinfo() {

cout << "ID: ";

cin >> ID;

cout << "Number of Students: ";

cin >> numOfStudent;

cout << "Number of Teachers: ";

cin >> numOfTeacher;

cout << "Number of TAs: ";

cin >> numOfTA;

}

void Course::disinfo()

{

cout<<endl;

cout<< "CourseID = "<< ID << endl;

cout<< "Number of student = " << numOfStudent << endl;

cout<< "Number of teacher = " << numOfTeacher << endl;

cout<< "Number of TA = " << numOfTA<< endl;

}

int main() {

//TODO

return 0;

}

## Answer q1.17

int main() {

    //TODO

    int n;

    cin>>n;

*Course*\* c=**new** *Course*[n];

    for(int i=0; i<n; i++){

        c[i].getinfo();}

    for(int i=0; i<n; i++){

      c[i].disinfo();}

    return 0;

}

# INLAB

## [Q 1.8] ClockType: Increment

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 |

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

## Answer q1.8

class *ClockType*

{

   public:

      void setTime(int, int, int);

      void printTime() const;

      void incrementSeconds(){

          sec++;

          if(sec == 60){

              sec = 0;

*this*->incrementMinutes();

          }

      }

      void incrementMinutes(){

          min++;

          if(min == 60){

              min=0;

*this*->incrementHours();

          }

      }

      void incrementHours(){

          hr++;

          if(hr == 24){

              hr=0;

          }

      }

      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

## [Q 1.1] ColoredCell

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 |

#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; };

};

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

// TODO

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;

}

## Answer q1.1

#include <iostream>

#include <string.h>

using namespace std;

enum *Color* {White, Yellow, Black, Red, Blue};

template<class *T*> class Cell {

protected:

*T* infor;

public:

    void set(*T* *\_infor*) { *this*->infor = *\_infor*; };

*T* get() { return *this*->infor; };

};

template<class *T*>class ColoredCell: public Cell<*T*> {

    //false: template<class T>class ColoredCell: public Cell

    //it dont consider template T for base class Cell

private:

*Color* mycolor;

public:

    void setColor(*Color* *c*);

*Color* getColor();

*T*\* get();

};

//Below is syntax which suitable for write header file(assignment)

//template<class T>

//void ColoredCell<T>::setColor(Color c)

// TODO

template<class *T*>

void ColoredCell<*T*>::setColor(*Color* *c*){

*this*->mycolor = *c*;

}

template<class *T*>

*Color* ColoredCell<*T*>::getColor(){

    return *this*->mycolor;

}

template<class *T*>

*T*\* ColoredCell<*T*>::get(){

    if(*this*->mycolor != White) return &(*this*->infor);

        //null is false, NULL//nullptr is true

        else return NULL;

}

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;

}

## [Q 1.7] ClockType: equalTime

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. |

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

Answer q1.7

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*& *yours*) const{

    if( *yours*.hr== hr && *yours*.min== min && *yours*.sec== sec ) return true;

    else return false;

}

## [Q 1.9] class Array: Constructor & Destructor

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 |

template <typename *T*>

class Array {

public:

    Array(int *size*, *T* *initValue*);

    ~Array();

private:

    int size;

*T* \* p;

};

// TODO

## Answer q1.9

template <typename *T*>

class Array {

public:

    Array(int *size*, *T* *initValue*);

    ~Array();

    //declare the print() method for the class Array (don't define the method)

    void print();

private:

    int size;

*T* \* p;

};

// TODO

template<typename *T*>

Array<*T*>::Array(int *size*, *T* *initValue*){

*this*->size = *size*;

*this*->p= **new** *T*[*size*];

    for(int i = 0; i < *size*; i++){p[i] = *initValue*;}

}

template<typename *T*>

Array<*T*>::~Array(){}

//must define constructor function

## [Q 1.10] class Array: Copy Constructor

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 (đươc 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 |

template <typename *T*>

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

## Answer q1.10

template <typename *T*>

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;

*this*->p = **new** *T*[*this*->size];

    for(int i = 0; i <*this*->size; i++){

        p[i] = *other*.p[i];

    }

}

## [Q 1.11] Book

Cho class như sau:

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

#### Hiện thực **constructor** và **destructor** của Booksao 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 |

// string.h is included

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

## Answer q1.11

// string.h is included

#include<string.h>

#include <iostream>

using namespace std;

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\* *Name*){

*this*->name= **new** char[strlen(*Name*)+1];

    //we need +1 to set '\0'

    //NOTE: char\* is the same as string

    for(int i=0; i<strlen(*Name*); i++){

        name[i]=*Name*[i];

    }

    name[strlen(*Name*)]='\0';

}

*Book*::~Book(){

**delete[]** name;

}

## [Q 1.12] Integer: Constructor & operator+

## [Q 1.15] class Array: Copy Constructor & Copy Assignment Operator

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 |

#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;

}

// TODO

# POSTLAB