Họ và tên : Đỗ Thái Dương  
MSSV : N22DCDK018  
Lớp : D22CQDK01-N

**Đề 2013**

**Câu 1:**

A)

#include<iostream>

using namespace std;

class ComplexNumber{

    private:

        int a, b;

    public:

        ComplexNumber();

        ComplexNumber(int x, int y);

        void nhap();

        void xuat();

        ComplexNumber operator + (const ComplexNumber& other);

        ComplexNumber operator – (constComplexNumber& other);

        ComplexNumber operator - ();

        ComplexNumber operator \* (const ComplexNumber& other);

};

B)

ComplexNumber::ComplexNumber(){

    a = 0;

    b = 0;

}

ComplexNumber::ComplexNumber(int x, int y){

    a = x;

    b = y;

}

void ComplexNumber::nhap(){

    cout << "Nhap phan thuc = ";

    cin >> a;

    cout << "Nhap phan ao = ";

    cin >> b;

}

void ComplexNumber::xuat(){

    if (a != 0 && b != 0) {

        cout << a << " + " << b << "√7";

    }

    else if(a != 0 && b != 0 && b < 0){

        cout << a << " " << b << "√7";

    } else if (a != 0 && b == 0) {

        cout << a;

    } else if (a == 0 && b != 0){

        cout << b << "√7";

    } else {

        cout << "0";

    }

    cout << endl;

}

ComplexNumber ComplexNumber::operator + (const ComplexNumber& other){

    ComplexNumber tong;

    tong.a = a + other.a;

    tong.b = b + other.b;

    return tong;

}

ComplexNumber ComplexNumber::operator -(const ComplexNumber& other){

    ComplexNumber hieu;

    hieu.a = a - other.a;

    hieu.b = b - other.b;

    return hieu;

}

ComplexNumber ComplexNumber::operator -(){

    ComplexNumber negation;

    negation.a = -a;

    negation.b = -b;

    return negation;

}

ComplexNumber ComplexNumber::operator \*(const ComplexNumber& other){

    ComplexNumber nhan;

    nhan.a = (a \* other.a) + (7 \* b \* other.b);

    nhan.b = (a \* other.b) + (b \* other.a);

    return nhan;

}

**Câu 2:**

- Hàm hủy ảo (virtual destructor) được sử dụng để đảm bảo rằng các tài nguyên được giải phóng đúng cách khi một đối tượng bị hủy. Hàm hủy ảo cho phép kế thừa nhiều lớp và đảm bảo rằng hàm hủy của lớp dẫn xuất được gọi trước khi hàm hủy của lớp cơ sở.

- Nếu không sử dụng hàm hủy ảo khi có các lớp kế thừa, có thể xảy ra tình trạng "slicing" khi một con trỏ hoặc tham chiếu đến đối tượng của lớp dẫn xuất được gán giá trị từ một đối tượng của lớp cơ sở. Trong trường hợp này, chỉ phần của đối tượng thực sự thuộc lớp cơ sở sẽ được giữ lại, và thông tin của lớp dẫn xuất sẽ bị mất.

**Câu 3:**

A)

- Không thể tạo ra đối tượng của lớp absList được vì lớp **absList** là một lớp trừu tượng (abstract class) vì nó có ít nhất một phương thức ảo thuần ảo (**pure virtual function**), có thể nhận diện bằng dấu bằng 0 (= 0) và được kết thúc bằng dấu chấm phẩy. Trong trường hợp này, các phương thức thuần ảo là **addFirst**, **getSubItem**, và **showAll**.

- Vì lớp **absList** có ít nhất một phương thức thuần ảo, nó không thể tạo ra đối tượng trực tiếp. Bạn không thể khởi tạo một đối tượng thuộc lớp trừu tượng.

B)

Dòng lệnh delete lnkLst sẽ chỉ làm hủy một lần duy nhất, không phải là một số lần như bạn nghĩ. Lớp linearList đã kế thừa từ lớp absList và định nghĩa lại hàm hủy (destructor), trong đó có gọi đến toán tử delete để giải phóng bộ nhớ của đối tượng con subLst (nếu đối tượng subLst không bằng NULL). Việc xóa một đối tượng chỉ xóa đối tượng đó mà thôi, nên chỉ có một lần hủy diễn ra.

C)

Sửa lại mã nguồn của phương thức countAll():

// Trong lớp linearList

virtual int countAll() override {

    int count = 1; // Bắt đầu với 1 vì linearList chứa ít nhất một phần tử (chính nó)

    absList\* current = subLst; // Bắt đầu từ phần tử tiếp theo sau linearList

    while (current != nullptr) {

        count++;

        current = current->getSubItem();

    }

    return count;

}

D)

Mã nguồn phương thức showAll(ostream& outDev) của lớp linearList chưa cài đặt xong:

virtual void showAll(ostream& outDev) override {

    outDev << getData() << " "; // Hiển thị giá trị của đối tượng linearList hiện tại

    absList\* current = subLst;

    while (current != nullptr) {

        current->showAll(outDev); // Đệ quy để hiển thị giá trị của các phần tử con

        current = current->getSubItem();

    }

}

E)

- Cấu trúc dữ liệu được cài đặt bởi lớp linearList thực chất là một danh sách liên kết (linked list).

- Danh sách liên kết là một cấu trúc dữ liệu mà mỗi phần tử (nút) có một con trỏ trỏ đến phần tử tiếp theo trong danh sách. Trong trường hợp của lớp linearList, mỗi phần tử được biểu diễn bởi một đối tượng linearList, và con trỏ subLst của mỗi đối tượng linearList trỏ đến đối tượng linearList tiếp theo trong danh sách.

Một số nhận xét:

- Một nhận xét quan trọng là trong cài đặt hiện tại, lớp linearList không cung cấp phương thức để thêm phần tử vào cuối danh sách hoặc xóa phần tử khỏi danh sách. Điều này giới hạn tính linh hoạt và khả năng sử dụng của cấu trúc dữ liệu linearList. Để tối ưu hơn, có thể cải thiện bằng cách thêm các phương thức như addLast() để thêm phần tử vào cuối danh sách và remove() để xóa phần tử khỏi danh sách.

- Cần triển khai bằng cách sử dụng đệ quy cho các phương thức như countAll() và showAll(), điều này có thể tạo ra mã nguồn dễ đọc và linh hoạt.