**TRƯỜNG ĐẠI HỌC CÔNG NGHỆ GIAO THÔNG VẬN TẢI**

**KHOA CÔNG NGHỆ THÔNG TIN**



**BÀI TẬP HỌC PHẦN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG C++**

**BÀI KIỂM TRA ÔN TẬP**

**Nhóm 4 : Đỗ Ngọc Thọ - 70DCHT23013**

**Trương Thị Hà Phương – 70DCHT21098**

**Lớp : 70DCHT22**

**Giảng viên: Phạm Thị Thuận**

Hà Nội, Tháng 6 Năm 2021

**MỤC LỤC**

[**PHẦN 1: LÝ THUYẾT 3**](#_Toc73701140)

[**1.1. Lập trình hướng đối tượng có mấy đặc trung (mỗi đặc trưng cho một ví dụ) 3**](#_Toc73701141)

[**1.2. Lớp là gì? Cho ví dụ 4**](#_Toc73701142)

[**1.3. Nêu các phương thức có thể có trong một lớp 4**](#_Toc73701143)

[**1.4. Cú pháp của phương thức 6**](#_Toc73701144)

[**1.5. Hàm bạn là gì? Cú pháp khai báo hàm bạn? 6**](#_Toc73701145)

[**1.6. Getter, setter là gì? Cho ví dụ 7**](#_Toc73701146)

[**1.7. Nạp chồng toán tử là gì? Vì sao phải nạp chồng 8**](#_Toc73701147)

[**1.8. Phân biệt phương thức tĩnh, ảo, thuần ảo 8**](#_Toc73701148)

[**1.9. Kế thừa là gì có mấy loại kế thừa, mục đích của kế thừa. cho ví dụ 9**](#_Toc73701149)

[**1.10. Đa hình là gì. Cho ví dụ 10**](#_Toc73701150)

[**1.11. Interface là gì? Cho ví dụ 11**](#_Toc73701151)

[**PHẦN 2: BÀI TẬP 12**](#_Toc73701152)

# **PHẦN 1: LÝ THUYẾT**

* 1. Lập trình hướng đối tượng có mấy đặc trung (mỗi đặc trưng cho một ví dụ)

Lập trình hướng đối tượng ( Object-oriented programming, viết tắt: OOP)

Lập trình hướng đối tượng có 4 đặc trưng:

* Tính trừu tượng (Abstraction): là tính chất không thể hiện cụ thể mà chỉ nêu tên vấn đề. Đó là một quá trình che giấu các hoạt động bên trong và chỉ hiển thị những tính năng thiết yếu của đối tượng tới người dùng.

Ví dụ: Một người sửu dụng điện thoại để gửi tin nhắn thì anh ta sẽ nhập nội dung tin nhắn, thông tin người nhận và ấn nút gửi. Khi anh ta bắt đầu gửi tin thì anh ấy không biết những gì diễn ra bên trong quá trình gửi mà chỉ biết được kết quả của tin nhắn đã được gửi đến người nhận thành công hay chưa.

* Tính đóng gói (Encapsulation): là sự che giấu bên trong dữ liệu riêng của mỗi đối tượng của lớp được khai báo và chỉ được truy xuất thông qua hệ thống các phương thức có sẵn của lớp.

Ví dụ:

class mayvitinh {

private:

string mausac;

int chieudai, chieurong;

public:

void input()

{

cout << "Nhap mau sac may: ";

fflush(stdin);

getline(cin, this->mausac);

cout << "Nhap chieu dai may: ";

cin >> this->chieudai;

cout << "Nhap chieu rong may: ";

cin >> this->chieurong;

}

void output()

{

cout << "Mau sac may: " << this->mausac << endl;

cout << "Chieu dai may: " << this->chieudai << endl;

cout << "Chieu rong may: " << this->chieurong << endl;

}

};

Tính đóng gói được thể hiện qua các *thuộc tính* mausac, chieudai, chieurong và *phương thức* input(), output() vào trong class mayvitinh. Ta không thể truy cập đến các private data hoặc gọi đến private methods của class từ bên ngoài class đó.

* Tính kế thừa (Inheritance): là kỹ thuật cho phép xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có; lớp đã có gọi là lớp Cha, lớp mới phát sinh gọi là lớp Con.

Ví dụ: - Lớp Cha là smartphone, có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành...

-Lớp Con là Iphone, Samsung, Oppo cũng có các thuộc tính: màu sắc, bộ nhớ, hệ điều hành...

* Tính đa hình (Polymorphism): Đa hình có nghĩa là nhiều hình thức, một đối tượng thuộc các lớp khác nhau có thể hiểu cùng một thông điệp theo cách khác nhau. Trong đó ‘Poly’ có nghĩa là nhiều, còn ‘morph’ có nghĩa là hình thức.

Ví dụ: Trong thực tế, ta có 2 con vật chó và mèo khi 2 con vật này nhận được mệnh lệnh là “hãy kêu” thì chó kêu “gâu gâu”, mèo kêu”meo meo”.

* 1. Lớp là gì? Cho ví dụ
* Lớp (Class): có thể coi là bản thiết kế của các đối tượng (object). Nó là một kiểu dữ liệu do người dùng định nghĩa, chức các thành viên dữ liệu và các hàm thành viên của riêng nó.
* Ví dụ: Xem xét các sinh viên. Mỗi sinh viên đều có thuộc tính chung là: tên, lớp, mã sinh viên, tuổi, quê quán, điểm toán, điểm lý... Do đó ta sẽ coi sinh viên là một lớp (class). Lớp sinh viên sẽ có các thành viên là: tên, lớp, mã sinh viên, tuổi, quê quán, điểm toán, điểm lý...
  1. Nêu các phương thức có thể có trong một lớp
* Phương thức khởi tạo (Constructor): Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng, hàm tạo được gọi khi đối tượng được cấp phát bộ nhớ.

-Định nghĩa ngoài lớp:

<tên\_lớp>::<tên\_lớp>([ds tham số])

{

//thân hàm

}

\*Chú ý: Hàm tạo là 1 phương thức đặc biệt vì:

+Tên hàm trùng với tên lớp.

+Không có kiểu dữ liệu trả về của hàm.

-Hàm tạo có 2 loại:

+Hàm tạo có đối số.

Diem (int x1, int y1) //hàm tạo có đối

{

x=x1

y=y1

}

+Hàm tạo không có đối số.

Diem() //hàm tạo không đối số

{

x=0;

y=0;

}

* Phương thức hủy (Destructor):

-Chức năng: Hủy bỏ, giải phóng các đối tượng khi nó hết phạm vi tồn tại.

-Hàm hủy cũng được định nghĩa trong lớp.

* Con trỏ this: là con trỏ trỏ đến đối tượng đang gọi hàm (phương thức), tồn tại ngầm trong mỗi lớp.
* Phương thức getter, setter:

-Ta cần xây dựng getter/setter để truy suất giá trị các thuộc tính (vì thuộc tính có phạm vi truy xuất private).

-Setter: truyền giá trị vào cho biến.

-Getter:lấy giá trị của biến.

-Khi các thuộc tính của lớp để gán phạm vi truy xuất vào private thì cần dùng đến getter/setter.

* 1. Cú pháp của phương thức
* Phương thức khởi tạo (Constructor):

<tên\_lớp>([ds tham số]);

* Phương thức hủy (Destructor):

~<tên\_lớp>();

* Con trỏ this:

This-> tên thuộc tính

* Phương thức getter,setter:

-Setter:

void set<tên thuộc tính> (<tham số giá trị mới>){

<tên thuộc tính> = <tham số giá trị mới>;

}

-Getter:

<kiểu dữ liệu thuộc tính> get<tên thuộc tính> () {

Return <tên thuộc tính>;

}

* 1. **Hàm bạn là gì? Cú pháp khai báo hàm bạn?**

Khái niệm:

* Hàm bạn là hàm tự do, không thuộc lớp. Tuy nhiên hàm bạn có quyền truy cập các thành viên private hoặc protected của một lớp. Hàm bạn không phải là hàm thành viên của lớp
* Một lớp trong c++ có thể có nhiều hàm bạn, hàm được khai báo bên trong lớp bắt đầu bằng từ khóa friend.

Đặc điểm của hàm bạn:

* Được xây dựng bên ngoài lớp.
* Không dùng tên lớp và toán tử phạm vi :: trong định nghĩa hàm bạn
* Hàm bạn của lớp định nghĩa bên ngoài của lớp.
* Được phép truy cập vào bất cứ thành viên nào của lớp mà nó nhận là bạn.
* Một hàm có thể là bạn của nhiều lớp.
* Hàm bạn không phải phương thức của lớp
* Hàm bạn của lớp được phép truy cập đến thành phần private của lớp.
* Hàm bạn không là thành viên của lớp nên không cho kế thừa.
* Khi truy cập hàm thành viên:
* <biến đối tượng>.<Tên hàm>; VD: p.sum();
* Truy cập hàm bạn:
* <Tên hàm> (biến đối tượng); VD: sum(p);
* Hàm bạn phá bỏ nguyên tắc che dấu thông tin.

Cú pháp:

class <Tên lớp>

{

// Khai báo các thành phần lớp như thông thường

// Khai báo hàm bạn

friend <Kiểu trả về> <Tên hàm bạn>([<Các tham số>]);

};

* **Kiểu\_dữ\_liêu tên\_hàm\_bạn ([ds tham số])**
* **{ thân hàm }**
  1. Getter, setter là gì? Cho ví dụ
* Getter: là truyền giá trị vào cho biến.

Setter: là lấy giá trị của biến.

* Ví dụ:

#include<iostream>

Using namespace std;

Class Employee{

Private:

//Private attribute

Int salary;

Public:

//Setter

Void setSalary(int s){

Salary = s;}

//Getter

Int getSalary() {

Return salary;

}

}

Int main() {

Employee myObj;

MyObj.setSalary(50000);

Cout<<myObj.getSalary();

Return 0;

}

* 1. Nạp chồng toán tử là gì? Vì sao phải nạp chồng
* Nạp chồng toán tử (overloading) : là khả năng một toán tử có thể thực hiện việc tính toán ứng với nhiều kiểu dữ liệu khác nhau.Các toán tử có thể được định nghĩa chồng như một phương thức của lớp hoặc như hàm bạn của lớp.
* Phải nạp chồng toán tử vì khi nạp chồng toán tử chúng ta xử lý các câu lệnh ở main trở nên tường minh và dễ hiểu hơn cho cả người code lẫn người đọc, do đó nó rất quan trọng. Nạp chồng toán tử được các lập trình viên rất hay sử dụng để chương trình trở nên có nghĩa hơn.
  1. Phân biệt phương thức tĩnh, ảo, thuần ảo
* Phương thức tĩnh:
* Phương thức tĩnh có thể được gọi ngay cả khi không có đối tượng nào của lớp tồn tại, hàm tĩnh được truy cập chỉ bằng cách dùng tên lớp và toán tử phạm vi.
  + - 1 phương thức tĩnh chỉ có thể truy cập các thành phần dữ liệu tĩnh. Có thể sử dụng phương thức tĩnh để xác định xem 1 số đối tượng của lớp đã được tạo ra hay không.
* Quy tắc gọi hàm tĩnh:
  + - Nếu lời gọi xuất phát từ 1 đối tượng của lớp nào thì hàm thành phần của lớp đó sẽ đc gọi.
  + Nếu lời gọi xuất phát từ 1 con trỏ kiểu lớp nào, thì hàm thành phần của lớp đó sẽ đc gọi bất kể con trỏ chứa địa chỉ của đối tượng nào.

Ví dụ:

A\*p,\*q,\*r; //p,q,r là các con trỏ kiểu A.

A a; // a là đối tượng kiểu A.

B a; // b là đối tượng kiểu B.

C c; // c là đối tượng kiểu C.

Phép gán con trỏ: con trỏ của lớp cơ sở có thể dùng để chứa địa chỉ các đối tượng của lớp dẫn xuất.

p=&a;

q=&b;

r=&c;

xét các lời gọi phương thức từ các con trỏ p,q,r:

p -> xuat();

q -> xuat();

r -> xuat();

-> Cả 3 câu lệnh trên đều gọi tới phương thức A::xuat(), vì các phương thức ở trên là phương thức tĩnh.

* Phương thức ảo:

- Những phương thức ở lớp cha được khai báo với từ khóa “virtual”- thì các lớp con kế thừa từ nó cũng sẽ cài đặt lại các phương thức “trùng tên” để cho phép có thể nạp chồng lại được.

- Tác dụng từ khóa virtual:

+ cho phép các lớp con nạp chồng lại đc phương thức có trùng tên của lớp cha(tức là ta new ra đối tượng con nào thì khi trỏ tới phương thức trùng tên, nó sẽ hiểu là gọi vào phương thức của đối tượng con đó).

+ nếu lớp cha ko cài đặt từ khóa virtual thì new ra bất kỳ lớp con nó luôn luôn vẫn gọi về lớp cha.

- Cú pháp: <virtual> <kiểu dữ liệu> <tên phương thức>;

* Phương thức thuần ảo:

- Những phương thức sẽ được khai báo ở lớp cha - còn việc định nghĩa sẽ do các lớp con kế thừa từ nó đảm nhận.

- Cú pháp: <virtual> <kiểu dữ liệu> <tên phương thức> (các phương thức tryền vào nếu có).

1. Kế thừa là gì có mấy loại kế thừa, mục đích của kế thừa. cho ví dụ

* Kế thừa (Inheritance) là Cho phép xây dựng một lớp mới dựa trên các định nghĩa của một lớp đã có.
* Có 5 loại kế thừa:

-Đơn kế thừa (Single Inheritance): nghĩa là một lớp chỉ được kế thừa từ đúng một lớp khác. Hay nói cách khác, lớp con chỉ có duy nhất một lớp cha.

-Đa kế thừa (Multiple Inheritance): là một tính năng của ngôn ngữ C++. Trong đó một lớp có thể kế thừa từ nhiều hơn một lớp khác. Nghĩa là một lớp con được kế thừa từ nhiều hơn một lớp cơ sở.

-Kế thừa đa cấp (Multilevel Inheritance): Trong kiểu kế thừa này, một lớp dẫn xuất được tạo từ một lớp dẫn xuất khác.

-Kế thừa phân cấp (Hierarchical Inheritance): Trong kiểu kế thwufa này, sẽ có nhiều hơn một lớp con được kế thừa từ một lớp cha duy nhất.

-Kế thừa lai (Kế thừa ảo) – Hybrid (Virtual) Inheritance: được thực hiện bằng cách kết hợp nhiều hơn một loại kế thừa.

* Mục đích của kế thừa:

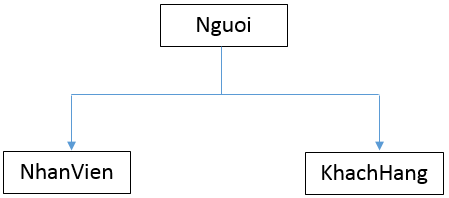
-Sử dụng lại code.

-Giảm số code cần thiết.

-Dễ bảo trì, sửa đổi về sau.

-Rõ ràng hơn về mặt logic trong thiết kế chương trình.

* Ví dụ: Một đối tượng NhanVien “là một” loại thuộc lớp NGƯỜI



* 1. Đa hình là gì. Cho ví dụ
* **Đa hình (polymorphism)** nghĩa là có nhiều hình thái khác nhau, là hiện tượng mà các đối tượng thuộc các class khác nhau có thể biểu diễn cùng một thông điệp theo các cách lhasc nhau. Tiêu biểu là, đa hình xuất hiện khi có một cấu trúc cấp bậc của các lớp và chúng là liên quan với nhau bởi tính kế thừa.

- Phương thức của lớp cha khi thực hiện sẽ được thay thế bằng một phương thức của lớp con thì phương thức này gọi là có tính đa hình.

- Tính đa hình giúp cho việc lập trình đơn giản và dễ mở rộng. Để cài đặt phương thức có tính đa hình ta dùng phương thức ảo và phương thức thuần ảo.

* Ví dụ: Hai con vật là con cho và con mèo, hai con vật này đều phát ra tiếng nhưng con mèo sẽ kêu “meo meo” còn con chó thì sủa “gâu gâu”.Hành động phát ra tiếng này tuy là một hành động nhưng khi được hai đối tượng khác nhau là chó và mèo thực hiện thì lại khác nhau.
  1. Interface là gì? Cho ví dụ
* Giao diện (interface): là tập các hàm, các luật đưa ra bắt lớp con kế thừa, tuân thủ theo.Nó là một chức năng mà ta có thể thêm và bất kì class nào. Chức năng ở đây không đồng nghĩa với phương thức (hoặc hàm).
* Interface có thể bao gồm nhiều hàm/phương thức và tất cả chúng cùng phục vụ cho một chức năng.
* Như vậy, Interface được dùng để mô tả một "bản thiết kế" cho một chức năng của class.
* Interface không phải là lớp. Các lớp dẫn xuất từ 1 Interface đều phải định nghĩa đầy đủ các phương thức và thuộc tính kế thừa từ Interface.
* Mục đích của một Interface là để định nghĩa những khả năng mà chúng ta muốn có trong một lớp, đảm bảo tính nhất quán, đồng bộ.
* Ví dụ:

interface class A {  
...  
}  
==> lớp A là một giao diện (interface);+Không thể tạo ra một thực thể của A; +A khai báo những phương thức mà không hiện thực (tức là không có thân của phương thức);

+Không thể hiện thực 1 phương thức trong A, A không chứa thuộc tính (biến thành viên).

# **PHẦN 2: BÀI TẬP**

* **Câu 23**: **Viết chương trình thực hiện các yêu cầu sau:**

1. Khai báo lớp PS1 cho các đối tượng là phân số với các thuộc tính: tử số, mẫu số.

Xây dựng phương thức nhập phân số (mẫu số khác 0), in phân số, tối giản phân số.

1. Xây dựng lớp PS2 kế thừa từ lớp PS1 và bổ sung:

Nạp chồng các toán tử: = (gán), < (nhỏ hơn), + (cộng).

1. Viết chương trình chính ứng dụng lớp PS2 để nhập một danh sách các đối tượng là các phân số (tối đa 10 phần tử). Tìm phân số có giá trị lớn nhất, tính tổng các phân số trong danh sách có giá trị nhỏ hơn 1/2.

Bài làm

#include<iostream>

#include<iomanip>

#include<math.h>

using namespace std;

class PS1{

protected:

int ts, ms;

public:

void nhap();

void xuat();

void toiGian();

};

void PS1::nhap(){

cout<<"Nhap tu so : ";

cin>>ts;

do{

cout<<"Nhap mau so : ";

cin>>ms;

if(ms==0){

cout<<"Nhap mau so khac 0!";

}

}while(ms==0);

}

int UCLN(int a, int b)

{

a=abs(a);

b=abs(b);

while(a!=b)

{

if(a>b)

a=a-b;

else

b=b-a;

}

return a;

}

void PS1::toiGian(){

int uc = UCLN(ts , ms);

this->ts = ts/uc;

this->ms = ms/uc;

this->xuat();

}

void PS1::xuat(){

if(ms < 0){

if(ms==-1)

cout<<-(ts);

else if(ts==ms)

cout<< ts/ms;

else

cout<<-(ts)<<"/"<<-(ms);

}else{

if(ms==1)

cout<<ts;

else if(ts==ms)

cout<<(float)ts/ms;

else

cout<<ts<<"/"<<ms;

}

}

class PS2:public PS1{

public:

PS2 operator = (PS2 a){

this->ts = a.ts;

this->ms = a.ms;

return \*this;

}

bool operator<(PS2 a){

if((float)ts/ms < (float)a.ts/a.ms)

return true;

else

return false;

}

bool operator>(PS2 a){

if((float)ts/ms > (float)a.ts/a.ms)

return true;

else

return false;

}

PS2 operator+(PS2 a){

PS2 kq;

kq.ts = ts\*a.ms + a.ts\*ms;

kq.ms = ms\*a.ms;

return kq;

}

bool operator<(float a){

if((float)ts/ms < a)

return true;

else

return false;

}

float operator+(float a){

return (float)ts/ms + a;

}

};

void nhapps(PS2 \*ps, int n){

for(int i = 0; i < n; i++){

cout<<"\nNhap phan so thu ["<<i+1<<"] :"<<endl;

ps[i].nhap();

}

}

void xuatps(PS2 \*ps, int n){

for(int i = 0; i < n; i++){

cout<<"\nPhan so thu ["<<i+1<<"] :"<<endl;

ps[i].toiGian();

}

}

void Max(PS2 \*ps, int n){

PS2 Max = ps[0];

int vt =0;

for(int i = 0; i < 4; i++){

if(ps[i] > Max)

vt = i;

}

cout<<"\nPhan so lon nhat la : ";

ps[vt].toiGian();

}

void Sum(PS2 \*ps, int n){

float sum = 0;

for(int i = 0 ; i < n; i++){

if(ps[i] < (float)1/2)

sum = ps[i] + sum;

}

cout<<"\nTong cac phan so nho hon 1/2 la : "<<sum;

}

int main(){

int n;

cout<<"Nhap so phan so: "; cin>>n;

PS2 \*ps = new PS2[10];

nhapps(ps,n);

xuatps(ps,n);

Max(ps,n);

Sum(ps,n);

}