Giới thiệu lập trình hướng đối tượng (OOP) với C++

- C++ là ngôn ngữ "lai" giữa lập trình cấu trúc và lập trình hướng đối tượng.

- Lập trình hướng đối tượng (OOP- Object-Oriented Programming) là một cách tư duy mới, tiếp cận hướng đối tượng để giải quyết vấn đề bằng máy tính. Thuật ngữ OOP ngày càng trở nên thông dụng trong lĩnh vực công nghệ thông tin.

- Lập trình hướng đối tượng (OOP) là một phương pháp thiết kế và phát triển phần mềm dựa trên kiến trúc lớp và đối tượng.

- Trong lập trình hướng đối tượng trong mỗi chương trình chúng ta có một số các đối tượng (object) có thể tương tác với nhau , thuộc các lớp (class) khác nhau, mỗi đối tượng tự quản lý lấy các dữ liệu của riêng chúng.

Khai báo lớp (class), đối tượng (object)

**i) Đối tượng (object)**

Đối tượng là một thực thể phần mềm bao bọc các thuộc tính và các phương thức liên quan.

**ii) Lớp (class)**

  Lớp (class) là một thiết kế (blueprint) hay một mẫu ban đầu (prototype) định nghĩa các thuộc tính và các phương thức chung cho tất cả các đối tượng của cùng một loại nào đó.

  Một đối tượng là một thể hiện cụ thể của một lớp.

**iii) Khai báo lớp và đối tượng**

***a. Khai báo lớp***

+ Lớp gồm 2 thành phần:

  - Thuộc tính (attribute)

  - Phương thức (method)

+ Khai báo:

class <Tên lớp>

{

public:

(danh sách các thành viên [bao gồm cả thuộc tính và phương thức] có đặc tính công cộng)

private:

(danh sách các thành viên [bao gồm cả thuộc tính và phương thức] có đặc tính riêng tư)

protected:

(danh sách các thành viên [bao gồm cả thuộc tính và phương thức] có đặc tính bảo tồn)

}; // dấu chấm phẩy chấm dứt câu lệnh

Copy

+ Việc sử dụng lớp trong một chương trình C++ có hai phần chính là phần định nghĩa lớp và phần khai báo và truy cập các thành viên của một đối tượng có kiểu là một lớp cho trước.

**Ví dụ:**

class SV {

// thuoc tinh

private:

char hoTen[30];

int tuoi;

float diem;

// phuong thuc

public:

void NhapDiem();

float TinhDiem();

};

Copy

***b. Khai báo đối tượng***

<Tên lớp> <tên đối tương> = <Tên lớp> ([Tham số khởi tạo]);

**Ví dụ:**

// t là đối tượng của lớp SV

SV sv1 = SV();

Copy

Phương thức khởi tạo (**constructor**)

Các phương thức thiết lập có nhiệm vụ thiết lập thông tin ban đầu cho một đối tượng thuộc về lớp ngay khi đối tượng được khai báo.

+ Đặc điểm của phương thức thiết lập:

  - Tên của phương thức thiết lập trùng với tên lớp.

  - Phương thức thiết lập không có giá trị trả về.

  - Một lớp có thể có nhiều phương thức thiết lập khác nhau.

  - Trong quá trình tồn tại của đối tượng, chỉ có một lần duy nhất mà phương thức thiết lập được gọi thực hiện mà thôi đó là khi đối tượng ra đời.

  - Phân loại phương thức thiết lập: Có thể chia phương thức thiết lập thành 3 loại

  - Phương thức thiết lập mặc định (default contructor): là phương thức thiết lập không nhận tham số đầu vào. Các thông tin ban đầu cho đối tượng của lớp bằng những giá trị mặc định (do lập trình viên quy định). Phương thức thiết lập mặc định không có đối số.

  - Phương thức thiết lập sao chép (copy contructor): là phương thức thiết lập nhận tham số đầu vào là 1 đối tượng thuộc cùng 1 lớp. Các thông tin ban đầu của đối tượng sẽ hoàn toàn giống thông tin của đối tượng tham số đầu vào.

  - Phương thức thiết lập có tham số: là phương thức thiết lập không thuộc 2 loại trên. Các thông tin ban đầu của đối tượng sẽ phụ thuộc vào giá trị các tham số của phương thức thiết lập.

**Ví dụ:**Xây dựng class CPhanSo

class CPhanSo

{

private:

int tu;

int mau;

public:

CPhanSo();

CPhanSo(const CPhanSo&);

CPhanSo(int);

CPhanSo(int,int);

};

// Khởi tạo mặc định

CPhanSo::CPhanSo()

{

tu = 0;

mau = 1;

}

//Khởi tạo sao chép

CPhanSo::CPhanSo(const CPhanSo& x)

{

tu = x.tu;

mau = x.mau;

}

//Khởi tạo 1 tham số

CPhanSo::CPhanSo(int t)

{

tu = t;

mau = 1;

}

// Khởi tạo 2 tham số

CPhanSo::CPhanSo(int t, int m)

{

tu = t;

mau = m;

}

Copy

Phương thức hủy (**destructor**)

Phương thức hủy có nhiệm vụ thu hồi lại bộ nhớ được cấp phát cho đối tượng thuộc lớp ngay khi đối tượng hết phạm vi hoạt động.

+ Các đặc điểm của phương thức hủy:

- Phương thức hủy có tên trùng với tên lớp (có thêm dấu ~ phía trước).

- Phương thức hủy không có giá trị trả về và cũng không nhận tham số đầu vào.

- Mỗi lớp chỉ có nhiều nhất một phương thức hủy. Nếu lập trình viên không xây dựng hàm hủy thì trình biên dịch sẽ tạo ra một phương thức hủy mặc định.

- Phương thức hủy chỉ được tự động gọi thực hiên một lần duy nhất (khi đối tượng hết phạm vi hoạt động).

- Phương thức hủy phải được xây dựng khi trong phương thức khởi tạo, ta có sử dụng các hàm cấp phát bộ nhớ thì ta bắt buộc phải xây dựng phương thức hủy để thu hồi các bộ nhớ này lại.

**Ví dụ:**

class CString

{

private:

char \*\_text;

int size;

public:

CString(char \*ch);//Phương thức khởi tạo

~CString(); // Phương thức hủy

};

// Định nghĩa phương thức khởi tạo

CString::CString( char \*ch )

{

size = strlen(ch) + 1;

//Cấp phát bộ nhớ cho biến \_text

\_text = new char[size];

if(\_text)

strcpy( \_text, ch);

}

// Định nghĩa phương thức hủy

CString::~CString()

{

// Thu hồi bộ nhớ cấp phát cho biến \_text

if (\_text)

delete[] \_text;

}

Copy

Kế thừa (**inheritance**)

- Kế thừa là một đặc trưng quan trọng trong lập trình hướng đối tượng (OOP). Sự kế thừa trong các ngôn ngữ lập trình như C++, Java, C#, VB.Net,.. cơ bản là giống nhau.

- Trong lập trình C++, có thêm khái niệm đa kế thừa mà các ngôn ngữ khác đã bỏ chức năng này do sự nhập nhằng khi sử dụng.

- Kế thừa là cách tạo lớp mới từ các lớp đã được định nghĩa từ trước.

**Ví dụ:**

// class B kế thừa class A.

class A{

....

};

class B:A{

...

};

Copy

\* ***Lưu ý:***

  - Một lớp cha có thể có nhiều lớp con (có phép kế thừa).

  - Đến lượt mình mỗi lớp con lại có thể có các con khác.

  - Trong C++ cho phép đa kế thừa (một lớp con có thể nhận hơn 1 lớp cha).

Phương thức ảo (**virtual**)

- Phương thức ảo là phương thức được định nghĩa ở lớp cơ sở (lớp cha) mà các lớp dẫn xuất (lớp con) muốn sử dụng phải định nghĩa lại. Dùng

từ khoá virtual để khai báo phương thức ảo:

- Cú pháp:

virtual <kiểu DL> <tên P.Thức>([tham số])

{

…

}

Copy

-  Phương thức khởi tạo không được là phương thức ảo nhưng phương thức hủy bỏ có thể là phương thức ảo.

- Dùng phương thức ảo chậm hơn phương thức thông thường vì khi thực hiện mới được xác định cụ thể.

Phương thức trừu tượng, và lớp trừu tượng (**abstract**)

  - Phương thức trừu tượng (thuần ảo) là phương thức ảo nhưng không có lệnh (phương thức rỗng). Phương thức thuần ảo có dạng:

  - Cú pháp:

abstract <kiểu DL> <tên ph.thức>([tham số]) = 0;

Copy

  - Lớp có phương thức ảo gọi là lớp trừu tượng (abstract class). Nếu một lớp thừa kế lớp trừu tượng mà không định nghĩa lại phương thức thuần ảo thì lớp thừa kế cũng là lớp trừu tượng.

\* **Lưu ý**:

  - không được tạo đối tượng thuộc lớp trừu tượng.

  - Thường chọn phương thức ở lớp cha, mà chưa thể xác định cách thực hiện, làm phương thức  thuần ảo. Ở lớp con ta sẽ định nghĩa lại

phương thức thuần ảo, để xác định cụ thể cách thức thực hiện.

Lớp bạn, hàm bạn (**friend**)

Trong khi viết các chương trình trong C++, đôi khi chúng ta cần cấp quyền truy xuất cho một hàm tới các thành viên không là các thành viên chung của một lớp. Một truy xuất như thế được thực hiện bằng cách khai báo hàm như là bạn của lớp. Có hai lý do có thể cần đến truy xuất này là:

- Có thể là cách định nghĩa hàm chính xác.

- Có thể là cần thiết nếu như hàm cài đặt không hiệu quả.

- Cách viết một hàm bạn: Để một hàm trở thành hàm bạn, có hai cách viết:

***Cách 1***: Đặt từ khóa friend trước các hàm được khai báo trong lớp, sau đó xây dựng hàm bên ngoài như bình thường.

class A

{

private:

//Cac thuoc tinh

public:

friend void h1(...);

friend int h2(...);

....

};

void h1(...)

{

...

}

int h2(...)

{

...

}

Copy

***Cách 2***: Dùng từ khóa friend để xây dựng hàm trong định nghĩa lớp.

class A

{

private:

//Cac thuoc tinh

public:

friend void h1(...)

{

...

}

friend int h2(...)

{

...

}

};

Copy

  - Tính chất của hàm bạn:

  + Trong thân hàm bạn của một lớp có thể truy nhập tới các thuộc tính của các đối tượng thuộc lớp này. Đây là khác nhau duy nhất giữa hàm bạn và hàm thông thường.

  + Hàm bạn không phải là phương thức của một lớp, lời gọi của hàm bạn giống như lời gọi của hàm thông thường.

Ví dụ 1:

+ **Yêu cầu**:

Xây dựng lớp cha (Nguoi), lớp con (HocSinh).

+ **Code**:

#include<conio.h>

#include<stdio.h>

#include<iostream>

#include<string.h>

using namespace std;

// lop cha

class Nguoi

{

protected:

char hoTen[30];

int namSinh;

private:

char soThich[50]; // so thich

public:

// khoi tao lop Nguoi

Nguoi(){

strcpy(hoTen,"Nguyen Van A");

namSinh=1990;

strcpy(soThich,"Lap trinh");

}

// In thông tin

void InTT();

};

// lop con

class HocSinh : public Nguoi

{

protected:

int maHS;

public:

void Nhap();

void InTT();

};

// InTT cua cha (Nguoi)

void Nguoi::InTT()

{

cout << "\n Nam sinh : " << namSinh;

cout << "\n Ho ten : " << hoTen;

cout << "\n So thich : " << soThich;

}

// InTT cua con (HocSinh)

void HocSinh::InTT()

{

cout << "\n Ma hoc sinh : " << maHS;

cout << "\n Nam sinh : " << namSinh;

cout << "\n Ho ten : " << hoTen;

}

// Nhap thong tin

void HocSinh::Nhap()

{

cout << "\n Nam sinh : "; cin>> namSinh;

cout << "\n Ho ten : "; gets(hoTen);

cout << "\n Ma hoc sinh: "; cin>> maHS;

}

// Ham main

int main(){

// khai bao doi tuong Nguoi

Nguoi a; a.InTT();

// khai bao doi tuong HocSinh

HocSinh t; t.Nhap(); t.InTT();

return 0;

}

Copy

Ví dụ 2:

+**Yêu cầu**: Sử dụng phương thức ảo.

+ **Code**:

#include<conio.h>

#include<stdio.h>

#include<iostream>

#include<string.h>

using namespace std;

class A

{

public:

//phuong thuc ao

virtual void Chao()

{

cout<<"\nA chao cac ban";

}

};

class B:public A

{

public:

void Chao()

{

cout<<"\nB chao cac ban";

}

};

class C:public A

{

public:

void Chao()

{

cout<<"\nC chao cac ban";

}

};

// Ham main

int main()

{

A a;

A \*pa= new A;

pa->Chao(); //goi chao cua A

B b;

pa=&b;

pa->Chao(); //goi chao cua B

C c;

pa=&c;

pa->Chao(); //goi chao cua C

return 0;

}

Copy

/\* Giải thích:

- Phương thức Chao() có tính đa hình: cùng lời gọi pa->chao() nhưng lần 1 gọi chao cua A, lan 2 gọi chao cua B, lan 3 goi chao cua C.

- Nếu trong lớp B, C không định nghĩa lại phương thức chào thì cả ba lần đều gọi chào của A.

- Nếu phương thức chao() trong lớp A không khai báo virtual thì phương thức chao() sẽ không có tính đa hình, khi đó cả ba lần đều gọi chào của A.

- Có thể gán địa chỉ của đt thuộc lớp con vào biến con trỏ, trỏ tới đt thuộc lớp cha nhưng không thể làm ngược lại (áp dụng nguyên tắc “con gán vào cha” đối với biến kiểu đối tượng hoặc biến kiểu con trỏ, trỏ tới đối tượng)

\*/

Ví dụ 3:

+**Yêu cầu**: Nhập một danh sách gồm giảng viên và sinh viên, in ra danh sách những người được thưởng. Biết rằng điều kiện được thưởng là giảng viên có số bài báo >3, sinh vien có điểm thi tốt nghiệp >8.

+ **Code**:

#include<conio.h>

#include<stdio.h>

#include<iostream>

#include<string.h>

using namespace std;

class Nguoi

{

char hoten[30];

public:

//phuong thuc ao

virtual void nhap()

{

cout<<"\nHo ten:"; cin.getline(hoten,30);

}

//phuong thuc thuc ao (truu tuong)

virtual int thuong()=0;

//phuong thuc ao

virtual void xuat()

{

cout<<"\nHo ten:"<<hoten;

}

};

// class sinhvien

class Sinhvien:public Nguoi

{

float dttn;

public:

//dinh nghia lai phuong thuc nhap

void nhap()

{

Nguoi::nhap();

cout<<"\nDiem thi tn:"; cin>>dttn;

}

//dinh nghia lai phuong thuc thuong

int thuong()

{

return (dttn>8?1:0);

}

void xuat()

{

cout<<"\n-Sinh vien:";

Nguoi::xuat();

cout<<"\nDiem thi tn:"<<dttn;

}

};

class Giangvien:public Nguoi

{

int sobaibao;

public:

void nhap()

{

Nguoi::nhap();

cout<<"\nSo bai bao:"; cin>>sobaibao;

}

int thuong()

{

return (sobaibao>3?1:0);

}

void xuat()

{

cout<<"\n-Giang vien:";

Nguoi::xuat();

cout<<"\nSo bai bao:"<<sobaibao;

}

};

// Ham main

int main()

{

Nguoi \*ds[100]; int k=0, chon, i;

while(1)

{

cout<<"\n\*Chon: Gv / Sv / Close (1,2,3):";

cin>>chon; cin.get();

if (chon==3) break;

if (chon==1) ds[k]=new Giangvien();

if (chon==2) ds[k]=new Sinhvien();

ds[k]->nhap(); k++;

}

cout<<"\n\*Danh sach nhung nguoi duoc thuong";

for (i=0; i<k; i++)

if (ds[i]->thuong()) ds[i]->xuat();

return 0;

}

Copy

Ví dụ 4: (hàm bạn, lớp bạn)

+ **Yêu cầu**: Viết hàm Area để tính diện tích hình chữ nhật, Area là hàm bạn của lớp Rectangle

+ **Code**:

#include<conio.h>

#include<stdio.h>

#include<iostream>

#include<string.h>

using namespace std;

class Rectangle

{

private:

int w;

int h;

public:

Rectangle(int, int);

// khai bao ham ban Area

friend int Area(Rectangle);

};

Rectangle::Rectangle(int w, int h){

this->w = w;

this->h = h;

}

// ham ban

int Area(Rectangle rec){

return (rec.w\*rec.h);

}

// ham main

int main()

{

Rectangle rec(2, 5);

cout<<"Dien tich HCN: "<<Area(rec);

return 0;

}