**Classes and object**

Định nghĩa class trong C++

Ví dụ: class người sẽ được khai báo như sau:

class Nguoi {

public://định danh truy cập

char ten[40];

char diachi[40];

char soDT[40];

};

Định nghĩa class kết thúc bằng dấu chấm phẩy(;).

Public: cho phép truy cập dữ liệu và hàm thành phần theo sau nó bởi đối tượng class bất cứ nơi đâu kể cả bên ngoài class

Định nghĩa đối tượng trong C++

Vd: Nguoi nguoi1, nguoi2;

Truy cập dữ liệu thành phần của class.

Ví dụ:

void main() {

Nguoi nguoi1;

cin.getline(nguoi1.ten, 20);

cin.getline(nguoi1.diachi, 40);

cout << "\nHo Ten: " << nguoi1.ten;

cout << "\nDia Chi: " << nguoi1.diachi;

system("pause");

}

Để định nghĩa hàm, có thể định nghĩa bên trong thân class:(như bình thường mình làm)

Định nghĩa bên ngoài class: sử dụng dấu :: trước tên hàm, sau tên class

char\* Nguoi::getName(){

return ten;

}

char \*Nguoi::getAddr(){

return diachi;

}

* Lưu ý quan trọng: ta phải sử dụng tên class trước định nghĩa hàm thành phần class khi định nghĩa nó ngoài class.
* Việc gọi hàm được thực hiện bằng cách đặt dấu chấm (.) giữa tên đối tượng và tên hàm.

Ví dụ:

Nguoi1.getName();

Nguoi1.getAddr();

**Class Access Modifiers**

ẩn dự liệu là một phương diện quan trọng trong lập trình hướng đối tượng, việc này cấm các hàm của chương trình truy cập trực tiếp vào các kiểu dữ liệu ngầm định của class

Quyền truy cập xác định bằng: public, protected,private thân class. Các từ khóa này đc gọi là access specifiers. Một class có thể có nhiều nhãn

Mặc định của class sẽ để khả năng truy cập của thành phần của class ở chế độ private nếu ta không khai báo nhãn định danh truy cập.

* Vùng ảnh hưởng của các nhãn

class Base {

public:

// public members go here

protected:

//protected members go here

private:

//private members go here

};

**Các thành phần public:**

Các thành phần hàm cùng dữ liệu ở chế độ này có thể đc truy cập một cách trực tiếp tại bất cứ đâu kể cả ngoài class nhưng trong khuôn khổ chương trình bởi đối tượng class đó.

Bạn có thể thiết lập hoặc lấy giá trị thành phần một cách trực tiếp không qua bất kì một hàm thành phần nào.

**Các thành phần private:**

Các dữ liệu thành phần và hàm thành phần ở chế độ private sẽ không thể bị truy cập hay xem xét từ bên ngoài class và hàm bạn – friend funtions có khả năng truy cập chúng

* Khi ta không khai báo thì trong class tự hiểu là private.

**Các thành phần protected:**

Tương tự private nhưng có một ưu điểm là các thành phần này có thể được truy cập bởi class chứa nó, class này là class dẫn xuất, sẽ học chương trình tiếp theo.

**Constructors & destructors**

Hàm tạo mặc định sẽ không có tham số:

class Nguoi {

char name[40];

char addr[40];

char phone[20];

public:

Nguoi() {//đây là hàm khởi tạo của class Nguoi

strcpy(name, "Tran Thi Thu Ha");

strcpy(addr, "Ha Noi");

strcpy(phone, "0989404047");

}

};

//khai báo bên ngoài

Nguoi::Nguoi() {

//...........

}

Trong trường hợp muốn truyền đối số vào hàm khởi tạo để gán giá trị theo mong muốn thì việc làm thêm các tham số cùng kiểu trong hàm tạo là ok.

Nguoi::Nguoi(char names[],char addrs[],char phones[] ) {//đây là hàm khởi tạo của class Nguoi

strcpy(name, names);

strcpy(addr, addrs);

strcpy(phone, phones);

}

Với hàm có tham số, ta có thể thực hiện việc khởi tạo với cú pháp như sau:

C1:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<string>

#define max 100;

using namespace std;

class Nguoi {

private:

string name;

string addr;

string phone;

public:

Nguoi(string na,string ad,string ph) {

name = na;

addr = ad;

phone = ph;

}

string getName() {

return name;

}

string getArr() {

return addr;

}

string getphone() {

return phone;

}

};

void main() {

string na = "Pham Huu Hoang";

string ad = "Ninh Thuan";

string ph = "0989 4040 47";

Nguoi n1(na,ad,ph);

cout << n1.getName() << endl;

cout << n1.getArr() << endl;

cout << n1.getphone() << endl;

system("pause");

}

C2:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<string>

#define max 100;

using namespace std;

class Nguoi {

private:

char name[40];

char addr[40];

char phone[40];

public:

Nguoi(char na[], char ad[], char ph[]) {

strcpy(name, na);

strcpy(addr, ad);

strcpy(phone, ph);

}

char \*getName() {

return name;

}

char \*getArr() {

return addr;

}

char \*getphone() {

return phone;

}

};

void main() {

char na[] = "Pham Huu Hoang";

char ad[] = "Ninh Thuan";

char ph[] = "0989 4040 47";

Nguoi n1(na, ad, ph);

cout << n1.getName() << endl;

cout << n1.getArr() << endl;

cout << n1.getphone() << endl;

system("pause");

}

hàm set và hàm get:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include<iostream>

#include<string>

#define max 100;

using namespace std;

class Nguoi {

private:

char name[40];

char addr[40];

char phone[40];

public:

Nguoi(char na[], char ad[], char ph[]) {

strcpy(name, na);

strcpy(addr, ad);

strcpy(phone, ph);

}

void setname(char na[]) {

strcpy(name, na);

}

void setarr(char ad[]) {

strcpy(addr, ad);

}

void setphone(char ph[]) {

strcpy(phone, ph);

}

void display(){

cout << getName() << endl;

cout << getArr() << endl;

cout << getphone() << endl;

}

char \*getName() {

return name;

}

char \*getArr() {

return addr;

}

char \*getphone() {

return phone;

}

};

void main() {

char na[] = "Pham Huu Hoang";

char ad[] = "Ninh Thuan";

char ph[] = "0989 4040 47";

Nguoi n1(na, ad, ph);

n1.setname("PHH");

n1.display();

system("pause");

}

**Destructors**:

Hàm hủy cũng là một hàm thành phần đặc biệt của class, nó tự động thực hiện khi một đối tượng kết thúc nhiệm vụ hoặc khi con trỏ trỏ tới class đó bị delete.

Dọn dẹp các dữ liệu đã được tạo ra trong quá trình thực thi chương trình. Chủ yếu là giải phóng tài nguyên bộ nhớ, đóng các file trước khi rời chương trình.

Hàm hủy không có kiểu trả về và không có tham số.

Định nghĩa hàm: trong học ngoài class, không nhất thiết phải có nội dung trong thân hàm hủy

Ví dụ:

~Nguoi() {

//say something

}

**Con trỏ this**

Mỗi đối tượng trong C++ có truy cập tới địa chỉ của bản thân nó thông qua một con trỏ quan trong gọi là con trỏ this.

Con trỏ this là một tham số ngầm định tới tất cả các hàm thành phần của class. Do đó, trong hàm thành phần, ta có thể sử dụng con trỏ this thay cho việc gọi đối tượng.

Hàm bạn(khai báo ngoài class) không có con trỏ this vì hàm bạn không phải là hàm thành phần của class. Chỉ có việc hàm thành phần của class mới có con trỏ this.

**Hàm bạn, lớp bạn – friend funcion & friend class:**

Hàm bạn của một class là hàm được định nghĩa ngoài vùng của một class có quyền truy cập tất cả thành phần pricate và protected của class.

Dù hàm nguyên mẫu của hàm bạn xuất hiện trong thân class nhưng hàm bạn không phải là hàm thành phần của class.

Để khai hàm bạn của class, ta đặt từ khóa friend trước kiểu trả về của hàm.

Ví dụ hàm bạn của class Nguoi:

class Nguoi {

private:

char \*name;

char \*addr;

char \*phone;

public:

friend void printName(Nguoi n) // hàm bạn

{

void setName(char \*na) {

char \*getName;

}

}

void printName(Nguoi n)//định nghĩa hàm bạn

{

cout << "\nTen: " << n.getName();

}

}

Để khai báo class lá class bạn của một class, ta khai báo thêm friend class định nghĩa class.

Ví dụ class B là bạn của class A ta khai báo:

class B {

friend class A;//A is friend class of class B

};

Khi class B là bạn của A, tất cả các hàm thành phần của B sẽ là bạn của class A.

Một class có thể là class bạn của nhiều class. Một class có thể có nhiều class bạn

Class trong C++ không có tính 2 chiều, tức là B là bạn của A nhưng điều ngược lại không đúng, để A là bạn của B thì ta phải khai báo friend class trong đinh nghĩa của B

**Hàm inline:**

* Trong C++, hàm inline có nhiều ưu điểm và thường được sử dụng phổ biến cùng với class. Nếu một hàm là hàm inline , trình biên dich sẽ đặt một bản sao code của hàm trong đó tại các vị trí mà hàm này đã được gọi chương trình.
* Bất kì thay đổi nào trong inline sẽ làm trình biên dịch phải biên dịch lại chương trình để update thông tin chính xác tương ứng với hàm này.
* Chức năng chủ yếu của hàm inline là giảm thời gian thực hiện chương trình nhưng việc sử dụng nhưng tăng kích thước chương trình. Hàm inline được áp dụng với các hàm có tần xuất sử dụng cao.
* Để một hàm là inline thì ta đặt từ khóa inline trả về của hàm.
* Trình biên dịch sẽ bỏ qua việc xác minh một hàm inline nếu nó được định nghĩa nhiều hơn 1 dòng.
* Các hàm thành phần của class được định nghĩa khi ta định nghĩa một hàm class được ngầm định là hàm inline cho dù nó không có khóa inline trước kiểu trả về của hàm trong định nghĩa hàm.

**Con trỏ tới class:**

* Tương tự như con trỏ tới cấu trúc, ta cũng tiến hành khai báo, bởi các thành phần của class như bình thường.
* Sự khác biệt ở đây là chúng ta sẽ sử dụng toán tử truy cập mũi tên thay vì dấu chấm.
* Ví dụ: với class Nguoi, trong hàm main ta có thể xuất thông tin ra màn hình qua con trỏ tới đối tượng như sau:

Nguoi \*n1;

Cout<<n1->getName( );//xuất tên

Cout<<n1->getAddr( );

**Các thành phần static:**

* Chúng ta có thể định nghĩa thành phần của class ở kiểu static bằng đặt từ khóa static trước kiểu dữ liệu trả về của nó trong class.
* Bằng việc khai báo một thành phần của class là static, ta ngầm định rằng, dù có bao nhiêu đối tượng được tạo thì cũng chỉ có duy nhất một bản sao của thành phần static.
* Thành phần static của class được dùng chung mọi đối tượng của class chứa nó, dữ liệu kiểu static sẽ được mặc định khởi tạo là 0 khi đối tượng đầu tiên được tạo ra nếu không chủ định khởi tại cho một giá trị khác.
* Ta không khởi tạo giá trị cho biến static trong class nhưng có thể làm vậy ở ngoài class thông qua **dấu ::** sau tên class, trước tên biến static.

**Trong class:**

class Nguoi {

char \*name;

char \*addr;

char \*phone;

public:

static int objectCount;//định nghĩa static

Nguoi() {

//hàm tạo

}

~Nguoi{

//hàm hủy

}

};

**Ngoài class:**

int Nguoi::objectCount = 0;

**Hàm thành phần kiểu static:**

* Bằng cách khai báo một hàm là static, bạn hàm đó khả năng độc lập với tất cả các đối tượng của class. Hàm static có thể được gọi ngay cả khi không có bất kì đối tượng nào tồn tại.
* Để truy cập hàm static của class, đung tên class:: tên hàm.
* Hàm thành phần static có thể truy cập được các biến và hàm thành phần khác cùng kiểu trong class và các hàm khác ngoài phạm vi class.
* Hàm thành phần static thuộc phạm vi class nhưng nó không có truy cập tới con trỏ this của class.
* Bạn có thể sử dụng hàm thành phần static để kiểm tra xem các đối tượng của class đã được tạo hay chưa.
* Ví dụ: dùng hàm static numberObject(); để trả về số các đối tượng của class được tạo.

//trong thân class: