**LÍ THUYẾT BÀI TẬP LỚN C++**

1.Lớp trong C++ là gì ?

* Trong C++, Lớp (class) là một nhóm các đối tượng tương tự. Nó là một khuôn mẫu mà từ đó các đối tượng được tạo ra
* Khai báo lớp :

class <tên\_lớp>

{

[quyền truy xuất:]

//khai báo các thành phần dữ liệu (thuộc tính) của lớp

[quyền truy xuất:]

//khai báo các thành phần hàm (phương thức) của lớp

};

Trong đó :

* <tên\_lớp> : do người dùng đặt , tuân các quy tắc về tên
* [quyền truy xuất:] : là khả năng truy xuất thành phần dữ liệu, ngầm định là private ( private :trong phạm vi lớp đó , public : ở mọi nơi nếu đối tượng tồn tại , protected : phạm vi lớp đó và các lớp con kế thừa )
* Khai báo các thành phần của lớp:
* Thuộc tính : <kiểu dữ liệu > <tên\_thuộc tính>;
* Phương thức :

Cách 1: Khai báo ngoài lớp và định nghĩa ngoài lớp

<kiểu trả về > tên lớp::<tên\_phương thức>([đối sô])

{

// <thân hàm>

}

Cách 2 : Khai báo trong lớp và định nghĩa ngay trong lớp

<Kiểu trả về> <Tên phương thức>([<Các tham số>]) { …}

* Khai báo đối tượng : <tên\_lớp> <tên\_đối\_tượng>;
* Thuộc tính : <tên\_đối\_tượng>.<tên\_thuộc tính>;

Nếu là con trỏ : <tên\_con\_trỏ>-><tên\_thuộc tính>;

* Phương thức : <tên\_đối\_tượng>.<tên\_hàm>([ds đối số]);

Với con trỏ: <tên\_con\_trỏ>->.<tên\_hàm>([đối số]);

2.Mảng và con trỏ đối tượng trong C++ là gì ?

* Mảng đối tượng:Khi cần khai báo một dãy các đối tượng cùng loại ta có thể sử dụng mảng đối tượng, cách khai báo và sử dụng giống mảng thông thường. Khai báo :

<tên\_lớp> <tên\_mảng>[spt];

* Con trỏ đối tượng : Khi cần cấp phát dữ liệu động ta có thể khai báo con trỏ đối tượng theo cú pháp sau:

<tên\_lớp> \*<tên\_con\_trỏ>;

Giải phóng đối tượng qua thao tác Delete : Delete <Tên đối tượng con trỏ>

* Khai báo mảng động với con trỏ :

<Tên lớp> \* <Tên biến mảng động> = new <Tên lớp> [<độ dài mảng>]

Giải phóng bộ nhớ : Delete [] <tên biến mảng động>;

3.Hàm tạo – Constructor

* Hàm tạo là một phương thức của lớp, dùng để tạo dựng một đối tượng mới.
* Chương trình dịch sẽ cấp phát bộ nhớ cho đối tượng sau đó sẽ gọi đến hàm tạo.
* Hàm tạo sẽ khởi gán giá trị cho các thuộc tính của đối tượng và có thể thực hiện một số công việc khác nhằm chuẩn bị cho đối tượng mới.
* Khai báo hàm tạo : <tên\_lớp>([ds tham số]);

Định nghĩa ngoài lớp : <tên\_lớp>::<tên\_lớp>([ds tham số])

{

//thân hàm

}

Trong đó :

* Hàm tạo phải được khai báo bên trong lớp
* Hàm khởi tạo phải có tên trùng với tên của lớp
* Không khai báo kiểu cho hàm tạo.
* Không có giá trị trả về
* Hàm khởi tạo có tính chất public
* Hàm tạo có thể có đối hoặc không có đối.
* Nếu không xây dựng thì chương trình tự động sinh hàm khởi tạo mặc định.
* Trong một lớp có thể có nhiều hàm tạo (cùng tên nhưng khác bộ đối).
* Phân loại hàm tạo : có đối số và không đối số
* Hàm khởi tạo có đối số

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

{

x=x1;

y=y1;

}

* Hàm khởi tạo không đối số

diem() //hàm tạo không đối

{

x=0;

y=0;

}

* Khai báo khởi tạo sao chép:

<tên\_lớp>(const <tên\_lớp> &<tên\_tham\_số>)

Đối tượng mới sẽ là bản sao của đối tượng đã có

Ví dụ :

diem (const diem &A) // Hàm tạo sao chép

{

x=A.x;

y=A.y;

}

* Hàm tạo có đối số mặc định.Cú pháp :

Tên\_lớp(KDL thamso1=gtr1, KDL thamso2=gtr2…)

{

// nội dung

}

Ví dụ :

diem (int x1=0, int y1=0) //hàm tạo có đối số mặc định

{

x=x1;

y=y1;

}

4.Hàm hủy – Destructor

* Nếu trong lớp không định nghĩa hàm hủy thì một hàm hủy mặc định không làm gì cả được phát sinh. Đối với nhiều lớp thì hàm hủy mặc định là đủ và không cần đưa vào một hàm hủy mới.
* Khai báo : ~<tên\_lớp>();
* Chức năng : Hủy bỏ, giải phóng các đối tượng khi nó hết phạm vi tồn tại
* Như vậy , hàm hủy
* Trùng tên với lớp và có dấu ~ ở trước
* Mỗi lớp có duy nhất một hàm hủy
* Không có đối số
* Không có giá trị trả về
* Không định nghĩa lại
* Thực hiện một số công việc trước khi hệ thống giải phóng bộ nhớ.
* Hàm khởi tạo có tính chất public
* Chương trình dịch tự động sinh hàm hủy mặc định

5.Con trỏ This

* Là con trỏ trỏ đến đối tượng đang gọi hàm (phương thức)
* Bất cứ khi nào một hàm thành phần được gọi, trình biên dịch sẽ gán địa chỉ của đối tượng đang gọi hàm thành phần đó cho con trỏ **this**
* Cho phép truy xuất các thành phần của đối tượng đang gọi hàm thành phần thông qua con trỏ **this**
* Là đối số không tường minh (Implicit argument) của lời gọi hàm thành phần không **static**

6.Toán tử chồng nhập xuất

* Nạp chồng toán tử (overload operator) là bạn định nghĩa lại toán tử đã có trên kiểu dữ liệu người dùng tự định nghĩa để dể dàng thể hiện các câu lệnh trong chương trình.
* Đối với toán tử <<, >> ta không thể xây dựng dưới dạng toán tử thông thường:
* Không thể là phương thức toán tử của lớp.
* Không thể là hàm toán tử thông thường.
* Khai báo toán tử nhập xuất phải khai báo là hàm bạn của lớp.
* Toán tử xuất dữ liệu:

ostream &operator<<(ostream &os, kiểu dữ liệu bien)

{

//Cac cau lenh xuat du lieu

return os;

}

* Toán tử nhập dữ liệu:

istream & operator>>(istream &is, kieu du lieu &bien)

{

//Cac cau lenh xuat du lieu

return is;

}

7.Mảng một chiều

* Mảng (array) trong C++ là một tập hoặc một nhóm các phần tử (dữ liệu) có kiểu dữ liệu đồng nhất(tương tự). Các phần tử của mảng được lưu trong các vùng nhớ liên tiếp.
* Mảng trong C++ là rất hữu ích nếu bạn muốn lưu trữ các phần tử tương tự. Giả sử bạn phải lưu trữ thông tin của 50 sinh viên. Bạn có thể làm điều này bằng việc sử dụng 50 biến, nhưng với cách này bạn sẽ khó quản lý vì bạn không thể truy cập giá trị của các biến này với chỉ 1 hoặc 2 dòng code.
* Lợi thế của mảng :
* Tối ưu hóa code.
* Dễ dàng để duyệt các phần tử của mảng.
* Dễ dàng sắp xếp dữ liệu.
* Truy cập ngẫu nhiên.
* Bất lợi của mảng : Kích thước cố định: Kích thước của mảng trong C++ phải được định nghĩa tại thời điểm khai báo, chúng ta không thể lưu số phần tử vượt quá kích thước này.
* Khai báo mảng : data\_type array\_name[array\_size];

Trong đó :