Class là gì?

Class hay lớp là một mô tả trừu tượng (abstract) của nhóm các đối tượng (object) có cùng bản chất, ngược lại mỗi một đối tượng là một thể hiện cụ thể (instance) cho những mô tả trừu tượng đó. Một class trong C++ sẽ có các đặc điểm sau:

- Một class bao gồm các thành phần dữ liệu (thuộc tính hay property) và các phương thức (hàm thành phần hay method).
- Class thực chất là một kiểu dữ liệu do người lập trình định nghĩa.
- Trong C++, từ khóa class sẽ chỉ điểm bắt đầu của một class sẽ được cài đặt.

Ví dụ về một class đơn giản, class Car. Một chiếc xe hơi vậy thì sẽ có chung những đặc điểm là đều có vô lăng, có bánh xe nhiều hơn 3, có động cơ… Đó là một class, một cái model hay mẫu mà người ta đã quy định là nếu đúng như vậy thì nó là xe hơi. Nhưng mà xe thì có thể có nhiều hãng khác nhau, BMW, Vinfast, Toyota… Thì mỗi hãng xe lại có những model xe khác nhau nhưng chúng đều là xe hơi. Vậy thì trong lập trình cũng vậy, class là quy định ra một mẫu, một cái model mà các thể hiện của nó (instance) hay đối tượng (object) phải tuân theo.

Khai báo class và sử dụng class

Ví dụ một class cơ bản:

```
class Person {
   public:
      string firstName; // property
      string lastName;  // property
      int age;          // property

      void fullname() { // method
         cout << firstName << ' ' << lastName;
      }
};
```

Cú pháp tạo object của một class và sử dụng các thuộc tính và phương thức:

```
Person person;

person.firstName = "Khiem";

person.lastName = "Le";

person.fullname(); // sẽ in ra màn hình là "Khiem Le"
```

Access modifiers & properties declaration

Access modifier là phạm vi truy cập của các thuộc tính và phương thức sẽ được khai báo bên dưới nó. Có 3 phạm vi truy cập trong C++ là public, private và protected.

- Các thuộc tính và phương thức khai báo public thì có thể được truy cập trực tiếp thông qua instance của class đó. Các thuộc tính nên khai báo là public nếu bạn không có ràng buộc điều kiện trước khi gán (người dùng có thể thoải mái gán giá trị) hoặc bạn không cần xử lý trước khi trả về giá trị thuộc tính;
- Các thuộc tính private thường được sử dụng khi bạn không mong muốn người khác có thể tùy ý gán giá trị hoặc là bạn muốn xử lý trước khi trả về giá trị.
- Đối với protected, các phương thức và thuộc tính chỉ có thể truy cập qua các class kế thừa nó hoặc chính nó.
-

Ví dụ của access modifier:

```
class MyClass
{
	public:
		int public_property;

	private:
		int _private_property;
};
```

Method declaration

Phương thức cũng giống như một hàm bình thường.

Đối với phương thức thì có hai cách định nghĩa thi hành: định nghĩa thi hành trong lúc định nghĩa class và định nghĩa thi hành bên ngoài class.

Định nghĩa thi hành bên trong class:

```
class Animal {
   public:
      string sound;

      void makeNoise() {
         cout << sound;
      }
};
```

Định nghĩa thi hành bên ngoài class:

```
class Animal {
   public:
      string sound;

      void makeNoise();
};

void Animal::makeNoise() {
   cout << sound;
}
```

Constructor

Constructor hay hàm dựng là một hàm đặc biệt, nó sẽ được gọi ngay khi chúng ta khởi tạo một object.

```
class Person {
        public:
            string firstName;
            string lastName;
            int age;

            Person(string _firstName, string _lastName, int _age)
            {
                firstName = _firstName;
                lastName = _lastName;
                age = _age;
            }

            void fullname() {
                        cout << firstName << ' ' << lastName;
            }
};
```

Static member

Static member hay thành viên tĩnh trong class C++ cũng tương tự như với static variable (biến tĩnh) trong function. Đối với function, sau khi thực hiện xong khối lệnh và thoát thì biến tĩnh vẫn sẽ không mất đi. Đối với class, thành viên tĩnh sẽ là thuộc tính dùng chung cho tất cả các đối tượng của class đó, cho dù là không có đối tượng nào tồn tại. Tức là bạn có thể khai báo nhiều object, mỗi object các thuộc tính của nó đều khác nhau nhưng riêng static thì chỉ có một và static member tồn tại trong suốt chương trình cho dù có hay không có object nào của nó hay nói ngắn gọn là dùng chung một biến static.

Đặc tính của lập trình hướng đối tượng

Có 4 đặc tính quan trọng của lập trình hướng đối tượng trong C++ mà chúng ta cần nắm vững sau đây.

Inheritance (Tính kế thừa) trong lập trình hướng đối tượng có ý nghĩa, một class có thể kế thừa các thuộc tính của một class khác đã tồn tại trước đó.

Khi một class con được tạo ra bởi việc kế thừa thuộc tính của class cha thì chúng ta sẽ gọi class con đó là **subclass trong C++**, và class cha chính là **superclass trong C++**.

Abstraction (Tính trừu tượng) trong lập trình hướng đối tượng là một khả năng mà chương trình có thể bỏ qua sự phức tạp bằng cách tập trung vào cốt lõi của thông tin cần xử lý.

Điều đó có nghĩa, bạn có thể xử lý một đối tượng bằng cách gọi tên một phương thức và thu về kết quả xử lý, mà không cần biết làm cách nào đối tượng đó được các thao tác trong class.

Ví dụ đơn giản, bạn có thể nấu cơm bằng nồi cơm điện bằng cách rất đơn giản là ấn công tắc nấu, mà không cần biết là bên trong cái nồi cơm điện đó đã làm thế nào mà gạo có thể nấu thành cơm.

Polymorphism (Tính đa hình) trong lập trình hướng đối tượng là một khả năng mà một phương thức trong class có thể đưa ra các kết quả hoàn toàn khác nhau, tùy thuộc vào dữ liệu được xử lý.

Ví dụ đơn giản, cùng là một class quản lý dữ liệu là các con vật, thì hành động sủa hay kêu của chúng được định nghĩa trong class sẽ cho ra kết quả khác nhau, ví dụ nếu là con mèo thì kêu meo meo, còn con chó thì sủa gâu gâu chẳng hạn.

Encapsulation (Tính đóng gói) trong lập trình hướng đối tượng có ý nghĩa không cho phép người sử dụng các đối tượng thay đổi trạng thái nội tại của một đối tượng, mà chỉ có phương thức nội tại của đối tượng có thể thay đổi chính nó.

Điều đó có nghĩa, dữ liệu và thông tin sẽ được đóng gói lại, giúp các tác động bên ngoài một đối tượng không thể làm thay đổi đối tượng đó, nên sẽ đảm bảo tính toàn vẹn của đối tượng, cũng như giúp dấu đi các dữ liệu thông tin cần được che giấu.

Ví dụ đơn giản, khi bạn dùng một cái iphone, bạn không thể thay đổi các cấu trúc bên trong của hệ điều hành iOS, mà chỉ có Apple mới có thể làm được điều này thôi.

Namespace là gì?

Tình huống:

Khi đang lập trình trong một file A bạn include 2 file B và C, nhưng 2 file này có cùng định nghĩa một hàm function() giống nhau về tên và tham số truyền vào, nhưng xử lý của mỗi hàm ở mỗi file là khác nhau, vấn đề đặt ra là code làm sao để trình biên dịch hiểu được khi nào bạn muốn gọi function của file B, khi nào bạn muốn gọi function của file C. Khi gọi hàm function() ở file A, trình biên dịch sẽ không biết được hàm function() bạn muốn gọi là hàm được định nghĩa ở file B hay file C. Vì vậy trình biên dịch chương trình sẽ báo lỗi.

Định nghĩa:

Namespace là từ khóa trong C++ được sử dụng để định nghĩa một phạm vi nhằm mục đích phân biệt các hàm, lớp, biến, ... cùng tên trong các thư viện khác nhau.

Template trong C++ là gì?

- *Template* (khuôn mẫu) là một từ khóa trong C++, và là một kiểu dữ liệu trừu tượng tổng quát hóa cho các kiểu dữ liệu int, float, double, bool...
- Template trong C++ có 2 loại đó là function template & class template.
- Template giúp người lập trình định nghĩa tổng quát cho hàm và lớp thay vì phải nạp chồng (overloading) cho từng hàm hay phương thức với những kiểu dữ liệu khác nhau.

Hàm ảo (virtual function) là gì?

Hàm ảo (virtual function) là một hàm thành viên trong lớp cơ sở mà lớp dẫn xuất khi kế thừa cần **phải định nghĩa lại**.

Hàm ảo được sử dụng trong lớp cơ sở khi cần đảm bảo hàm ảo đó sẽ được **định nghĩa lại** trong lớp dẫn xuất. Việc này rất cần thiết trong trường hợp con trỏ có kiểu là lớp cơ sở trỏ đến đối tượng của lớp dẫn xuất.

Hàm ảo là một phần không thể thiếu để thể hiện tính đa hình trong kế thừa được hỗ trợ bởi nguồn ngữ C++.

Lưu ý: Con trỏ của lớp cơ sở có thể chứa địa chỉ của đối tượng thuộc lớp dẫn xuất, nhưng ngược lại thì không được.

Hàm ảo chỉ khác hàm thành phần thông thường khi được gọi từ **một con trỏ**. Sử dụng hàm ảo khi muốn con trỏ đang trỏ tới đối tượng của lớp nào thì hàm thành phần của lớp đó sẽ được gọi mà không xem xét đến kiểu của con trỏ.

Vector là gì?

Giống như là mảng (array), vector trong C++ là một đối tượng dùng để chứa các đối tượng khác, và các đối tượng được chứa này cũng được lưu trữ một cách liên tiếp trong vector.

Tuy nhiên, nếu như số lượng phần tử (size) của một mảng là cố định, thì ở vector, nó hoàn toàn có thể thay đổi trong suốt quá trình làm việc của chương trình

Modifiers

1. **push_back()**: Hàm đẩy một phần tử vào vị trí sau cùng của vector. Nếu kiểu của đối tượng được truyền dưới dạng tham số trong push_back() không giống với kiểu của vector thì sẽ bị ném ra.

ten-vector.push_back(ten-cua-phan-tu);

2. **assign()**: Nó gán một giá trị mới cho các phần tử vector bằng cách thay thế các giá trị cũ.

ten-vector.assign(int size, int value);

3. **pop_back()**: Hàm pop_back () được sử dụng để xóa đi phần tử cuối cùng một vector.

4. **insert()**: Hàm này chèn các phần tử mới vào trước phần tử trước vị trí được trỏ bởi vòng lặp. Chúng ta cũng có thể chuyển một số đối số thứ ba, đếm số lần phần tử được chèn vào trước vị trí được trỏ.

5. **erase()**: Hàm được sử dụng để xóa các phần tử tùy theo vị trí vùng chứa

6. **emplace()**: Nó mở rộng vùng chứa bằng cách chèn phần tử mới vào

7. **emplace_back()**: Nó được sử dụng để chèn một phần tử mới vào vùng chứa vector, phần tử mới sẽ được thêm vào cuối vector

8. **swap()**: Hàm được sử dụng để hoán đổi nội dung của một vector này với một vector khác cùng kiểu. Kích thước có thể khác nhau.

9. **clear()**: Hàm được sử dụng để loại bỏ tất cả các phần tử của vùng chứa vector.

Dưới đây là một số điểm khác nhau giữa Vector và List:

Cấu trúc dữ liệu:
Vector: Lưu trữ dữ liệu dưới dạng mảng động. Các phần tử được lưu trữ liên tiếp trong bộ nhớ.
List: Lưu trữ dữ liệu dưới dạng danh sách liên kết hai chiều. Mỗi phần tử trong danh sách được lưu trữ trong một node riêng biệt, và các node được liên kết với nhau.

Truy cập phần tử:
Vector: Có thể truy cập phần tử bằng cách sử dụng chỉ số (indexing). Điều này cho phép truy cập nhanh đến phần tử cụ thể.
List: Cần duyệt qua các phần tử từ đầu đến cuối để truy cập một phần tử cụ thể. Truy cập phần tử trong List tốn thời gian hơn so với Vector.

Chèn và xóa phần tử:
Vector: Chèn và xóa phần tử tại vị trí bất kỳ trong Vector tốn thời gian, vì các phần tử phía sau cần được dịch chuyển để tạo chỗ cho phần tử mới hoặc để điền vào khoảng trống sau khi xóa.
List: Chèn và xóa phần tử tại vị trí bất kỳ trong List có hiệu suất tốt hơn so với Vector, vì không cần dịch chuyển các phần tử.

Kích thước động:
Vector: Có kích thước động, tức là có thể thay đổi kích thước của Vector theo nhu cầu bằng các phương thức như push_back() hoặc resize().
List: Có kích thước động và không có giới hạn về số lượng phần tử.