**POLYMORPHISM**

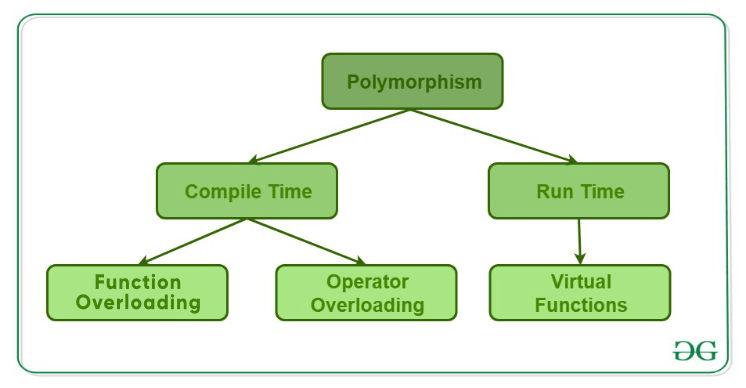
1. **Tính Đa hình**
2. **Khái niệm**

Tính đa hình cho phép các chức năng (method) giống nhau được thực thi khác nhau trên các đối tượng khác nhau.

***Ví dụ:*** 2 lớp Nhân viên và lớp Sinh viên, đều kế thừa từ lớp Người. Khi đó, ta thêm vào mỗi lớp một phương thức show():

* show() của lớp Người sẽ giới thiệu tên và tuổi của người đó.
* show() của lớp Nhân viên sẽ giới thiệu nhân viên đó có tiền lương là bao nhiêu
* show() của lớp Sinh viên sẽ giới thiệu là sinh viên đó học ở lớp nào.

1. **Phân loại**

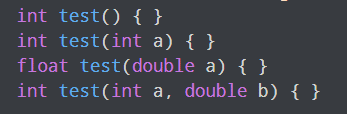


* 1. **Compile time Polymorphism (Đa hình thời gian biên dịch)**

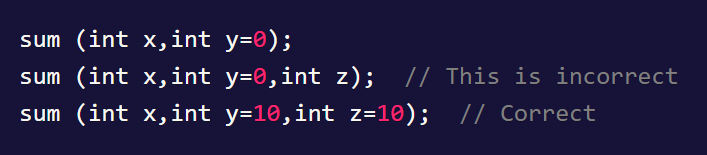
Đa hình thời gian biên dịch sử dụng phương thức nạp chồng hàm hoặc nạp chồng toán tử

1. **Function Overloading (Nạp chồng hàm)**

Khi có nhiều hàm có cùng tên nhưng khác tham số thì các hàm này được gọi là hàm nạp chồng. Các hàm có thể được nạp chồng bằng cách thay đổi số lượng đối số hoặc / và thay đổi kiểu của đối số.



* Ở đây, cả 4 hàm đều là hàm nạp chồng.
* Chú ý rằng kiểu trả về của cả 4 hàm này không giống nhau. Các hàm nạp chồng có thể có hoặc không có các kiểu trả về khác nhau nhưng chúng phải có các đối số khác nhau. Ví dụ:

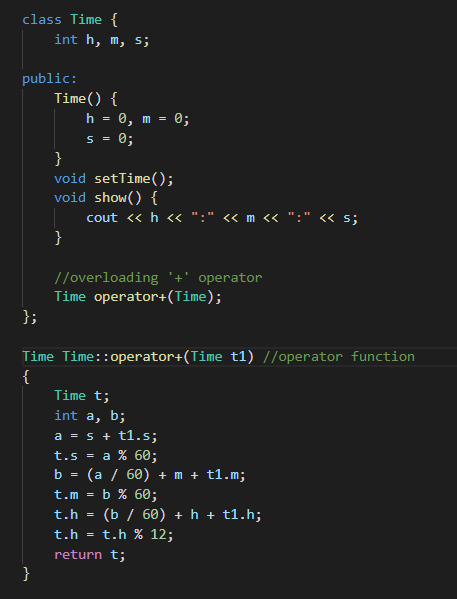


1. **Operator Overloading (Nạp chồng toán tử)**

Nạp chồng toán tử là phương pháp mà chúng ta có thể thay đổi chức năng của một số toán tử cụ thể để thực hiện một số tác vụ khác nhau.

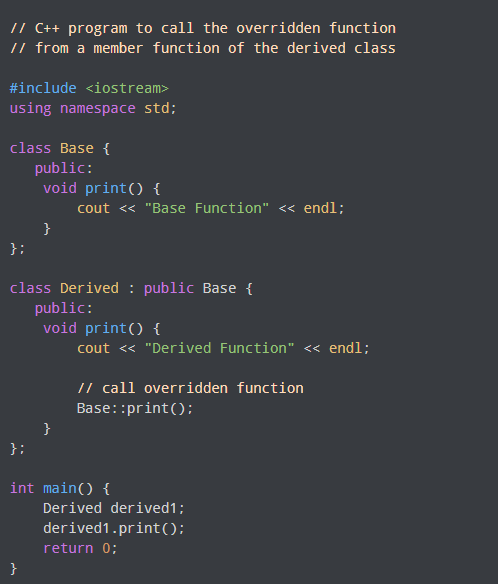
**Ví dụ:** Nạp chồng toán tử + để cộng hai đối tượng lớp Time

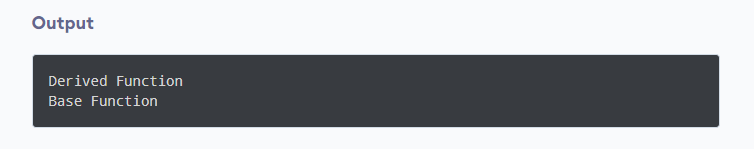
🡺Trong khi phép cộng hai số bình thường trả về kết quả tổng, chúng ta đã nạp chồng toán tử + để thực hiện phép cộng hai đối tượng lớp Time. Chúng ta thêm các giá trị giây, phút và giờ một cách riêng biệt để trả về giá trị mới của thời gian.



* 1. **Runtime Polymorphism (Đa hình thời gian chạy)**

Loại đa hình này sử dụng chức năng Ghi đè. Mặt khác, việc ghi đè hàm xảy ra khi một lớp dẫn xuất có định nghĩa cho một trong các hàm thành viên của lớp cơ sở. Hàm cơ sở đó được cho là bị ghi đè.



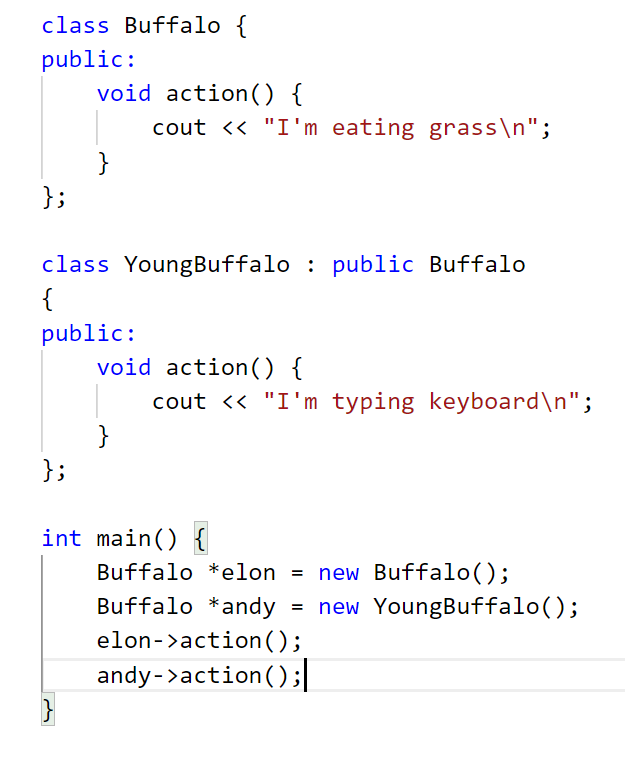


1. **Phương thức ảo – Từ khoá Virtual**

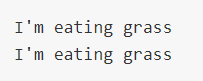
Hàm ảo là thành phần của một lớp, được khai báo trong lớp cơ sở nhưng lại được khai báo lại (ghi đè) trong lớp dẫn xuất với từ khoá **virtual** ở đằng trước.

Về cơ bản, một hàm ảo được sử dụng trong lớp cơ sở để đảm bảo rằng hàm được ghi đè. Điều này đặc biệt áp dụng cho các trường hợp con trỏ của lớp cơ sở trỏ đến một đối tượng của lớp dẫn xuất.

**Ví dụ:**

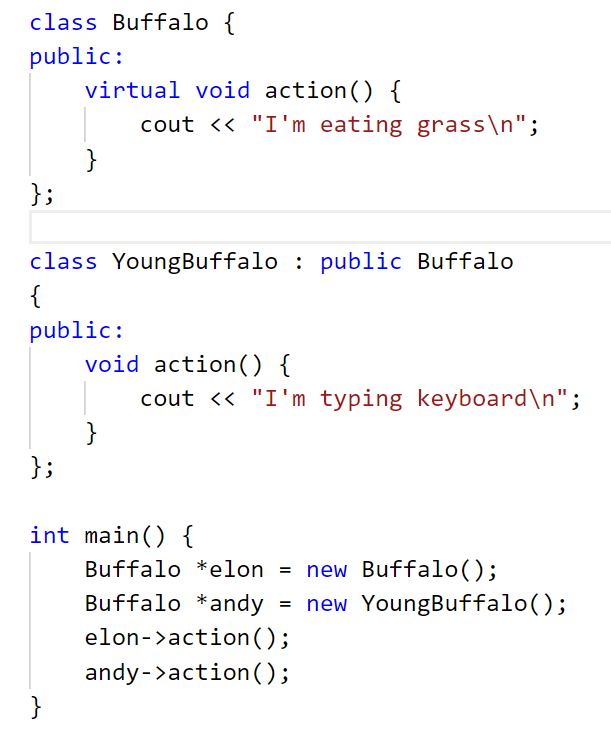
****

**Output:**

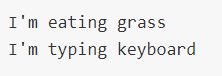
****

* Mặc dù andy được tạo ra từ constructor của class YoungBuffalo thế nhưng nó hành xử lại như thể nó là một Buffalo

**🡺** Vì vậy chúng ta sẽ fix với vấn đề này với virtual như sau:



**Output:**

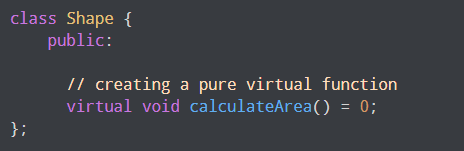


1. **Phương thức thuần ảo**
2. **Pure virtual function**

Một hàm thuần ảo (hoặc abstract function (hàm trừu tượng)) trong C ++ là một hàm ảo mà chúng ta không có phần triển khai, chúng ta chỉ khai báo nó. Một hàm thuần ảo được khai báo bằng cách gán 0 trong khai báo.

Hàm thuần ảo được sử dụng nếu một hàm không có bất kỳ công dụng nào trong lớp cơ sở nhưng hàm phải được thực thi bởi tất cả các lớp dẫn xuất của nó

**Ví dụ:** Giả sử, chúng ta tạo các lớp Triangle, Square và Circle từ lớp Shape. Chúng ta muốn tính diện tích của tất cả các hình này. Trong trường hợp này, chúng ta có thể tạo một hàm thuần có tên là CalculArea () trong Shape. Vì nó là một hàm thuần ảo, nên tất cả các lớp dẫn xuất Triangle, Square và Circle phải bao gồm hàm CalculArea () với việc triển khai.

****

* **Chú ý:** Cú pháp = 0 không có nghĩa là chúng ta đang gán 0 cho hàm. Đó chỉ là cách chúng ta định nghĩa các hàm thuần ảo.

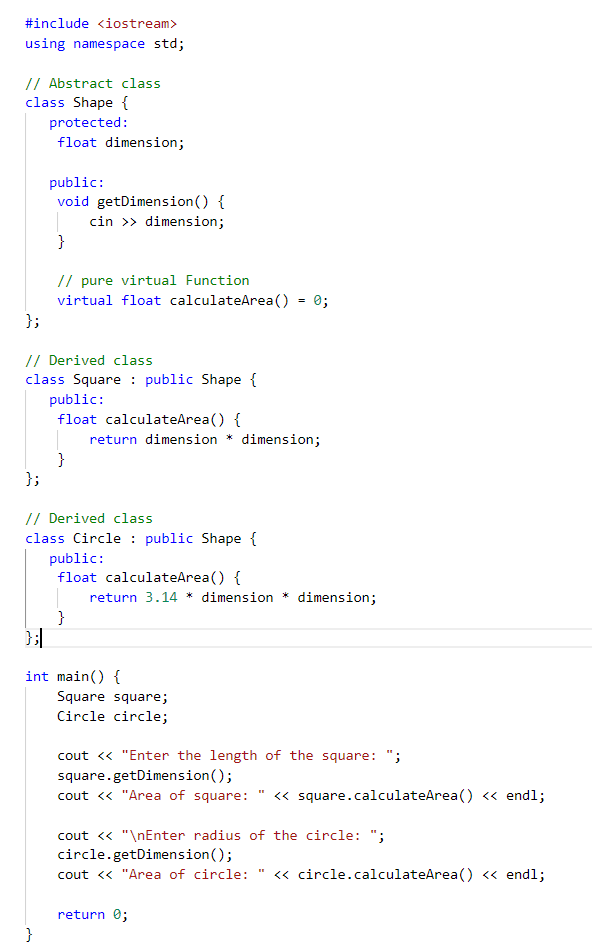
1. **Abstract class**

Một lớp chứa ít nhất một hàm thuần ảo, được gọi là abstract class (lớp trừu tượng)

Trong ví dụ trên, lớp Shape là một lớp trừu tượng.

Chúng ta không thể tạo các đối tượng của một lớp trừu tượng. Tuy nhiên, chúng ta có thể tạo các lớp dẫn xuất từ nó và sử dụng các thành viên dữ liệu và các hàm thành viên của nó (ngoại trừ các hàm thuần ảo).

**Ví dụ:** Chương trình C ++ để tính diện tích hình vuông và hình tròn

****