Class & Struct II

Lập trình nâng cao

Nội dung chính

Chủ yếu là các vấn đề cú pháp

- Quyền truy nhập private/public cho biến/hàm thành viên
- class so với struct
- Khởi tạo hằng thành viên
- Hàm bạn (friend)
- Cài chồng toán tử
- Tách cài đặt hàm thành viên ra khỏi định nghĩa
- Tách file .h và .cpp

public / private?

```
struct Vector {
private:
  double x;
  double y;

public:
  Vector add(Vector other) {... }
  void print() {...}
};

// tại một hàm không phải thành viên của struct/class
      Vector v;
      v.x = 1.0;          //Lỗi! x private
      v.print();          //Ok. print() public
…
```

x và y là các thành viên được khai báo là private

add() và print() là các thành viên public

public / private?

```
struct Vector {
private:
  double x;
  double y;

public:
  Vector add(Vector other) {... }
  void print() {...}
};
```

Thành viên private của một struct/class là thành viên chỉ có thể được truy nhập ở **bên trong định nghĩa và cài đặt** của struct/class đó.

Thành viên public của một struct/class là thành viên mà có thể truy nhập được từ **bất cứ đâu** trong phạm vi của biến struct/class.

```
// tại một hàm không phải thành viên của struct/class
        Vector v;
        v.x = 1.0;      //Lỗi! x private
        v.print();      //Ok. print() public
…
```

Thành viên của struct mặc định là public

```
struct Vector {
  double x;
  double y;

  Vector add(Vector other) {... }
  void print() {...}
};

// bên ngoài struct/class
      Vector v;
      v.x = 1.0; //truy nhập biến thành viên x của v
      v.print(); //truy nhập hàm thành viên print() của v
…
```

x,y, add(), print() nghiễm nhiên public mà không cần gì ngoài khai báo thông thường

Class giống hệt struct
ngoại trừ quyền truy nhập mặc định

```
struct Vector {
private:
  double x;
  double y;

public:
  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

```
class Vector {
private:
  double x;
  double y;

public:
  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

hoàn toàn tương đương

Class giống hệt struct
ngoại trừ quyền truy nhập mặc định

```
struct Vector {
private:
  double x;
  double y;

  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

```
class Vector {
// không cần khai báo private:
  double x;
  double y;

  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

hoàn toàn tương đương

mặc định, thành viên class là private

Cách khai báo thông dụng cho class

```
class Vector {
  double x;
  double y;

public:
  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

```
class Vector {
public:
  Vector add(Vector other)
  {... }
  void print()
  {...}

private:
  double x;
  double y;
};
```

Tại sao cần cả struct lẫn class?

- Có struct là vì kế thừa struct của C
- Class là thuật ngữ quen thuộc của lập trình hướng đối tượng (C++ là ngôn ngữ hướng đối tượng)
- Tuy nhiên: cú pháp của struct C và struct C++ khác nhau.

Không được dùng struct C trong code C++ và ngược lại!

Class / struct

- Khi nào nên dùng class, khi nào nên dùng struct?
- Thông lệ:
  - dùng struct cho cấu trúc không cần che private
  - dùng class cho các cấu trúc còn lại

  Tuy nhiên, tùy chọn của từng người.
- Class và struct đều dùng để định nghĩa lớp đối tượng. Mỗi biến thuộc lớp đó là một đối tượng.
- Từ nay ta gọi:

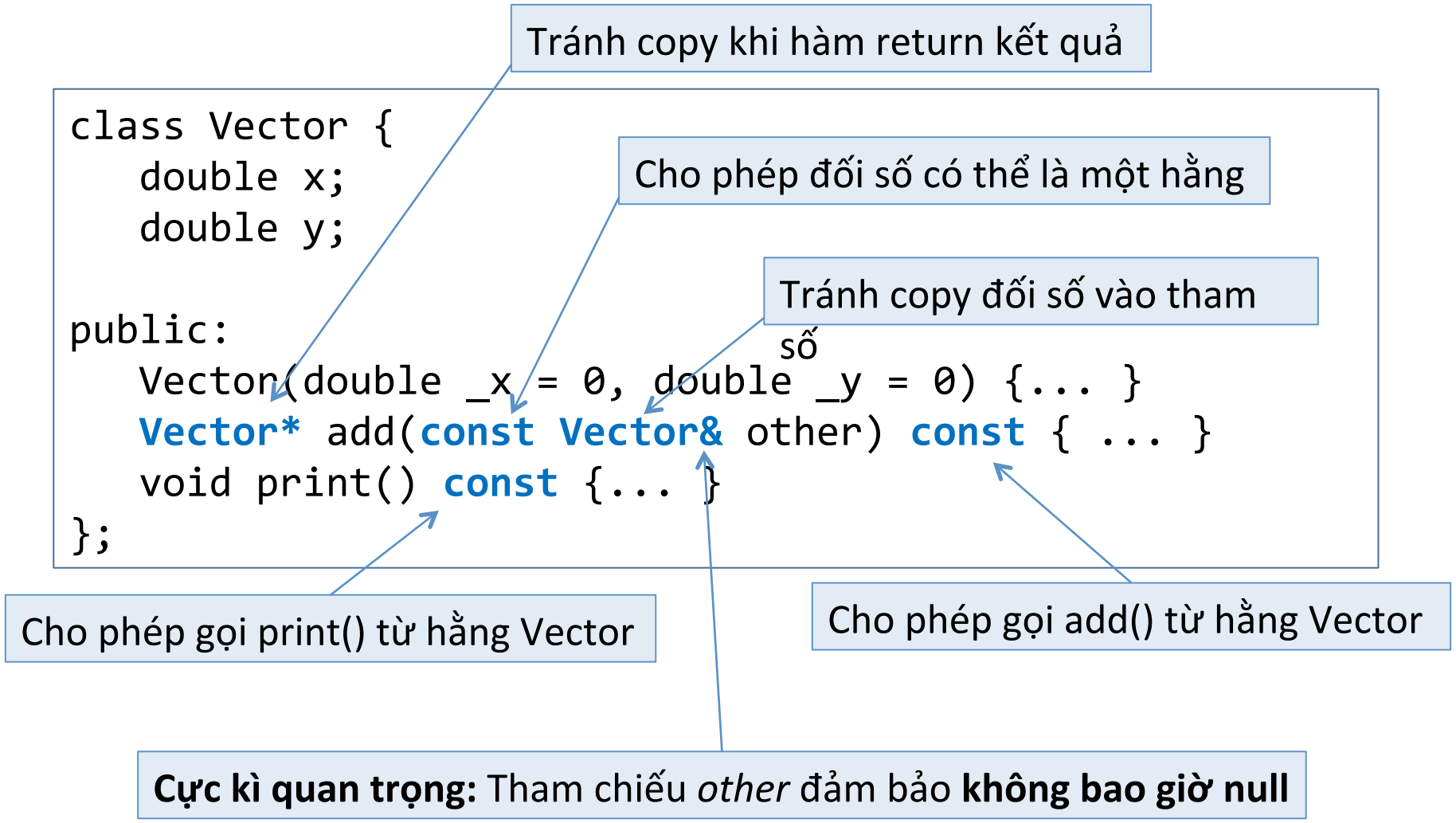
```
Vector v;                       // v là đối tượng (thuộc lớp) Vector
Vector* p = new Vector();  // p trỏ tới một đối tượng Vector
```

Ôn lại best practice

```
class Vector {
  double x;
  double y;

public:
  Vector add(Vector other)
  {... }
  void print()
  {...}
};
```

Hãy chỉnh lại vì code nãy giờ bỏ const và không quan tâm tối ưu hóa để code ngắn và đơn giản dễ đọc



Chỉnh lại vì code nãy giờ bỏ const và không quan tâm tối ưu hóa để code ngắn và đơn giản dễ đọc

```
class Vector {
   double x;
   double y;

public:
   Vector(double _x = 0, double _y = 0) {
       x = _x; y = _y;
   }

   Vector* add(const Vector& other) const {
      return new Vector(x + other.x, y + other.y);
   }

   void print() const {
       cout << "(" << x << "," << y << ")";
   }
};
```

Hằng thành viên dữ liệu

```
class Screen {
   const int width;        // hằng thành viên dữ liệu
   const int height;       // không thể thay đổi giá trị

public:
   Screen(double w, double h) {
      width = w;     // lỗi cú pháp
      height = h;    // lỗi cú pháp
   }

   void change() {
      width = 3;     // lỗi cú pháp và lỗi ngữ nghĩa
   }
};
```

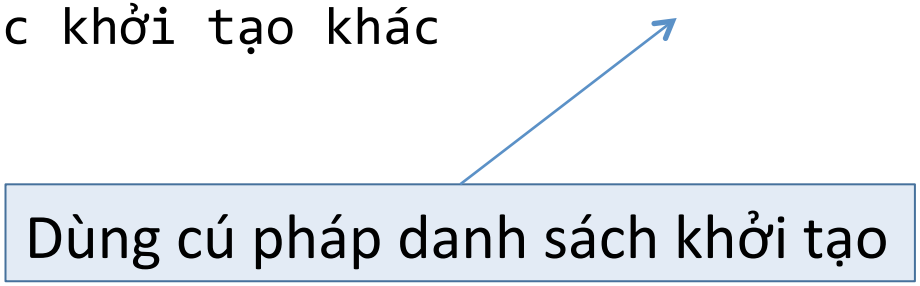
Làm thế nào để khởi tạo width và height?

Hằng thành viên dữ liệu

```
class Screen {
    const int width;          // hằng thành viên dữ liệu
    const int height;         // không thể thay đổi giá trị

public:
    Screen(double w, double h) : width(w), height(h) {
        // các việc khởi tạo khác
    }

    void change() {
        width = 3;   //  sai ngữ nghĩa nên phải xóa bỏ
    }
};
```



Làm sao để truy nhập biến thành viên private?

```
class Vector {
private:
   double x;
   double y;
...
};
```

X và y đang là các thành viên private,
Ta muốn truy cập x, y từ một hàm không phải thành viên của Vector

Phải làm sao?

```
void someTask(Vector v1, Vector v2) {
   double xx, yy;
   xx = v1.x + v2.x; // lỗi biên dịch
   yy = v1.y + v2.y; // lỗi biên dịch
   ...
}
```

Truy nhập biến thành viên qua setter, getter

```
class Vector {
private:
   double x;
   double y;
public:
   double getX() { return x;}
   double getY() { return y;}
...
};
```

Các hàm không phải thành viên của Vector sẽ dùng getX() và getY() để lấy giá trị

Kết quả: **TẤT CẢ** các hàm không phải thành viên của Vector đều được đọc giá trị của x, y

```
void someTask(Vector v1, Vector v2) {
   double xx, yy;
   xx = v1.getX() + v2.getX(); //ok
   yy = v1.getY() + v2.getY(); //ok ...
}
```

Truy nhập
biến thành viên qua setter, getter

```
class Vector {
private:
   double x;
   double y;
public:
   double getX() { return x;}
   double getY() { return y;}
...
};
```

Các hàm không phải thành viên của Vector sẽ dùng getX() và getY() để lấy giá trị

Nếu muốn chỉ 1-2 hàm được đọc giá trị x,y thì làm thế nào?

Kết quả: **TẤT CẢ** các hàm không phải thành viên của Vector đều được đọc giá trị của x, y

```
void someTask(Vector v1, Vector v2) {
   double xx, yy;
   xx = v1.getX() + v2.getX();
   yy = v1.getY() + v2.getY(); ...
}
```

Khai báo một hàm là friend

```
class Vector {
    double x;
    double y;
    friend void someTask(Vector v1, Vector v2);
    ...
};

void someTask(Vector v1, Vector v2) {
    double xx, yy;
    xx = v1.x + v2.y; //ok
    v1.x = v2.y; //ok ...
}

int otherTask(Vector v) {
    double a = v.x;  // lỗi biên dịch
}
```

Khai báo rằng someTask() là friend của Vector

Hàm được Vector nhận là friend được **đọc và ghi** các thành viên private

Hàm không phải friend của Vector không được truy cập.

Khi nào nên dùng friend?

- Nếu có thể thay thế một hàm friend bằng một hàm thành viên thì nên làm
- Chỉ dùng khi nào không tránh được:
  - Không thể chuyển thành hàm thành viên
  - Không thể cho setter và getter public (ai cũng dùng được)
  - Sẽ thấy ví dụ khi học về template

Định nghĩa lại toán tử operator overload

- Ta đã có thể làm:

  Vector sum = v1**.add**(v2);

- Nếu ta muốn dùng dấu cộng thì làm thế nào?

  Vector sum = v1 **+** v2;

- **Operator Overload – Định nghĩa lại toán tử mà ta muốn để dùng được cho kiểu dữ liệu ta muốn.**

Ví dụ: định nghĩa phép cộng Vector

```
class Vector {
  double x;
  double y;

public:
  Vector operator+(const Vector& other) const {
    Vector sum(x + other.x, y + other.y);
    return sum;
  }

  Vector (double _x = 0, double _y = 0)
    : x(_x), y(_y) {}
};
```

Tên hàm là phải là operator+, operator-, operator*, ….

Kết quả: với Vector v1, v2, v3, ta có thể viết:

Vector s = v1 + v2 + v3;

Câu hỏi

```
class Vector {
  double x;
  double y;

public:
  Vector* operator+(const Vector& other) const {
      return new Vector(x + other.x, y + other.y);
  }

  Vec
    :
};
```

Return con trỏ nhanh hơn return một đối tượng Vector.
Có nên giảm thời gian sao chép giá trị trả về bằng cách này không?

Liệu với Vector v1, v2, v3, ta có thể viết biểu thức sau?

(v1 + v2 + v3)

Con trỏ **this** của đối tượng

- Bên trong hàm thành viên, từ khóa this cho ta con trỏ tới đối tượng hiện đang chạy hàm thành viên đó.

```
class Vector {
    double x;
    double y;

public:
    bool equals(const Vector& other) {
        if (this == &other) return true;
        return (x == other.x && y == other.y);
    }

    void print() const {
        cout << "(" << x << "," << y << ")";
    }
...
```

Con trỏ **this** của đối tượng

- Bên trong hàm thành viên, từ khóa this cho ta con trỏ tới đối tượng hiện đang chạy hàm thành viên đó.

```
class Vector {
    double x;
    double y;

public:
    Vector(double _x = 0, double _y = 0) {
        x = _x; y = _y;
    }

    void print() const {
        cout << "(" << x << "," << y << ")";
    }
};
```

Con trỏ **this** của đối tượng

- Bên trong hàm thành viên, từ khóa this cho ta con trỏ tới đối tượng hiện đang chạy hàm thành viên đó.

```
class Vector {
    double x;
    double y;

public:
    Vector(double x = 0, double y = 0) {
        this->x = x; this->y = y;
    }

    void print() const {
        cout << "(" << this->x << "," << this-> y <<")";
    }
};
```

Template class

```
template <class T>
class MyPair {
  T a, b;
public:
  mypair (T first, T second) {a=first; b=second;}
  T getmax ();
};

template <class T>
T MyPair<T>::getmax () {
  T retval = a>b ? a : b;
  return retval;
}
```

Template class

```
template <class T>
class MyPair {
  T a, b;
public:
  mypair (T first, T second) {a=first; b=second;}
  T getmax ();
};

template <class T>
T MyPair<T>::getmax () {
  T retval = a>b ? a : b;
  return retval;
}
```

```
int main () {
  MyPair <int> myobject (100, 75);
  cout << myobject.getmax();
  return 0;
}
```

Xem thêm

- http://www.cplusplus.com/doc/tutorial/templates/

- learncpp.com

Tách cài đặt hàm ra khỏi định nghĩa class/struct

```
class Vector {
  double x;
  double y;
public:
  Vector(double _x = 0, double _y = 0);
  Vector add(Vector other);
  void print() const;
};

Vector::Vector (double _x, double _y) { ... }
Vector Vector::add(Vector& other) {... }
void Vector::print() {...}
```

Khai báo các hàm thành viên ở **bên trong** khối {} của struct/class

Tên struct/class để phân biệt với cài đặt của các hàm thông thường

Định nghĩa các hàm thành viên đặt **bên ngoài** khối {} của struct/class

Tách cài đặt hàm ra khỏi định nghĩa class/struct

```
class Vector {
  double x;
  double y;
public:
  Vector (double _x = 0, double _y = 0);
  ...
};

Vector::Vector (double _x, double _y) {
      x = _x;
      y = _y;
}
```

Giá trị mặc định của tham số phải đặt tại khai báo hàm thành viên, không đặt tại định nghĩa hàm

Tách class/struct ra file riêng để tái sử dụng

File vector.h

```
#include <iostream>

using namespace std;

class Vector {
  double x;
  double y;

public:
  Vector(double _x = 0, double _y = 0);
  Vector* add(const Vector& other) const;
  void print() const;
};

Vector::Vector (double _x, double _y) {
  x = _x; y = _y;
}

Vector* Vector::add(const Vector& other) const {
  return new Vector(x + other.x, y + other.y);
}

void Vector::print() const {
  cout << "(" << x << "," << y << ")";
```

```
#include <iostream>
#include "vector.h"

using namespace std;

int main() {
  Vector a(1,2);
  cout << &a << ": ";
  a.print();
  Vector b(10,20);
...
```

File program.cpp

Tách class/struct ra file riêng để tái sử dụng

File vector.h

```
#include <iostream>

using namespace std;

class Vector {
  double x;
  double y;

public:
  Vector(double _x = 0, double _y = 0);
  Vector* add(const Vector& other) const;
  void print() const;
};

Vector::Vector (double _x, double _y) {
  x = _x; y = _y;
}

Vector* Vector::add(const Vector& other) const {
  return new Vector(x + other.x, y + other.y);
}

void Vector::print() const {
  cout << "(" << x << "," << y << ")";
```

```
#include <iostream>
#include "vector.h"

using namespace std;

int main() {
  Vector a(1,2);
  cout << "a = ";
  a.print();
  Vector b(10,20);
  ...
```

File program.cpp

Lợi ích:

Có thể tái sử dụng cài đặt cấu trúc Vector trong nhiều dự án khác nhau.

Chưa ổn:

Ai dùng vector.h có thể nhìn thấy toàn bộ cài đặt và có thể sửa mã nguồn

Tách tiếp

File vector.h

```
class Vector {
  double x;
  double y;

public:
  Vector(double _x = 0, double _y = 0);
  Vector* add(const Vector& other) const;
  void print() const;
};
```

```
#include <iostream>
#include "vector.h"

using namespace std;
```

Lợi ích:

- Vẫn có thể tái sử dụng toàn bộ cài đặt cấu trúc Vector trong nhiều dự án khác nhau. Chỉ cần có file vector.h và file nhị phân (không phải mã nguồn của vector.cpp) → Che được chi tiết cài đặt.

```
#include "vector.h"
#include <iostream>

using namespace std;

Vector::Vector (double _x, double _y) {
  x = _x; y = _y;
}

Vector* Vector::add(const Vector& other)
const {
```

File vector.cpp

File program.cpp

Tránh lỗi lặp include khi có nhiều file cùng include một thư viện

```
#ifndef VECTOR_H
#define VECTOR_H
class Vector {
  double x;
  double y;

public:
  Vector(double _x = 0, double _y = 0);

  Vector* add(const Vector& other)
const;
  void print() const;
};
#endif
```

Biên dịch thế nào?

- Bài thực hành