Lập trình hướng đối tượng

Constructor & Destructor
Variable scope & lifetime

Nội dung của module gồm có:

1. Làm quen với hàm tạo và hàm hủy
2. Tầm vực và vòng đời của biến
3. Các bài tập vận dụng

Bộ môn Công nghệ phần mềm
Khoa Công nghệ thông tin
Đại học Khoa học tự nhiên TP HCM

1

Làm quen với hàm tạo và hàm hủy

1.1. Constructor

Hàm tạo / Hàm dựng (constructor) của một lớp đối tượng là một phương thức đặc biệt **được tự động gọi** thực hiện khi **đối tượng thuộc lớp đó được tạo lập**.

Các tính chất của hàm dựng:

- Được tự động gọi thực hiện khi đối tượng được tạo lập.
- Không có giá trị trả về (nhưng có thể có các tham số).
- Một lớp đối tượng có thể có nhiều hàm dựng khác nhau.
- Trong C++, hàm dựng có tên trùng với tên lớp.

Bất kỳ lớp đối tượng nào cũng có hàm dựng. Trong trường hợp chúng ta không khai báo hàm dựng nào cho lớp đối tượng, hàm dựng mặc định (default constructor) không tham số sẽ được tự động thêm vào.

Tutorial 1: Làm quen với việc hàm tạo được tự động gọi của một lớp.

Yêu cầu: Viết lớp **PhanSo** biểu diễn kiểu dữ liệu phân số.

PhanSo.h

```
#pragma once // Chỉ include một lần
#include <string>
using namespace std;

class PhanSo
{
private:
    int mTuSo;
    int mMauSo;
public:
    PhanSo();
    string ToString();
};
```

PhanSo.cpp

```
#include "PhanSo.h"
#include <sstream>

PhanSo::PhanSo()
{
    mTuSo = 0;
    mMauSo = 1;
}

string PhanSo::ToString()
{
    stringstream ss;
    ss << mTuSo << "/" << mMauSo;
    return ss.str();
}
```

Code hàm main như sau:

main.cpp

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

using namespace std;

void main()
{
    PhanSo a;

    cout << a.ToString();

    cin.get();
}
```

Những điểm cần lưu ý của Tutorial trên

1. Coding convention (Phong cách lập trình)

Đặt tên biến private của lớp như thế nào là hợp lí?

Đây là một chủ đề gây tranh cãi vì có rất nhiều trường phái.

Xét thuộc tính tử số của phân số.

Nếu viết theo trường phái Hungarian notation thì đầy đủ sẽ phải là: m_iTuSo. Trong đó:

m bắt đầu ám chỉ là member (thành viên của một lớp).

i ám chỉ biến có kiểu là số nguyên (integer).

Phân tách giữa m và i là một dấu gạch dưới _ (underscore)

Tên của kiểu dữ liệu được viết theo **PascalCase**, tức chữ cái phân tách các từ được viết hoa.

Xuất phát từ trường phái cơ bản trên, có thêm một số cách viết sau:

+ _iTuSo: bỏ đi m vì không cần thiết. Khi thấy _ tự động hiểu là thành viên private.

+ _TuSo: bỏ luôn i (kiểu dữ liệu) với lí do các IDE hiện đại dư sức nhắc nhở biến thuộc kiểu gì.

+ _tuSo: đi theo phong cách **camelCase**, tức chữ cái đầu viết thường, các chữ đầu tiên của các từ sau viết hoa.

+ tuSo: bỏ luôn gạch dưới ở đầu. Cách này này có thể gây nhầm lẫn và không phân biệt được đâu là thuộc tính của lớp, đâu là biến cục bộ nếu cả hai cùng viết thường ở đầu. Cần sử dụng thêm con trỏ this để chống mập mờ và dĩ nhiên là cách viết này dài dòng hơn.

2. Sử dụng stringstream để nối chuỗi string và biến nguyên

Có nhiều cách khác để nối chuỗi và số nguyên như sử dụng sprintf, dùng std::to_string.

3. Hàm ToString tạo kết xuất chuỗi cho đối tượng.

4. Hàm tạo của lớp PhanSo được gọi tự động nên một biến kiểu PhanSo khi vừa khai báo sẽ có giá trị là 0/1.

1.2. Destructor

Hàm hủy *(destructor)* của một lớp đối tượng là một phương thức đặc biệt **được tự động gọi thực hiện** khi đối tượng thuộc lớp đó bị hủy đi.

Các tính chất của hàm hủy

- Được tự động gọi thực hiện khi đối tượng bị hủy đi.
- Không có giá trị trả về lẫn tham số.
- Một lớp đối tượng chỉ có duy nhất một hàm hủy.
- Trong C++, hàm hủy có tên trùng với tên lớp và thêm dấu "~" phía trước.

Mẹo: Không cần quan tâm lắm đến hàm hủy nếu như lớp không có thuộc tính nào có kiểu **con trỏ**! Điều này dẫn đến hệ quả thú vị là **Rule of three** trong C++ (khi đã viết 1 trong 3 hàm destructor, copy constructor, copy assignment operator thì phải viết luôn đầy đủ 2 hàm còn lại – Ngoại lệ là RAII). Xem thêm trong phần 4, các vấn đề mở rộng.

Tutorial 2: Làm quen với việc hàm hủy được gọi tự động.

Yêu cầu: viết lớp IntegerArray để biểu diễn lớp mảng động các số nguyên.

Ý tưởng thiết kế và cài đặt ban đầu

- Để biểu diễn lớp mảng động ta sẽ dùng một con trỏ kiểu int để có thể cấp phát động bộ nhớ: int* mpData.
- Tuy nhiên ta không thể cấp phát tùy ý tùy theo thời điểm cần thêm hay bỏ bớt số trong mảng. Vậy nên ta sẽ cấp phát trước một mảng các số nguyên có kích thước maximum, khi thêm số nguyên vào mảng nếu quá kích thước maximum này thì sẽ cấp phát lại từ đầu.
- Do vậy ban đầu lớp sẽ có 3 thành phần chính:
  - mpData: con trỏ chứa dữ liệu mảng
  - mCurrentSize: kích thước mảng hiện tại có dữ liệu
  - mMaxSize: kích thước mảng tối đa
- Ta sẽ cần có một cách để thêm các số nguyên vào mảng động này, vậy nên sẽ cần có hàm **Add**.
- Ta cũng sẽ cần một cách thức để truy cập đến các phần tử của mảng, vậy nên sẽ có hàm **GetAt**

IntegerArray.h

```
#pragma once

class IntegerArray
{
private:
    const int INITIAL_SIZE = 100;

private:
    int* mpData;
    int mCurrentSize;
    int mMaxSize;

public:
    IntegerArray();
    ~IntegerArray();

    void Add(int num);
    int GetAt(int iIndex);
};
```

Nếu là Visual Studio 2010 thì cần thêm static đằng trước thành static const int…

IntegerArray.cpp

```
#include "IntegerArray.h"
#include <iostream>
using namespace std;

IntegerArray::IntegerArray()
{
    mpData = new int[INITIAL_SIZE];
    mCurrentSize = 0;
    mMaxSize = INITIAL_SIZE;
}

IntegerArray::~IntegerArray()
{
    if (NULL != mpData)
        delete[] mpData;

    cout << "Ham huy duoc goi";
}

void IntegerArray::Add(int num)
{
    mpData[mCurrentSize - 1] = num;
    mCurrentSize++;
}

int IntegerArray::GetAt(int iIndex)
{
    return mpData[iIndex];
}
```

Câu hỏi:

Hàm Add như vậy viết có đúng không? Làm sao để cải tiến?

Quan sát kết quả tương ứng với hàm main sau:

main.cpp

```
#include <iostream>
using namespace std;

#include "IntegerArray.h"

void main()
{
    IntegerArray a;

    cin.get();
}
```

Câu hỏi: Tại sao lại có dòng chữ "Ham huy duoc goi" xuất ra? Có gì hủy biến đâu?

Tiến hành sửa hàm main như sau:

```
void main()
{
    IntegerArray a;

    IntegerArray* b = new IntegerArray();
    delete b;

    cin.get();
}
```

Câu hỏi:

1. Hàm hủy được gọi bao nhiêu lần? Có phải là hàm hủy của cùng một đối tượng không?
2. Ở đây có hai đối tượng a và b, hiệu chỉnh lớp để biết rõ hàm hủy của đối tượng nào được gọi. (Gợi ý: gán tên/id khi khởi tạo, trong hàm hủy in ra id này để biết)

Những điểm cần lưu ý của Tutorial trên

1. Việc hủy dùng hàm delete: tại sao phải có [] (delete[])?

2. Để truy cập một phần tử của mảng động, phải thông qua hàm GetAt, việc gọi như thế này sẽ không tự nhiên. Có cách nào dùng toán tử chỉ mục - [], ví dụ như khi thao tác với phần tử i của mảng, ta viết a[i] được không?

(*Gợi ý*: nạp chồng – overload - hàm operator[])

3. Mỗi khi tạo ra mảng, thì một vùng nhớ tương đương với hằng số INITIAL_SIZE luôn được cấp phát. Ta muốn tránh sự lãng phí như vậy bằng cách cho phép lúc tạo, chỉ định luôn kích thước ban đầu. Làm thế nào làm được như vậy?

(*Gợi ý*: viết constructor có đối số là số nguyên, là kích thước ban đầu)

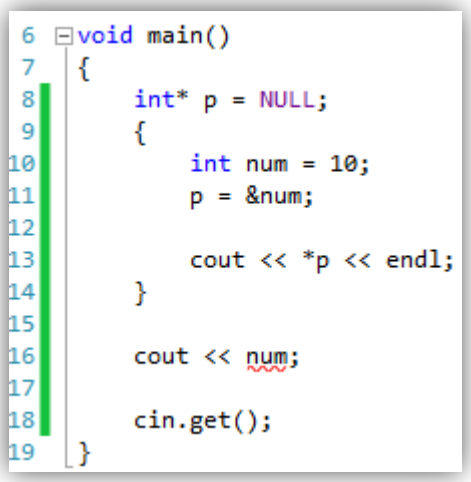
4. Làm sao biết được kích thước hiện tại của mảng?

(*Gợi ý*: viết hàm GetLength(), hoặc viết hàm size() giống lớp vector. Câu hỏi đặt ra là nên đặt tên theo kiểu nào, GetLength() hay size() / Size())

2

Tầm vực và vòng đời của biến

Tutorial 3: Tìm hiểu về tầm vực và vòng đời của biến



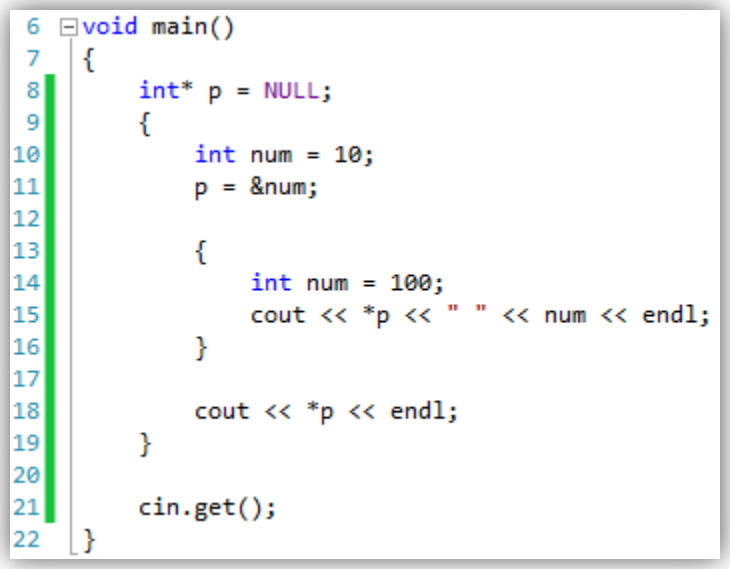
```
6  void main()
7  {
8      int* p = NULL;
9      {
10         int num = 10;
11         p = &num;
12
13         cout << *p << endl;
14     }
15
16     cout << num;
17
18     cin.get();
19 }
```

Câu hỏi:

1. Đoạn code trên có thể biên dịch và chạy được không? Thử giải thích tại sao.

2. Thử comment dòng lệnh 16 (phím tắt: Ctrl + K > C) để biên dịch được và chạy thử, quan sát kết quả. (Để uncomment, phím tắt là Ctrl + K > U).

Tutorial 4: Khối lệnh lồng nhau.



```
6  void main()
7  {
8      int* p = NULL;
9      {
10         int num = 10;
11         p = &num;
12
13         {
14             int num = 100;
15             cout << *p << " " << num << endl;
16         }
17
18         cout << *p << endl;
19     }
20
21     cin.get();
22 }
```

Câu hỏi:

1. Viết lại đoạn code trên. Trước khi chạy thử thử đoán xem kết quả là gì?

2. Tại sao kết quả lại như vậy?

Chú ý biến p vẫn được nhìn thấy bên trong đoạn code thứ hai, tuy nhiên biến num được khai báo lại trong đoạn code lồng trong cùng che mất biến num ở bên ngoài.

Tutorial 5: Vòng đời của biến

Xét một mảng số nguyên a, đoạn chương trình sau sắp xếp tăng dần a sử dụng hai vòng lặp và in lại mảng a sau khi sắp xếp.

```
7  void main()
8  {
9      int a[] = { 6, 7, 5, 10, 33 };
10     int n = 5;
11
12     for (int i = 0; i < n; i++)
13     {
14         for (int j = i + 1; j < n; j++)
15         { // Xét biến temp
16             // Cứ bắt đầu mỗi vòng lặp một biến temp lại được tạo ra
17             int temp = a[i]; // Bắt đầu phạm vi xuất hiện
18             a[i] = a[j];
19             a[j] = temp;     // Ra khỏi phạm vi
20             // Biến temp sẽ bị hủy
21         }
22
23         cout << temp;
24     }
25
26     for (int i = 0; i < n; i++)
27     {
28         cout << a[i] << " ";
29     }
30
31     cin.get();
32 }
```

Câu hỏi & thảo luận:

1. Thử biên dịch đoạn code trên. Nếu không biên dịch được thì lí do lỗi là ở dòng nào? Tại sao?

2. Comment dòng lệnh bị lỗi và biên dịch rồi chạy lại. Quan sát kết quả.

3. Xét biến temp. Dòng 17 và 19 giới hạn scope (tầm vực) của biến. Bằng chứng là ra ngoài tầm vực này, tại dòng 23 (cout << temp;) biến này không thể được truy xuất nữa. Cuối mỗi lần lặp biến temp này sẽ bị hủy. Tuy nhiên khi bắt đầu vòng lặp mới, biến temp **MỚI** lại được tạo ra. Ví dụ trên minh họa sự khác nhau giữa tầm vực và vòng đời của một biến.

3

Bài tập vận dụng

1. Viết lớp biểu diễn một điểm trong không gian 2 chiều Oxy. Viết hàm main nhập vào hai điểm và xuất ra màn hình hai điểm này.

2. Viết lớp biểu diễn đoạn thẳng trong không gian 2 chiều Oxy. Viết hàm main nhập vào một đường thẳng và xuất ra màn hình đường thẳng này. Cho biết độ dài của đường thẳng.

3. Viết lớp biểu diễn tam giác, hình vuông, hình chữ nhật trong không gian 2 chiều Oxy. Viết hàm main demo việc nhập các hình này, xuất các hình này ra. Cho biết chu vi, diện tích của các hình vừa nhập.

4. Viết lớp biểu diễn đa giác trong không gian 2 chiều Oxy. Viết lớp nhập đa giác, xuất ra đa giác này. Cho biết chu vi của đa giác (giả định đa giác đã là đa giác lồi).

* Nâng cao:

a. Viết hàm kiểm tra đa giác có phải là đa giác lồi hay không.

b. Viết hàm tính diện tích đa giác.

c. Viết hàm kiểm tra một điểm có thuộc đa giác hay không.

4

Các vấn đề mở rộng và đọc thêm

1. Hàm tạo thường được đặt ở phần public của một lớp. Tuy nhiên có những trường hợp đặc biệt hàm tạo được đặt ở phần private. Như vậy là có thể thấy nếu hàm tạo nằm ở private thì ta không cách gì tạo ra một đối tượng của lớp này được. Vậy ngữ cảnh của việc này là khi nào?

Tham khảo thêm về mẫu thiết kế Singleton.

http://en.wikipedia.org/wiki/Singleton_pattern

2. Rule of three: Một khi đã lỡ viết 1 trong 3 hàm destructor, copy constructor, copy assignment operator thì phải viết luôn 2 hàm còn lại.

http://en.wikipedia.org/wiki/Rule_of_three_(C%2B%2B_programming)