Hàm tạo và hủy đối tượng

Nguyễn Khắc Huy

Ký hiệu UML

- Một lớp đối tượng được mô tả bằng một hình chữ nhật chia thành 3 phần
  - Tên lớp
  - Các thuộc tính
  - Các hành vi

Professor
- name : String - employeeID : UniqueId - hireDate : Date - status : String - discipline : String - maxLoad : Integer
+ submitFinalGrade() + acceptCourseOffering() + setMaxLoad() + takeSabbatical() + teachClass()

profA: Professor
- name = “Chau Ngo” - employeeID = 23068 - hireDate : 15/7/2007 - status = “abroad” - discipline = “Math” - maxLoad = 8

Nội dung

- Hàm dựng
- Hàm hủy
- Hàm toán tử
- Bài tập

Hàm dựng

- Khi đối tượng vừa được tạo:
  - ✓**Giá trị các thuộc tính bằng bao nhiêu?**
  - ✓Một số đối tượng cần có thông tin ban đầu.
  - ✓Giải pháp:
    - Xây dựng phương thức khởi tạo.
    - Người dùng quên gọi?!

PhanSo
- Tử số??
- Mẫu số??
- **Khởi tạo**

HocSinh
- Họ tên??
- Điểm văn??
- Điểm toán??
- **Khởi tạo**

Hàm dựng ra đời!!

Hàm dựng

- Hàm dựng là 1 hàm đặc biệt, khác với các hàm thông thường khác và có các đặc điểm sau:
  - Hàm có tên trùng với tên của lớp đối tượng
  - Không có kiểu trả về (nhưng có thể có tham số)
  - Tự động được gọi sau khi đối tượng thuộc lớp được tạo lập
  - Có thể nạp chồng nhiều hàm dựng.

Hàm dựng

- Ví dụ

```
class PhanSo
{
private:
    int     m_iTuSo;
    int     m_iMauSo;
public:
    PhanSo(int iTuSo, int iMauSo);
    PhanSo(int iGiaTri);
};
```

```
void main()
{
    PhanSo   p1(1, 2);
    PhanSo   *p2 = new PhanSo(5);
}
```

Hàm dựng

- Hàm dựng mặc định (default constructor):
  - ✓**Khi lớp không có hàm dựng?**
    - ➔ Trình biên dịch cung cấp hàm hựng mặc định.
  - ✓Tính chất:
    - – Không tham số.
    - – Khởi tạo mặc định các thuộc tính.

```
class PhanSo
{
private:
        int         m_iTuSo;
        int         m_iMauSo;
};
```

```
void main()
{
    PhanSo   p1;
}
```

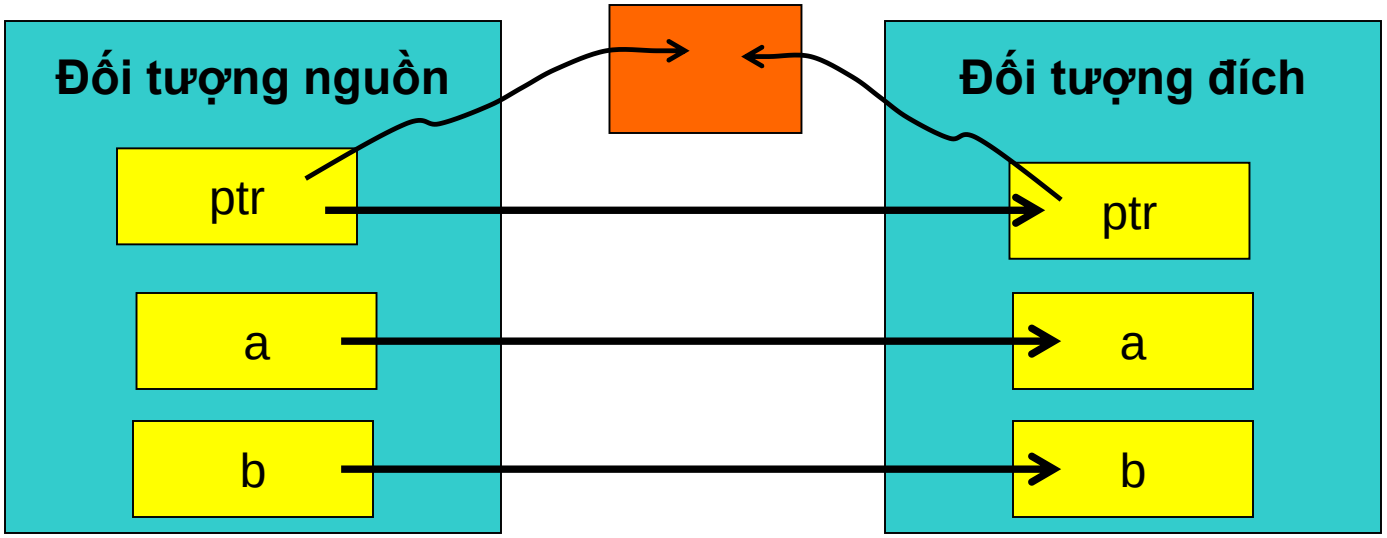
Hàm dựng sao chép mặc định

- Mỗi lớp, nếu không định nghĩa 1 hàm dựng sao chép thì trình biên dịch sẽ cung cấp 1 hàm dựng sao chép mặc định. Hàm này giúp khởi tạo 1 đối tượng thuộc lớp này bằng 1 đối tượng khác thuộc cùng lớp. Ví dụ:

```
int main()
{
        Rectangle a;
        Rectangle b(a); // gọi copy constructor
        Rectangle c = a; // gọi copy constructor
}
```

Hàm dựng sao chép mặc định (tt)

- Hàm dựng sao chép mặc định chỉ sao chép từng bit (bitwise copy) của các thành phần trong đối tượng nguồn sang đối tượng đích



Hàm dựng sao chép mặc định (tt)

- Do tính chất sao chép từng bit (bitwise copy) của hàm dựng mặc định, nếu đối tượng có chứa con trỏ và nó đang trỏ tới 1 vùng nhớ nào đó thì việc sao chép sẽ gây ra vấn đề nghiêm trọng.
  - Cụ thể, khi đó 2 biến con trỏ của 2 đối tượng khác nhau cùng trỏ tới 1 vùng nhớ.

Hàm dựng sao chép

- Cần lưu ý vào đặc thù của lớp đối tượng mà có nên xây dựng hàm dựng sao chép hay không.
  - Cụ thể: khi đối tượng có thành phần dữ liệu là con trỏ

```
Test::Test(const Test& src)
{
        iSize = src.iSize;
        ptr = new int [iSize];
        for (int i=0; i<iSize; ++i)
              ptr[i] = src.ptr[i];
};
```

Hàm dựng sao chép

- Hàm dựng sao chép (copy constructor):
  - ✓Có tham số là đối tượng cùng lớp.
  - ✓Dùng khởi tạo từ đối tượng cùng loại.
  - ✓Lớp không có hàm dựng sao chép?
    - ➔ Trình biên dịch cung cấp.

```
class PhanSo
{
private:
        int         m_iTuSo;
        int         m_iMauSo;
public:
        PhanSo(const PhanSo &p);
};
```

```
void main()
{
    PhanSo   p1(1, 2);
    PhanSo   p2(p1);
    PhanSo   p3 = p2;
}
```

Hàm dựng

- Một lớp nên có tối thiểu 3 hàm dựng sau:
  - Hàm dựng mặc định.
  - Hàm dựng có đầy đủ tham số.
  - Hàm dựng sao chép.

```
class PhanSo
{
private:
        int         m_iTuSo;
        int         m_iMauSo;
public:
        PhanSo();
        PhanSo(int iTuSo, int iMauSo);
        PhanSo(const PhanSo &p);
};
```

Toán tử gán bằng

- Cũng tương tự như hàm dựng sao chép, nếu mỗi lớp đối tượng không có toán tử gán bằng thì trình biên dịch sẽ tạo 1 hàm toán tử gán bằng mặc định
- Hàm này cũng có chức năng tương tự như hàm dựng sao chép mặc định: sao chép từng bit của đối tượng nguồn cho đối tượng đích.

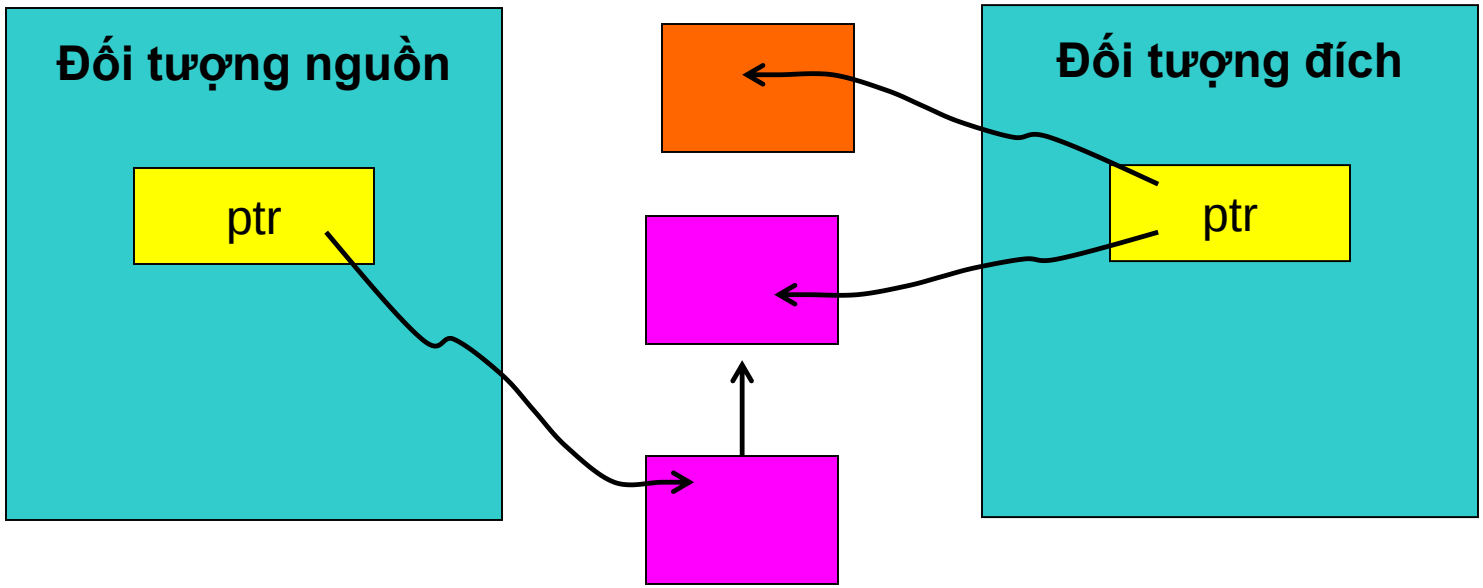
Toán tử gán bằng

- Do vậy, nếu lớp đối tượng có biến con trỏ và có nhu cầu gán bằng 1 đối tượng khác.

  ➔ cần xây dựng toán tử gán bằng cho lớp

- Lưu ý: toán tử gán bằng khác hàm dựng sao chép 1 số điểm sau:
  - Xóa phần bộ nhớ nó đang kiểm soát trước khi gán bằng đối tượng mới.
  - Kiểm tra kỹ việc đối tượng tự gán bằng chính nó.

Toán tử gán bằng



- Xóa bỏ vùng nhớ nó đang kiểm soát
- Copy vùng nhớ và trỏ quả vùng nhớ mới

Ví dụ

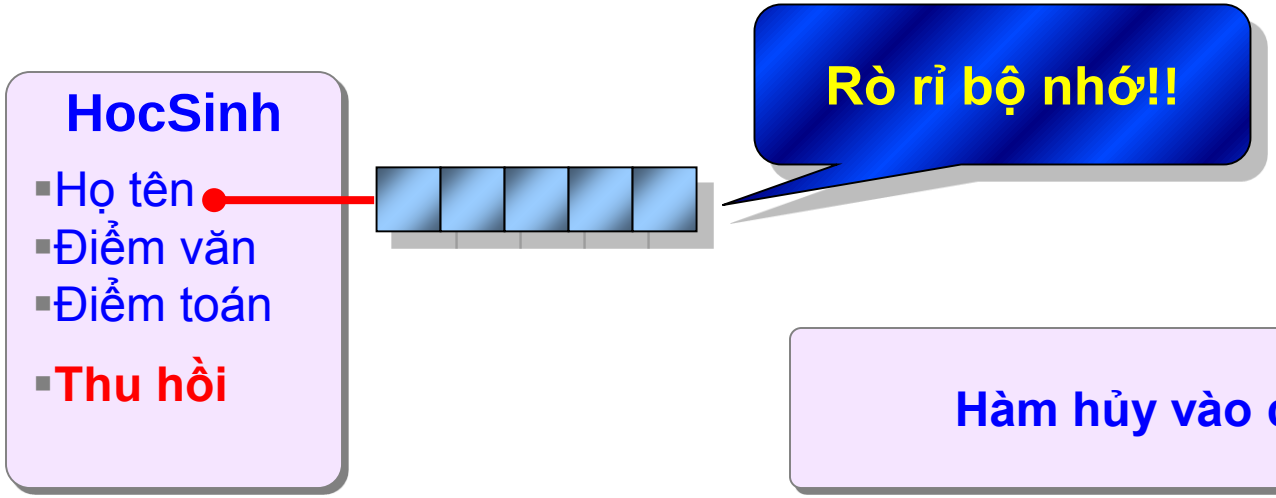
```
Test& Test::operator=(const Test& src)
{
    if (this != &src)
    {
       delete [] ptr;
       iSize = src.iSize;
       ptr = new int [iSize];
       for (int i=0; i<iSize; ++i)
             ptr[i] = src.ptr[i];
    }
    return *this;
}
```

Nội dung

- Hàm dựng
- Hàm hủy
- Hàm toán tử
- Bài tập

Hàm hủy

- Vấn đề rò rỉ bộ nhớ (memory leak):
  - ✓Khi hoạt động, đối tượng có cấp phát bộ nhớ.
  - ✓**Khi hủy đi, bộ nhớ có được thu hồi?**
  - ✓**Làm cách nào để thu hồi?**
    - Xây dựng phương thức thu hồi.
    - Người dùng quên gọi?!



Hàm hủy

- Tính chất hàm hủy (destructor):
  - ✓Tự động thực hiện khi đối tượng bị hủy.
  - ✓Không có giá trị trả về lẫn tham số.
  - ✓Mỗi lớp có duy nhất một hàm hủy.
  - ✓Trong C++, hàm hủy có tên **~<Tên lớp>**

```
class HocSinh
{
private:
        char      *m_sHoTen;
        float     m_fDiemVan;
        float     m_fDiemToan;
public:
        ~HocSinh() { delete m_sHoTen; }
};
```

```
void main()
{
    HocSinh   h;
    HocSinh   *p = new HocSinh;
    delete p;
}
```

Cần ghi nhớ

Bộ 3 hàm sau luôn đi chung với nhau:

- Hàm dựng sao chép (copy constructor)
- Toán tử gán bằng (assignment operator)
- Hàm hủy (destructor)

Nội dung

- Hàm dựng
- Hàm hủy
- Hàm toán tử
- Bài tập

Hàm toán tử

- Khái niệm hàm toán tử:
  - ✓**Có thể dùng toán tử đặt tên hàm?**
  - ✓Trong C++, dùng từ khóa operator.

    PhanSo operator +(***const PhanSo &p1, const PhanSo &p2***);
  - ✓**Hệ quả?**
    - Định nghĩa lại cách thực hiện toán tử.

      PhanSo p1, p2;

      PhanSo p3 = p1 + p2;
    - Định nghĩa nhiều cách thực hiện khác nhau cho toán tử bằng nạp chồng hàm.

      PhanSo operator +(***const PhanSo &p, int iNumber***);

      float opeartor +(***const PhanSo &p, float iNumber***);

Hàm toán tử

- Ưu điểm:
  - ✓Thực hiện toán tử trên kiểu dữ liệu tự định nghĩa.

```
PhanSo p1, p2;
PhanSo p3 = p1 + p2;
```

```
HocSinh h1, h2;
if (h1 > h2)
    h1++;
```

- Hạn chế:
  - ✓Không thể tạo toán tử mới.
  - ✓Không thể định nghĩa lại toán tử trên kiểu cơ bản.
  - ✓Ngôi của toán tử giữ nguyên.
  - ✓Độ ưu tiên của toán tử không đổi.
  - ✓**Đôi khi gây nhầm lẫn!!**

Hàm toán tử

- Phân loại hàm toán tử:
  - ✓Toán tử độc lập:
    - – Không thuộc lớp nào.
    - – Ngôi của toán tử là số tham số truyền vào.

      PhanSo operator +(***const PhanSo &p1, const PhanSo &p2***);

      bool operator >(***const PhanSo &p1, const PhanSo &p2***);
  - ✓Toán tử thuộc lớp:
    - – Là phương thức của lớp.
    - – Ngôi của toán tử: đối tượng của lớp + số tham số.

      PhanSo PhanSo::operator +(***const PhanSo &p***);

      bool PhanSo::operator >(***const PhanSo &p***);
  - ✓**Cách sử dụng 2 loại là như nhau!!**

Hàm chồng là gì?

- Có nhiều “cách cài đặt” khác nhau cho 1 hàm
- Trong C++, các hàm chồng của 1 hàm được phân biệt bằng số lượng và kiểu của các tham số truyền vào.
- Lưu ý: kiểu trả về của hàm không được xét để phân biệt hàm chồng.

Chồng toán tử

- Dùng để định nghĩa thêm cho các toán tử đã có sẵn các hành động mới dành cho kiểu dữ liệu do người dùng mới tạo ra.
- Ví dụ: các toán tử +, -, *, / đã có sẵn cho các kiểu dữ liệu chuẩn (int, float…) trong C++
- Khi có 1 kiểu dữ liệu mới, ví dụ: **PhanSo**, người dùng cần mở rộng định nghĩa của các toán tử này cho kiểu dữ liệu mới

Hàm toán tử

- Toán tử có thể định nghĩa lại:

Ngôi	Nhóm	Toán tử
1 Ngôi (Unary)	Tăng giảm	++, --
	Dấu số học	+, -
	Logic	!, ~
	Con trỏ	*, &
	Ép kiểu	int, float, double, …
2 Ngôi (Binary)	Số học	+, -, *, /, %, +=, -=, *=, /=, %=
	So sánh	>, <, ==, >=, <=, !=
	Logic	&&, ||, &, |
	Nhập xuất	<<, >>
	Gán	=
	Lấy chỉ số mảng	[ ]

Hàm toán tử

- Toán tử không thể định nghĩa lại:

Toán tử	Ý nghĩa
.	Truy xuất phần tử
.*	Truy xuất con trỏ phần tử
::	Toán tử ::
? :	Toán tử điều kiện
#	Chỉ thị tiền xử lý
# #	Chỉ thị tiền xử lý

Hàm toán tử

- Những lưu ý khi định nghĩa lại toán tử:
  - Ngôi: số lượng tham số.
  - Toán hạng: kiểu dữ liệu tham số.
  - Kết quả: kiểu trả về.

- Ví dụ:
  - Toán tử >
  - Toán tử =
  - Toán tử [ ]

Một số lưu ý khi viết chồng toán tử

- Tránh thay đổi ý nghĩa nguyên thủy của toán tử đó
- Các cặp toán tử có cùng chức năng, ví dụ x=x+y và x+=y phải được viết cùng nhau và có cùng chức năng.
- Nếu toán tử chồng không là hàm thành viên của lớp thì nên sử dụng từ khóa **friend** thay vì truy xuất đến các thành phần dữ liệu 1 cách phức tạp

Cú pháp chung

<kiểu trả về> **operator** <toán tử> (danh sánh tham số)

Ví dụ:

```
bool HoTen::operator==(const HoTen& rhs)
{
    return((sTen==rhs.sTen) && (sHo==rhs.sHo));
}
```

Nội dung

- Hàm dựng
- Hàm hủy
- Hàm toán tử
- Một số vấn đề
- Bài tập

Con trỏ this

- Xét đoạn code sau:

  Đoạn code này có đúng không? Về cú pháp và ngữ nghĩa? Làm vì mục đích gì?

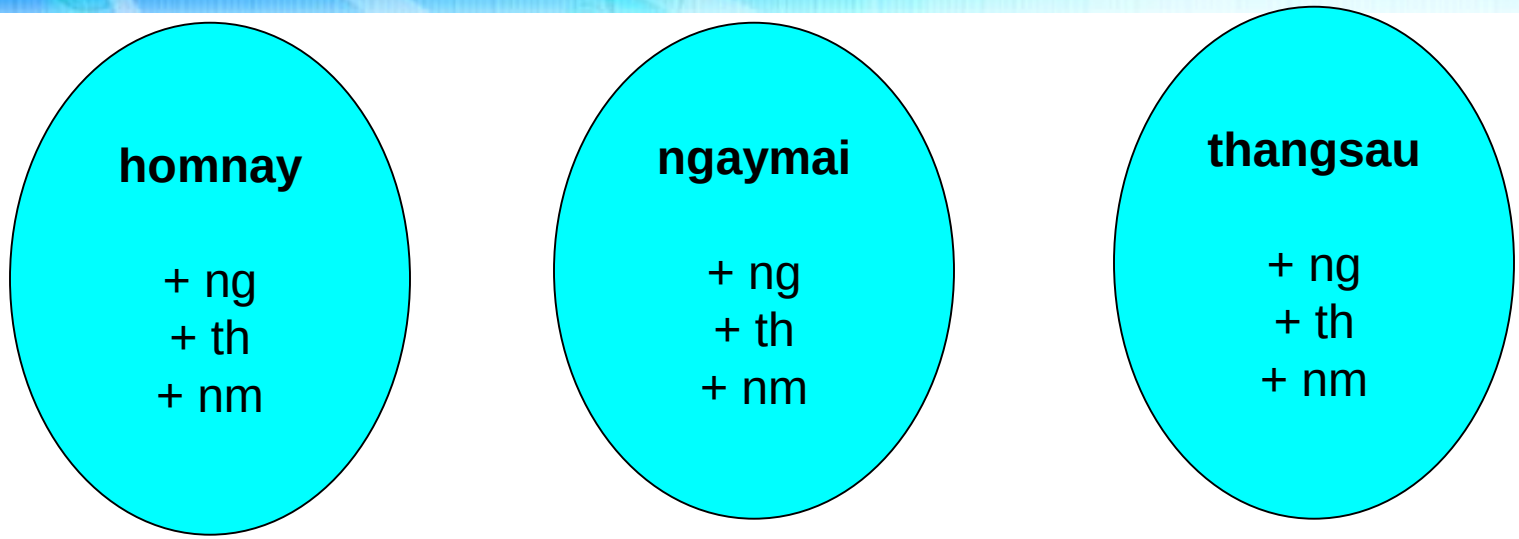
```
Ngay::Ngay(int ng, int th, int nm)
{
      ng = ng;
      th = th;
      nm = nm;
}
```

Con trỏ this (tt)

- Trong hàm main:

```
int main()
{
  Ngay homnay(18, 9, 2008);
  Ngay ngaymai(19, 9, 2008);
  Ngay thangsau(18, 10, 2008);
  ….
}
```

Con trỏ this (tt)




```
int Ngay::layThang() {
    return th;
}
```

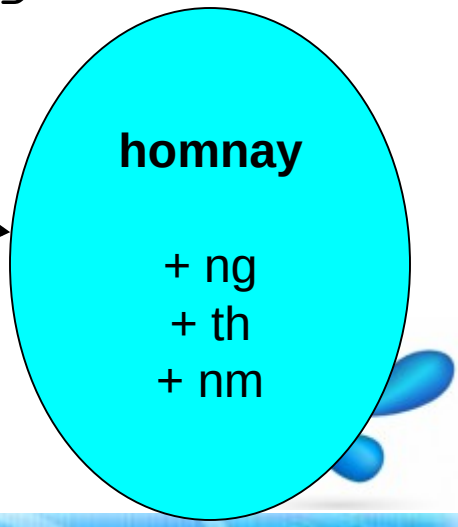
Làm sao trong phần cài đặt, chúng ta biết được `ng`, `th` hay `nm` nào đang được dùng?

Con trỏ this (tt)

- Trong C++, trình biên dịch tự động thêm vào trong các đối số của hàm 1 con trỏ **this**
- Con trỏ **this** trỏ tới đối tượng tương ứng hiện tại

```
int thang = homnay.layThang();
```

```
int Ngay::layThang(Ngay* const this)
{
    return this->th;
}
```



Đoạn code ban đầu

```
Ngay::Ngay(int ng, int th, int nm)
{
      ng = ng;
      th = th;
      nm = nm;
}
```

- Đúng về cú pháp
- Đúng về ngữ nghĩa
- Nhưng, sai mục đích

Đoạn code

- Phải được viết lại như sau:

```
Ngay::Ngay(int ng, int th, int nm)
{
      this->ng = ng;
      this->th = th;
      this->nm = nm;
}
```

Tuy nhiên, trong trường hợp này do các đối số bị trùng tên với thành phần dữ liệu nên mới xảy ra vấn đề vừa nêu. Nếu không, hàm sẽ tự động ngầm hiểu con trỏ this cho các biến có tên thuộc lớp đối tượng tương ứng

Con trỏ this trong các hàm

- Trong các hàm, con trỏ **this** (trỏ tới đối tượng được khởi tạo tương ứng) được truyền vào hàm 1 cách không tường minh
- Các đối khác được khai báo bình thường trong hàm

Ví dụ:

```
float Diem::tinhKhoangCach(Diem d) {
  return sqrt((x - d.x)*(x - d.x)+(y - d.y)*(y - d.y))
}
```

Tóm tắt

- Hàm dựng:
  - ✓Khởi tạo thông tin ban đầu cho đối tượng.
  - ✓Tự động thực hiện khi đối tượng được tạo lập.
  - ✓Mỗi lớp có thể có nhiều hàm dựng.
- Hàm hủy:
  - ✓Dọn dẹp bộ nhớ cho đối tượng.
  - ✓Tự động thực hiện khi đối tượng bị hủy.
  - ✓Mỗi lớp có duy nhất một hàm hủy.

Tóm tắt

- Hàm toán tử:
  - ✓Hàm có tên là toán tử.
  - ✓Dùng định nghĩa lại toán tử.
  - ✓Ràng buộc:
    - Ngôi của toán tử giữ nguyên.
    - Độ ưu tiên của toán tử không đổi.
    - Không thể tạo toán tử mới.
    - Không thể định nghĩa lại toán tử cho kiểu cơ bản.
  - ✓Có 2 loại hàm toán tử:
    - Toán tử độc lập.
    - Toán tử thuộc lớp.

Nội dung

- Hàm dựng
- Hàm hủy
- Hàm toán tử
- Bài tập

Bài tập

Bài tập 3.1:

Bổ sung vào lớp **phân số** những phương thức sau:

(Nhóm tạo hủy)

- ✓ Khởi tạo mặc định phân số = 0.
- ✓ Khởi tạo với tử và mẫu cho trước.
- ✓ Khởi tạo với giá trị số nguyên cho trước.
- ✓ Khởi tạo từ một phân số khác.

(Nhóm toán tử)

- ✓ Toán tử số học: +, -, *, /, =, +=, -=.
- ✓ Toán tử so sánh: >, <, ==, >=, <=, !=.
- ✓ Toán tử một ngôi: ++, --.
- ✓ Toán tử ép kiểu: (float), (int).
- ✓ Toán tử nhập, xuất: >>, <<.

Bài tập

Bài tập 3.2:

Bổ sung vào lớp **số phức** những phương thức sau:

(Nhóm tạo hủy)

- ✓ Khởi tạo mặc định số phức = 0.
- ✓ Khởi tạo với phần thực và phần ảo cho trước.
- ✓ Khởi tạo với giá trị thực cho trước.
- ✓ Khởi tạo từ một số phức khác.

(Nhóm toán tử)

- ✓ Toán tử số học: +, -, *, /, =, +=, -=.
- ✓ Toán tử so sánh: >, <, ==, >=, <=, !=.
- ✓ Toán tử một ngôi: ++, --.
- ✓ Toán tử ép kiểu: (float), (int).
- ✓ Toán tử nhập, xuất: >>, <<.

Bài tập

Bài tập 3.3:

Bổ sung vào lớp **đơn thức** những phương thức sau:

(Nhóm tạo hủy)

- ✓ Khởi tạo mặc định đơn thức = 0.
- ✓ Khởi tạo với hệ số và số mũ cho trước.
- ✓ Khởi tạo với hệ số cho trước, số mũ = 0.
- ✓ Khởi tạo từ một đơn thức khác.

(Nhóm toán tử)

- ✓ Toán tử số học: +, -, *, /, =, +=, -=.
- ✓ Toán tử so sánh: >, <, ==, >=, <=, !=.
- ✓ Toán tử một ngôi: ++, --.
- ✓ Toán tử ép kiểu: (float), (int).
- ✓ Toán tử nhập, xuất: >>, <<.

Bài tập

□ Bài tập 3.4:

Bổ sung vào lớp **học sinh** những phương thức sau:

(Nhóm tạo hủy)

- ✓ Khởi tạo với họ tên và điểm văn, toán cho trước.
- ✓ Khởi tạo với họ tên cho trước, điểm văn, toán = 0.
- ✓ Khởi tạo từ một học sinh khác.

(Nhóm toán tử)

- ✓ Toán tử gán: =.
- ✓ Toán tử so sánh (ĐTB): >, <, ==, >=, <=, !=.
- ✓ Toán tử nhập, xuất: >>, <<.

Bài tập

□ Bài tập 3.5:

Bổ sung vào lớp **mảng** những phương thức sau:

(Nhóm tạo hủy)

- ✓ Khởi tạo mặc định mảng kích thước = 0.
- ✓ Khởi tạo với kích thước cho trước, các phần tử = 0.
- ✓ Khởi tạo từ một mảng int [ ] với kích thước cho trước.
- ✓ Khởi tạo từ một đối tượng IntArray khác.
- ✓ Hủy đối tượng IntArray, thu hồi bộ nhớ.

(Nhóm toán tử)

- ✓ Toán tử gán: =.
- ✓ Toán tử mảng: [ ].
- ✓ Toán tử ép kiểu: (int *).
- ✓ Toán tử nhập, xuất: >>, <<.

Lời cảm ơn

- Nội dung được xây dựng dựa trên slide trình bày của Thầy Đinh Bá Tiến, Thầy Nguyễn Minh Huy.