Lập trình hướng đối tượng (P.1)

Kỹ thuật Lập trình (CO1027)

Ngày 25 tháng 4 năm 2021

ThS. Trần Ngọc Bảo Duy

Khoa Khoa học và Kỹ thuật Máy tính

Trường Đại học Bách Khoa, ĐHQG-HCM

Tổng quan

1. **Tại sao phải có LTHĐT?**
2. **Các khái niệm cơ bản**
   - Khái niệm
   - Con trỏ this
   - Tính khả kiến
3. **Nguyên tắc thiết kế lớp**
4. **Phương thức đặc biệt**
   - Hàm khởi tạo
   - Hàm hủy
   - Định nghĩa toán tử

ThS.
Trần Ngọc Bảo Duy

So sánh chuỗi giữa C và C++

```
1 int main() {
2     char* str1 = "Bach␣Khoa";
3     cout << strlen(str1);
4
5     string str2 = "Bach␣Khoa";
6     cout << str2.length();
7 }
```

- Với `str1`, muốn sử dụng hàm `strlen` và truyền nó vào như một tham số.

ThS.
Trần Ngọc Bảo Duy

So sánh chuỗi giữa C và C++

```
1 int main() {
2     char* str1 = "Bach Khoa";
3     cout << strlen(str1);
4
5     string str2 = "Bach Khoa";
6     cout << str2.length();
7 }
```

- Với `str1`, muốn sử dụng hàm `strlen` và truyền nó vào như một tham số.
- Với `str2`, ta có thể gọi `length()` thông qua toán tử . chứ không cần truyền thông qua hàm khác.

ThS.
Trần Ngọc Bảo Duy

So sánh chuỗi giữa C và C++

```
1 int main() {
2     char* str1 = "Bach Khoa";
3     cout << strlen(str1);
4
5     string str2 = "Bach Khoa";
6     cout << str2.length();
7 }
```

- Với `str1`, muốn sử dụng hàm `strlen` và truyền nó vào như một tham số.
- Với `str2`, ta có thể gọi `length()` thông qua toán tử . chứ không cần truyền thông qua hàm khác.
  Rõ ràng, `struct` trong C không thể định nghĩa hàm trong thân của nó.

ThS.
Trần Ngọc Bảo Duy

So sánh chuỗi giữa C và C++

```
1 int main() {
2     char* str1 = "Bach Khoa";
3     cout << strlen(str1);
4
5     string str2 = "Bach Khoa";
6     cout << str2.length();
7 }
```

- Với `str1`, muốn sử dụng hàm `strlen` và truyền nó vào như một tham số.
- Với `str2`, ta có thể gọi `length()` thông qua toán tử . chứ không cần truyền thông qua hàm khác.
  Rõ ràng, `struct` trong C không thể định nghĩa hàm trong thân của nó. Tại sao lại có thể gọi một hàm như một thành viên như vậy?

ThS.
Trần Ngọc Bảo Duy

Kiểu `struct` trong C

Xét đoạn chương trình:

```
1  struct Date {
2      int day, month, year;
3  };
4
5  int main() {
6      int a;
7      double d;
8      Date c = { 17, 4, 2021 };
9
10     return 0;
11 }
```


ThS.
Trần Ngọc Bảo Duy

Kiểu `struct` trong C

Xét đoạn chương trình:

```
1  struct Date {
2      int day, month, year;
3  };
4
5  int main() {
6      int a;
7      double d;
8      Date c = { 17, 4, 2021 };
9
10     return 0;
11 }
```

Bộ biên dịch cấp phát 3 vùng nhớ có tên: `a`, `d`, và `c` trên **STACK**:

1. `d`: 4 bytes
2. `d`: 8 bytes
3. `c`: 12 bytes

ThS.
Trần Ngọc Bảo Duy

Kiểu `struct` trong C

Xét đoạn chương trình:

```
1  struct Date {
2      int day, month, year;
3  };
4
5  int main() {
6      int a;
7      double d;
8      Date c = { 17, 4, 2021 };
9
10     return 0;
11 }
```

Bộ biên dịch cấp phát 3 vùng nhớ có tên: `a`, `d`, và `c` trên **STACK**:

1. `d`: 4 bytes
2. `d`: 8 bytes
3. `c`: 12 bytes

Cả 3 vùng nhớ này đều **THỤ ĐỘNG**, **CHỈ CÓ CÔNG NĂNG LÀ CHỨA** các giá trị của kiểu được mô tả.

ThS.
Trần Ngọc Bảo Duy

Kiểu `struct` trong C

Vì thụ động, nên khi cần xử lý dữ liệu, thực hiện:

- Tạo ra hàm
- Gọi hàm và truyền dữ liệu vào

ThS.
Trần Ngọc Bảo Duy

Kiểu `struct` trong C

Vì thụ động, nên khi cần xử lý dữ liệu, thực hiện:

- Tạo ra hàm
- Gọi hàm và truyền dữ liệu vào: **Ví dụ như hàm `print`.**

```
1  struct Date { int day, month, year; };
2
3  void print(Date& d) {
4      cout << d.day << "/" << d.month
5           << "/" << d.year;
6  }
7
8  int main() {
9      Date c = {20, 5, 2017};
10     print(c);
11 }
```


Nhược điểm của tính thụ động

Xét ví dụ sau:

```
1  struct Date { int day, month, year; }
2
3  void change(Date& d) {
4      d.day = 32;
5  }
6
7  int main() {
8      Date c = {20, 5, 2017};
9      change(c);
10 }
```


ThS.
Trần Ngọc Bảo Duy

Nhược điểm của tính thụ động

Xét ví dụ sau:

```
1  struct Date { int day, month, year; }
2
3  void change(Date& d) {
4      d.day = 32;
5  }
6
7  int main() {
8      Date c = {20, 5, 2017};
9      change(c);
10 }
```

Nếu có một hàm `change` như trên, trong thân hàm, thay đổi giá trị của `d` truyền vào thành `32`, thì chính `d` cũng không thể **kháng cự** lại được.

ThS.
Trần Ngọc Bảo Duy

Nhược điểm của tính thụ động

Xét ví dụ sau:

```
1  struct Date { int day, month, year; }
2
3  void change(Date& d) {
4      d.day = 32;
5  }
6
7  int main() {
8      Date c = {20, 5, 2017};
9      change(c);
10 }
```

Nhược điểm của tính thụ động

1. Khó đảm bảo được ràng buộc trên dữ liệu.
2. Khó biết được dữ liệu có thể được xử lý bởi hàm nào.

ThS.
Trần Ngọc Bảo Duy

Nhược điểm của tính thụ động

Xét ví dụ sau:

```
1  struct Date { int day, month, year; }
2
3  void change(Date& d) {
4      d.day = 32;
5  }
6
7  int main() {
8      Date c = {20, 5, 2017};
9      change(c);
10 }
```

Nhược điểm của tính thụ động

1. Khó đảm bảo được ràng buộc trên dữ liệu.
2. Khó biết được dữ liệu có thể được xử lý bởi hàm nào.

Lập trình hướng đối tượng có tính chất **đóng gói** (encapsulation) có thể giải quyết được vấn đề trên.

ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

1. **Lớp (class)**: là một kiểu dữ liệu do người lập trình tạo ra, tương tự với `struct`.

ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

1. **Lớp (class)**: là một kiểu dữ liệu do người lập trình tạo ra, tương tự với `struct`.

2. **Đối tượng (object, instance):** là một biến tạo ra từ kiểu lớp.
   Ví dụ: Giả sử ta có lớp `X`.
   Lệnh `X c;` sẽ tạo ra một đối tượng, đặt tên là c có kiểu là `X`, nghĩa là có một vùng nhớ tên c được tạo ra.

ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

1. **Lớp (class)**: là một kiểu dữ liệu do người lập trình tạo ra, tương tự với `struct`.

2. **Đối tượng (object, instance):** là một biến tạo ra từ kiểu lớp.
   Ví dụ: Giả sử ta có lớp `X`.
   Lệnh `X c;` sẽ tạo ra một đối tượng, đặt tên là c có kiểu là `X`, nghĩa là có một vùng nhớ tên c được tạo ra.

Phân biệt giữa lớp và đối tượng

- Lớp như cái khuôn để từ đó tạo ra các đối tượng.
- Đối tượng là một biến cụ thể mang kiểu là lớp đối tượng đó.

Các khái niệm cơ bản

❸ **Mô tả lớp có gì khác mô tả một `struct` trong C?**
Khi mô tả kiểu này, cần mô tả:

ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

3. **Mô tả lớp có gì khác mô tả một `struct` trong C?**
   Khi mô tả kiểu này, cần mô tả:
   1. Các **dữ liệu (thuộc tính - attributes)** mà một đối tượng của lớp phải có.

      Ví dụ: Một đối tượng thuộc lớp **Cat** (để mô tả con mèo) phải có chiều cao, cân nặng, màu sắc, ...

ThS.
Trần Ngọc Bảo Duy

3 Mô tả lớp có gì khác mô tả một `struct` trong C?

Khi mô tả kiểu này, cần mô tả:

1. Các **dữ liệu (thuộc tính - attributes)** mà một đối tượng của lớp phải có.

   Ví dụ: Một đối tượng thuộc lớp **Cat** (để mô tả con mèo) phải có chiều cao, cân nặng, màu sắc, ...

2. Các **hàm (phương thức - methods)** để mô tả các hành động mà đối tượng có thể thực hiện được.

   Ví dụ: Một đối tượng thuộc lớp **Cat** (để mô tả con mèo) có các hành vi như đi, nằm, kêu, ...

   Chính nhờ việc có thể mô tả được hành động của đối tượng mà đã thể hiện được **tính chủ động** của đối tượng (không chỉ là một vùng nhớ thụ động).

ThS.
Trần Ngọc Bảo Duy

ThS.
Trần Ngọc Bảo Duy

3. **Mô tả lớp có gì khác mô tả một `struct` trong C?**
   Khi mô tả kiểu này, cần mô tả:
   1. Các **dữ liệu (thuộc tính - attributes)** mà một đối tượng của lớp phải có.
   2. Các **hàm (phương thức - methods)** để mô tả các hành động mà đối tượng có thể thực hiện được.

Kiểu `struct`

- Trong C: Mô tả kiểu này không có mô tả hàm/phương thức như kiểu lớp.
- Trong C++: Có thể mô tả được, vì `struct` thực ra đã là kiểu lớp.

Các khái niệm cơ bản

4. **Tạo đối tượng:** Về bản chất, lớp cũng giống như cấu trúc hay các kiểu dữ liệu khác.

ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

4. **Tạo đối tượng:** Về bản chất, lớp cũng giống như cấu trúc hay các kiểu dữ liệu khác.

    1. Tạo tĩnh trên STACK:

```
1 MyClass obj;
2 MyClass obj(10, 15);
```

    2. Tạo động trên HEAP:

```
1 MyClass *ptr = new MyClass;
2 MyClass *ptr = new MyClass(10, 15);
3 // ...
4 delete ptr;
```


Các khái niệm cơ bản

4. **Tạo đối tượng:** Về bản chất, lớp cũng giống như cấu trúc hay các kiểu dữ liệu khác.

   3. Tạo mảng tĩnh trên STACK:

```
1 MyClass obj[SIZE];
```

   4. Tạo mảng động trên HEAP:

```
1 MyClass *ptr = new MyClass[SIZE];
2 // ...
3 delete[] ptr;
```


ThS.
Trần Ngọc Bảo Duy

Các khái niệm cơ bản

❺ Truy xuất thuộc tính và gọi phương thức:

ThS.
Trần Ngọc Bảo Duy

5 Truy xuất thuộc tính và gọi phương thức:

1. Đối tượng trên STACK:

```
1 MyClass obj;
2 // ...
3 obj.attr1;
4 obj.method1();
```

2. Đối tượng trên HEAP:

```
1 MyClass *ptr = new MyClass;
2 // ...
3 ptr->attr1;
4 ptr->method1();
```


ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

Ví dụ

Hãy tạo ra một kiểu **Date** theo yêu cầu sau:

- Một đối tượng của **Date** có phải chứa được dữ liệu về ngày (day), tháng (month), và năm (year).
- Có thể đón nhận lời yêu cầu **print**. Một khi nó (đối tượng) nhận được yêu cầu này, nó in ra màn hình ngày, tháng, và năm mà nó đang giữ.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

Ví dụ

Hãy tạo ra một kiểu **Date** theo yêu cầu sau:

- Một đối tượng của **Date** có phải chứa được dữ liệu về ngày (day), tháng (month), và năm (year).
- Có thể đón nhận lời yêu cầu **print**. Một khi nó (đối tượng) nhận được yêu cầu này, nó in ra màn hình ngày, tháng, và năm mà nó đang giữ.
- **NÂNG CAO:** Có thể nhận lời yêu cầu **changeDay** kèm theo một giá trị ngày mới. Một khi nó (đối tượng) nhận yêu cầu này, nó sẽ kiểm tra giá trị mới đó có nằm trong đoạn [1, 31] không, nếu sai nó ném (throw) lỗi `std::invalid_argument`. Ngược lại, nó sẽ thiết lập giá trị lại cho giá trị ngày mà nó đang giữ.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1  class Date {
2  public:
3      int day, month, year;
4
5      void print() {
6          cout << day << "/" << month
7                       << "/" << year;
8      }
9  };
```


ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1 class Date {
2 public:
3     int day, month, year;
4
5     void print() {
6         cout << day << "/" << month
7                     << "/" << year;
8     }
9 };
```

- Dòng 1: Từ khóa `class` đi kèm tên lớp `Date` dùng để khai báo một lớp đối tượng.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1 class Date {
2 public:
3     int day, month, year;
4
5     void print() {
6         cout << day << "/" << month
7                     << "/" << year;
8     }
9 };
```

- Dòng 1: Từ khóa `class` đi kèm tên lớp `Date` dùng để khai báo một lớp đối tượng.
- Dòng 1 - 9: được đóng trong { }; là thân, là định nghĩa lớp `Date`.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1 class Date {
2 public:
3     int day, month, year;
4
5     void print() {
6         cout << day << "/" << month
7                    << "/" << year;
8     }
9 };
```

- Dòng 1: Từ khóa `class` đi kèm tên lớp `Date` dùng để khai báo một lớp đối tượng.
- Dòng 1 - 9: được đóng trong { }; là thân, là định nghĩa lớp `Date`.
- Dòng 3: Mô tả thuộc tính cho đối tượng của lớp `Date`.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1 class Date {
2 public:
3     int day, month, year;
4
5     void print() {
6         cout << day << "/" << month
7                     << "/" << year;
8     }
9 };
```

- Dòng 1: Từ khóa `class` đi kèm tên lớp `Date` dùng để khai báo một lớp đối tượng.
- Dòng 1 - 9: được đóng trong { }; là thân, là định nghĩa lớp `Date`.
- Dòng 3: Mô tả thuộc tính cho đối tượng của lớp `Date`.
- Dòng 5 - 8: Mô tả phương thức cho đối tượng của lớp `Date`: khai báo + định nghĩa hàm.

ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
1 class Date {
2 public:
3     int day, month, year;
4
5     void print() {
6         cout << day << "/" << month
7                     << "/" << year;
8     }
9 };
```

- Dòng 1: Từ khóa `class` đi kèm tên lớp `Date` dùng để khai báo một lớp đối tượng.
- Dòng 1 - 9: được đóng trong { }; là thân, là định nghĩa lớp `Date`.
- Dòng 3: Mô tả thuộc tính cho đối tượng của lớp `Date`.
- Dòng 5 - 8: Mô tả phương thức cho đối tượng của lớp `Date`: khai báo + định nghĩa hàm.
- Dòng 2: từ khóa `public` để nói đến tính khả kiến (visibility) của các thành viên (thuộc tính và phương thức).

ThS.
Trần Ngọc Bảo Duy

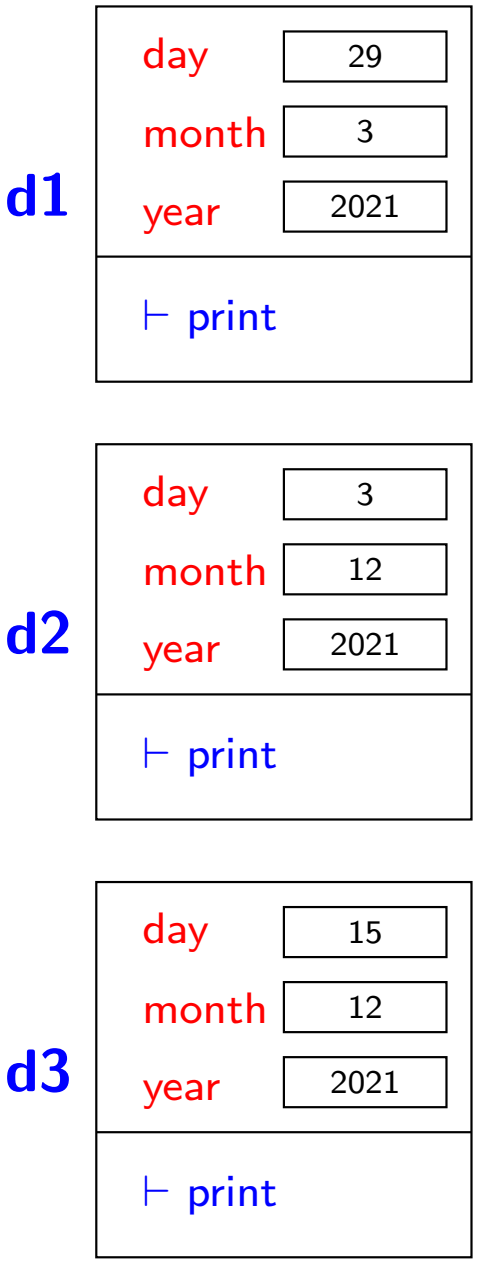
Khái niệm cơ bản: Ví dụ

```
int main() {
    Date d1 = {29, 3, 2021};
    Date d2 = {3, 12, 2021};
    Date d3 = {15, 12, 2021};

    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```


ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
int main() {
    Date d1 = {29, 3, 2021};
    Date d2 = {3, 12, 2021};
    Date d3 = {15, 12, 2021};

    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```



ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

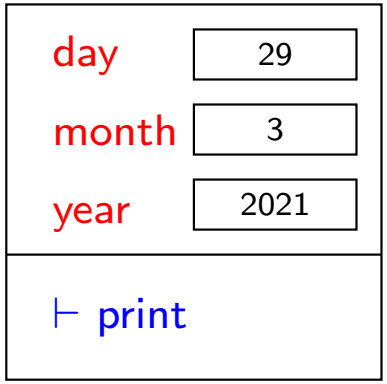
```
int main() {
    Date d1 = {29, 3, 2021};
    Date d2 = {3, 12, 2021};
    Date d3 = {15, 12, 2021};

    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```

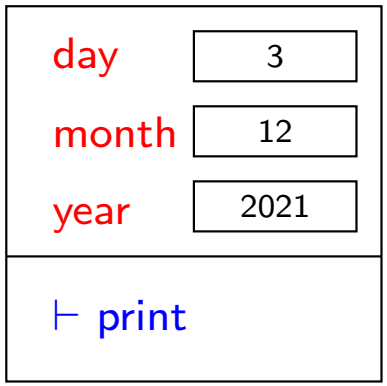
Kết quả xuất ra màn hình:

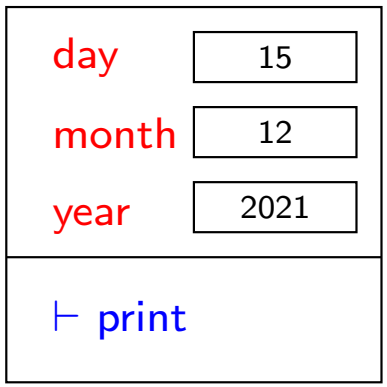
29/3/2021

3/12/2021

15/12/2021





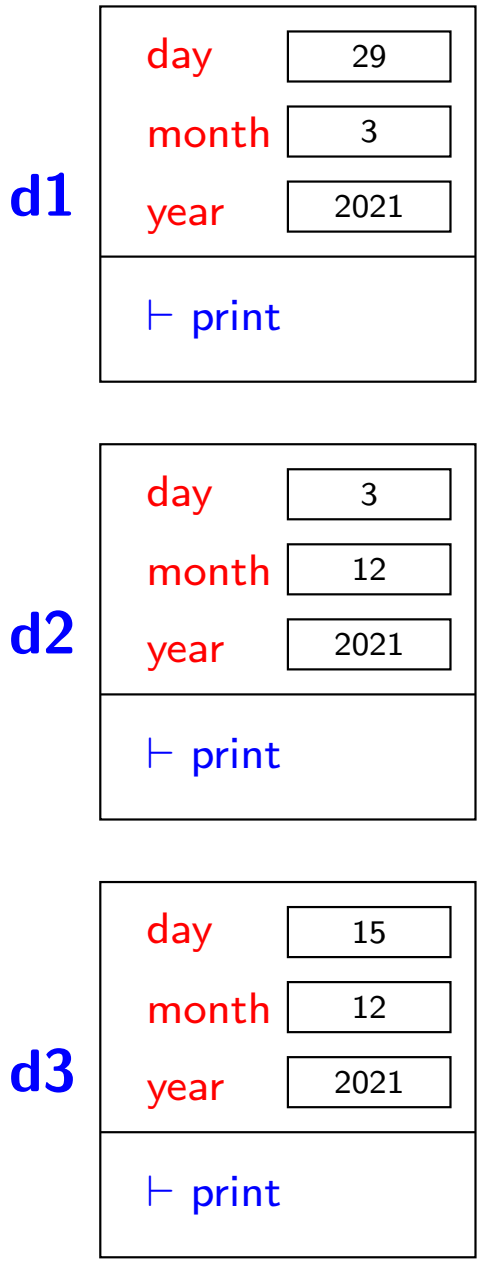


ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
int main() {
    Date d1 = {29, 3, 2021};
    Date d2 = {3, 12, 2021};
    Date d3 = {15, 12, 2021};

    d1.print();
    d2.print();
    d3.print();

    return 0;
}
```

Từ khóa `public` có nghĩa là thành viên có thể truy cập bất kỳ nơi nào, kể cả ngoài định nghĩa của lớp nên có thể truy cập được trong thân hàm `main` như `print`.



ThS.
Trần Ngọc Bảo Duy

Khái niệm cơ bản: Ví dụ

```
class Date {
public:
    int day, month, year;

    void print() {
        cout << day << "/" << month
                    << "/" << year;
    }

    void changeDay(int newDay) {
        if (new_day < 1 || new_day > 31)
            throw invalid_argument("Invalid day");

        day = newDay;
    }
};
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ `this`

Địa chỉ đến (byte đầu tiên của) đối tượng được lưu trong biến `this` (là từ khóa). Địa chỉ này chỉ có thể dùng được bên trong các hàm thành viên của đối tượng.

- Bên trong **print** của **d1**: `this` chỉ đến vùng **d1**.
- Bên trong **print** của **d2**: `this` chỉ đến vùng **d2**.

ThS.
Trần Ngọc Bảo Duy

Con trỏ this

```
class Date {
public:
    int day, month, year;

    void print() {
        cout << day << "/" << month
                    << "/" << year;
    }
};
```


Con trỏ this

```
class Date {
public:
    int day, month, year;

    void print() {
        cout << day << "/" << month
                    << "/" << year;
    }
};
```

có thể viết lại thành:

```
class Date {
public:
    int day, month, year;

    void print() {
        cout << this->day << "/" << this->month
                          << "/" << this->year;
    }
};
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ this

```
1  class Date {
2  public:
3      int day, month, year;
4
5      void print() {
6          // ...
7      }
8
9      void setDay(int day) {
10
11     }
12 };
```


ThS.
Trần Ngọc Bảo Duy

Con trỏ this

```
1  class Date {
2  public:
3      int day, month, year;
4
5      void print() {
6          // ...
7      }
8
9      void setDay(int day) {
10
11     }
12 };
```

Tại dòng số 10, có hai biến cùng tên là `day`:

- `day`: của thông số.
- `day`: dữ liệu của lớp.

ThS.
Trần Ngọc Bảo Duy

Con trỏ this

```
1  class Date {
2  public:
3      int day, month, year;
4
5      void print() {
6          // ...
7      }
8
9      void setDay(int day) {
10
11     }
12 };
```

Mặc nhiên, day của thông số là ưu tiên. Do đó để gán day của thông số vào day dữ liệu của lớp thì dùng this:

```
this->day = day;
```


ThS.
Trần Ngọc Bảo Duy

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến

- **Tính khả kiến (visibility)** là tính chất cho biết thuộc tính và phương thức (gọi chung là thành viên) của lớp được nhìn thấy và dùng được (còn gọi là truy cập được) ở nơi nào của chương trình.
- Có 3 mức khả kiến:
  1. `public`:
  2. `protected`:
  3. `private`:

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến

- **Tính khả kiến (visibility)** là tính chất cho biết thuộc tính và phương thức (gọi chung là thành viên) của lớp được nhìn thấy và dùng được (còn gọi là truy cập được) ở nơi nào của chương trình.
- Có 3 mức khả kiến:
  1. `public`: Thành viên có thể được nhìn thấy và truy xuất được ở bất kỳ nơi nào của chương trình.
  2. `protected`:
  3. `private`:

Tính khả kiến

- **Tính khả kiến (visibility)** là tính chất cho biết thuộc tính và phương thức (gọi chung là thành viên) của lớp được nhìn thấy và dùng được (còn gọi là truy cập được) ở nơi nào của chương trình.
- Có 3 mức khả kiến:
  1. `public`:
  2. `protected`: Thành viên có thể được nhìn thấy và truy xuất được:
     - ở các phương thức thành viên của lớp đó.
     - ở các phương thức của các lớp dẫn ra (lớp con) từ lớp đó.
  3. `private`:

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến

- **Tính khả kiến (visibility)** là tính chất cho biết thuộc tính và phương thức (gọi chung là thành viên) của lớp được nhìn thấy và dùng được (còn gọi là truy cập được) ở nơi nào của chương trình.
- Có 3 mức khả kiến:
  1. `public`:
  2. `protected`:
  3. `private`: Thành viên **CHỈ CÓ THỂ** được nhìn thấy và truy xuất được ở các phương thức thành viên của lớp đó.

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến

Tính khả kiến	`public`	`private`
Các phương thức của lớp	x	x
Các phương thức hoặc hàm ngoài lớp	x	

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến: Ví dụ

```
1  class Foo {
2  private:
3      int value;
4
5  public:
6      void print() {
7          cout << this->value << endl;
8      }
9  };
10
11 class Bar {
12 public:
13     void func() {
14         Foo obj;
15         obj.print();
16     }
17 };
```


ThS.
Trần Ngọc Bảo Duy

Tính khả kiến: Ví dụ

```
1  class Foo {
2  private:
3      int value;
4
5  public:
6      void print() {
7          cout << this->value << endl;
8      }
9  };
10
11 class Bar {
12 public:
13     void func() {
14         Foo obj;
15         obj.print();
16     }
17 };
```

Đối với lớp `Foo`: Cho dù `value` được khai báo với tính `private`, nhưng nó vẫn truy cập được trong phương thức `print`.

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến: Ví dụ

```
1  class Foo {
2  private:
3      int value;
4
5  public:
6      void print() {
7          cout << this->value << endl;
8      }
9  };
10
11 class Bar {
12 public:
13     void func() {
14         Foo obj;
15         obj.print();
16     }
17 };
```

Đối với lớp `Foo`: Cho dù `value` được khai báo với tính `private`, nhưng nó vẫn truy cập được trong phương thức `print`.

TÓM LẠI: hàm thành viên luôn luôn truy cập được các biến thành viên và hàm thành viên khác, không quan tâm tính khả kiến là gì.

ThS.
Trần Ngọc Bảo Duy

Tính khả kiến: Ví dụ

```
1  class Foo {
2  private:
3      int value;
4
5  public:
6      void print() {
7          cout << this->value << endl;
8      }
9  };
10
11 class Bar {
12 public:
13     void func() {
14         Foo obj;
15         obj.print();
16     }
17 };
```

Đối với lớp Bar, trong hàm func:

- Gọi hàm print vì print của Foo có tính public.
- TUY NHIÊN: nếu nó truy cập vào biến value do value có tính private.

ThS.
Trần Ngọc Bảo Duy

Nguyên tắc thiết kế lớp

Nguyên tắc

- Mỗi lớp biểu diễn một tập hợp các đối tượng có liên quan chặt chẽ, có sự giống nhau về nhiều mặt (dữ liệu lưu trữ hay hành động của đối tượng).

ThS.
Trần Ngọc Bảo Duy

Nguyên tắc thiết kế lớp

Nguyên tắc

- Mỗi lớp biểu diễn một tập hợp các đối tượng có liên quan chặt chẽ, có sự giống nhau về nhiều mặt (dữ liệu lưu trữ hay hành động của đối tượng).
- Do đó:
  - Cần dùng tính khả kiến để đảm bảo chỉ cung cấp ra bên ngoài (dùng tính `public`) những dữ liệu và phương thức được lựa chọn cẩn thận.
  - Đồng thời, cần che dấu (tính `private`) những chi tiết về hiện thực cũng như những phương thức và dữ liệu nội bộ.

ThS.
Trần Ngọc Bảo Duy

Gợi ý cách hiện thực

- Thường che dấu dữ liệu của lớp: dùng tính `private`.
- Chỉ cho phép bên ngoài truy xuất dữ liệu thông qua những phương thức được thiết kế sẵn: dùng tính `public`.
- Đối với mỗi thuộc tính:
  - Bổ sung một phương thức cho phép bên ngoài đọc giá trị của thuộc tính: gọi là **getter, có tính public**.
  - Bổ sung một phương thức cho phép bên ngoài ghi giá trị mới vào thuộc tính: gọi là **setter, có tính public**.
- **Getter**: Lấy giá trị thuộc tính, định dạng, xuất ra ngoài.
- **Setter**: Lấy giá trị được truyền vào, kiểm tra có tính hợp lệ của dữ liệu truyền vào:
  - báo lỗi
  - hoặc, gán vào thuộc tính.

ThS.
Trần Ngọc Bảo Duy

Thiết kế lớp: Ví dụ

Với lớp `Date` nói trên:

- **day, month, year**: nên là `private`.
- Với mỗi thuộc tính: có cặp **getter** và **setter**
- Có thể cung cấp một số phương thức tiện ích khác:
  - Lấy `Date` dưới dạng một chuỗi theo định dạng.
  - In `Date` in màn hình.
  - Tính số ngày nằm giữa hai đối tượng `Date`.

ThS.
Trần Ngọc Bảo Duy

Thiết kế lớp: Ví dụ

```
class Date {
private:
    int day, month, year;

public:
    int getDay() { return this->day; }
    void setDay(int day) { this->day = day; }
    int getMonth() { return this->month; }
    void setMonth(int month) { this->month = month; }
    int getYear() { return this->year; }
    void setYear(int year) { this->year = year; }

    string toString() {
        return to_string(day) + "/" + to_string(month)
                              + "/" + to_string(year);
    }

    void print() {
        cout << day << "/" << month
                    << "/" << year << endl;
    }
}
```


ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo

Giả sử ta đã có lớp `Date` trên:

```
1 int main() {
2     Date d;
3     d.setDay(29);
4     d.setMonth(3);
5     d.setYear(2021);
6     d.print();
7
8     return 0;
9 }
```

- Dòng 1: Khai báo một đối tượng `d` có kiểu `Date`. o các thuộc tính `day, month, year` đều có tính `private` nên không thể khởi tạo như trước.

Hàm khởi tạo

Giả sử ta đã có lớp `Date` trên:

```
1  int main() {
2      Date d;
3      d.setDay(29);
4      d.setMonth(3);
5      d.setYear(2021);
6      d.print();
7
8      return 0;
9  }
```

- Dòng 1: Khai báo một đối tượng `d` có kiểu `Date`. Do các thuộc tính `day`, `month`, `year` đều có tính `private` nên không thể khởi tạo như trước.

ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo

Giả sử ta đã có lớp `Date` trên:

```
1  int main() {
2      Date d;
3      d.setDay(29);
4      d.setMonth(3);
5      d.setYear(2021);
6      d.print();
7
8      return 0;
9  }
```

- Dòng 1: Khai báo một đối tượng d có kiểu `Date`. o các thuộc tính `day, month, year` đều có tính `private` nên không thể khởi tạo như trước.
- Dòng 3, 4, 5: Vì vậy, ta phải dùng các **setter** để thiết lập dữ liệu ban đầu của đối tượng.

Hàm khởi tạo

Giả sử ta đã có lớp `Date` trên:

```
1 int main() {
2     Date d;
3     d.setDay(29);
4     d.setMonth(3);
5     d.setYear(2021);
6     d.print();
7
8     return 0;
9 }
```

- Dòng 1: Khai báo một đối tượng `d` có kiểu `Date`. o các thuộc tính `day, month, year` đều có tính `private` nên không thể khởi tạo như trước.
- Dòng 3, 4, 5: Vì vậy, ta phải dùng các **setter** để thiết lập dữ liệu ban đầu của đối tượng.
- Dòng 6: Lời gọi `print` in ra màn hình 29/3/2021.

ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo

Giả sử ta đã có lớp `Date` trên:

```
1 int main() {
2     Date d;
3     d.setDay(29);
4     d.setMonth(3);
5     d.setYear(2021);
6     d.print();
7
8     return 0;
9 }
```

- Dòng 1: Khai báo một đối tượng `d` có kiểu `Date`. o các thuộc tính `day, month, year` đều có tính `private` nên không thể khởi tạo như trước.
- Dòng 3, 4, 5: Vì vậy, ta phải dùng các **setter** để thiết lập dữ liệu ban đầu của đối tượng.
- Dòng 6: Lời gọi `print` in ra màn hình 29/3/2021.

$\Rightarrow$ Việc tạo đối tượng `d` quá dài dòng, cần gọi đến 3 **setter**.

ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo

- **Đặc điểm cú pháp**:
  1. Không có kiểu trả về.
  2. Tên hàm cũng chính là tên lớp.
  3. Có thể có nhiều hàm khởi tạo, chỉ cần khác chữ ký (overloading).
- **Đặc điểm ngữ nghĩa**: Chỉ được gọi **1 và chỉ 1** lần duy nhất – chính là lúc tạo ra đối tượng.

ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo

Hàm khởi tạo

- **Đặc điểm cú pháp**:
  1. Không có kiểu trả về.
  2. Tên hàm cũng chính là tên lớp.
  3. Có thể có nhiều hàm khởi tạo, chỉ cần khác chữ ký (overloading).
- **Đặc điểm ngữ nghĩa**: Chỉ được gọi **1 và chỉ 1** lần duy nhất – chính là lúc tạo ra đối tượng.

Có 3 loại hàm khởi tạo:

1. Khởi tạo mặc nhiên (default constructor).
2. Khởi tạo sao chép (copy constructor).
3. Khởi tạo do người dùng tự định nghĩa (user-defined constructor).

ThS.
Trần Ngọc Bảo Duy

Khởi tạo mặc nhiên

- Đặc điểm cú pháp:
  - Không mô tả tường minh bất kỳ các loại hàm khởi tạo nào khác.
  - Hàm khởi tạo không có thông số, tức là tạo đối tượng không truyền bất cứ đối số nào.

Ví dụ:

Nếu `X` là một lớp thì dòng lệnh `X a;` hoặc `X a();` sẽ tạo ra đối tượng a và khởi tạo bằng khởi tạo mặc nhiên.

ThS.
Trần Ngọc Bảo Duy

Khởi tạo mặc nhiên

- Đặc điểm cú pháp:
  - Không mô tả tường minh bất kỳ các loại hàm khởi tạo nào khác.
  - Hàm khởi tạo không có thông số, tức là tạo đối tượng không truyền bất cứ đối số nào.
- Bộ thực thi sẽ tìm hàm khởi tạo không có thông số do người lập trình định nghĩa:
  1. Nếu không thấy, nhưng thì lại thấy có hàm khởi tạo khác $\rightarrow$ **báo lỗi**.
  2. Nếu không thấy bất kỳ hàm khởi tạo nào $\Rightarrow$ gọi hàm khởi tạo mặc nhiên, khởi tạo các thuộc tính vào trạng thái mặc nhiên.

ThS.
Trần Ngọc Bảo Duy

Khởi tạo mặc nhiên: Ví dụ

```
1  class Date {
2  private:
3      int day, month, year;
4
5  public:
6      Date() {
7          this->day = 1;
8          this->month = 1;
9          this->year = 1970;
10     }
11
12     // Other methods
13 };
```


ThS.
Trần Ngọc Bảo Duy

Khởi tạo sao chép (Copy constructor)

- Đặc điểm cú pháp: có prototype là một trong các dạng sau:

```
MyClass(const MyClass& other);
MyClass(MyClass& other);
MyClass(volatile const MyClass& other);
MyClass(volatile MyClass& other);
```


ThS.
Trần Ngọc Bảo Duy

Khởi tạo sao chép (Copy constructor)

- Đặc điểm ngữ nghĩa: Khởi tạo sao chép có thể được gọi khi:
  1. Tạo đối tượng và truyền vào đối số là đối tượng

```
MyClass a;
MyClass b(a);
```


ThS.
Trần Ngọc Bảo Duy

Khởi tạo sao chép (Copy constructor)

- Đặc điểm ngữ nghĩa: Khởi tạo sao chép có thể được gọi khi:
  1. Tạo đối tượng và truyền vào đối số là đối tượng

```
MyClass a;
MyClass b(a);
```

  2. Tạo đối tượng và khởi gán bằng đối tượng khác

```
MyClass a;
MyClass b = a;
```


ThS.
Trần Ngọc Bảo Duy

Khởi tạo sao chép (Copy constructor)

- Đặc điểm ngữ nghĩa: Khởi tạo sao chép có thể được gọi khi:
  3. Truyền đối tượng bằng trị vào hàm

```
void foo(MyClass x) { // ... }
int main() {
    MyClass a;
    foo(a);

    return 0;
}
```


ThS.
Trần Ngọc Bảo Duy

Khởi tạo sao chép (Copy constructor)

- Đặc điểm ngữ nghĩa: Khởi tạo sao chép có thể được gọi khi:
  4. Hàm trả về giá trị có kiểu lớp.

```
MyClass bar() {
    MyClass x;
    // ...
    return x;
}

int main() {
    MyClass a = bar();

    return 0;
}
```


ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo do người lập trình định nghĩa

- Đặc điểm cú pháp: Hàm khởi tạo có danh sách tham số không rỗng khác với khởi tạo copy.
- Đặc điểm ngữ nghĩa: Hàm khởi tạo có kèm các thông số phù hợp với các yêu cầu của bài toán thiết kế lớp.

ThS.
Trần Ngọc Bảo Duy

Hàm khởi tạo do người lập trình định nghĩa

- Đặc điểm cú pháp: Hàm khởi tạo có danh sách tham số không rỗng khác với khởi tạo copy.
- Đặc điểm ngữ nghĩa: Hàm khởi tạo có kèm các thông số phù hợp với các yêu cầu của bài toán thiết kế lớp.

Ví dụ: Khi lớp `MyClass` có hàm khởi tạo:

```
MyClass(int a, double* ptr);
```

Hàm khởi tạo đó sẽ được gọi khi:

```
double list[] = {10.5, 20.5, 40.5};
MyClass obj(1, list);
```


Hàm hủy (Destructor)

- Đặc điểm cú pháp:
  - Không có kiểu trả về.
  - Tên hàm có dạng: `~MyClass`. Ở đó, `MyClass` là tên lớp.
  - Không có thông số.

Hàm hủy (Destructor)

- Đặc điểm cú pháp:
  - Không có kiểu trả về.
  - Tên hàm có dạng: `~MyClass`. Ở đó, `MyClass` là tên lớp.
  - Không có thông số.

- Đặc điểm ngữ nghĩa: Chỉ được gọi 1 và chỉ 1 lần duy nhất – chính là lúc cần huỷ đối tượng.
  1. Khi ra khỏi tầm vực (kể cả kết thúc hàm) $\rightarrow$ huỷ các đối tượng trong tầm vực vừa ra.
  2. Khi người lập trình chủ động gọi `delete`.

ThS.
Trần Ngọc Bảo Duy

Công dụng của hàm khởi tạo và hàm hủy

1. **Hàm khởi tạo**: Dùng để khởi tạo tình trạng (trạng thái) ban đầu cho đối tượng khi được tạo ra.
2. **Hàm hủy**: Làm công việc dọn dẹp để chuẩn bị cho đối tượng bị huỷ.

ThS.
Trần Ngọc Bảo Duy

Một số công dụng thường gặp

Trường hợp	Hàm khởi tạo	Hàm hủy
Đơn giản	Khởi gán các biến, đưa đối tượng về trạng thái ban đầu nào đó	Không cần
Một biến thành viên có kiểu con trỏ, **cần được cấp phát động**	Gọi đến `new` để xin bộ nhớ, và khởi động giá trị	Gọi đến `delete` để giải phóng bộ nhớ.
Đọc/ ghi dữ liệu xuống file	Mở file, chuẩn bị cho việc ghi/đọc	Đóng file

ThS.
Trần Ngọc Bảo Duy

Định nghĩa lại toán tử (Operator overloading)

Giả sử có lớp `Complex` để mô tả các số phức. Trên tập số phức, phép cộng là tồn tại. Tuy nhiên:

```
1 Complex a;
2 Complex b;
3 Complex c = a + b;
```

Toán tử + ở dòng số 3 là không hợp lý vì kiểu `Complex` là kiểu do người dùng tự định nghĩa, C++ chưa có định nghĩa cho phép + cho kiểu này.

ThS.
Trần Ngọc Bảo Duy

Định nghĩa lại toán tử (Operator overloading)

Giả sử có lớp `Complex` để mô tả các số phức. Trên tập số phức, phép cộng là tồn tại. Tuy nhiên:

```
1 Complex a;
2 Complex b;
3 Complex c = a + b;
```

Toán tử + ở dòng số 3 là không hợp lý vì kiểu `Complex` là kiểu do người dùng tự định nghĩa, C++ chưa có định nghĩa cho phép + cho kiểu này.

C++ cho phép định nghĩa lại nhiều toán tử có sẵn để chúng có thể chấp nhận các toán hạng là các đối tượng.

```
+ - * / = < > += -= *= /= << >>
< <= > >= == != <= >= ++ -- \% \& ^ ! |
~ &= ^= |= \&\& || \%= [] () , ->* ->
new delete new[] delete[]
```


ThS.
Trần Ngọc Bảo Duy

Định nghĩa lại toán tử (Operator overloading)

Cú pháp định nghĩa lại toán tử:

```
<rettype> operator<symbol>(<class_name>& rhs);
<rettype> operator<symbol>(const <class_name>& rhs);
```

trong đó:

- `<rettype>` là kiểu dữ liệu trả về sau khi thực hiện toán tử.
- `<symbol>` là toán tử cần định nghĩa lại.
- `<class_name>` là tên lớp cần định nghĩa lại toán tử.

ThS.
Trần Ngọc Bảo Duy

Định nghĩa lại toán tử (Operator overloading): Ví dụ

```
class Complex {
private:
    double real, img;

public:
    Complex() { real = img = 0.0; }
    Complex(double real, double img) {
        this->real = real;
        this->img = img;
    }
    // Other constructors and methods

    Complex& operator+(const Complex& rhs) {
        Complex res(this->real + rhs.real,
                    this->img + rhs.img);
        return res;
    }

    bool operator>(const Complex& rhs) {
        double mod1 = sqrt(real * real + img * img);
        double mod2 = sqrt(rhs.real * rhs.real,
                           rhs.img * rhs.img);
        return mod1 > mod2;
    }
};
```


ThS.
Trần Ngọc Bảo Duy

Toán tử gán

Nếu một thuộc tính thành viên được **cấp phát động**, thì:

- Cần phải định nghĩa lại toán tử gán
- Cần phải định nghĩa lại hàm khởi tạo copy.

ThS.
Trần Ngọc Bảo Duy

Toán tử gán

Nếu một thuộc tính thành viên được **cấp phát động**, thì:

- Cần phải định nghĩa lại toán tử gán
- Cần phải định nghĩa lại hàm khởi tạo copy.

Một số prototype phổ biến:

```
MyClass& operator=(const MyClass& rhs);
MyClass& operator=(MyClass& rhs);
const MyClass& operator=(const MyClass& rhs);
const MyClass& operator=(MyClass& rhs);
MyClass operator=(const MyClass& rhs);
MyClass operator=(MyClass& rhs);
```


ThS.
Trần Ngọc Bảo Duy

Vận dụng

Hãy thiết kế và hiện thức lớp `Complex` trong C++ để mô tả số phức $z = a + bi$ với các yêu cầu sau:

- Thuộc tính: phần thực và phần ảo.
- Các phương thức:
  1. Các hàm khởi tạo và hàm hủy phù hợp nếu có.
  2. Các phương thức **getter** và **setter** cho từng dữ liệu.
  3. Phương thức lấy module của số phức.
  4. Phương thức lấy argument của số phức.
  5. Phương thức lấy số phức liên hợp của số phức.
  6. Các phương thức cộng, trừ, nhân số phức được hiện thực ở dưới dạng toán tử $+, -, *$.