Bài 18: Đa hình compile time

Nguyễn Hoàng Anh

Function Overloading

- **Nạp chồng hàm** (Function Overloading) là việc định nghĩa **nhiều hàm cùng tên** nhưng **khác tham số** trong cùng một phạm vi.
- Trình biên dịch sẽ chọn hàm phù hợp dựa trên kiểu và số lượng đối số khi gọi hàm.

Function Overloading

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

void print(int a){ cout << "Integer: " << a << endl; }

void print(double b){ cout << "Double: " << b << endl; }

void print(string s){ cout << "String: " << s << endl; }

int main()
{
    print(5);
    print(3.14);
    print("Hello");
    return 0;
}
```

Function Overloading

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

// 1 method có thể có nhiều input parameter, return type khác nhau
class TinhToan{
    private:
        int a;
        int b;
    public:
        int tong(int a, int b){
            return a+b;
        }
        double tong(int a, int b, int c, double d){
            return (double)a+b+c+d;
        }
        double tong(int a, double b){
            return (double)a+b;
        }
};
```

Operator Overloading

- **Nạp chồng toán tử** (Operator Overloading) là việc định nghĩa lại cách hoạt động của các toán tử (+, -, =, ==, <<, >>,...) cho các kiểu dữ liệu do người dùng định nghĩa (**class/struct**).
- Cú pháp:

```
<return_type> operator symbol (parameter)
{
    // logic của toán tử
}
```

Operator Overloading

- Các toán tử có thể định nghĩa lại:

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

- Các toán tử không thể định nghĩa lại:
  - Toán tử . (chấm)
  - Toán tử phạm vi ::
  - Toán tử điều kiện ?:
  - Toán tử sizeof

Operator Overloading

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

class Complex
{
    private:
        double realPart;    // phần thực
        double imagPart;    // phần ảo

    public:
        Complex(double real = 0, double imag = 0): realPart(real), imagPart(imag){}

        // nạp chồng toán tử +
        Complex operator + (const Complex other) const
        {
            Complex result;
            result.realPart = Complex::realPart + other.realPart;
            result.imagPart = Complex::imagPart + other.imagPart;
            return result;
        }

        // nạp chồng toán tử so sánh bằng (==)
        bool operator == (const Complex other) const
```

This Pointer

- **this** là một **con trỏ ẩn** (ẩn danh) có sẵn trong mọi hàm thành viên (method) của class. Nó trỏ đến đối tượng hiện tại mà hàm đang được gọi trên.

This Pointer

Đặc điểm	Chi tiết
1️⃣ Chỉ xuất hiện trong hàm thành viên	**this** chỉ tồn tại trong hàm thành viên của class, không có trong các hàm **static** hoặc hàm global.
2️⃣ Trỏ đến đối tượng hiện tại	**this** là con trỏ trỏ đến đối tượng gọi hàm hiện tại.
3️⃣ Có kiểu là con trỏ đến class	Trong class **Person**, thì this có kiểu là **Person***
4️⃣ Là **constant pointer**	Không thể thay đổi giá trị của con trỏ (**Person const *this**)
5️⃣ Phân biệt biến thành viên và tham số cùng tên	**this→** giúp truy cập chính xác biến trong class.
6️⃣ Không dùng được trong **static function**	Vì static function không gắn với bất kỳ object nào ⇒ không có **this**.

This Pointer

```
class Student
{
    private:
        std::string name;

    public:
        void setName(std::string name)
        {
            this->name = name; // this giúp phân biệt rõ ràng
        }
};
```

Tham trị (Pass by Value)

- Tham trị là cách truyền một bản sao của biến vào hàm. Mọi thay đổi trong hàm không ảnh hưởng đến biến gốc bên ngoài.

```
void modify(int x){ x = x + 10;}

int main()
{
    int a = 5;
    modify(a);
    cout << a << endl; // Output: 5
}
```

Tham chiếu (Pass by Reference)

- Tham chiếu là cách tạo ra **một tên khác** để truy cập cùng một vùng nhớ của một biến đã tồn tại.

```
type& referenceName = variable;
```

  - **type**: kiểu dữ liệu
  - **&**: ký hiệu cho tham chiếu (khác với con trỏ)
  - **referenceName**: tên tham chiếu
  - **variable**: biến đã khai báo

Tham chiếu (Pass by Reference)

```
int a = 10;
int& ref = a; // ref là một tham chiếu đến a
ref = 20;
std::cout << a << std::endl; // Kết quả: 20
```

```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```

Operator Overloading

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

class Complex
{
    private:
        double realPart;    // phần thực
        double imagPart;    // phần ảo

    public:
        Complex(double &real = 0, double &imag = 0): realPart(real), imagPart(imag){}

        // nạp chồng toán tử +
        Complex operator + (const Complex &other) const
        {
            Complex result;
            result.realPart = this->realPart + other.realPart;
            result.imagPart = this->imagPart + other.imagPart;
            return result;
        }

        // nạp chồng toán tử so sánh bằng (==)
        bool operator == (const Complex &other) const
```

Tổng kết

- **Function Overloading**
- **Operator Overloading**
- **Tham Trị - Tham Chiếu**
- **Con trỏ This**