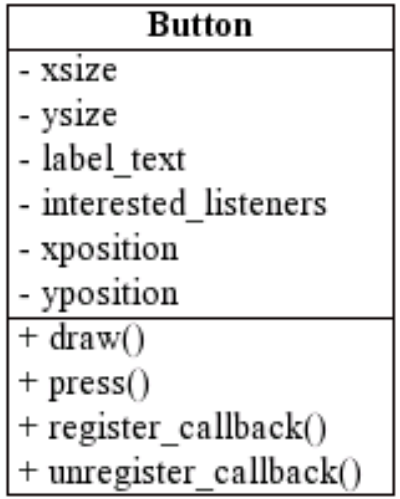
Lecture 4

TOC

面向对象编程

面向对象编程（Object-Oriented Programming，OOP）是一种程序设计范式，它以对象作为基本单元，将数据和行为封装在对象中，以提高代码的重用性、灵活性和可维护性。



C++中的类

C++ 中，我们用 `class`，`struct` 关键字来定义类，类的成员变量和成员函数可以分别用 `public`，`protected`，`private` 关键字来修饰限定访问权限。

```
class Box {
public:
    double length;
    double breadth;
    double height;
    double getVolume(void);
protected:
    double width;
private:
    double depth;
};
```

- public: 可以被类的外部访问
- protected: 可以被类的派生类访问
- private: 只能被类的成员函数访问

> struct 和 class 的区别在于默认的访问权限不同，struct 默认为 public，class 默认为 private。

类成员的修饰符

- `static`：静态成员变量和静态成员函数属于类本身，而不是类的对象。静态成员变量在类的所有对象中共享，静态成员函数可以直接访问静态成员变量。
- `const`：常量成员变量，只能在进入构造函数体前初始化。
- `mutable`：可变成员变量，即使在常量成员函数中也可以修改。

非static成员变量可以在类的声明中初始化成员变量，这样无需在构造函数中初始化。static成员变量需要在类的定义外初始化(例外：const static int/enum 成员变量)。

静态成员函数只能访问类的静态成员变量:

```
class Foo{
  static int bar;
  double baz;

  static void foo(){
    bar += 2; // Okay
    baz--; // Error
  }
}
```

类的成员函数

类的非静态成员函数可以访问类的任何成员变量，也可以调用其他成员函数。

```
class Box {
public:
    double length=1.1;
    double breadth{2.2};
    double height;

    double getVolume(void);
};

double Box::getVolume(void) {
    return length * breadth * height;
}

int main() {
    Box box1;
    box1.length = 5.0;
    box1.breadth = 6.0;
    box1.height = 7.0;
    double volume = box1.getVolume();
    return 0;
}
```

构造函数

构造函数是一种特殊的成员函数，用于初始化对象的成员变量。它们在对象创建时自动调用。

```
class Box {
public:
    double length;
    double breadth;
    double height;

    Box(double l, double b, double h) { // constructor
        length = l;
        breadth = b;
        height = h;
    }
    double getVolume(void);
};
```

cpp中编译器可以生成的构造函数有：

- 默认构造函数 `Box() {}`
- 拷贝/复制构造函数 `Box(const Box &obj) {}`
- 移动构造函数 `Box(Box &&obj) noexcept {}`

C++ 中，如果没有定义构造函数，编译器会自动生成一个默认构造函数，它会调用类类型成员的默认构造函数，但不初始化内置类型的值。同时，默认生成的复制构造函数和赋值运算符会对对象的成员变量进行逐个复制。

如果定义了构造函数，编译器不会生成默认构造函数。

如果定义了复制构造函数，编译器不会生成默认的移动构造函数。反之亦然。

如果要显示定义默认的(复制)构造函数，可以使用 `= default` 关键字。

```
class Box {
public:
    double length;
    double breadth;
    double height;

    Box();
    Box(const Box &obj);
    Box(Box &&obj) noexcept;
};

Box::Box() = default;
Box::Box(const Box &obj) = default;
Box::Box(Box &&obj) noexcept = default;
```

带单个参数的构造函数

```
struct Box {
  int width;
  explicit Box(int w) : width(w) {}
};
```

`explicit` 关键字可以防止隐式转换，只能用于带单个参数的构造函数。

```
Box b = 10; // Error
```

成员初始化列表(member initializer list)

成员初始化列表可以在构造函数体之前初始化成员变量，这样可以避免在构造函数体中对成员变量进行赋值。对于 `const` 成员变量和引用类型成员变量，必须使用成员初始化列表进行初始化。

```
struct Foo{
  int b;
  const int a;

  Foo(int x, int y): a(x), b(y) {}
}
```

成员初始化顺序

成员初始化列表中的初始化顺序与成员变量在类中的声明顺序一致，而不是成员初始化列表中的顺序。

```
b -> a
```

复制构造函数和复制运算符

复制构造函数用于创建一个新对象，它的参数是一个对象的引用。

```
struct A{
private:
  int a;

public:
  A(const A &obj): a(obj.a) {}
  A& operator=(const A &obj) {
    a = obj.a;
    return *this;
  }
}
```

同一种类的不同实例之间可以在成员函数中互相访问对方成员，无视访问权限。

移动构造函数和移动赋值运算符

移动构造函数用于直接复用原对象的资源创建一个新对象，它的参数是一个右值引用。

```
struct A{
  int *a;
  A(A &&obj): a(obj.a) {
    obj.a = nullptr;
  }
  A& operator=(A &&obj) {
    a = obj.a;
    obj.a = nullptr;
    return *this;
  }
}
```

移动构造会转移资源句柄的所有权，而不是复制资源。

析构函数

析构函数是一种特殊的成员函数，用于释放对象的资源。它在对象销毁时自动调用。

```
class Box {
public:
    double length;
    double breadth;
    double height;
    int *useless;

    Box(double l, double b, double h) : length(l), breadth(b), height(h) {
        useless = new int[100];
    }

    ~Box() {
        // 释放资源
    }
};
```

析构函数不接受参数，不允许重载，不允许有返回值。析构函数不应抛出异常，否则可能会导致程序终止。

杂

列表初始化(List-initialization)

C++11 引入了列表初始化语法，可以用花括号 `{}` 初始化对象。

```
int arr[3] = {1, 2, 3};
int a = {1}; // Okay

struct Foo{
  int a;
  float b;
};

Foo b{1, 2.3f};
```

Reference

const 成员函数

const 成员函数不会修改对象的成员变量，可以在 const 对象上调用。

```
class Box {
public:
    double length;
    double breadth;
    double height;

    double getVolume(void) const {
        return length * breadth * height;
    }

    void setLength(double l) {
        length = l;
    }
};

const Box a;
double volume = a.getVolume();
a.setLength(10.0); // Error
```

浅拷贝和深拷贝

- 浅拷贝：只复制对象的成员变量，不复制句柄的资源。
- 深拷贝：复制对象的成员变量，同时复制句柄的资源。

```
int *a = new int[100];

int *b = a; // 浅拷贝
int *c = new int[100];
std::memcpy(c, a, 100 * sizeof(int)); // 深拷贝
```

运算符重载

运算符重载是一种特殊的成员函数，用于重定义运算符的行为。

```
class Box {
public:
    double length;
    double breadth;
    double height;

    Box operator+(const Box& b) {
        Box box;
        box.length = this->length + b.length;
        box.breadth = this->breadth + b.breadth;
        box.height = this->height + b.height;
        return box;
    }
};

Box a, b;
Box c = a + b;
```

C++ 中有一些运算符不能被重载，如 `.` `::` `?:` `sizeof` `typeid` `.*` `->*` 等

Reference

this 指针

`this` 指针是一个隐式参数，指向当前对象的地址。它可以用于访问对象的成员变量和成员函数。常用来区分成员变量和局部变量/函数参数。

```
class Foo{
  int a;
public:
  Foo(int a) {
    this->a = a;
  }

  Foo& operator=(const Foo &obj) {
    this->a = obj.a;
    return *this; // 返回当前对象
  }
};
```

友元函数和友元类

友元函数和友元类可以访问类的私有成员。

```
class Matrix {
private:
    int data[3][3];
public:
friend std::ostream& operator<<(std::ostream& os, const Matrix& m);
};

std::ostream& operator<<(std::ostream& os, const Matrix& m) {
    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            os << m.data[i][j] << " ";
        }
        os << std::endl;
    }
    return os;
}
```

Q&A