面向对象（Object-oriented programming，OOP）

概念

面向对象（Object-oriented programming，OOP）的概念，是一种抽象概念，意思是在程序设计中可以将你的问题抽象成一个对象。每个对象都是一个独立的实体，它具有自己的属性与方法，可以和其他对象进行交互与通信。

以游戏人物为例子

我们可以将每个游戏人物都看作一个对象，每个游戏人物都具有自己的属性和方法。 以下是一个对象中可能具有的属性和方法：

属性：

- 名字
- 生命值
- 能量值
- 等级
- 经验值

方法：

- 攻击
- 防御
- 升级

Code（OOP）

```
class GameCharacter {
 private:
   // 属性
   std::string name;
   int health;
   int energy;
   int level;
   int experience;

 public:
   // 构造函数
   GameCharacter(std::string n, int h, int e, int l, int exp) :
       name(n), health(h), energy(e), level(l), experience(exp) {}

   // getter and setter 方法
   std::string getName() const { return name; }
   void setName(std::string n) { name = n; }
   int getHealth() const { return health; }
   void setHealth(int h) { health = h; }
   int getEnergy() const { return energy; }
   void setEnergy(int e) { energy = e; }
```

```
    int getLevel() const { return level; }
    void setLevel(int l) { level = l; }
    int getExperience() const { return experience; }
    void setExperience(int exp) { experience = exp; }

    void attack(GameCharacter& enemy);
    void levelUp();
};
```

访问权限

在下一步之前需要提一下，C++中，对象的访问权限有三种，分别是public、private和protected。

概念

- public表示公开的，意味着可以在任何地方访问，包括类内部和类外部。
- private表示私有的，意味着只能在类的内部访问，类外部无法访问。
- protected表示受保护的，意味着只能在类的内部以及其派生类中访问。

访问权限表格

根据访问权限总结出不同的访问类型，如下所示：

访问	public	protected	private
同一个类	yes	yes	yes
派生类	yes	yes	no
外部的类	yes	no	no

默认情况下，C++ 中的成员变量和成员函数是私有的。因此，如果要让类中的成员变量或成员函数在类外部可以访问，必须将其声明为 public。 使用不同的访问权限可以帮助我们控制数据的访问范围，从而提高代码的安全性和可维护性。例如，私有成员可以防止类外部的代码直接修改类中的数据，保护数据的完整性。另外，使用继承时，可以通过不同的访问权限来控制子类是否能够访问父类中的成员。

OOP中的继承（Inheritance）

概念

在OOP中，继承是一种将已有类的属性和方法引入到新的类中的机制。它是面向对象编程中的重要概念之一，也是实现代码复用和提高代码可维护性的关键手段之一。

基类 & 派生类

通过继承，一个类可以派生出一个新的类，新类会自动拥有原类的属性和方法。在C++中，继承 = 派生类名+基类的名称和访问权限。基类是被继承的类，而派生类是继承基类而得到的新类。形式如下:

```
class derived-class: access-specifier base-class
```

其中，访问修饰符 access-specifier包括（public、protected、private ），base-class 是之前定义过的某个类的名称。如果未使用访问修饰符 access-specifier，则默认为 private。以游戏人物为例子:

Code（Inheritance）

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

// 基类，游戏人物类
class GameCharacter {
public:
    GameCharacter(string n, int hp, int atk) : name(n), healthPoints(hp),
attack(atk) {}
    virtual void attackEnemy() = 0; // 纯虚函数
protected:
    string name; // 游戏人物名称
    int healthPoints; // 生命值
    int attack; // 攻击力
};

// 派生类，战士类
class Warrior : public GameCharacter {
public:
    Warrior(string n, int hp, int atk, int r) : GameCharacter(n, hp, atk),
rage(r) {}
    void attackEnemy() override {
        cout << name << "使用剑攻击敌人，造成" << attack << "点伤害" << endl;
    }
    void useSkill() {
        if (rage >= 100) {
            cout << name << "使用愤怒打击，造成" << attack * 2 << "点伤害" <<
endl;
            rage -= 100;
        } else {
            cout << name << "愤怒值不足，无法使用技能" << endl;
        }
    }
private:
    int rage; // 愤怒值
};

int main() {
    GameCharacter* gc1 = new Warrior("Arthur", 100, 20, 0);
    gc1.attackEnemy();
    Warrior* w1 = dynamic_cast<Warrior*>(gc1);
    w1.useSkill();
    delete gc1;
    return 0;
}
```

以上代码运行结果：

```
Arthur使用剑攻击敌人，造成20点伤害
Arthur愤怒值不足，无法使用技能
```

继承类型

C++中的继承方式包括公有继承、私有继承和保护继承。不同的继承方式会影响到派生类对基类成员的访问权限。具体而言：

- 公有继承（public inheritance）：派生类可以访问基类中的公有成员和保护成员，但不能访问基类中的私有成员。
- 私有继承（private inheritance）：派生类可以访问基类中的公有成员、保护成员和私有成员，但这些成员在派生类中都变成了私有成员。
- 保护继承（protected inheritance）：派生类可以访问基类中的公有成员和保护成员，但这些成员在派生类中都变成了保护成员，不能被外部访问。

我们几乎不使用 protected 或 private 继承，通常使用 public 继承。通过继承，派生类可以获得基类的所有属性和方法，同时也可以通过重写基类的虚函数来实现多态性。此外，派生类还可以添加自己的新属性和方法，从而实现对基类的扩展和改进。继承是OOP中非常重要的概念，它可以有效地提高代码复用性和可维护性，是面向对象编程的核心之一。

多继承（Multiple Inheritance）

允许一个派生类继承自多个基类。以游戏角色为例，我们可以定义多个基类来描述不同方面的角色属性，然后让一个派生类继承这些基类，从而组合出一个完整的游戏角色。

Code（Multiple Inheritance）

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

// 角色的基类
class Character {
public:
    virtual void printDescription() {
        cout << "I am a character." << endl;
    }
};

// 攻击属性的基类
class Attack {
public:
    virtual void attack() {
        cout << "I am attacking." << endl;
    }
};

// 魔法属性的基类
```

```
class Magic {
public:
    virtual void castSpell() {
        cout << "I am casting a spell." << endl;
    }
};

// 战士角色类，同时继承自Character和Attack两个基类
class Warrior : public Character, public Attack {
public:
    void printDescription() {
        cout << "I am a warrior." << endl;
    }
};

// 法师角色类，同时继承自Character和Magic两个基类
class Wizard : public Character, public Magic {
public:
    void printDescription() {
        cout << "I am a wizard." << endl;
    }
};

// 剑法大师角色类，同时继承自Warrior和Magic两个角色类
class SwordMaster : public Warrior, public Magic {
public:
    void printDescription() {
        cout << "I am a sword master." << endl;
    }
};

int main() {
    Warrior warrior;
    warrior.printDescription();
    warrior.attack();

    Wizard wizard;
    wizard.printDescription();
    wizard.castSpell();

    SwordMaster swordMaster;
    swordMaster.printDescription();
    swordMaster.attack();
    swordMaster.castSpell();

    return 0;
}
```

以上代码运行结果：

```
I am a warrior.
I am attacking.
```

```
I am a wizard.
I am casting a spell.
I am a sword master.
I am attacking.
I am casting a spell.
```