**什么是多态？**

多态是面向对象编程中的一个重要概念。它指的**是同一个实体（如函数或运算符）在不同情况下表现出不同的行为。**例如，在 C++ 中，`+` 运算符用于执行两个特定的函数。当它与数字（整数和浮点数）一起使用时，它执行加法。当我们将 `+` 运算符与字符串一起使用时，它执行字符串连接。

C++ 中的多态可以通过以下几种方式实现：

- **函数重载：**可以使用相同名称但具有不同参数（类型或参数数量）的两个函数。根据参数的数量/类型，调用不同的函数。

- **运算符重载：**可以重载运算符，只要我们对用户定义的类型（如对象或结构）进行操作。我们不能对基本类型（如 int、double 等）使用运算符重载。

- **函数覆盖：**派生类中的虚函数必须与基类中被覆盖的虚函数具有相同的函数名、参数个数、参数类型、参数顺序和返回类型。

- **虚函数：**虚成员是可以在派生类中重新定义的成员函数，同时保留其通过引用调用的属性。

多态允许我们**编写更通用、更可重用的代码，并且能够更容易地扩展和维护**。例如，我们可以编写一个处理多种不同类型对象的函数，而不需要为每种类型都编写一个单独的函数。这样，当我们添加新类型时，只需要确保新类型满足多态性要求，而无需修改处理函数。

18-19年：

**1、c++不允许实例化抽象类的对象。因此抽象类中不需要设计构造函数。**

**2、在异常处理时，异常抛出后，会导致所有程序被终止运行**

**3、通过常量对象只能直接调用常成员数和静态成员函数。**

**4、如果类B和类A中都只由成员函数组成，则sizeof(B)一定等于sizeof(A).**

**5、若有类A.类B,且B b;A a=b,则类B一定是通过public继承方式从类A派生的。**

**6、对已知函数void func(int a=0,int b=0,int c=0):用func(2)进行调用，参数匹配时，各参数的实参值分别为a=0,b=0,c=2.**

**7、对象可以隐式或显式地使用this指针来引用自身数据成员和函数成员，this指针是对象自身的一部分，this指针占用的内存大小可以反映在对对象进行sizeof的运算结果中，：**

**8、根据里氏替换原则、需要基类对象的任何地方都可以使用其派生类的对象来替代、程序的行为没有变化。**

**9、虚函数是派生类对基类的覆盖而不是重载，其判断依据是参数的个数、类型、顺序、返回类型必须相同。**

**10、静态数据成员在main函数之前构造、生命周期等同于全局对象、所以不需要其他声明情况下就可以在程序的任何地方访问到静态数据成员**

1. 错误。C++ 不允许实例化抽象类的对象，但是抽象类仍然可以定义构造函数。因为抽象类的派生类的构造函数可能需要调用基类的构造函数来初始化基类部分。

2. 错误。当异常被抛出时，程序并不会立即终止运行。相反，程序会在调用栈中查找能够处理该异常的异常处理程序。如果找到了异常处理程序，程序将继续执行；否则，程序将终止运行。

3. 正确。常量对象只能调用常量成员函数和静态成员函数，因为常量成员函数不会修改对象的状态，而静态成员函数不依赖于对象的状态。

4. 错误。即使类 `B` 和类 `A` 中都只由成员函数组成，`sizeof(B)` 和 `sizeof(A)` 的值也不一定相等。这是因为，如果类中有虚函数，那么对象中会包含一个指向虚函数表的指针。如果类 `B` 和类 `A` 的虚函数表不同，那么 `sizeof(B)` 和 `sizeof(A)` 的值可能不同。此外，如果类使用了多重继承或虚继承，那么对象的大小也可能受到影响。

5. 错误。这种情况下，类 `B` 可能是通过公有继承、保护继承或私有继承方式从类 `A` 派生的。也有可能类 `B` 和类 `A` 没有任何继承关系，但是定义了从类 `B` 到类 `A` 的类型转换运算符或者类 `A` 有一个接受类 `B` 对象作为参数的构造函数。

6. 错误。对于函数 `void func(int a=0,int b=0,int c=0)`，使用 `func(2)` 进行调用时，参数匹配的结果是：`a=2, b=0, c=0`。

7. 错误。对象可以隐式或显式地使用 `this` 指针来引用自身数据成员和函数成员，但是 `this` 指针并不是对象自身的一部分，它不占用内存空间。因此，对对象进行 `sizeof` 运算时，并不会包含 `this` 指针占用的内存大小。

8. 正确。根据里氏替换原则，需要基类对象的任何地方都可以使用其派生类的对象来替代，程序的行为没有变化。

9. 正确。虚函数是派生类对基类的覆盖而不是重载，其判断依据是参数的个数、类型、顺序、返回类型必须相同。

10. 错误。静态数据成员在 main 函数之前构造，生命周期等同于全局对象。所以，在程序的任何地方都可以访问到静态数据成员（只要它在作用域内）。但是需要注意访问权限（如公有、保护或私有）。

二、

1. **请说出static在c++语言中的全部作用？（3分）**

在 C++ 语言中，`static` 关键字有以下几种作用：

1. 定义静态局部变量：在函数内部使用 `static` 关键字定义静态局部变量，它的生命周期与程序的执行时间相同，但只能在定义它的函数内部访问。静态局部变量在程序执行到它的定义处时被初始化一次，之后每次调用函数时都保持它的值不变。

2. 定义静态全局变量：在全局作用域内使用 `static` 关键字定义静态全局变量，它的生命周期与程序的执行时间相同，但只能在定义它的文件内部访问。静态全局变量在 `main` 函数之前被初始化。

3. 定义静态成员变量：在类定义中使用 `static` 关键字定义静态成员变量，它是类的所有对象共享的一个变量。静态成员变量在 `main` 函数之前被初始化。

4. 定义静态成员函数：在类定义中使用 `static` 关键字定义静态成员函数，它不依赖于类的对象，可以通过类名直接调用。静态成员函数不能访问非静态数据成员和非静态成员函数。

5. 控制函数和变量的链接性：在全局作用域内使用 `static` 关键字定义函数或变量时，可以控制它们的链接性。具有静态链接性的函数或变量只能在定义它们的文件内部访问。

1. **请解释c++的友元关系的概念？并说明友元关系的副作用？(2分)**

在 C++ 中，友元关系是一种允许非成员函数或其他类访问类的私有和保护成员的机制。可以通过在类定义中使用 `friend` 关键字来声明友元函数或友元类。

友元关系破坏了类的封装性，因为它允许外部函数或类直接访问类的私有和保护成员。这意味着，如果类的实现发生了变化，那么与之相关的友元函数或友元类也可能需要进行相应的修改。此外，过多地使用友元关系会使代码变得难以理解和维护。

因此，在设计类时，应谨慎使用友元关系，并尽量通过公有成员函数来提供对类的操作。只有在确实需要访问类的内部实现时，才应考虑使用友元关系。此外，应尽量减少与类相关的友元函数或友元类的数量，以降低代码的复杂性。

3、**class A**

**{**

**public:**

**static int s\_a;**

**int const c\_a;**

**int d\_a;**

**};**

**class B :public A**

**{**

**public:**

**int d\_b;**

**int& r\_b;**

**};**

**class C**

**{**

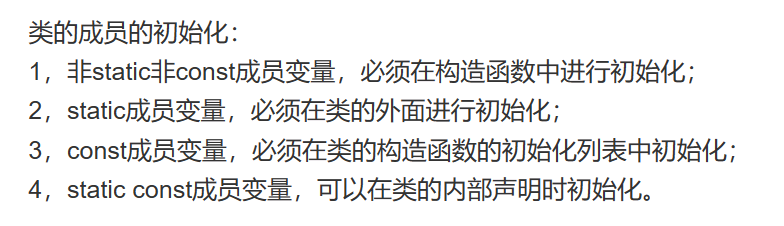
**public:**

**B b\_c;**

**int d\_c;**

**};**

**请为以上3个类添加有参构造函数，初始化各类中的数据成员**



class A

{

public:

static int s\_a;

int const c\_a;

int d\_a;

A(int sa, int ca, int da) : c\_a(ca), d\_a(da)

{

s\_a = sa;

}

};

int A::s\_a = 0;

class B :public A

{

public:

int d\_b;

int& r\_b;

B(int sa, int ca, int da, int db, int& rb) : A(sa, ca, da), d\_b(db), r\_b(rb)

{

}

};

class C

{

public:

B b\_c;

int d\_c;

C(int sa, int ca, int da, int db, int& rb, int dc) : b\_c(sa, ca, da, db, rb), d\_c(dc)

{

}

};

**4、请说出静态绑定和动态绑定的不同之处，并解释虚函数的工作机理。**

静态绑定和动态绑定是两种不同的函数调用机制。

静态绑定是在编译时确定被调用函数的过程。编译器根据函数的名称、参数类型和数量来确定被调用的函数。静态绑定通常用于非虚成员函数和非成员函数的调用。

动态绑定是在运行时确定被调用函数的过程。它通常用于虚成员函数的调用。当通过基类指针或引用调用虚成员函数时，程序会根据指针或引用所指向的对象的实际类型来确定调用哪个版本的虚成员函数。这种机制被称为动态绑定或运行时多态。

虚函数的工作机制依赖于虚函数表和虚表指针。每个包含虚函数的类都有一个虚函数表，其中存储了该类的虚函数地址。每个包含虚函数的类的对象都有一个虚表指针，它指向该类的虚函数表。当通过基类指针或引用调用虚成员函数时，程序会根据虚表指针来查找虚函数表，并根据虚函数表中存储的地址来确定调用哪个版本的虚成员函数。

1. 存在问题：类A直接依赖类B，假如要将类A改为依赖类C，则必须通过修改类A的代码来实现。请给出一个无需修改类A代码的解决方案。

6、

补充设计一个小汽车类Car，并能支持完成以下代码调用的相应功能：

#include<iostream>

using namespace std;

class Car

{

char\* pName;/\*发动机名字\*/

int iId;/\*车架号，自动产生具有唯一性\*/

int iI;/\*行车里程数，初始化为0\*/

public:

/\*需要补充的代码\*/

};

int main()

{

Car c1("捷达发动机");

Car c2("奥迪发动机");

Car c3(c1);

c3 = c1;

c2 = c2 + 3;/\*又跑了3公里\*/

++c2;/\*又跑了1公里\*/

cout << c2 << endl;/\*输出小汽车c2的信息\*/

}

#include<iostream>

#include<cstring>

using namespace std;

class Car

{

char\* pName; /\*发动机名字\*/

int iId; /\*车架号，自动产生具有唯一性\*/

int iI; /\*行车里程数，初始化为0\*/

static int idCount; //静态成员变量，用于生成唯一的车架号

public:

Car(const char\* name) : iI(0) {

pName = new char[strlen(name) + 1];

strcpy(pName, name);

iId = ++idCount;

}

Car(const Car& car) : iI(car.iI), iId(++idCount) {

pName = new char[strlen(car.pName) + 1];

strcpy(pName, car.pName);

}

~Car() {

delete[] pName;

}

Car& operator=(const Car& car) {

if (this != &car) {

delete[] pName;

pName = new char[strlen(car.pName) + 1];

strcpy(pName, car.pName);

iI = car.iI;

}

return \*this;

}

Car& operator+(int x) {

iI += x;

return \*this;

}

Car& operator++() {

++iI;

return \*this;

}

friend ostream& operator<<(ostream& os, const Car& car) {

os << "发动机名字: " << car.pName << endl;

os << "车架号: " << car.iId << endl;

os << "行车里程数: " << car.iI << endl;

return os;

}

};

int Car::idCount = 0;

int main()

{

Car c1("捷达发动机");

Car c2("奥迪发动机");

Car c3(c1);

c3 = c1;

c2 = c2 + 3; /\*又跑了3公里\*/

++c2; /\*又跑了1公里\*/

cout << c2 << endl; /\*输出小汽车c2的信息\*/

}

七综合编程题。(12分)

各大通信运营商提供的手机运营业务主要包括通话业务,流量业务,短信业务等等,不同的业务具有不同资费方案。例如不同运营商具有不同通话业务基本资费标准,根据用户的不同需求还提供有月租套餐、包年套餐等,月租套餐一般包括基本的月基本费用〔常规包含一定的本地或异地通话时间和数据流量),超出部分主叫费用/分钟和费用/流量等等。(具体业务资费规则可参考同学自己的资费情况）。随着手机业务需求的不断发展，允许新的运营商加入竞争，并可能推出新的业务类型，新的资费方案。特别是5G即将全面上线，对于流量业务资费响颇大。

请你设计并实现手机业务管理系统,请以收费为关注点设计类的体系和主要接口、并给出调用程序（比如main函数)模拟张同学在2019年1月到2019年6月期间手机业务的费用情况.

#include <iostream>

#include <vector>

using namespace std;

class Service {

public:

virtual double getCharge() const = 0;

virtual ~Service() {}

};

class CallService : public Service {

public:

CallService(double rate, int minutes) : rate(rate), minutes(minutes) {}

virtual double getCharge() const {

return rate \* minutes;

}

private:

double rate;

int minutes;

};

class DataService : public Service {

public:

DataService(double rate, int bytes) : rate(rate), bytes(bytes) {}

virtual double getCharge() const {

return rate \* bytes;

}

private:

double rate;

int bytes;

};

class SMSService : public Service {

public:

SMSService(double rate, int count) : rate(rate), count(count) {}

virtual double getCharge() const {

return rate \* count;

}

private:

double rate;

int count;

};

class Customer {

public:

void addService(Service\* service) {

services.push\_back(service);

}

double getTotalCharge() const {

double total = 0.0;

for (vector<Service\*>::const\_iterator it = services.begin(); it != services.end(); ++it) {

total += (\*it)->getCharge();

}

return total;

}

private:

vector<Service\*> services;

};

int main() {

Customer customer;

// 通话业务

CallService call1(0.2, 100);

CallService call2(0.2, 200);

customer.addService(&call1);

customer.addService(&call2);

// 流量业务

DataService data1(0.1, 1024);

DataService data2(0.1, 2048);

customer.addService(&data1);

customer.addService(&data2);

// 短信业务

SMSService sms1(0.1, 50);

SMSService sms2(0.1, 100);

customer.addService(&sms1);

customer.addService(&sms2);

cout << "Total charge: " << customer.getTotalCharge() << endl;

return 0;

}

13-14年：

1. 类A的常成员函数中，this指针的类型是const A\* .
2. 友元函数可以是全局函数，也可以是其它类的成员函数.
3. 有函数void f(A)，语句块B b; f(b); 能编译通过并执行，则类B一定是类A的公有派生类
4. 重载的析构函数应与重载的构造函数一一对应.
5. 函数的形参类型是const A& ，则实参类型必须是const A.
6. 通过在自定义类中定义运算符重载函数，可以改变或扩充运算符的语义，但不能改变运算符的结合性和优先级等固有性质.
7. 可以定义嵌套类，但不能定义嵌套的名字空间.
8. 在一个类首次实例化前，类中的静态数据成员一定会先被初始化.
9. 静态成员函数不能直接访问本类的非静态成员，非静态成员函数也不能直接访问本类的静态成员.
10. 用const修饰的成员函数只能通过常对象调用.

二、

1、既然每个类的成员函数都是确定的，为什么还要通过建立虚表实现多态呢  
 虚表是用来实现多态的。多态是面向对象编程中的一个重要概念，它允许在运行时根据对象的实际类型来调用相应的方法。虚表是一种用于支持多态的技术，它包含了一个类的虚函数指针，这些指针指向该类的虚函数实现。当我们使用基类指针或引用调用一个虚函数时，编译器会在运行时通过查找虚表来确定应该调用哪个函数。

虽然每个类的成员函数都是确定的，但是当我们使用基类指针或引用来调用一个虚函数时，编译器无法在编译时确定应该调用哪个函数，因为它只知道指针或引用的静态类型，而不知道它所指向的对象的实际类型。因此，编译器需要在运行时通过查找虚表来确定应该调用哪个函数。这就是为什么需要通过建立虚表来实现多态的原因。

1. 信息时代人们高度依赖移动即时通信工具，通信工具可以分解为移动通信功能和通信软件的叠加功能。常见的移动通信功能实现方式有:3G、4G、WiFi、蓝牙、微波、Z1Gbee等，负责完成无线信号的传递;通信软件负责信号的呈现并完成人机交互模式，常见的有:QQ、微信、飞信、泡泡、阿里旺旺等;随着技术的发展新的移动通信方式和通信软件也会发生新的变化，给出通信工具系统实现所需类的设计，需要满足可复用、可扩展、可维护的特点(8分)

五、题目四中所把到的所有通信软件一般都有群的功能，群可以模拟生活圈、工作圈、同学圈、生意圈等相对固定组织，也可以模拟讨论组、读书会、畅读吧、学习班等临时性的组织，这些组织可以相互交叉、相互叠加，即每个组织下面都可以有若干个其他的组织。这些组织的本质都是群，其核心功能是群成员的信息共享。给出以上所描述的群的设计，实现群的信息共享的基本功能，体现群内可以任意个数、任意深度组合不同性质子群的特点。(7分)

四、  
为了设计一个可复用、可扩展、可维护的通信工具系统，我们可以使用面向对象的设计方法。在这种方法中，我们可以定义几个类来表示系统中的不同组件，例如移动通信功能和通信软件。

首先，我们可以定义一个抽象基类 `MobileCommunication` 来表示移动通信功能。这个类可以定义一些虚函数来表示移动通信功能的通用接口，例如发送和接收数据。然后，我们可以定义一些派生类来表示不同的移动通信实现方式，例如 `3G`、`4G`、`WiFi`、`Bluetooth`、`Microwave` 和 `Zigbee` 等。这些派生类可以重写基类中的虚函数，以实现特定的移动通信功能。

其次，我们可以定义一个抽象基类 `CommunicationSoftware` 来表示通信软件。这个类可以定义一些虚函数来表示通信软件的通用接口，例如登录、发送消息和接收消息等。然后，我们可以定义一些派生类来表示不同的通信软件，例如 `QQ`、`WeChat`、`Fetion`、`PaoPao` 和 `AliWangWang` 等。这些派生类可以重写基类中的虚函数，以实现特定的通信软件功能。

最后，我们可以定义一个类 `CommunicationTool` 来表示通信工具。这个类可以包含一个指向 `MobileCommunication` 类型对象的指针和一个指向 `CommunicationSoftware` 类型对象的指针。这样，我们就可以在运行时根据需要选择不同的移动通信实现方式和通信软件。

这种设计方法具有很好的可复用性、可扩展性和可维护性。当需要添加新的移动通信实现方式或通信软件时，只需要定义一个新的派生类即可。而且，由于使用了面向对象的设计方法，代码结构清晰，便于管理和维护。

下面是一个简单的示例代码，展示了如何使用面向对象的设计方法来实现上述设计：

```c++

// 定义一个抽象基类 MobileCommunication 来表示移动通信功能

class MobileCommunication {

public:

virtual ~MobileCommunication() {}

virtual void send(const string& data) = 0;

virtual string receive() = 0;

};

// 定义一个派生类 WiFi 来表示 WiFi 移动通信实现方式

class WiFi : public MobileCommunication {

public:

void send(const string& data) override {

// 实现 WiFi 发送数据的逻辑

}

string receive() override {

// 实现 WiFi 接收数据的逻辑

return "";

}

};

// 定义一个抽象基类 CommunicationSoftware 来表示通信软件

class CommunicationSoftware {

public:

virtual ~CommunicationSoftware() {}

virtual void login(const string& username, const string& password) = 0;

virtual void sendMessage(const string& message) = 0;

virtual string receiveMessage() = 0;

};

// 定义一个派生类 QQ 来表示 QQ 通信软件

class QQ : public CommunicationSoftware {

public:

void login(const string& username, const string& password) override {

// 实现 QQ 登录的逻辑

}

void sendMessage(const string& message) override {

// 实现 QQ 发送消息的逻辑

}

string receiveMessage() override {

// 实现 QQ 接收消息的逻辑

return "";

}

};

// 定义一个类 CommunicationTool 来表示通信工具

class CommunicationTool {

public:

CommunicationTool(MobileCommunication\* mc, CommunicationSoftware\* cs)

: mc(mc), cs(cs) {}

void login(const string& username, const string& password) {

cs->login(username, password);

}

void sendMessage(const string& message) {

cs->sendMessage(message);

}

string receiveMessage() {

return cs->receiveMessage();

}

private:

MobileCommunication\* mc; // 移动通信功能

CommunicationSoftware\* cs; // 通信软件

};

```

五、

为了设计一个支持群功能的通信软件，我们可以定义一个类 Group 来表示群。这个类可以包含一些数据成员来表示群的属性，例如群名称、群描述、群成员列表等。此外，这个类还可以包含一些成员函数来实现群的核心功能，例如添加成员、删除成员、发送消息和接收消息等。

由于每个群下面都可以有若干个其他的群，因此我们可以在 Group 类中定义一个数据成员来表示子群列表。这个数据成员可以是一个 vector<Group\*> 类型的变量，用于存储指向子群对象的指针。这样，我们就可以在 Group 类中定义一些成员函数来管理子群，例如添加子群、删除子群和获取子群列表等。

为了实现群成员的信息共享功能，我们可以在 Group 类中定义一个成员函数 sendMessage 来发送消息。当一个群成员调用这个函数时，它会将消息发送给所有其他的群成员。此外，我们还可以定义一个成员函数 receiveMessage 来接收消息。当一个群成员调用这个函数时，它会从消息队列中获取一条消息并返回。

下面是一个简单的示例代码，展示了如何使用面向对象的设计方法来实现上述设计：

#include <iostream>

#include <vector>

#include <string>

using namespace std;

// 定义一个类 Group 来表示群

class Group {

public:

// 构造函数

Group(const string& name, const string& description)

: name(name), description(description) {}

// 添加成员

void addMember(const string& member) {

members.push\_back(member);

}

// 删除成员

void removeMember(const string& member) {

members.erase(remove(members.begin(), members.end(), member), members.end());

}

//这行代码是一个使用erase-remove惯用法的例子，它用于从vector容器中删除所有等于`member`的元素。`remove(members.begin(), members.end(), member)`函数会将所有不等于`member`的元素移到vector的前面，并返回一个指向第一个等于`member`的元素的迭代器。然后，`members.erase()`函数会删除从这个迭代器到vector末尾的所有元素¹。这种方法可以高效地从vector容器中删除多个元素，而不需要手动遍历容器并逐个删除元素。

// 发送消息

void sendMessage(const string& message) {

// 将消息发送给所有其他的群成员

for (const string& member : members) {

cout << "Send message to " << member << ": " << message << endl;

}

}

// 接收消息

string receiveMessage() {

// 从消息队列中获取一条消息并返回

return "";

}

// 添加子群

void addSubgroup(Group\* group) {

subgroups.push\_back(group);

}

// 删除子群

void removeSubgroup(Group\* group) {

subgroups.erase(remove(subgroups.begin(), subgroups.end(), group), subgroups.end());

}

private:

string name; // 群名称

string description; // 群描述

vector<string> members; // 群成员列表

vector<Group\*> subgroups; // 子群列表

};

八、编程题(12分)。

欧洲杯激战正酣，组委会编写软件实现赛事的管理，需要管理的内容有:

1. 球员。信息:编号、姓名、位置、身价、红黄牌数目等。需要具有的功能有:攻击、防守、犯规等。

2、教练。信息:编号、姓名、身价等。需要具有的功能有:阵型布置、战术指挥、场上球员更换等。  
3、裁判。信息:编号、姓名等;需要具有的功能有:犯规确认，红牌、黄牌、点球判罚、进球确认等。

4、球队。信息:编号、名称、队员、教练、积分等。

再main函数里需要调试的功能有：

每次比赛后需要统计所有球员红黄牌数目以确定出场受限情况;

每次比赛后需要统计球队和球员进球情况以确定球队积分和球员身价;每次比赛后需要统计每个球员的上场时间;

每次比赛后需要统计球队积分、阵型有效性评估等，以确认教练身价;每次比赛后可能阵前换帅（更换教练);

在以上描述的基础上，完成比赛的管理模拟，给出类的设计并给出main函数测试代码。