

计算机学院_高级语言程序设计_课程实验报告

| | | |
|--|-------|------------------|
| 实验题目： 多态（二） 动态_运行时多态 | | 学号： 202200130048 |
| 日期： 2023. 4. 21 | 班级： 6 | 姓名： 陈静雯 |
| Email： 1205037094@qq. com | | |
| <p>实验步骤与内容：</p> <ol style="list-style-type: none">1. 编写一个哺乳动物类 Mammal, 再由此派生出狗类 Dog, 二者都声明 speak() 成员函数, 该函数在基类中被声明为虚函数, 声明一个 Dog 类的对象, 通过此对象调用 speak 函数, 观察运行结果。2. 请编写一个抽象类 Shape, 在此基础上派生出类 Rectangle 和 Circle, 二者都有计算对象面积的函数 getArea()、计算对象周长的函数 getPerim()。注意不同的派生类可能需要额外的成员数据。（Shape 的析构函数之前应该加 virtual）3. 定义一个基类 BaseClass, 从它派生出类 DerivedClass。BaseClass 有成员函数 fn1()、fn2() 打印该函数调用信息, fn1() 是虚函数; DerivedClass 也有成员函数 fn1()、fn2()。在主函数中声明一个 DerivedClass 的对象, 分别用 BaseClass 和 DerivedClass 的指针指向 DerivedClass 的对象, 并通过指针调用 fn1()、fn2(), 观察运行结果。父指针（父类指针指子类对象）实现动态多态练习。4. 分析第 8 章 PPT 例 8-5, 非多态类型, 无虚析构函数, 造成内存泄漏的例子, 分析出错原因? (a) 在 main() 中增加语句 Base b1=Derived(); 请分析输出结果。 (b) 在 main() 中增加语句 Derived d1; fun(&d1); 程序会出问题吗?5. 阅读学习 cast 中几种不同的 cast 类型和用法, 理解并分析其中 p3. cpp 的运行结果。6. 第 8 章 PPT, P69 例 8-9 dynamic_cast 练习, 并做如下修改, 分析运行结果。 | | |
| <p>结论分析与体会：</p> <ol style="list-style-type: none">1. | | |
| <pre>#include <iostream> using namespace std; class mammal{ public: mammal(int x):m(x){}</pre> | | |

```
virtual void speak(){  
  
    cout<<"mammal"<<endl;  
  
}  
  
~mammal(){}  
  
private:  
  
    int m;  
  
};  
  
class dog:public mammal{  
  
public:  
  
    dog(int x,int y):mammal(x),n(y){}  
  
    void speak(){  
  
        cout<<"dog"<<endl;  
  
    }  
  
    ~dog(){}  
  
private:  
  
    int n;  
  
};  
  
int main(){  
  
    dog dd(1,2);  
  
    dd.speak();  
  
}
```

```
or-ggdx1y2.af1' --pid=Microsoft-MIEngine-Pid-31nf4ekx.hsa --dbgE
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
dog
PS D:\code_repository\code>
```

分析：派生类的虚函数覆盖了基类的虚函数，dog.speak 调用的是派生类的函数

2.

```
#include <iostream>

using namespace std;

class shape{

public:

    shape(){}

    virtual getarea(){.....}

    virtual getperim(){.....}

    virtual ~shape(){.....}

};

class rectangle:public shape{

public:

    rectangle(int x,int y):l(x),w(y){}

    void getarea(){

        int s=l*w;

        cout<<s<<endl;

    }

    void getperim(){

        int c=2*(l+w);
```

```
        cout<<c<<endl;

    }

    ~rectangle(){}

private:

    int l;

    int w;

};

class circle:public shape{

public:

    circle(int x):r(x){}

    void getarea(){

        double s=3.14*r*r;

        cout<<s<<endl;

    }

    void getperim(){

        double c=2*3.14*r;

        cout<<c<<endl;

    }

    ~circle(){}

private:

    float r;

};
```

```
int main(){

    rectangle rr(2,3);

    circle cc(3);

    rr.getarea();

    cc.getperim();

}
```

3.

```
#include <iostream>

using namespace std;

class baseclass{

public:

    baseclass(){

    }

    virtual void fn1(){

        cout<<"base fn1"<<endl;

    }

    void fn2(){

        cout<<"base fn2"<<endl;

    }

    ~baseclass(){

    }

};

class derivedclass:public baseclass{
```

public:

derivedclass(){}

void fn1(){

cout<<"derive fn1"<<endl;

}

void fn2(){

cout<<"derived fn2"<<endl;

}

~derivedclass(){}

};

int main(){

derivedclass d;

baseclass *p=&d;

derivedclass *pp=&d;

p->fn1();

p->fn2();

pp->fn1();

pp->fn2();

}

```
or-bqinfjcv.54h --pid=Microsoft-MEngine-Pid-11p0raud.don --db
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
derive fn1
base fn2
derive fn1
derived fn2
```

分析：父类指针指向子类对象，子类的虚函数已经把父类的同名虚函数覆盖了，所以父类指针调用 fn1 时会调用子类的虚函数，而调用 fn2 时调用的是父类的函数

4.

```
Microsoft-MIEngine-Out-cs.jkrhjo.k0j --stdin=Microsoft-MIEngine-Err
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
Base destructor
```

分析：没有调用派生类的析构函数，即 p 所指向的派生类的内存空间没有被释放，造成内存泄漏。应该把基类的析构函数声明为虚函数

(a)

```
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
Base destructor
Derived destructor
Base destructor
Base destructor
PS D:\code_repository\codes>
```

分析：Base *b = new Derived(); 中的 derived 无法析构，调用 fun 只能析构一个 base。而 Base b1=Derived(); 构造 derived 时会先构造一个 base，同时 b1 是 base 类，所以析构时会有一个 derived 析构，两个 base 析构。析构顺序与构造顺序相反。

(b)

```
24 }
25 void fun(Base* b) {
26     delete b; //只执行基类析构
27 }
28 int main() {
29     Base *b = new Derived();
30     fun(b);
31     Derived d1;
32     fun(&d1);
33     return 0;
34 }
```

调用堆栈

- [1] 因 PAUSE 已暂停
 - ntdll.dll!ntdll!Rtl
 - ntdll.dll!ntdll!Rtl
 - ntdll.dll!ntdll!Rtl
 - ntdll.dll!ntdll!Rtl
 - ntdll.dll!ntdll!Rtl
 - ntdll.dll!ntdll!Rtl
 - msvcrt.dll!msvcrt!f
 - fun(Base * b) 0
 - main() 000.cpp
- > [2] 已暂停
- > [3] 已暂停
- > [4] 已暂停

断点

- All C++ Exceptio...

问题 输出 调试控制台 终端


```
uncher.exe' '--stdin=Microsoft-MIEngine-In-2ylxb
icrosoft-MIEngine-Out-rcoyyjh3.zka' '--stderr=Mi
or-atjwmchw.3gi' '--pid=Microsoft-MIEngine-Pid-o
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
Base destructor
Base destructor
```

分析：在析构 d1 时报错，因为无法析构 derived，内存泄漏。

5.

```
or=umauscxe.b03 --pid=Microsoft-MIEngine-Pid-25jtrwqx.1f1  
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'  
unsafe_dynamic_cast_1  
PS D:\code_repository\codes> □
```

▼  p3.cpp C:\Users\cjt\w\Downloads\cast

 dynamic_cast of 'Base b' to 'class Derived*' can never succeed gcc [行 21,

分析:

static_cast 不能用来在不同类型的指针之间互相转换,也不能用于整型和指针之间的互相转换,也不能用于不同类型的引用之间的转换。所以转换不成功。

reinterpret_cast 不检查安全性,总是进行转换

dynamic_cast 专门用于将多态基类的指针或引用,强制转换为派生类的指针或引用,而且能够检查转换的安全性。对于不安全的指针转换,转换结果返回 NULL 指针。dynamic_cast 不能用于将非多态基类的指针或引用强制转换为派生类的指针或引用。Base b 中没有虚函数,不是多态基类

pd = dynamic_cast<Derived*> (&d); 是安全的转换

6.

(a)

```
Base::fun1()  
0x61fe08 0  
Derived1::fun1()  
Derived1::fun2()  
Derived2::fun1()  
Derived2::fun2()  
PS D:\code_repository\codes> □
```

分析: base 转换为 derived 不安全,返回 null, 另外两个成功且安全转换
父类指针指向子类对象调用函数调用的是子类的函数。

(b)

The screenshot shows a C++ IDE with a code editor and a debugger window. The code editor displays the following code:

```
19 b->fun1();
20 //尝试将b转换为Derived1指针
21 Derived1 *d = static_cast<Derived1 *>(b);
22 //判断转换是否成功
23 if (d != 0){
24     d->fun2();
25 }
26 else {
27     cout<<b<<" "<<d<<endl;
28 }
29
30 }
31 int main() {
32     Base b;
33     fun(&b);
}
```

The debugger window on the left shows the call stack with the following entries:

- > [2] 已暂停
- > [3] 已暂停
- > [4] 已暂停

The terminal window at the bottom shows the following output:

```
xe=D:\mingw64\bin\gdb.exe' '--interpreter=mi'
Base::fun1()
[]
```

分析：static_cast 不能用来在不同类型的指针之间互相转换，也不能用于整型和指针之间的互相转换，也不能用于不同类型的引用之间的转换。所以报错。