计算机学院 高级程序语言设计 课程实验报告

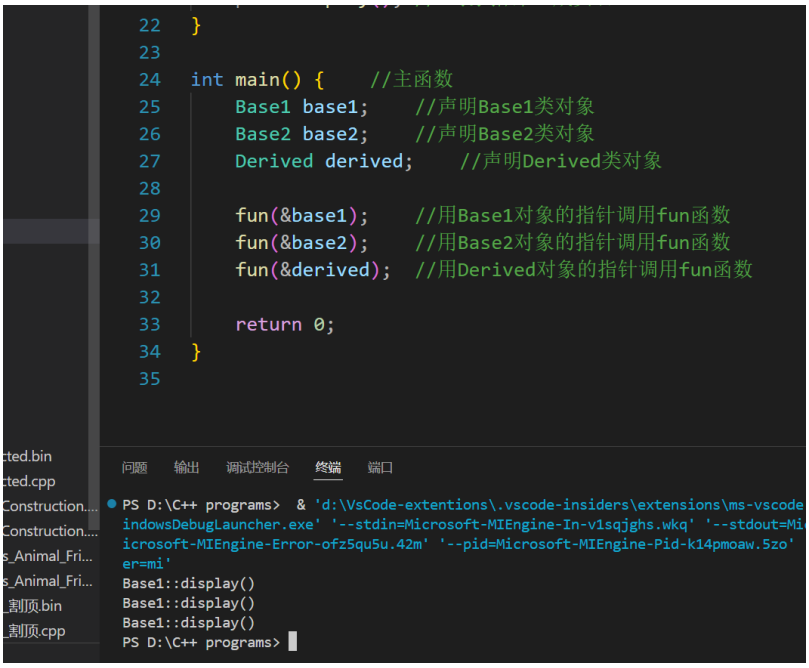
实验题目：多继承，继承中的类型兼容、构造、析构函数执行顺序及访问控制		学号：202300130150
日期：2024.4.11	班级： 4	姓名： 王成意

实验目的：

1. 实践多继承
2. 掌握继承中的类型兼容
3. 掌握继承中的构造、析构函数、初始化执行顺序及访问控制；
4. 掌握继承中同名隐藏问题的解决方案，虚基类。

实验步骤与内容：

1. 练习 7-3.cpp，继承中的类型兼容，截图并解释运行结果。



```
}

int main() {    //主函数
    Base1 base1;    //声明Base1类对象
    Base2 base2;    //声明Base2类对象
    Derived derived;    //声明Derived类对象

    fun(&base1);    //用Base1对象的指针调用fun函数
    fun(&base2);    //用Base2对象的指针调用fun函数
    fun(&derived);  //用Derived对象的指针调用fun函数

    return 0;
}
```

```
问题  输出  调试控制台  终端  端口
PS D:\C++ programs>  & 'd:\VsCode-extentions\.vscode-insiders\extensions\ms-vscode
indowsDebugLauncher.exe' '--stdin=Microsoft-MIEngine-In-v1sqjghs.wkq' '--stdout=Mi
icrosoft-MIEngine-Error-ofz5qu5u.42m' '--pid=Microsoft-MIEngine-Pid-k14pmoaw.5zo'
er=mi'
Base1::display()
Base1::display()
Base1::display()
PS D:\C++ programs>
```

结果：在基类对象出现的场合使用了派生对象进行替代，但是替代之后由于 fun（）函数内容，只发挥了基类的作用。30 行 Fun 函数运行时通过 ptr 指针只能指向 base2 中的 base1 的 display，而不是 base2 自己的 display。Derived 同理。

2. 练习例 7-5.cpp， 截图解释运行结果，为什么输出的顺序是这样的？

```
er=mi'
Constructing Base2 2
Constructing Base1 1
Constructing Base3 *
Constructing Base1 3
Constructing Base2 4
Constructing Base3 *
Destructing Base3
Destructing Base2
Destructing Base1
Destructing Base3
Destructing Base1
Destructing Base2
```

解释：

36 行构造 derived 类的对象 obj，调用 26 行其构造函数，先按照 23 行 derived 的继承顺序：2，1，3 构造，再按照参数表：

```
private:     //派生类的私
    Base1 member1;
    Base2 member2;
    Base3 member3;
};
```

进行接下来的构造。

验证：改变为：

```
private:     //派生类的私
    Base2 member2;
    Base3 member3;
    Base1 member1;
};
```

结果相应改变：

```
Constructing Base2 2
Constructing Base1 1
Constructing Base3 *
Constructing Base2 4
Constructing Base3 *
Constructing Base1 3
Destructing Base1
```

此题 main()函数中增加如下一句，会出现什么错误，请解释。

```
    Derived obj1;
```

编译结果：

```
正在启动生成...
cmd /c chcp 65001>nul && D:\mingw64\bin\g++.exe -fdiagnostics-color=always -g "D:\C++ programs\tmp\
tmp\7_5.exe" -std=c++17
D:\C++ programs\tmp\7_5.cpp: In function 'int main()':
D:\C++ programs\tmp\7_5.cpp:37:13: error: no matching function for call to 'Derived::Derived()'
     Derived obj1;
             ^~~~
D:\C++ programs\tmp\7_5.cpp:26:5: note: candidate: 'Derived::Derived(int, int, int, int)'
     Derived(int a, int b, int c, int d): Base1(a), member2(d), member1(c), Base2(b)
     ^~~~~~~
D:\C++ programs\tmp\7_5.cpp:26:5: note:   candidate expects 4 arguments, 0 provided
D:\C++ programs\tmp\7_5.cpp:23:7: note: candidate: 'constexpr Derived::Derived(const Derived&)'
 class Derived: public Base2, public Base1, public Base3 {
       ^~~~~~~
D:\C++ programs\tmp\7_5.cpp:23:7: note:   candidate expects 1 argument, 0 provided
D:\C++ programs\tmp\7_5.cpp:23:7: note: candidate: 'constexpr Derived::Derived(Derived&&)'
D:\C++ programs\tmp\7_5.cpp:23:7: note:   candidate expects 1 argument, 0 provided
```

解释：没有默认构造函数，但是加上：

```
Derived() = default;
```

发现：

```
D:\C++ programs\tmp\7_5.cpp:26:5: note: 'Derived::Derived()' is implicitly deleted because the default definition would be i
ormed:
    Derived() = default;
```

发现原因在于基类没有默认构造函数。添加后：

```
6   public:
7       Base1() = default;
8       Base1(int i) { cout << "Constr
9       ~Base1() { cout << "Destructin
10  };
11
12  class Base2 {   //基类Base2，构造函
13  public:
14
15      Base2() = default;
16      Base2(int j) { cout << "Constr
```

编译通过，结果：

```
Constructing Base2 2
Constructing Base1 1
Constructing Base3 *
Constructing Base2 4
Constructing Base3 *
Constructing Base1 3
Constructing Base3 *
Constructing Base3 *
Destructing Base1
Destructing Base3
Destructing Base2
Destructing Base3
Destructing Base1
Destructing Base2
Destructing Base1
Destructing Base3
Destructing Base2
Destructing Base3
Destructing Base1
Destructing Base2
PS D:\C++ programs>
```

练习例 7-6.cpp， 尝试做如下修改：

（1） 在 Base1 中增加 fun(int a)函数；

（2）在 Base2 增加 fun(int a,int b)函数；

（3）在 main()函数中增加如下语句：

d.fun(1);

d.fun(2,3);

测试是否可以通过编译？（即 Derived 类中的 fun()是否同名隐藏了父类形参不同的同名函数？）

不可以：

```
tmp\7_6.exe" -std=c++17
D:\C++ programs\tmp\7_6.cpp: In function 'int main()':
D:\C++ programs\tmp\7_6.cpp:41:12: error: no matching function for call to 'Derived::fun(int)'
     d.fun(1);
            ^
D:\C++ programs\tmp\7_6.cpp:22:10: note: candidate: 'void Derived::fun()'
     void fun() { cout << "Member of Derived" << endl; } //同名函数成员
          ^~~
D:\C++ programs\tmp\7_6.cpp:22:10: note:   candidate expects 0 arguments, 1 provided
D:\C++ programs\tmp\7_6.cpp:42:15: error: no matching function for call to 'Derived::fun(int, int)'
     d.fun(2, 3);
               ^
D:\C++ programs\tmp\7_6.cpp:22:10: note: candidate: 'void Derived::fun()'
     void fun() { cout << "Member of Derived" << endl; } //同名函数成员
          ^~~
D:\C++ programs\tmp\7_6.cpp:22:10: note:   candidate expects 0 arguments, 2 provided
```

即 Derived 类中的 fun()同名隐藏了父类形参不同的同名函数。

4.（1） 在例 7_7.cpp 的 main()函数中增加输出语句(如下),问有几个 var0?

答：2 个：

```
Member of Base0
Member of Base0
2  3
```

如果再增加输出语句 cout<<d.var0<<" "<<d.Base0::var0<<endl; 会怎样？为什么？

```
D:\C++ programs\tmp\7_7.cpp: In function 'int main()':
D:\C++ programs\tmp\7_7.cpp:34:13: error: request for member 'var0' is ambiguous
    cout<<d.var0<<"  "<<d.Base0::var0<<endl;
            ^~~~
D:\C++ programs\tmp\7_7.cpp:7:9: note: candidates are: 'int Base0::var0'
    int var0;
        ^~~~
D:\C++ programs\tmp\7_7.cpp:7:9: note:                 'int Base0::var0'
D:\C++ programs\tmp\7_7.cpp:34:34: error: 'Base0' is an ambiguous base of 'Derived'
    cout<<d.var0<<"  "<<d.Base0::var0<<endl;
                                ^~~~
```

解释：
第一个输出中“var0”有二义性，第二个输出中“base0”有二义性。

（2） 练习第 7_8.cpp，将 main 函数改为如下，解释运行结果。
结果：

```
1  1  1
Member of Base0
2  2  2
3  3  3
```

解释：
程序使用了虚基类，使得 base2，base1 继承的 base0 是同一个，都是 1，这样不会出现二义性，因此在

```
cout<<d.var0<<"  "<<d.Base1::var0<<"  "<<d.Base2::var0
<<endl;
```

这一句中输出的结果都是 1；
32 行更改了虚基类中的值，并且直接访问虚基类的函数成员，会出现“Mebmber of Base0”；在

```
cout<<d.var0<<"  "<<d.Base1::var0<<"  "<<d.Base2::var0
<<endl;
```

这一句中输出的结果都是 2；
随后 36，37 行两次修改，使 base0 中 var0 值最后定格在 3，输出：

```
cout<<d.var0<<"  "<<d.Base1::var0<<"  "<<d.Base2::var0
<<endl;
```

结果为 3 个 3.

结论分析与体会：

非常好的实验，使我深刻理解了多继承，继承中的类型兼容、构造、析构函数执行顺序及访问控制，尤其是关于 c++编译报隐式删除的错误，是因为成员函数不支持拷贝。