H10 PB18000290 胡毅翔

1.阅读 C++:94—类继承（菱形继承、虚继承(virtual 虚基类)），请自行编写简单的具有菱形继承的 C++程序（B 和 C 从 A 继承，D 从 B、C 继承），

1）练习用 nm 和 demangle 的方法分析程序中的 2 个名字的改编；

g++ c.cpp

得到 a.out

nm a.out

部分输出：

000000000000091e W _ZN1A2aaEi

0000000000000958 W _ZN1B4funcEv

nm --demangle a.out

部分输出：

000000000000091e W A::aa(int)

0000000000000958 W B::func()

分析：

_Z mangled 符号的前缀

N nested names 的前缀

1B,2aa 等 <length, id>

E 结束标志

v,i 等 参数类型

故_ZN1A2aaEi 表示 A::aa(int)；_ZN1B4funcEv 表示 B::func()。

2）试分析在虚继承和非虚继承下的对象布局和方法表布局，并给出你的分析方法。

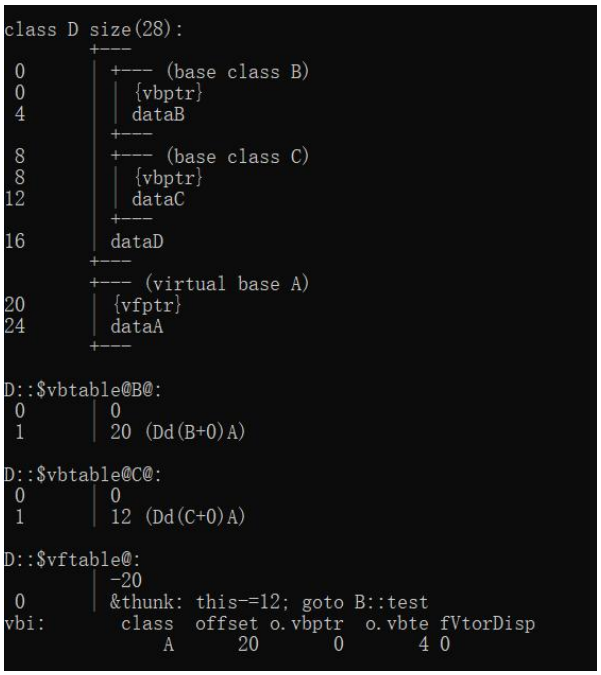
使用 Visual Studio 中的开发者命令行提示，在 c.cpp(非虚继承),c1.cpp(虚继承)所在目录下，输入命令

cl /d1 reportSingleClassLayout[ClassName] [FileName]

即可得到 ClassName 所对应的类的内存布局，本次实验所得如下：

虚继承：

D 的对象布局：



可以看出，虚继承时类 D 的对象布局与非虚继承时(见后文)有明显不同。

地址(按字节)	虚继承	非虚继承
0	vbptr 继承自父类 B 中的指针	dataA
4	dataB	dataB
8	vbptr 继承自父类 C 中的指针	dataA
12	dataC	dataC
16	dataD	dataD
20	vfptr 虚函数表指针	
24	dataA	

虚继承之所以能够实现在菱形继承中只保存一份共有基类，关键在于 vbptr 指针。Vbptr 即(Virtual base table pointer)，该指针指向了一个虚表，虚表中第一项记录了 vbptr 与本类的偏移地址；第二项记录了 vbptr 到共有基类元素的偏移量(具体可见上图中的 D::$vbtable$@B@ 和 D::$vbtable$@C@)。

虚函数表指针 vfptr 与 vbptr 同理。由上图可知，D 中保存的 A 部分有个虚函数表指针。表中指向了该类中的虚函数 test，根据上图信息知，D 中的 test 函数同 B 中的 test 函数。

B 的对象布局：

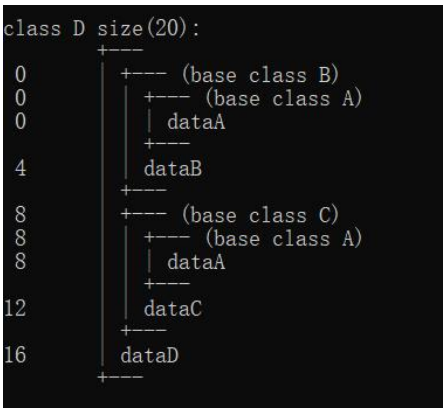
```
class B size(16):
        +---
 0      | {vbptr}
 4      | dataB
        +---
        +--- (virtual base A)
 8      | {vfptr}
12      | dataA
        +---

B::$vbtable@:
 0      | 0
 1      | 8 (Bd(B+0)A)

B::$vftable@:
        | -8
 0      | &B::test

B::test this adjustor: 8
vbi:       class  offset o.vbptr  o.vbte fVtorDisp
             A       8       0       4 0
```

C 的对象布局：

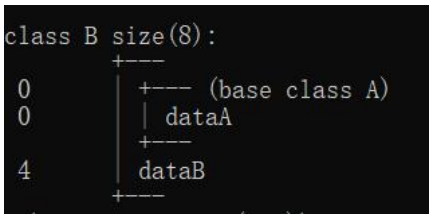
```
class C size(16):
        +---
 0      | {vbptr}
 4      | dataC
        +---
        +--- (virtual base A)
 8      | {vfptr}
12      | dataA
        +---

C::$vbtable@:
 0      | 0
 1      | 8 (Cd(C+0)A)

C::$vftable@:
        | -8
 0      | &A::test
vbi:       class  offset o.vbptr  o.vbte fVtorDisp
             A       8       0       4 0
```

非虚继承：

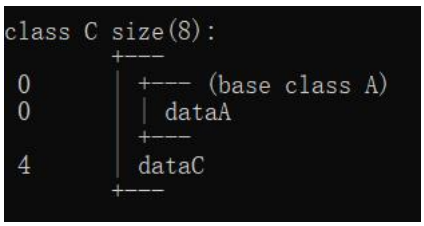
D 的对象布局：



B 的对象布局：



C 的对象布局：



2.

1）试从产生的汇编代码总结 gcc 处理 inline 的特征，内联是在编译的什么阶段被处理？

在未选择优化时，gcc 按照对普通函数的处理方式调用 f，而没有展开。在-O2 优化中对内联函数进行了展开。内联是在编译的链接阶段被处理。

2）试说明编译器进行了哪些优化而得到带-O2 选项生成的汇编码。

对 f()进行了展开。展开后删除了 f()中的死代码。运用了复写传播，减少了复写语句。公共子表达式 a 也用 1 进行了替换。

3）如果将 inline.h 第 1 行的 static 去掉，执行 gcc inline.c inline1.c，产生如下错误，试说明原因，并指出这是在编译的哪个阶段产生的错误。

这是在链接的阶段产生的错误。

原因是在 inline.c 和 inline1.c 都 include 了 inline.h，但没有使用 static 对函数 f()的作用域加以限制，展开后，两个 f()函数的作用域发生了重叠。故在链接 inline.o 和 inline1.o 时，报错出现 f 的重复定义。