Assignment 8

Fuyun Wang

191300051

一、概念题

1

多继承是指派生类可以有一个以上的基类。

1. 基类与派生类中的成员同名
2. 重复继承

解决办法：

1. 类明限定
2. 同名覆盖
3. 虚基类

2

聚合与组合

在聚合关系中，被包含的对象与包含它的对象独立的存在和消亡，被包含的对象可以脱离包含它的对象而存在。

聚合类的成员对象一般采用对象指针来表示，当然也可以使用vector。

在组合关系中，被包含的对象不能够独立于包含它的对象独立的存在和消亡。被包含的对象由包含它的对象创建，并随着包含它的对象的消失而消失。

组合类的成员对象一般直接是对象，是静态变量，但也可以不是，但一定要在包含它的对象中创建与消亡。

继承与组合

继承就是子类继承父类的特征和行为，使得子类对象（实例）具有父类的实例域和方法，或子类从父类继承方法，使得子类具有父类相同的行为。组合是通过对现有对象进行拼装即组合产生新的具有更复杂的功能。

继承的优缺点
优点：
支持扩展，通过继承父类实现，但会使系统结构较复杂
易于修改被复用的代码
缺点：
代码白盒复用，父类的实现细节暴露给子类，破坏了封装性
当父类的实现代码修改时，可能使得子类也不得不修改，增加维护难度。
子类缺乏独立性，依赖于父类，耦合度较高
不支持动态拓展，在编译期就决定了父类
组合的优缺点
优点：
代码黑盒复用，被包括的对象内部实现细节对外不可见，封装性好。
整体类与局部类之间松耦合，相互独立。
支持扩展
每个类只专注于一项任务
支持动态扩展，可在运行时根据具体对象选择不同类型的组合对象(扩展性比继承好)
缺点：
创建整体类对象时，需要创建所有局部类对象。导致系统对象很多。

二.编程题

1

构造函数调用顺序是

```
Object Base D1 Object D2 Mid Object D1 Object Base Object D2 Final
```

首先Final间接继承了Base，所以首先调用Base的构造函数

所以前两个是Object，Base

随后构造Mid，首先调用D1,D2的构造函数，因为D1,D2都只有一个虚基类所以直接调用自身的构造函数，因为D2有一个数据成员o，所以要先调用Object的构造函数，所以顺序为

D1,Object,D2,MID.

随后构造Object，object没有继承和对象成员，所以就只有一个Object。

随后构造D1,因为D1只有一个虚基类，所以只有一个D1.

因为Final有一个D2的数据成员o，所以调用o的构造函数，即

Object，Base，Object，D2

最后调用Final的构造函数。

析构函数与上述顺序相反。

2 仿照课堂上的例子，实现通用指针归并排序，可以对double类型进行排序。

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

bool doubleCmp(const void *a, const void *b)
{
    return *(double *)a < *(double *)b;
}
void Merge(void *base, unsigned l, unsigned imid, unsigned r, unsigned element_size, bool (*cmp)(const void *, const void *))
{
    int n1 = imid - l;
    int n2 = r - imid;
    char lbase[(n1 +1) * element_size];
    char rbase[(n2 +1) * element_size];
    memcpy((char *)lbase, (char *)base + l * element_size, n1 * element_size);
    memcpy((char *)rbase, (char *)base + imid * element_size, n2 * element_size);
    // *(double *)(lbase+(n1)*element_size)=1000000000;
    // *(double *)(rbase+(n2)*element_size)=1000000000;
    for (int i = 0; i < n1; i++)
    {
        cout << "a1:" << *(double *)(lbase + i * element_size) << "\t";
    }
    for (int i = 0; i < n2; i++)
    {
        cout << "b1:" << *(double *)(rbase + i * element_size) << "\t";
    }
    cout << endl;
    int i = 0, j = 0;
    for (int k = l; k < r; k++)
    {
        if ((j>=n2||cmp((char *)lbase + i * element_size, (char *)rbase + j * element_size))&&i<n1)
        {
            memcpy((char *)base + k * element_size, (char *)lbase + i * element_size, element_size);
            i++;
        }
        else
        {
            memcpy((char *)base + k * element_size, (char *)rbase + j * element_size, element_size);
```

```
            j++;
        }
    }
    for (int i = 0; i < n1; i++)
    {
        cout << "a2:" << *(double *)(lbase + i * element_size) <<
"\t";
    }
    for (int i = 0; i < n2; i++)
    {
        cout << "b2:" << *(double *)(rbase + i * element_size) <<
"\t";
    }
    cout << endl;
}
void SplitSort(void *base, unsigned l, unsigned r, unsigned
element_size, bool (*cmp)(const void *, const void *))
{
    for (int i = l; i < r; i++)
    {
        cout << *(double *)((char *)base + i * element_size) << "\t";
    }
    cout << endl;
    if (l >= r - 1)
        return;
    int imid = (l + r) / 2;
    SplitSort(base, l, imid, element_size, cmp);
    SplitSort(base, imid, r, element_size, cmp);
    Merge(base, l, imid, r, element_size, cmp);
}

void MergeSort(void *base, unsigned count, unsigned element_size, bool
(*cmp)(const void *, const void *))
{
    SplitSort(base, 0, count, element_size, cmp);
}
int main()
{
    double a[10] = {10.0, 9.0, 8.0, 7.0, 6.0, 5.0, 4.0, 3.0, 2.0,
1.0};
    MergeSort(a, 10, sizeof(double), doubleCmp);
    for (int i = 0; i < 10; i++)
    {
        cout << a[i] << " ";
    }
}
```