HW8

181220076 周韧哲

一. 概念题

1. 多继承是指一个类可以有两个或两个以上的直接基类。多个直接父类中出现同名成员时产生的二义性，同一个父类通过不同继承路径产生的二义性。当一个派生类从多个基类派生，而这些基类又有一个共同基类，当对该基类中说明的成员进行访问时，就会产生二义性，c++中可使用虚基类使这个公共基类在派生类中只产生一个对象来避免这种问题。还可以采用基类名受限访问来解决第一个命名冲突问题。
2. 继承容易进行新的实现，因为其大多数可继承而来，易于修改或扩展那些被复用的实现，缺点是破坏了封装性，因为这会将父类的实现细节暴露给子类。组合仅能通过被包含对象的接口来对其进行访问，封装性好，实现上的相互依赖性比较小，每一个类只专注于一项任务。缺点是会导致系统中的对象过多，为了能将多个不同的对象作为组合块来使用，必须仔细地对接口进行定义。如果类A和类B毫不相关，不可以为了使B的功能更多些而让B继承A的功能，而应该组合；若在逻辑上B是A的“一种”，则允许B继承A的功能，若在逻辑上A是B的“一部分”，则不允许B继承A的功能，而是要用A和其它东西组合出B。

二. 编程题

1. 构造顺序：Object，Base，Derived1，Derived2，Object，Object，Base，Derived1。因为基类的声明次序决定了对基类构造函数的调用次序，虚基类的构造函数优先非虚基类构造函数执行。析构顺序：Derived1，Base，Object，Object，Derived2，Derived1，Base，Object，即相反。

2.
```c
int double_compare(const void *p1, const void *p2){
    if(*(double*)p1 < *(double*)p2)
        return -1;
    else if(*(double*)p1 > *(double*)p2)
        return 1;
    else
        return 0;
}

void merge(void *base, unsigned int count, unsigned int element_size,
int (*cmp)(const void*, const void*), int p, int q, int m){
    int n1 = m - p + 1;
    int n2 = q - m;
    void* L = malloc(n1*element_size);
    void* R = malloc(n2*element_size);
    for (int i = 0; i < n1; i++) {
        char *p1 = (char*) base + (p+i)*element_size;
        char *l = (char*) L + i*element_size;
        for(int k=0;k<element_size;k++){
            l[k] = p1[k];
        }
    }
    for (int i = 0; i < n2; i++) {
        char *p1 = (char*) base + (i+m+1)*element_size;
        char *r = (char*) R + i*element_size;
        for(int k=0;k<element_size;k++){
            r[k] = p1[k];
        }
```

```
    }
    int i = 0, j = 0;
    for (int r = p; r <= q; r++) {
        if(j>=n2){
            char *target = (char*) base + r*element_size;
            char *src = (char*) L + i*element_size;
            for(int k=0;k<element_size;k++){
                target[k] = src[k];
            }
            i++;
        }else if(i>=n1){
            char *target = (char*) base + r*element_size;
            char *src = (char*) R + j*element_size;
            for(int k=0;k<element_size;k++){
                target[k] = src[k];
            }
            j++;
        }else{
            if((*cmp)((char*)L + i*element_size, (char*)R +
j*element_size) == -1){
                char *target = (char*) base + r*element_size;
                char *src = (char*) L + i*element_size;
                for(int k=0;k<element_size;k++){
                    target[k] = src[k];
                }
                i++;
            }else{
                char *target = (char*) base + r*element_size;
                char *src = (char*) R + j*element_size;
                for(int k=0;k<element_size;k++){
                    target[k] = src[k];
                }
                j++;
            }
        }
    }
}

void mergesort(void *base, unsigned int count, unsigned int
element_size, int (*cmp)(const void*, const void*), int p, int q){
    if(p<q){
        int m = (p+q)/2;
        mergesort(base, count, element_size, cmp, p, m);
        mergesort(base, count, element_size, cmp, m+1, q);
        merge(base, count, element_size, cmp, p, q, m);
    }
}

void merge_sort(void *base, unsigned int count, unsigned int
element_size, int (*cmp)(const void *, const void *)){
    mergesort(base, count, element_size, cmp, 0, count-1);
}
```