计算机学院　高级语言程序设计　课程实验报告

实验题目：实验五 对象数组，指针，动态内存、深拷贝浅拷贝		学号：202200400053
日期：2024-04-11	班级： 2202	姓名： 王宇涵
Email：1941497679@qq.com		

实验目的：

1. 练习对象数组
2. 练习动态内存的使用（深刻认识指针的灵活性与潜在的危险）
3. 练习动态数组的封装,vector 的使用
4. 探索浅拷贝与深拷贝不同
5. 掌握 string、getline()的使用

实验软件和硬件环境：

软件环境 ：VSCODE

硬件环境 ：Legion Y7000P

实验步骤与内容：

1. 运行学生用书第 6 章实验 6，实验任务（2）lab6_2。（针对 lab6_1 改写）

关键重构操作 ：

```
// 开辟每一行的空间
int **matrix = new int*[3];
//对于每一行开辟新的空间
for (int i = 0; i < 3; i++) {
    matrix[i] = new int[3];
}

//赋值
int count = 1;
for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
        matrix[i][j] = count++;
    }
}
```

输出结果 ：

```
Original Matrix:
1 2 3
4 5 6
7 8 9

Transposed Matrix:
1 4 7
2 5 8
3 6 9
```

2. 参照学生用书第 6 章习题 6-18，找出习题 6-20（SimpleCircle)中的错误。

答：没有正确释放 int *itsRadius 开辟的内存空间, 修改后为

```
SimpleCircle::~SimpleCircle() {
    delete itsRadius;
}
```

3. 分析实验附件.zip 中的 c6test.cpp，测试如下修改，对比不同运行结果。

1) 请注释 exec 函数中以下语句（第 44 行）的含义：

Object *a=new Object, &b(*new Object(*a));// 注释此句的含义

含义为：创建了一个指向 Object 对象的指针 a，并通过 new 关键字动态分配了内存来创建一个新的 Object 对象。此时 count 值增加，因为构造函数被调用。

接着，又创建了一个匿名的 Object 对象，并通过引用 b 引用了这个对象, 将 b 和匿名对象关联起来, 这个对象通过复制构造函数从指针 a 拷贝而来，然后 count 值再次增加，因为构造函数被调用。

2）如果此句中的&b(*new Object(*a))改为如下两种形式，其含义及对运行结果的影响有什么不同？

a) *b(new Object(*a));

b）b(*new Object(*a))

a) 匿名对象指针调用 object a 复制构造函数初始化, 并将该地址赋值给 b, 因此只调用了一次复制构造函数, 对运行结果无影响.

b) 匿名对象指针调用 object a 复制构造函数初始化, b 再调用匿名对象指针指向的匿名对象的复制构造函数, 因此调用了两次复制构造函数, 对运行结果有影响.

3）此题如何避免内存泄漏？cc 需要 delete 吗？

答：在 exec() 函数中，通过 new Object 创建了一个匿名的 Object 对象，并将其引用给 b。但是，没有在程序的任何地方释放这个匿名对象所占用的内存。因此我们要解决这个潜在的内存泄漏问题，需要在创建匿名对象后，在不再需要它时手动释放其内存。可以通过将其分配给指针来实现，然后在合适的时机使用 delete 释放内存.

cc 不需要 delete 来释放对应的内存, 因为它是一个引用, 而不是指针.

4. 浅拷贝与深拷贝

练习第六章 PPT，例 6-21 浅拷贝，讨论其中的问题，上面的 P103 例 6-16 有这种问题吗？为什么？

练习例 6-21

```
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
Destructor called.
```

发现系统产生运行时错误.

因为调用复制构造函数的时候直接复制指针，则两个变量共有同一个空间，释放内存的时候会重复释放两次相同的空间，会产生报错.

例 6-16 不会产生类似的问题，因为两个变量 new 开辟空间时分配的是不同的空间，释放内存不会产生问题.

5. 练习第六章 PPT，P116-120 vector 应用举例，讨论它与上面的动态数组有什么不同？

思考，它是怎样实现动态扩展数组空间大小的？

答： **不同点**：vector 可以自动管理动态内存，并提供了一系列方便的成员函数用于访问和操作元素。动态数组需要使用 new 和 delete 关键字，需要手动分配和释放内存，否则会造成内存泄漏。

实现动态扩展数组空间大小过程如下：

当向 vector 添加元素时，如果当前的内部数组已经无法容纳更多的元素，vector 会分配一个更大的内存空间，通常是当前容量的两倍。且 vector 通常还会在当前容量用尽之前就进行预先分配，以减少动态扩展的次数，提高效率。

Vector 复制元素时将当前的元素从旧的内存空间复制到新的内存空间。

Vector 释放空间时释放旧的内存空间，以防止内存泄漏。

6. 练习 string 类、getline 函数

（1）练习第六章 PPT，例 6-24。

（2）运行学生用书第 6 章实验 6，实验任务（6）lab6_6。要求用 string 类替代字符数组实现。

（string 对象为什么不怕随便复制，造成原字符串对象泄漏？）

(1) 测试

```
hi c++, ok
City: hi c++; state:  ok
```

(2) 测试样例

```
int main() {
    const int NUM_EMPLOYEES = 2;
    Employee employees[NUM_EMPLOYEES] = {
        Employee("约翰·多", "123主街", "任意城镇", "12345"),
        Employee("简·史密斯", "456橡树街", "他镇", "54321"),
    };

    // 显示每个雇员的信息
    for (int i = 0; i < NUM_EMPLOYEES; ++i) {
        cout << "雇员 #" << i + 1 << ": " << endl;
        employees[i].display();
        cout << endl;
    }

    // 修改雇员的姓名
    employees[0].change_name("约翰·史密斯");
    cout << "修改后的姓名：" << endl;
    employees[0].display();
    return 0;
}
```

```
雇员 #1:
姓名：约翰·多
街道地址：123主街
城市：任意城镇
邮编：12345

雇员 #2:
姓名：简·史密斯
街道地址：456橡树街
城市：他镇
邮编：54321
```

为什么 string 不怕对象泄露?

答 ：string 类提供了封装好的字符串管理机制，使得可以安全地对字符串进行复制、赋值和销毁，而不必担心内存泄漏问题. string 使用动态内存分配来存储字符串内容，这意味着它会根据需要自动调整内存大小，以容纳字符串的长度, 不同的对象一般占用不同的内存空间.

7. 综合应用

课本 6.7 综合实例——个人银行账户管理程序，即例 6-25。

（字符串表示银行账号，账户数组，设计 Date 日期类)

测试样例 ：

```
int main() {
    Date date(2008, 11, 1); //起始日期
    //建立账户数组
    SavingsAccount accounts[] = {
            SavingsAccount(date, "03755217", 0.015),
            SavingsAccount(date, "02342342", 0.015)
    };
    const int n = sizeof(accounts) / sizeof(SavingsAccount); //账户总数
    //11月份的几笔账目
    accounts[0].deposit(Date(2008, 11, 5), 5000, "salary");
    accounts[1].deposit(Date(2008, 11, 25), 10000, "sell stock 0323");
    //12月份的几笔账目
    accounts[0].deposit(Date(2008, 12, 5), 5500, "salary");
    accounts[1].withdraw(Date(2008, 12, 20), 4000, "buy a laptop");
    //结算所有账户并输出各个账户信息
    cout << endl;
    for (int i = 0; i < n; i++) {
        accounts[i].settle(Date(2009, 1, 1));
        accounts[i].show();
        cout << endl;
    }
    cout << "Total: " << SavingsAccount::getTotal() << endl;
    return 0;
}
```

输出结果 ：

```
2008-11-1       #03755217 created
2008-11-1       #02342342 created
2008-11-5       #03755217       5000    5000    salary
2008-11-25      #02342342       10000   10000   sell stock 0323
2008-12-5       #03755217       5500    10500   salary
2008-12-20      #02342342       -4000   6000    buy a laptop

2009-1-1        #03755217       17.77   10517.8 interest
03755217        Balance: 10517.8
2009-1-1        #02342342       13.2    6013.2  interest
02342342        Balance: 6013.2
Total: 16531
```

探索 ： 什么样的程序需要考虑深拷贝？

答 ： 以下几种情况的程序:

1. 自定义类含有指针成员： 如果你的自定义类包含指针成员，并且这些指针指向了动态分配的内存，则需要确保在拷贝对象时，复制指针指向的数据而不仅仅是指针本身。
2. 使用动态分配内存的容器： 如果你使用了诸如 vector、list、map 等动态分配内存的标准容器，并且容器中存储了指针或者自定义对象，那么在进行拷贝时需要考虑深拷贝，以确保每个副本都有自己独立的内存空间。

结论分析与体会：

本次实验我通过练习对象数组掌握了对象数组的使用, 通过练习动态内存的使用深刻认识了指针的灵活性与潜在的危险, 通过练习动态数组的封装, vector 的使用掌握了动态开辟空间的原理, 此外我还探索浅拷贝与深拷贝不同, 理解了实际开发中很可能出现的 bug, 最后掌握 string、getline()的使用, 不再局限于使用字符数组来存储字符.

经过这次实验, 我更加懂得了理论课的知识, 并提升了对于不同知识点的理解, 收获良多

就实验过程中遇到的问题及解决处理方法，自拟 1－3 道问答题：

1. 解释 Int &b (a)，其中 a 是 Int 类型的匿名对象?

答 ：代表 b 引用了 a.

2. 如何进行深拷贝 ？

答 ：不是单纯的复制地址, 而是重新开辟一块空间, 并赋对应的值.