20220326-C++练功

1.过程描述

1.1 C++ Prime

头文件防护 C++ 复制代码

```
为了防止头文件被include多次，在定义头文件内容时，可以采用以下方法：
#ifndef func_A_
#define func_A_
……
#endif
这样不能防止头文件被包含多次，但能让编译器忽略第一次包含之外的内容
```

C++存储数据的方案 C++ 复制代码

```
1.自动存储持续性：
在函数定义或代码块中声明的变量，在程序开始执行时被创建，在执行完后它们使用的内存被释放
2.静态存储持续性：
在函数定义外定义和使用static定义的变量，在程序整个运行过程中都存在。静态变量未被初始化时将被设置为0
->函数（和代码块）中的static的作用域为函数内，但与自动变量不同的是，它在函数没有执行时也留在内存中（无链接性）
->函数外的static的作用域为当前文件（内部链接性），而其它静态变量（全局变量）的作用域为全部文件（外部链接性）
3.线程存储持续性：
使用关键字thread_local声明的变量，声明周期与所属的线程一样长
4.动态存储持续性：
用new分配的内存将一直存在，使用delete或程序结束才被释放。又被称为自由存储或堆（动态内存由运算符new和delete控制，而不是由作用域和链接性规则控制。因此可以在一个函数中分配动态内存，而在另一个函数中将其释放。
float * p_fees=new float[20]

编译器通常使用三块独立的内存：
1.用于静态变量；
2.用于自动变量；
3.用于动态存储。
```

使用栈来管理自动变量（LIFO：后进先出）

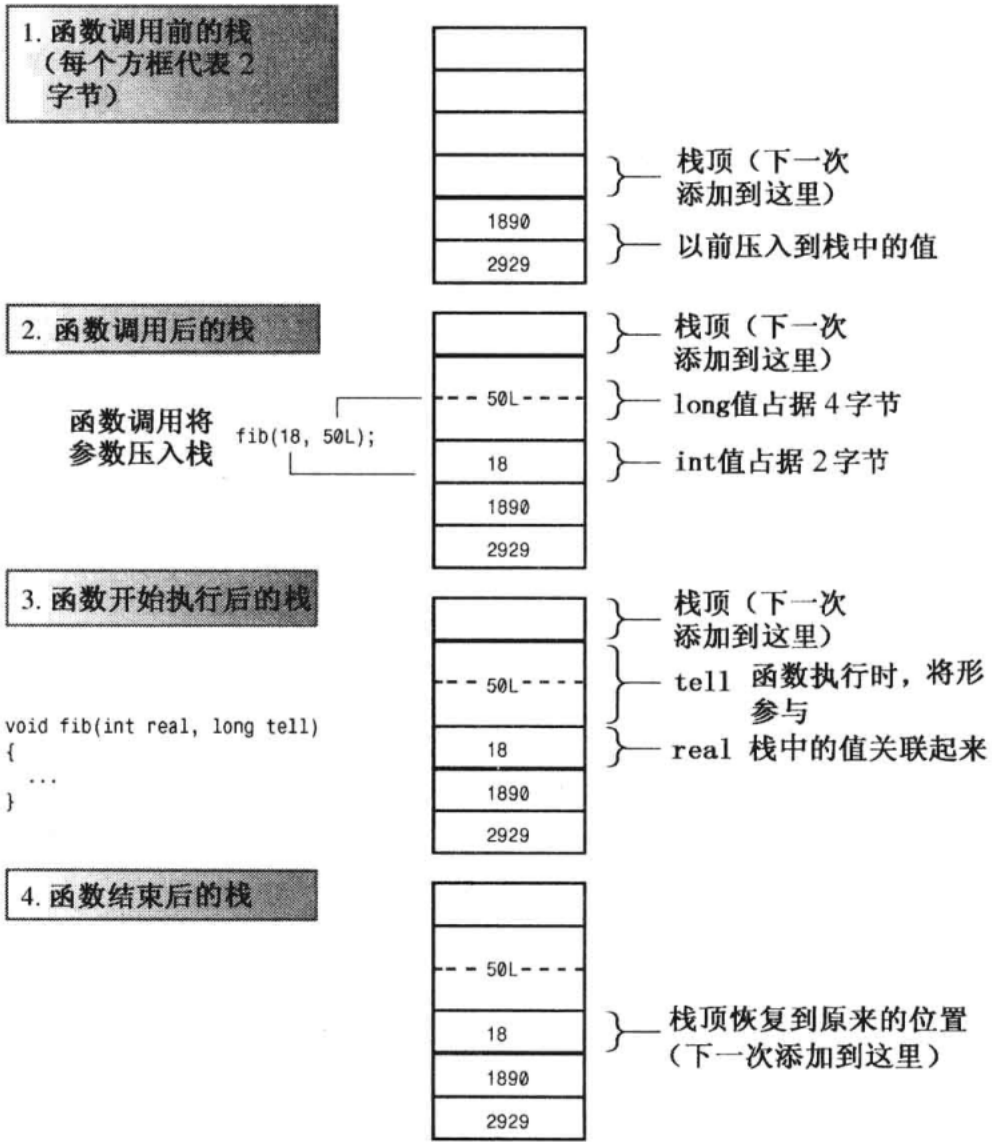

图 9.3　使用栈传递参数

extern引用声明　　C++　复制代码

```
为了满足全局变量只能定义一次的需求，C++提供了两种变量声明：
1.定义声明：给变量分配存储空间
2.引用声明：不给变量分配存储空间，而是引用已有的变量。引入extern关键字。
extern int a; //a在其它地方定义
extern int b=10; //因为初始化了，认为是定义声明
当在函数中定义了一个同名的b时，函数将使用b的局部定义；
C++中提供了作用域解析运算符（：：），使用::b将使用b的全局版本
为了解决C++和C对于函数符号名称的不同处理，可以在函数原型中指出要使用的约定：
extern "C" void spiff(int);
extern "C++" void spaff(int);

```

mutable关键字 C++ 复制代码

```
mutable用来指出，即使结构（或类）变量为const，其某个成员也可以被修改
struct Profile
{
    mutable int age;
    const char* name;
};
int main()
{
    const Profile Willian = { 20, "Willian" };
    Willian.age = 30;
    cout << Willian.age;
}
```

const关键字 C++ 复制代码

```
const表明，内存被初始化后，程序便不能再对它进行修改。
在默认情况下全局变量的链接性为外部的，但const全局变量的链接性为内部的（跟使用static作用相似）
const a=10;//作用域为当前文件
extern const a=10;//使用extern关键字覆盖默认的内部链接性
```

new运算符 C++ 复制代码

```
new用于动态分配内存，使用new和new[]分别调用如下函数：
void * operator new(std::size_t);
void * operator new[](std::size_t);
使用delete和delete[]分别调用如下函数：
void operator delete(void *);
void operator delete[](void *);

std::size_t是一个typedef，对应于合适的整形
int *pi=new int--->int *pi=new(sizeof(int));
int *pi=new int[40]--->int *pi=new(40*sizeof(int));

1.使用new初始化
内置标量类型
int * a=new int(9);
结构与数组
struct profile{double x;double y;double z};
profile* willian=new profile{2.3,4.5,6.7};
int* ar=new int[4]{1,3,4,5};
列表初始化用于单值变量
int *pin=new int{6};

2.定位new运算符
new通常负责在堆（heap）中找到一个足以满足要求的内存块。而定位new能指定要使用的位置
#include <new>
char buffer1[50];
struct profile{double x;double y;double z};
profile *pt1, *pt2;
pt1=new profile;//new的常规用法，将结构体放在heap里
pt2=new(buffer1) profile;//指定放在buffer1里

例子1：
double* pd1, * pd2;
int i;

pd1 = new double[N];
pd2 = new(buffer)double[N];

for (i = 0; i < N; i++)
{
    pd2[i] = pd1[i] = 1000 + 20.0 * i;
}

cout << "Memory addresses:\n" << "heap: " << pd1 << " static: " <<
(void*)buffer << endl;//p1是double指针，buffer是char指针，所以使用void*对
buffer进行强制转换
```

```
44  cout << "Memory contents:\n";
45  for (i = 0; i < N; i++)
46  {
47      cout << pd1[i] << " at " << &pd1[i] << "; ";
48      cout << pd2[i] << " at " << &pd2[i] << endl;
49  }
50
51  double* pd3, * pd4;
52  pd3 = new double(N);
53  pd4 = new(buffer) double(N);//内存位置还是buffer;
54
55  for (i = 0; i < N; i++)
56  {
57      pd4[i] = pd3[i] = 2000 + 20.0 * i;
58  }
59
60  cout << "Memory addresses:\n" << "heap: " << pd3 << " static: " <<
    (void*)buffer << endl;//p1是double指针，buffer是char指针，所以使用void*对
    buffer进行强制转换
61  cout << "Memory contents:\n";
62  for (i = 0; i < N; i++)
63  {
64      cout << pd3[i] << " at " << &pd3[i] << "; ";
65      cout << pd4[i] << " at " << &pd4[i] << endl;
66  }
67
68  double* pd5;
69  pd5 = new(buffer + N * sizeof(double)) double(N);//在buffer旁边的位置分配内
    存
70
71  for (i = 0; i < N; i++)
72  {
73      pd5[i] = 3000 + 20.0 * i;
74  }
75
76  cout << "Memory addresses:\n" << " static: " << (void*)(buffer + N *
    sizeof(double)) << endl;//p1是double指针，buffer是char指针，所以使用void*对
    buffer进行强制转换
77  cout << "Memory contents:\n";
78  for (i = 0; i < N; i++)
79  {
80      cout << pd5[i] << " at " << &pd5[i]  << endl;
81  }
82
83  delete[] pd1;
84  delete[] pd2;//按理说应该是会报错的，因为delete只能用于这样的指针：指向常规new运算
    符分配的堆内存
85  delete[] pd3;
```

```
delete[] pd4;
delete[] pd5;
```

void*

C++ | 复制代码

```
如果函数的参数或返回值可以是任意类型指针，那么应声明其类型为void*;
1.任何类型的指针都可以直接赋值给void指针
double b=9.8;
double *pt=&b;//void*能存放任意类型对象的地址
void* pv=&b;
pv=pb;//pv能存放任意类型的指针

2.void指针需类型转换才能赋值给其它类型
double b=9.8;
void* pv=&b;
double* pt=pv;//错误
double* pt=(double*)pv;//（显式类型转换）

3.void指针能和其它类型的指针直接比较指针存放的地址值是否相同
pv==pt;

4.void指针是强制类型转换后才可以正常对其操作
double b = 9.8;
void* pv = &b;
cout << *pv<<endl;//报错
cout << *(double*)pv << endl;//OK

5.void指针可以通过null和nullptr初始化
void* pv=0;
void* pv=nullptr;

6.当void指针作为函数的输入和输出时，表示可以接收任意类型的输出指针和输出任意类型的指针。参考new跟delete
new用于动态分配内存，使用new和new[]分别调用如下函数：
void * operator new(std::size_t);
void * operator new[](std::size_t);
使用delete和delete[]分别调用如下函数：
void operator delete(void *);
void operator delete[](void *);
```

构造函数 C++ 复制代码

```
1.构造函数用来对对象进行初始化（不能在类的声明中对变量进行初始化赋值，因为在创建对象前没有用于存储值的空间），初始化方式：
DemoClass demo(初始化值);
DemoClass demo=DemoClass(初始化值）；
DemoClass *ptr=new DemoClass(初始化值);
还可以用列表来进行初始化
DemoClass demo={初始化值};
DemoClass demo{初始化值};


要在类中定义常量有两种方法：
1)枚举
class Profile{
    private:
        enum{months=12};
        double arr[months];
}
用这种方式声明枚举不会创建类数据成员，所有对象中都不包含枚举
2) static
class Profile{
    private:
        static const int months=12;
        double arr[months];
}
该常量将与其它静态变量存储在一起，而不是存储在对象中。所有这个常量将被所有对象共享
2.构造函数的定义方式：
DemoClass::DemoClass(){};
DemoClass::DemoClass(atrr)
     :A_Var(atrr){}//将A_VAR的值初始化为atrr


构造函数可以有多个，但注意要符合重载的要求
一般有一个默认的构造函数(没参数）以及一个有参数的构造函数
DemoClass::DemoClass(){}
DemoClass::DemoCalss(int a,double b){}
在创建对象时两种方式都可以，即可以有实参也可以没有，相当于两种不同的初始化方法


3.可以用const DemoClass ObjDmo来创建对象，但必须确保对象所调用的函数不对这个对象进行修改。解决方法：
类的某个成员函数如果这样声明： void func() const;表示该函数将不能对调用的对象进行修改
```

this指针 C++ 复制代码

```
this指针指向用来调用成员函数的对象（this被作为隐藏参数传递给方法）。每个成员函数都有一个this指针，指向调用对象，如果要引用整个调用对象，可以使用表达式*this

const demo& demo::PriceCMP(demo & s) const
{
    if (s.acquire() > price)
    {return s;}
    else
    {return *this;}
}

当要创建多个对象时，可以用对象数组
demo demoArr[10]=
{
    demo(初始值1),
    demo(初始值2),
}

const demo* a = &objDemo1;
a=&a->PriceCMP(objDemo2); //注意这里是先执行&后面的函数调用，然后再用&得到pointer

★this还需要再多琢磨一下，369页const以及引用还是有点混乱
```

1.2 香港的治与乱

- 香港人身上有种脱离时代的集体优越感；漂泊感；避难者心态，希望与内地保持安全距离；受害者心理；弱者的高度敏感及警戒心态
- 通过少数对香港经济社会拥有巨大影响力的富豪来对香港社会进行某种形式的管制是英国与北京都认可的方式，因为能降低沟通成本，然而这种脱离群众的管治必然会带来很大的问题
- 这些富豪为了避免成为众矢之的，也有足够的动机操纵他们的媒体机器，引导民众将香港经济社会的结构性矛盾归咎于一国两制
- 香港的上一代过分关注内地脏乱差的一面，而忽视其崛起与发展，这种心态传承到年轻一代身上，成为了恐惧与厌恶情绪的来源。
- 两边没有有效的沟通管道，相互缺乏了解
- http://www.aisixiang.com/data/101206-2.html

2.结果输出

今天主要看了存储跟类的入门，知识点比较零散。C++有很多比较微妙的特性，想一口吃成个大胖子确实不太实际，后续得多找一些实际的项目练练手。果然还是没能在三天内解决战斗，明天争取把类部分看完，后续大块的还有STL跟模板、元编程之类的东西，可能得之后再抽些零碎的时间好好学习一下。