20220325-C++练功

1.过程描述

1.1 C++ Prime

函数指针 C++ 复制代码

```
函数的地址是存储其机器语言代码的内存的开始地址。函数名就表示函数的地址
process(think)//传入的是函数的地址,能在内部调用think()函数
process(think())//传入的是函数的返回值
----------------------------------------
1.声明函数指针
double pam(int)
double (*pf)(int) //表示pf是一个指向函数的指针，注意跟double *pf(int)有区别，这表示pf()是一个返回double类型指针的函数；
pf=pam;//pf现在指向pam()函数

假如现在要将pam()函数的地址传递给es函数，则es的函数原型:
void es(int lines,double (*pf)(int));
传递地址：es(50,pam);
----------------------------------------
2.使用指针来调用函数
(*pf)扮演的角色与函数名相同，因此使用时把它看作函数名就行
double pam(int)
double (*pf)(int)
double x=pam(5)
double y=(*pf)(5)//c++也允许double y=pf(5)，但这种不够明显
----------------------------------------
3.示例1
double besty(int);
double pam(int);
void estimate(int lines, double (*pf)(int));
int main()
{
    int codes;
    cin >> codes;
    estimate(codes, besty);
    estimate(codes, pam);
}

double besty(int lines)
{return 0.5 * lines;}
double pam(int lines)
{return 0.05 * lines;}

void estimate(int lines, double (*pf)(int)) //注意这里的声明
{
    cout << "lines will take " << (*pf)(lines) << endl;
}
----------------------------------------
4.示例2
//下面三种声明是一样的
```

```cpp
const double* f1(const double ar[], int n);
const double* f2(const double[], int);
const double* f3(const double*, int);

int main()
{
    //-----------------------------------------------------
    double av[3] = { 11.2,22.3,33.4 };
    const double* (*p1)(const double*, int) = f1;
    //这里也可以用typedef,将p_fun变为一个类型名
    //typedef double* (*p_fun)(const double*, int);
    //p_fun p1 = f1;
    auto p2 = f2;
    cout << (*p1)(av, 3) << endl; //输出的是数组第一个元素所在的地址
    cout << *(*p1)(av, 3) << endl; //输出11.2
    cout << p2(av, 3) << endl;//输出第二个元素的地址
    cout << *p2(av, 3) << endl;//输出22.3
    //-----------------------------------------------------
    const double* (*pa[3])(const double*, int) = { f1,f2,f3 };//pa是一个指针数组
    auto pb = pa;//pb是指向pa第一个元素的指针
    for (int i = 0; i < 3; i++)
    {
        cout << pa[i](av, 3) << endl;
        cout << *pa[i](av, 3) << endl;
    }
    for (int i = 0; i < 3; i++)
    {
        cout << pb[i](av, 3) << endl;
        cout << *pb[i](av, 3) << endl;
    }
    //-----------------------------------------------------
    auto pc = &pa;

    cout << (*pc)[0](av, 3) << endl;//输出第一个元素的地址
    cout << *(*pc)[0](av, 3) << endl;//输出第一个元素
    //-----------------------------------------------------
    const double* (*(*pd)[3])(const double*, int) = &pa; //属实有点变态
    const double* pdb = (*pd)[1](av, 3);
    cout << pdb << endl;
    cout << *pdb << endl;
}

const double* f1(const double *ar, int n){return ar;}
const double* f2(const double ar[], int n){return ar + 1;}
const double* f3(const double ar[], int n){return ar + 2;}
```

引用变量 C++ 复制代码

```
1.创建引用变量
int a=10;
int & b=a; //这里&不是地址运算符，而是指向int的引用。这里a和b指向相同的值和内存单元。假如b+1，则a也将+1
int* pt;
pt = &a;
引用跟指针的区别在于，必须在声明引用时将其初始化，而指针可以先声明再赋值。引用意味着一旦与某个变量关联起来，将一直效忠于它。
int & b=a 等价于 int* const ptr=&a,b的角色与*ptr一样
-----------------------------------------------------------------
2.将引用用作函数形参
例子：swap。指针和引用都能实现交换
void swap(int& c, int& d);
int main()
{
    int a = 10;
    int b = 30;
    swap(a, b);
    cout << a << "," << b << endl;
}
void swap(int & c, int & d)
{
    int temp;
    temp = c;
    c = d;
    d = temp;
}

void swap(int* c, int* d);
int main()
{
    int a = 10;
    int b = 30;
    swap(&a, &b);
    cout << a << "," << b << endl;
}
void swap(int* c, int * d)
{
    int temp;
    temp = *c;
    *c = *d;
    *d = temp;
}

```

```
//如果不想修改原来的数据，要尽可能使用const声明。当数据比较小时，引用节省内存的效应不是很明显，尽量用按值传递的方式。当数据比较大时再考虑使用引用
void swap(const int& c, const int& d);
-------------------------------------------------------------
3.返回引用
返回引用在某些情况下效率更高。需要注意不能返回函数终止时不再存在的内存引用
struct Profile
{
    int age;
    string name;
};
Profile& func1(Profile& prof)
{
    return prof;  //调用之后返回的就是Willian;
}

Profile func2(Profile& prof)
{
    return prof; //调用之后返回的是Willian的副本;
}

int main()
{
    Profile Willian = { 89,"Willian" };
    Profile Mary = { 64,"Mary" };
    cout << &Willian<< endl;
    cout << &func1(Willian) << endl; //跟上面的地址一样
    cout << &func2(Willian) << endl;//会报错，这也侧面说明返回的只是一个副本

    func2(Willian) = Mary;
    cout << Willian.name; //还是Willian
    func1(Willian) = Mary;
    cout << Willian.name; //变成了Mary

    auto a = func1(Willian); //这里直接将Willian复制到a，效率更高
    auto b = func2(Willian); //这里先将结果复制到一个临时位置，再复制到b
}

```

函数模板 C++ 复制代码

```
1.模板允许以泛型的方式进行编程，其实可以理解为在函数基础上进一步提高自由度，允许根据需要传入任意数据类型。同一个算法可以用于不同类型。
template <typename T>
//这里typename改成class也可以
void Swap(T& a, T& b)
{
    T temp;
    temp = a;
    a = b;
    b = temp;
}
int main()
{
    int a = 10;
    int b = 20;
    Swap(a, b);
    cout << b << endl;
}
模板允许重载，只要参数列表不同就行；另外也并非所有的形参的类型都必须为T，可以自行规定好
-----------------------------------------------
2.显式具体化
下面是三个函数声明（模板）。非模板函数优先于具体化和常规模板，具体化优先于常规模板
1）.非模板函数
void Swap(structDemo&,structDemo&);
2）.常规模板
template<typename T>
void Swap(T&,T&);
3）.具体化
template<>void Swap<structDemo>(structDemo&,structDemo&);

在定义时：
template<typename T>
void Swap(T&,T&){……}

template<>void Swap<structDemo>(structDemo&,structDemo&){……}

3.实例化和具体化
->下面为显式实例化
template void Swap<structDemo>(structDemo&,structDemo&);
将使用Swap模板生成一个使用structDemo类型的函数定义
->下面为显示具体化
template<> void Swap<structDemo>(structDemo&,structDemo&);
不使用Swap模板生成一个使用structDemo类型的函数定义，必须在后面补充自己的函数定义

4.decltype关键字
```

```
可以用于不知道变量类型的时候
template<class T1,class T2>
void ft(T1 x,T2 y)
{
    ……
    decltype(x+y) xpy=x+y;
}


double x=5.5;
decltype(x) w;//w为double类型

long indeed(int);
decltype(indeed(3)) m;//m为int类型，这里并不会实际调用函数，而是通过查看函数原型来获悉返回类型

double xx=4.4;
decltype ((xx)) r2=xx;//r2为double&,注意加了两个括号

int j=3;
int& k=j;
int& m=j;
decltype(k+m) var;//var是int类型，注意k+m不是引用

一种更复杂的用法
template<class T1,class T2>
auto ft(T1 x,T2 y)->decltype(x+y)
{
   return x+y
}

```

1.2 知识大融通

- 科学是一种解放了的宗教
- 追求客观的真实性，而非神的显现，这是另一种满足宗教饥渴的方法
- 伊卡洛斯式的精神

  希腊建筑师兼发明家代达罗斯替克里特岛的国王米诺斯建造一座路线设计非常巧妙的迷宫，用来关住米诺斯那个牛头人身的儿子弥诺陶洛斯。但国王担心迷宫的秘密走漏，于是下令将代达罗斯和他的儿子伊卡洛斯一同关进那座迷宫里高高的塔楼，以防犯他们逃脱。

  为了逃出，代达罗斯设计了飞行翼。然而，飞行翼是以蜡结合鸟羽制成，不能耐高热，代达罗斯告诫儿子：“飞行高度过低，蜡翼会因雾气潮湿而使飞行速度受阻；而飞行高度过高，则会因强烈阳光照射的高热而灼烧，造成蜡翼融化。”

他们父子从岛上的石塔展翅飞翔逃出，年轻的伊卡洛斯因初次飞行所带来的喜悦感受，他越飞越高，因太接近太阳而使蜡翼融化，最终导致坠海身亡。父亲代达罗斯目睹此景，悲伤的飞回家乡，并将自己身上的那对蜡翼悬挂在奥林帕斯山的阿波罗神殿（Temple of Apollo）里，从此不再想飞翔。

- 哲学在知性的综合上扮演一个重要的角色，它让我们意识到许多世纪以来思想发展的连续性和威力，同时也帮助我们展望未来，给定未知事物某种形象，这向来是哲学被赋予的天职。

结果输出

今天主要完成了C++函数部分，本来打算把类也开个头，结果有点没心思了，盯了大半天电脑眼睛也快撑不住。明天估计没办法结束类的部分，乐观的情况下两天能完成，第一项任务大概率是要超时了。之前听别人说C++很难学明白还觉得有点不以为然，现在终于是领教到了。所有东西都花功夫细细思考的话估计一个月都看不完。这第一项任务的目标也不是把prime彻底吃透，而是初步窥探一下函数、类、指针这几个大魔头，更深入的了解只能靠在实际的项目中达成了。明后两天会根据情况调整之前的规划。