http://wdxtub.com/2016/04/16/thin-csapp-2/

Basic

1.继承时 : 还需添加修饰符

class B : public A{};

**struct，class，interface之间的关系**

①C++中的struct实际上是可以直接操作成员，无函数的class，而class默认下所有成员，方法访问修饰符都是private。两者都是类的概念，只是访问权限不同

②interface和class，class中有具体方法的实现，而interface中只是带有virtual关键字修饰的函数声明。另外接口中的所有声明未实现的方法必须是public修饰的，这一点与struct不谋而合，故可以在struct 中声明虚函数，从而将class转换为一个接口。不仅如此，C++的多重继承此时也就是自然而然的继承多个接口了。

抽象程度：低→高

Class → abstract class → interface

1. C++中同样如果继承接口，那么就必须实现所有接口中声明的方法。而Java中extend和implement关键字反而有一点混淆了class和interface
2. 创建对象 class A

Java A a = new A( );

C++ ①A a;

②A\* a = new A( );

③A\* a = new A;

含有纯虚函数的抽象类，是不能实例化对象的。

1. 随机数

srand(unsigned(time(0));

int rand()%N, [0,n)

1. 多态

C++多态的实现依赖于，关键字virtual 指针和引用

前提book ：public material 都有check\_in（）方法

如下

①

book b;

material& m = b;

m.check\_in();//实现多态

②

book\* b = new book();

material\* m = b;

m->check\_in()//实现多态

ps:动态绑定只有在指针或引用调用虚函数时，才会发生

1. 数组特性

①不允许将一个数组赋值到另一个数组（不能进行值传递））

②使用数组名，其实是在使用指向第一个元素的指针

让函数返回数组：

①返回数组指针

②返回数组引用

1. 指针数组 和 指向数组的指针

①int\* pi[10];//从右往左看 元素类型为整型指针←数组名为pi←10个元素数组

②int (\*pi)[10];//先看括号内的内容 一个指针←指向10个元素的数组→整型

1. inline

作用：用于修饰函数，减少开销（调用函数时，保存寄存器，函数调用结束恢复）

inline string make\_plural(size\_t ctr, const string &word, const string &ending = "s"){

return ctr > 1 ? word + ending : word;

}

具体使用可以在类内函数声明时，也可以在类外函数定义时

在类中声明并定义一个函数，该函数是隐式内联。当然也可以在类外部定义函数时用inline关键字修饰添加内联

1. 函数指针

bool (\*pF) (const string&,const string&);//声明一个指向特定函数（两个形参，返回值为bool类型）的指针

bool\* pf (const string,const string&);//一个返回bool类型指针的函数

使用：

bool b1 = lengthCompare("hello", "lala");//直接调用

bool b2 = (\*fp)("hello", "lala");//对指针进行解引用

bool b3 = fp("hello", "lala");// 等价

1. 传递函数

void useBigger(const string &s1, const string &s2,

bool fp(const string &,const string &));//将函数作为形参传入

void useBigger(const string &s1, const string &s2,

bool(\*fp)(const string &, const string &));//传递函数指针

常量指针： int \*const p；

指向常量的指针： const int\* p；

指向常量的常量指针： const int \*const p;

sizeof(类型，struct) 获取的是个类型或结构体，所占内存的的大小。单位为Byte，若指向该类型的指针为 \*pbuffer。

memcpy（dest\*，src\*，sizeof（……））；//将从src\*所指内存地址起，将sizeof字节复制到dest

1. static ：局部静态成员，类的静态成员 都存在栈中，用static关键字修饰的变量在使用之前已分配，直至程序结束才销毁。

类的静态成员一般不再类内进行初始化

private:

static constexpr int period = 30 //在类内部将period初始化为一个常量表达式

class Account {

private:

//每个类的实例即对象都有的成员

string owner;

double amount;

//每个对象共享的类的静态成员

static double interestRate;

static double initRate(double d) {

return d;

}

static Account a;//不完全类型的静态成员

Account\* a2;//不完全的指针类型

//Account a3;//普通数据成员不允许是不完全类型

public:

void caclulate() {

amount += amount \* interestRate;

}

static double rate() {

return interestRate;

}

static void rate(double d) {

interestRate = d;

}

};

//类外初始化静态成员

double Account::interestRate = initRate(6.5);

void main() {

/\*shared\_ptr<string> pl;

pl = make\_shared<string>("fjdlfj");

cout << (\*pl) << endl;\*/

Account::rate(3.9);

double d = Account::rate();

Account a;

Account\* a2 = &a;

a.rate();

a2->rate();

system("pause");

}

OOP

15.

this

默认是一个类类型的常量指针

将this声明为指向常量的指针

const int \*const this；

Data\_sales func（）const {

return this->isbn;

}

1. 编译器分两步处理类：
   1. 首先编译成员
   2. 然后函数
2. 构造函数

//constructor

Sales\_data() = default;//显式声明默认构造函数

//类似Java的赋值

Sales\_data(const string& bookNo) {

this->bookNo = bookNo;

}

//C++中的初始化

Sales\_data(const string& bookNo) : bookNo(bookNo) {}

: bookNo(bookNo) 构造函数初始初始化（这里即为初始化过程）

C++中构造函数在执行函数体时，初始化已经完成了。此时只能进行赋值操作。

这里就涉及到了 初始化 和 赋值 的区别

1. 在类作用域外面定义方法时，需使用类名加作用域运算符

Sales\_data::Sales\_data();

Sales\_data::newFunc();

1. class 与 struct

class 默认所有成员方法访问修饰符为private：

struct public：

1. 友元

class Sales\_data {

friend ostream& print(ostream&, const Sales\_data&);//将print函数作为该类的友元

……

……

};

ostream& print(ostream&, const Sales\_data&);//声明

令其他其他类作为友元 friend Windows\_mgr;

令其他类某个成员函数作为友元 friend Windows\_mgr::clear();

1. 清空数组的快捷方式

由于数组存储特性（内存地址连续）

memset（数组名，0，个数）

1. 类的静态成员 5.25 17

类的对象成员存在于任何对象之外，对象中不包含任何与静态成员有关的数据

使用：

①Account：：rate（）；

虽然该静态方法不属于任何一个对象，但仍然可以使用对象调用

②account.rate();

③account2->rate();//account2为指向Account对象的指针

注意：

带有类内初始值设定项的成员必须为常量

1. new

C++ ①new A;

②new A();

都返回一个A对象的内存地址，

若有构造函数①②都分配内存地址，并根据具体构造函数初始化

无 ①只分配内存地址②进行默认初始化

Java new A();

STL

1. IO无拷贝或赋值

使用fstream对文件进行读写之后，需对当前的fstream（ifstream，ofstream）关闭，才能对下一文件进行读写

1. 顺序容器

vector：

vector<int> v1;//默认构造初始化,vector中为空

vector<int> v2(10);//10个值初始化元素

vector<int> v3(10, 1);//10个特定元素

vector<int> v4 = { 12,34 };//列表初始化

vector<int> v5(v4);//拷贝初始化

vector<int> v6(v5.begin(), v5.end());//迭代器范围初始化

注：vector虽然支持动态扩容，但也只是在容器元素末尾的添加操作。对没有初始化（或容器元素个数为0）的vector使用索引运算符[]会引发vector subscript out of range异常

vector<int> vi;

vi[0] = 1;//这样将引发边界异常

insert将一个vector添加到另一个同类型vector的尾部

列表初始化：

vector<int> vi(10, -1);//初始化为10个-1

list<string> lists(10, "hi");//同理

//当不指定初始化的特定元素时，容器的元素值为值初始化的值

forward\_list<int> fli(10);

deque<string> ds(10);

对vector使用索引运算符时，若索引符对应位置无具体数据。那么将引发越界错误

注意：只有顺序容器支持在构造函数中制定大小

数组的初始化：

array<int, 10> arr;//在声明容器元素类型的同时还需要制定容器大小

数组类型：

array<string, 5>::size\_type arrayT;

与内置数组：

#include<array>支持赋值和拷贝操作

array<int, 5>ia2 = { 1,2,3,4,5 };//列表初始化

array<int, 5>newArr;

newArr = ia2;

对容器的操作可能会引起迭代器失效，故每次操作后还需更新迭代器，

如果在一个循环中插入/删除deque,vector,string中，不要保存end()返回的迭代器，而应在每次操作后重新获取。因为在一次操作后，迭代器失效。

string:

string str(char[]);//以一个字符数组来构造初始化string对象，行为是未定义的

1. 泛型算法

①搜索

容器： find(container.begin(),container.end(),value);

内置数组： int a[10] = { 1,2,3,4,5,6,7,8,9,10 };

int\* result = find(begin(a), end(a), 5);

②计算

accumulate(container.begin(),container.end(),0);//和的初始值为0

③ copy copy(li.begin(), li.end(), vi.begin());//这里vi的大小必须>=li大小，不然会报异常

vector<int> vi;

vi.reserve(10);//虽然给vi预分配了10个大小，但此时仍然是空的

fill\_n(vi.begin(), 10, 9);

④sort & unique

vector<string> vs = { "the","quick","red","fox","jump","over","the","slow","red","turtle" };

sort(vs.begin(), vs.end());

auto iter = unique(vs.begin(), vs.end());//unique()返回最后一个不重复元素之后的元素

vs.erase(iter, vs.end());//将之后的重复元素删除

1. 使用谓词

stable\_sort(vs.begin(), vs.end(), isShorter);//先按长度排序，若长度相同再按首字母顺序排序

1. lambda表达式

形式：[capture list] (parameter list) -> return type {function body}

可为[ ] 可为空 可为空

可调用对象

函数

函数指针

重载了函数调用运算符的类

lambda

<https://sword/svn/server_mahjong>

使用方式：

①int i = 3;

auto iter = find\_if(vs.begin(), vs.end(),

[i](const string& s) {return s.size() > i; });

//找到vs中第一个长度>3的 string

②for\_each(iter, vs.end(),

[](const string& s) {cout << s << endl; });//再次使用lambda

Template program

C++ 所有标准容器都定义了 != 运算符，用 != 比 < 运算符更好

1. initializer\_list<T>

auto al = {1,2,3,4} //此时al的类型就是一个 initializer\_list<int>

这是一个通用容器，可以在构造函数中作为参数传入初始化具体类型的容器成员

template<class T>

class S

{

public:

std::vector<T> v;

S(std::initializer\_list<T> l) : v(l) {

std::cout << "constructed with a << l.size() << " - element list\n";

}

};

其他

#pragma pack(4) 可设置struct的对齐方式。//VS中默认对齐字节数min（8，struct中最大变量字节数）

对齐规则：（以4字节为例）

①按4字节数对齐，若当前类型没把4字节占满。则作一定偏移，保证下一类型的偏移量是该类型所在字节数的整数倍。

如：

#pragma pack（4）

struct one{

char c；//1

short s；//2

float f；//4

}

该struct内存中变量的存储方式为：

c \_ s s

f f f f

//注：第二个空出来的字节，是由于类型的偏移量必须为所在字节数的整数倍所造成的,这里的偏移量是相对于第一个字节。例如char c就从第一个字节开始故偏移量为0

字节，有符号，无符号数

1个字节是8位二进制数 \_ \_ \_ \_ \_ \_ \_ \_

C++ 里的各数据类型，分别占多少字节,此时能表示的数范围与字节数有关

如 unsinged short 范围： [-2^(2\*8-1）,-1] U [0,2^(2\*8-1)]

short [0,2^(2\*8)-1]

原码码，反码，补码

规定：正数的反码，补码与原码相同

负数：符号位不变，反码，补码

8位有符号数1111 1111 -127

反码为1000 0000

补码为1000 0001

由于+128的二进制，表示为1000 0000，正数的补码就是响应的负数，

即 -128 为1000 0000

因此在有符号数具体为下：[-127 –1],[0,127],[-128]这里的-128单独用1000 0000表示

有符号数的转换过程：

不断加1

整数0~127 即 00000000~01111111

到达127后继续加1

得：10000000~11111111 -128~-1

链接过程：

初始文件：

main.c

int sum(int\* a,int n);

int array[2] = {1,2};

int main(){

int val = sum(array,2);

return val;

}

sum.c

int sum(int\* a,int n){

int I;

int s = 0;

for(int =0;i<n;i++){

s += a[i];

}

return s;

}

Linux(Ubuntu环境)

1. 预编译器将C文件转为中间文件（ASCII）

cpp main.c main.i

cpp sum.c sum.i

1. C编译器将中间文件转为汇编文件（ASCII）

gcc –S main.i -o main.s

gcc –S sum.i -o sum.s

没找到ccl 编译器，用gcc代替

1. 汇编器将汇编文件转为可重定位目标文件

as main.s -o main.o

as sum.s -o sum.o

1. 链接器将目标文件链接为一个可执行文件mainsum

gcc main.o sum.o -o mainsum

1. 运行

./mainsum

C++ primer第十三章

拷贝控制

控制初始化：

默认构造函数

拷贝构造函数

控制赋值操作：

拷贝赋值运算符

赋值运算符本质就是一个名为 operator= 的函数，进行赋值操作其实就是在调用该函数