quiz 3个满90的同学，还是非常厉害的

may lead to 是为了严谨 A. n = 0, B vec 为空都是没有undefined behavior的

首先强调一点,vector 所有操作，除了at以外，都是不检查越界的（原因=>为了效率，快，否则一堆if，跳转，异常处理...绝对没有直接读内存来的快）

eg. 在空vector上去访问front，back，pop_back...都会导致undefined behaviors

编译器直接假定你所有的操作都是合法的，提升效率

B. 可能回觉得他是一个无限循环，因为一直在往里加元素，但实际并不是

push_back reallocate，可能会导致元素搬家，使得指向vector的指针或引用失效（指向原来的地方，原来的地方失效）

range based for loop本质是使用迭代器->实质仍然是指针或引用，只是做了一层包装(以后课上会讲)

所以有可能会使迭代器失效，指向不存在的位置->undefined behavior

C. 默认构造函数，空串

D. free/delete/delete []空指针都是无害的

delete 1.析构函数 2.内存释放函数，释放这只指针的内存 但空指针的话永远不会做任何事

3.只能出现在等号右边的是rvalue，其余为lvalue

前置递增/递减++i是左值，后置为右值

CD先解引用在取地址/先取地址再解引用其实就是迷惑作用，无论我们是解引用一个地址，还是解引用一个指针，我们都会得到存在于那片内存上的对象，所以解引用得到的一定是个对象，为左值（除非你重载了运算符）；取地址一定得到的是右值

EF完全等价。F：解引用，一定是左值

4.默认构造函数=default，移动构造函数=default

析构函数有输出

其实我们第一版的quiz里是没有default的，我们可以根据5法则推断出这两个是default的

首先我们可以在析构函数里调用这个类的成员的，所以成员的销毁一定是在执行这个函数体之后，否则就不可能访问类成员

C.编译器会默认生成一个形如C选项的移动构造函数

这里std::move不能省掉。 我们上节课也说过，右值引用其实是左值

所以s(other.s)这里就是使用的string的拷贝构造函数，而不是移动构造函数，所以是不能省略的

5.拷贝赋值运算符什么时候使用。

我们已经说明了他有拷贝赋值运算符合移动赋值运算符(如果不说的话，我们就不知道右值究竟是被移动还是被拷贝)

拷贝左值，移动右值

A.返回函数内部的result，并且是by value返回的，所以得到的是右值=》移动赋值给a

B.=不是赋值运算符，=就是初始化用的一个语法，是在用拷贝构造函数

C.指针的赋值和我们类型的赋值没有任何关系，就是普通的赋值运算

BD的区别就在于我们是不是在声明这个实例，生命的话就是普通的一个初始化，否则就是赋值

6.const成员函数

A.只能在const的对象上调用，这是肯定不对的，应该这样说：

在const的对象上只能调用const的成员函数，不能调用nonconst成员函数

抓住一个原则，就是底层const不能被去除

const成员函数，是向this指针添加底层cosnt，然后所有经由this访问的成员，都是带有const的成员

B. 在const成员函数里面调用一个nonconst成员函数，其实就相当于要去除this指针上带有的底层const，这是不可以的，除非我们用const_cast绕过这个机制

C.const成员的this指针是cosnt X*,我们说过了这是在this上添加底层const

D.是不是听起来听像对的，其实一个函数是否有const，不是他是否修改成员决定的，要从他的意义上考虑。

如果说这个成员函数在调用的前后对于用户来说，对象的状态没有发生改变，否则他是const，不然不带const。

待会我们会看一个具体的例子，就是看起来没有修改任何东西，但是他不能是const

7.定义数组是说调用他的默认构造函数的

new不给初始化器，调用的就是默认构造

C.其实我们调用了构造函数，只不过这个函数没有传入任何参数，这≠默认构造函数

D.传参，要看调用他的时候是怎么写的，你要写成fun({})，那就有可能是默认构造函数的

这取决于传参是的写法

8.让你写var的类型，而不是编译器会用什么类型代替auto

*放在这里只是为了给编译器一个提示，让编译器帮我们推断他的类型

就算不给*，*编译器也会识别成int *，*

3.14f->float， 浮点型字面值默认是double

这里的命名其实已经给我们提示了

vs： vector《string》引用，遍历第一个vector

var遍历内层的vector

std不写不扣分，但推荐写上

int,double内置类型(居然有人写std::int)

Folder 里有一个容器保存所有的message

现在大家学过的容器有vector/数组...

在这个问题里比较合适的一个容器是一个叫set的容器

不希望拷贝，那么我们容器里存的不应该是message本身，而应该是message的

指针

message.h

首先我们message里存的东西使用string来存的

然后我们include set， cs101: set用平衡树/红黑树？来存的

消息类 m_content表示消息的内容，set表示消息存在于哪些folder之中

放folder指针是因为我们不希望每次都拷贝folder的内容

然后我们还有一个folder类，表示他存了哪些message

我们编译一下，发现他报错了，这是因为他编译器编译的时候遇到了folder，他只会向上去查找

所以我们需要在前面对folder做一个声明，不然编译器不知道folder是什么。

所以我们在前面**声明**一个folder类，而不去定义他

不定义的话就是incomplete type，我们可以使用他的指针/引用,但不能使用他的成员

如果我们希望右值被移动，加上std::move, by value的传进来，这样在传参的过程中，如果传递的实参是左值，就会拷贝初始化，如果是右值，就会移动初始化

="""相当于把默认构造函数也给写了

先把message搞过来

incomplete type CE

因为我们folder没有定义，只有声明

有人说我们把folder放到前面是不是就可以了

其实是一样的，message和folder是互相嵌套的，谁放在前面都没有用

所以我们要把AddToAll的定义放到folder的定义之后

#define在编译过程中完成替换

但我们尽量不要用define来定义函数

#define sqr(a,b) sqrt(a* *a+b* * b)

为什么会出现多次#include<>

我们肯定不会闲的没事连续include两个相同的东西

但我们每个文件里，都可能有iostream/cstdio等来过编译=》需要头文件的保护

sum.hpp中，我们只声明函数，甚至可以删除int a, int b的a和b

通常我们.hpp中进行类及其函数的声明，在.cpp中定义

我们运行一下发现，他是不能直接运行的

然后看我们的代码，include了hpp，有了声明，我们可以发现他是可以编译的

g++ sum.cpp a.cpp -o a1

这里我们要手动给他加上和sum.cpp一起编译

---

g++ -shared sum.cpp -o libsum.so

ls 一下文件夹中多了个libsum.so

g++ a.cpp -o a2 -Wl,-rpath . -L. -lsum

-Wl,-rpath .: 链接到libsum.so

首先让他知道在哪->当前目 "."(上层目录是..)

-lsum: 链接到libsum.so

./a2

这件事情告诉我们，在没有main函数的这个文件，但我们定义了一个函数，我们可以把这个文件单独编译出来，然后给他链接到我们想要的文件中

libsum.so so:shared object

动态链接库

windows下通常为dll(dynamic link library)

动态链接库有什么好处呢：c++可能会编译的非常非常慢

这点你们在把代码提交到oj上的时候可能有所感受，他会编译非常非常长的时间

以后我们上专业课的时候，编译大项目也会编译非常长的时间

内联函数

我们不用define处理函数，但我们希望简单函数在编译的时候直接带入

来减少调用函数带来的时间空间的开销

这样他其实可以让编译器展开这个函数，还没有define那样需要注意的问题

但这里的inline只是向编译器提出一个请求，编译器可以选择不展开

eg. 递归函数无法内敛展开/函数非常复杂，编译器可以拒绝请求，但不报错

所以我们可以说define一个很好的代替者就是inline function

提醒大家复习一下，五一回来不久可能我们就会期中考试了

然后大家可能不到做作业的时候是不会复习的

下学期cs101每周1quiz，1homework，3周一次pa