# 头文件：

* < iomanip>
* srand((unsigned)time(NULL))
* <initializer\_list>
* initializer\_list<T>
* <limits>
* (std::numeric\_limits<T>::max)()
* (std::numeric\_limits<T>::max)()

# 关于默认构造函数

* vector<A>(0);这个**不会**调用A的默认构造函数！（如果A没有默认构造函数，会出现"尝试引用被删除的函数"的错误）
* vector<A>(1);这个**会**调用A的默认构造函数！
* vector<A>();**不会**调用A的默认构造函数！（如果A没有默认构造函数，也不会出现"尝试引用被删除的函数"的错误）
* 调用默认构造函数的方法 vector<A> v; 或者显式调用 vector<A> v=vector<A>(); 这两者都不会调用A的默认构造函数，也不会出现"尝试引用被删除的函数"的错误
* 当调用vector<T>的默认构造函数时，vector<T>的默认构造函数不会调用类型T的默认构造函数

# 若类类型b含有为类类型a的数据成员

如果对b进行默认初始化 若b的数据成员a1含有类内初始值，那么用该初始值初始化a1,否则调用类类型a的默认构造函数

# 指针的数组如何定义（动态数组）

(int\*)\* a;首先a是一个指针，指针指向的元素是int\*

# extern：

* 当头文件a.h中定义了一个变量，并在b.cpp,c.cpp中引入了该头文件，那么会产生重复定义的错误（无论是否使用宏来避免重复声明）。解决方案：在a.h中在变量前添加extern，将其变为实例化声明，这样使用宏就能避免重复的声明，然后在任一个cpp中对其进行定义，cpp文件只要包含该声明变量的头文件即可
* 也可以在一个cpp中定义，其余cpp文件用extern进行声明

6、嵌套try语句，当内层try的代码抛出异常且被内层try的catch捕获后，外层的try会继续执行剩余语句

int main() {

STACK<int> S(1);

try {

if (true){

try {

S.PUSH(1);

S.PUSH(2);

S.PUSH(3);

}

catch (out\_of\_range error) {

cout << "Inside:" << error.what() << endl;

}

}

cout << "here" << endl;

}

catch (out\_of\_range error) {

cout << error.what() << endl;

}

catch (logic\_error error) {

cout << error.what() << endl;

}

system("pause");//定义在iostream中

return 1;

}

# 含有动态指针的类，要自定义析构函数以及其他五个函数

# delete销毁一个对象，delete[]销毁一个序列。

* 对于基本类型的数组来说，两者没什么区别
* 如果数组包含的是类的对象指针，则两者有区别。delete只会引起第一个元素析构，而delete[]会依序调用所有元素的析构

# 头文件互相包含引发的大量未声明错误

//a.h

#pragma once

#include "b.h"

class A{

};

//b.h

#pragma once

//加上这句"class A;"

#include "a.h"

class B{

A a;

};

//main.cpp

include "b.h"

int main(){

return -1;

}

三个文件如下，类型B中包含有A的对象，因此需要包含A对象的头文件。

但是a.h中引入了b.h。在编译时，相当于把class B的定义放到了class A之前，因此class B中类型A是未定义的。

解决方法是：加上标记为红色的这句，在B定义前，前置声明一下A即可。或者直接删掉a.h文件中的#include "b.h"，头文件相互包含本身就不合理！！！