这部分包含部分CPP的基础知识，是需要背诵的八股文，到时候看情况随机应变即可。

全局静态变量在声明文件之外是不可见的，作用域是：定义之处-》文件结尾。

局部静态变量作用域是局部，当局部退出后，不会销毁，当下一次声明并且使用时，数值保持不变。

静态函数：仅可在本cpp内使用，别的地方是不可行的。

静态成员：共享数据，多个对象共享数据。

类的静态函数：不需要对象名，实现中不能直接引用类里面的非静态成员。类名：：<静态成员函数名><参数表>

四种cast转换，强制类型转换。

const_cast 将const转为非const

static_cast 隐式转换 void*转指针、非const转const

dynamic_cast 动态类型转换，含有虚函数的类实现多态的核心原理。

reinterpret_cast 强制转换，容易出错，什么都能转。

有const指针，但是没有const引用

C++11的智能指针

智能指针可以只管理一个指针。当超出了类的作用域时，会自动释放内存空间。

auto_ptr

unique_ptr 仅允许一个指向

```
unique_ptr<string> pu1(new string ("hello world!"));
```

shared_ptr 共享指针？使用计数器来表明资源被几个指针共享。

shared_ptr . use_count() 可以查看资源的所有者个数

两个shared_ptr互相引用会引发死锁问题。

weak_ptr 指向shared_ptr时，不会增加个数。

函数指针：

典型的看一下代码例子即可

```
char * fun(char * p)  {…}    // 函数fun
char * (*pf)(char * p);       // 函数指针pf
pf = fun;             // 函数指针pf指向函数fun
pf(p);              // 通过函数指针pf调用函数fun
```

多态：

静态多态：主要是重载，在编译时已经确定了。

动态多态：主要是虚函数，指针会通过虚函数表，指向实际实现的函数。也就实现了子类的多态。

C++默认栈的最大值：1M

C++调用C需要 extern C

1、new分配内存按照数据类型进行分配，malloc分配内存按照指定的大小分配；

2、new返回的是指定对象的指针，而malloc返回的是void*，因此malloc的返回值一般都需要进行类型转化。

3、new不仅分配一段内存，而且会调用构造函数，malloc不会。

4、new分配的内存要用delete销毁，malloc要用free来销毁；delete销毁的时候会调用对象的析构函数，而free则不会。

5、new是一个操作符可以重载，malloc是一个库函数。

6、malloc分配的内存不够的时候，可以用realloc扩容。扩容的原理？new没用这样操作。

7、new如果分配失败了会抛出bad_malloc的异常，而malloc失败了会返回NULL。

8、申请数组时： new[]一次分配所有内存，多次调用构造函数，搭配使用delete[]，d
多次调用析构函数，销毁数组中的每个对象。而malloc则只能sizeof(int) * n。

函数调用时，会给他分配函数栈，专门用于存储函数的各种参数。

昨天看cpp的时候，头有点晕，就没有完全看明白，我直接裂开。

静态函数要引用非静态成员时，可以通过对象来引用。这一点在现在是非常明确的。

面经一下子背太多，背的人头有点晕，八股文。

一、C++四种类型转换

C++支持C风格的类型转换，但是C风格的转换太过随意，指针可以任意指向，且没有统一的关键字和标识符，在大型系统中，不利于代码排查。

static_cast(target value) 这个是强制类型转换的风格。
还有一种转换是dynamic_cast 运行时转换，这二者的区别是什么呢?

1.static_cast 转换 静态转换

用于类结构中基类和派生类之间的各种转换。

上行转换：派生类-》基类 安全的

下行转换：缺少动态检查，不安全的

用于基本数据类型之间的转换，由int转换为char，安全性问题由程序员保证。

用于空指针-》目标类型的指针

把任何表达式转为void类型

主要用于执行非多态的转换操作，用于代替C语言的各种基本的转换过程。隐式转换建议使用static_cast进行表明和替换。

void*指针?

2.dynamic_cast强制类型转换

只有在派生类之间相互转换时，才会使用dynamic_cast进行运行时检查，需要有虚函数表里面的信息。

只有一个类定义了虚函数，才会有虚函数表。

基类的子类也是有虚函数表的，所以可以使用dynamic_cast强制转换。

转换时，只有真实指向目标指针，才会转换成功的。

否则，会返回null或者抛出异常。

3.const_cast转换

常量指针-》非常量指针

常量引用-》非常量引用

4.reinterpret_cast转换

从底层对数据重新解释，可以进行肆无忌惮的转换。高危操作。

二、著名的字符串的区别

```
const char * arr = "123"  字符串123保存在常量区，由于在常量区，本来就不能修改，加不加const是一样的
char * brr = "123"   字符串123保存在常量区，同样无法修改
const char crr[] = "123" 这里的123本来是在栈上，由于const，可能会放在常量区
char drr[] = "123"  临时变量，保存在栈区，可以通过指针去修改，这一点总是固定的
```

定义的常量必须初始化。

局部变量，常量放在栈区。

全局对象，常量放在全局、静态存储区。

字面值常量，比如一串硬编码字符串，常常放在常量存储区。

const保证不会对对象作出任何的修改，如果确认不会被更改，就应该加上const，这样无论什么情况都可以。

同时定义两个函数，一个带const，一个不带cosnt，参数是const或者不是const，均等价于函数重载。

虚函数表如何实现运行时多态?

子类重写父类虚函数，在虚函数表中，函数地址会被替换。通过本机制实现了多态，还算是比较精妙的。

C语言压栈顺序：先压参数，再压参数个数 argc argument count

argument value

三、STL容器与算法

1.map和set区别

均是C++的关联容器，底层实现均是红黑树（RB-Tree）Red Black Tree 红黑树

map和set转调用红黑树的接口直接实现的。

map中的元素是key-value对，键值对，关键字起到索引作用

set中迭代器是const的，不允许修改元素的值。map允许修改value的。

如果find能解决问题，应该尽量使用find函数，这一点是明确的。

2.STL的allocator

STL的分配器用于封装STL容器在内存管理上的底层细节。

new：调用operator new 配置内存 调用构造函数构造内容

delete：调用析构函数 调用：：operator delete释放内存

STL将这两个部分区分开来，内存配置：allocate负责。

3.STL迭代器删除元素

1.对于序列容器vector、deque来说，使用erase(itertor)以后，后边的每个元素都会失效，但是都会往前移动一个位置，之后会返回下一个有效的迭代器。

2.对于关联容器map、set来说，删除一个元素，不会影响下一个元素的迭代器。

3.对于list，erase也会返回下一个有效的iterator

4.STL的基本组成

容器、迭代器、仿函数、算法、分配器、适配器。

分配器allocator分配存储空间，算法通过迭代器获取容器内容，仿函数协助算法完成操作，适配器用来套接适配仿函数。

5.map和unordered_map的种种操作?

map：红黑树 键值对 所有元素会根据元素的键值自动排序，不允许键值重复。

适用于有序键值对不重复映射。

multimap：允许键值重复

6.STL迭代器的作用

Iterator迭代器 又叫做Cursor模式，是一种设计模式。

提供一种方法顺序访问一个聚合对象的内部的各个元素，但是又不暴露该对象的内部表示。

resize改变当前含有的数量，reserve(len)确定这个容器允许放入多少个对象。

BSS segment，专门用于存放未初始化的全局变量和静态变量。

我真的是直接醉了。