待解决问题：

1. 对象相加，P19
2. 养成命名习惯，看库函数方法
3. unique\_ptr
4. 总结文件的空白（空格、tab等）、换行、文件结束符号，方便操作文件
5. 模板和类的区别？模板里元素必须同一类型？
6. 标准库类型和内置的差别
7. const深入理解
8. 注意容器，智能指针，多线程
9. return vector<int>（）形式
10. 实例名（参数），调用operator（）类函数
11. 常用类和函数：<https://www.runoob.com/cplusplus/cpp-tutorial.html>

set、map、list：

<https://www.jianshu.com/p/3cd0c9c31519>

* C++标准模板库

C++ STL（标准模板库）是一套功能强大的 C++ 模板类，提供了通用的模板类和函数（一般函数和类函数），这些模板类和函数可以实现多种流行和常用的算法和数据结构，如向量、链表、队列、栈。

C++ 标准模板库的核心包括以下三个组件：模板和类的区别，对于类模板，实例化时，需要在模板后跟一对尖括号，在括号内放上信息表示到底实例化什么类型，如vector<int> ivec;

* 容器（Containers）：容器是用来管理某一类对象的集合。C++ 提供了各种不同类型的容器，比如 deque、list、vector、map 等。
* 算法（Algorithms）：算法作用于容器（类函数）。它们提供了执行各种操作的方式，包括对容器内容执行初始化、排序、搜索和转换等操作。
* 迭代器（iterators）：P97迭代器用于遍历对象集合的元素。这些集合可能是容器，也可能是容器的子集。有三种不同含义：可能时迭代器概念本身，也可能是指容器定义的迭代器类型，还可能是指迭代器对象。类似于指针类型，所有标准库容器都可以使用迭代器，但是只有少数的几种支持下标运算符（像数组一样）。vector和string都支持下标运算和迭代器，string不是容器，支持很多容器相关的操作（push\_back）等。操作迭代器有点类似于指针。
* 适配器：P329,三个顺序容器适配器：stack、queue、priority\_queue。一个容器适配器接受一种已有的容器类型，使其行为看起来像一种不同的类型。
* 泛型算法：P335，标准库并没有给每个容器添加大量功能，而是提供了一组算法，这些算法中的大多数都独立于任何特定容器，对迭代器而不是对容器进行操作，不能（直接）添加或删除元素。标准库提供超过100种算法，重点是理解，见P770-附录A.2。

综上，迭代器是容器的一种操作。

* 迭代器(iterator)常见函数：标准迭代器运算符-P96；迭代器支持的运算-P99

容器使用迭代器时要定义，如：vector<int>::iterator 实例= vector<int>实例.begin();iterator能读写vector<int>实例的元素，const\_iterator只能读vector<int>实例的元素，不能写

1. begin(),end()：const\_iterator
2. iterator实例 ->first：vector<int>实例的索引
3. iterator实例 ->second：vector<int>实例的内容（元素）
4. 迭代器运算符：\*(取引用)、
5. iterator：普通迭代器，类始于指向容器中不同元素的指针。
6. const\_iterator：常普通迭代器，指向固定。
7. 插入迭代器（insert）：绑定再容器上可用来向容器插入元素
8. 流迭代器（stream）：绑定在输入或输出流上，可用来遍历所关联的IO流
9. 反向迭代器（reverse）：这些迭代器是向后而不是向前移动的。
10. 移动迭代器（move）：不是拷贝其中的元素，而是移动它们。

* 容器通用操作（函数）：

1. 实例.empty()
2. 实例.size()
3. 实例.push\_back(t)
4. 实例.front(),实例.back()：访问首元素和最后的元素。

顺序容器：按它们在容器中的位置来顺序访问和保存的。P292, 定义和初始化-P299、P87;容器赋值运算-P302；容器操作-P295，P91;访问元素-P310；删除元素-P311；改变容器大小-P314；容器大小管理操作-P318；额外的string操作-P320

1. Vector：p86，可变大小数组。支持快速随机访问，在尾部之外的位置插入或删除元素可能很慢。支持下标运算。vector是一个类模板，也称为容器，表示对象的集合，其中所有对象类型都一样，如：可用{}做列表初始化，允许用push\_back动态添加元素至尾部
2. deque：双端队列。支持快速随机访问。在头尾位置插入/删除速度很快
3. list：双向链表。只支持双向顺序访问。在list中的任何位置进行插入/删除操作速度都很快
4. forward\_list：单向链表。只支持单向顺序访问。在链表的任何位置进行插入/删除操作速度都很快。
5. array：固定大小数组。支持快速随机访问。不能添加或删除元素。
6. string：与vector相似的容器，但专门用于保存字符。随机访问快。在尾部插入/删除快。

关联容器：P373，按关键字访问和保存的。Key中存放元素得地址？

哈希函数：哈希的过程中需要使用哈希函数进行计算。哈希函数是一种映射关系，根据数据的关键词 key （源程序中变量名），通过一定的函数关系，计算出该元素存储位置（生成目标文件在存储中的地址）的函数。

表示为：

address = H [key]

<https://blog.csdn.net/u012835097/article/details/79407591>

支持P295表9.2普通容器操作和类型别名，不支持顺序容器的位置相关的操作。

1. 实例.first：关键字
2. 实例.second：值
3. 按关键字作下标读取值：value=实例[key]
4. value\_type：map的值类型，如set<string>：：value\_type v1。
5. key\_type：关键字类型，关键字是const，不能更改，为只读
6. map：又称关联数组；保存关键字-值（元素）对，值类型（value\_type）为pair。类似于字典，可以像数组一样使用，其下标可以不是int了，另外，一些操作和pair相似。Map.count（key），可以查找是否有下标。可以有4个参数，有2个默认参数，比较参数，一般为key，allocator-pair。
7. set：关键字即值，即保存关键字的容器。每个元素只包含一个关键字。

<https://www.cnblogs.com/zyxStar/p/4542835.html>

1. count(a)：返回a的个数
2. mutimap：关键字可重复出现的map
3. mutiset：关键字可重复出现的set

无序集合：如果关键字类型固有就是无序的，或者性能测试发现问题可以用哈希技术解决，就可以使用无序容器。无序容器的性能依赖于哈希函数的质量和桶的数量和大小。默认情况下，无序容器使用关键字类型的==运算符来比较元素，它们还使用一个hash<key\_type>类型的对象来生成每个元素的哈希值。

1. unordered\_map：用哈希函数组织的map
2. unordered\_set：用哈希函数组织的set
3. unordered\_mutimap：哈希组织的map,关键字可以重复出现
4. unordered\_mutiset：哈希组织的set,关键字可以重复出现
5. pair：p379,是标准库类型。pair操作P380表11.2。pair <int,int> x,x.first即第一个元素，x.second即第二个元素，make\_pair(x[2])将其索引和内容配对

make\_pair:数组对之类，

<https://blog.csdn.net/u011499425/article/details/52756088>

* 泛型算法：分类-P365表10.5，优先使用容器成员函数，即容器特有的操作，再使用通用操作，一般来说，特有操作更快P369

1. P344定制操作lambda：替代简单的==或<比较条件，改变排序规则。短小、一两个地方使用时用，类似于内联函数。
2. fill：填充数值，

<https://blog.csdn.net/cv_jason/article/details/80894886>

1. reshape：

<https://blog.csdn.net/u010916338/article/details/84066369>

1. std库：
2. chrono：计时，ORBSLAM2中
3. 自己总结

* 定义一个变量，C++的变量名是让编译器识别的，如int a；定义了int类的对象a，那么编译器编译时会生成为a开辟相应字节数地址（相对地址，起始地址用于寻址）b存放a的值（对象）
* 引用指向同一个变量，编译器编译后就是同一个变量；指针有自己的地址，在该地址保存了变量的地址。
* P60类型别名：typedef、using
* P102,105,数组的两个特殊性质：不允许拷贝和赋值；使用数组时编译器一般会把它转换成指针。
* 确定不需要改变变量值时，应将其变为const，防止误操作。
* P57顶层const（指针本身，int const \*p1）和底层const（所指对象，const int \*p2）概念。引用变量绑定（等号前绑定到等号后）在非引用变量上。
* P213调用函数一般比求等价表达式的值要慢一些，调用函数有压栈等额外开销，内联函数（inline）可避免函数调用的开销，类似于头文件中的宏定义函数。定义在类内（声明和定义都在class{}内）是隐式的inline函数。
* 初始化方式：默认初始化P40；直接初始化方式P76；值初始化P88
* P49预处理器是运行于编译过程（编译器）之前的一段程序，如#ifdef #ifndef，#开头的，assert(用于调试特别好，NDEBUG相当于它的开关)-P215，
* P68预处理变量：#define定义的
* P220不能把普通引用绑定到const对象上，但可以把const引用绑定到普通对象上
* P222不能定义函数类型和数组类型的形参或返回值，但形参或返回值可以是指向数组或函数的指针
* P247拷贝的是副本，占用了新的内存，引用和指针内容不占用新的内存
* P254编译器找变量声明时，从局部作用域向全局作用域找
* P259初始化和赋值的区别：赋值是先初始化在赋值，效率较低，还有些需要初始化，故需养成使用构造函数初始值的习惯。
* P340注意容器类型和元素类型
* #define常量是通过包含头文件用，const常量是通过extern来引用
* cin（istream类）是输入流，键盘输入的变量，按enter结束输入，否则一直在等待，按enter后通过>>传给变量；cout是输出流，通过<<将变量赋给cout（ostream类），cout会自动在显示器上显示，endl是换行的变量
* const int sz = get\_size()//其具体值在运行时获得，不是常量表达式
* 编辑：使用IDE写代码的过程，像写文件一样，在硬盘（存储器）进行

编译：将代码变成机器可执行的文件（二进制等），存放于硬盘（存储器）

运行：将可执行文件从硬盘（存储器）调入运行内存（内存）执行，会占用运存空间

* 对象是指会在内存上开辟存储空间的，程序是在运存上运行，内存有限，会经常开辟、释放内存
* 一般前面是参数，后面是被操作变量，如：typedef int zhang；
* 一定要用名称（变量名、名称）和对象（内存上数据）的思想思考问题，如在传递参数时把传参看成被传参数赋值给参数，那么引用和指针会改变被传参数
* 内置类是库自带类型，类类型是自己定义的类
* 转义：n在C语言中有特殊意义：一个字母，那么可以通过\来转义，变成换行的意思
* strin和vector是容器，可称容器的迭代器，可用迭代器像下标方式一样访问容器元素，返回的是迭代器类型
* int \*ptr；其中ptr自己占用一个地址，为一个对象，\*ptr解引用才是它指向的对象
* 越界出错，可能取到一个未知的地址，取得一个未知的数
* 常见、未知错误：溢出（下标、值范围）、未定义（指针未定义）
* 左结合： 优先级相同时，从左到右结合，右结合：优先级相同时，由右向左结合。如：=是右结合
* 指针是指P，指针本身
* assert函数：先计算表达式 expression ，如果其值为假（即为0），那么它先向stderr打印一条出错信息，然后通过调用 abort 来终止程序运行
* set类元素都是唯一的，可以方便排序，

<https://blog.csdn.net/byn12345/article/details/79523516>

* new新建的是一个指针实例
* inline宏函数

<https://blog.csdn.net/K346K346/article/details/52065524>

* 运算符重载operator

<https://blog.csdn.net/zhangzheng_1986/article/details/81080407>

* 在C++中，直接=的话，会将两个值类型指向同一个地址这样的话，修改任何一个对象，另一个对象也会跟着变化？？Clone执行了浅拷贝~~~，拷贝后是一个新的对象，对新对象操作不影响原对象。非std类才需这样？
* 实例名（参数），调用operator（）类函数
* 如下代码，这不同于赋值操作，改变temp就改变image
* P145：static\_cast < type-id > ( exdivssion ) 该运算符把exdivssion转换为type-id类型，但没有执行时类型检查来保证转换的安全性。

<https://www.cnblogs.com/yxwkf/p/4090364.html>

* reserve减少重新分配次数，局部运算中，内存会被释放，那么多给点内存也没事。

<https://blog.csdn.net/zhangchen1003/article/details/49070341>

* 只有全局变量一直保存在内存中，局部变量在不断建立和释放
* iterator可以把迭代器指针像数一样操作
* erase：删除容器中position所指位置的元素。返回值是指向被删元素之后的那个元素(即下一个元素)的迭代器。

<https://www.cnblogs.com/wangkundentisy/p/9023977.html>

* front（）函数：取的是内容

<https://blog.csdn.net/laobai1015/article/details/51568456>

* 对于类中的类指针，可以使用new创建指针实例类中类指针实例=new 类名（参数）；对于类中的类实例，可以使用类中类实例=类名（参数）
* Ubuntu上编译器有CLion（用edu结尾邮箱可以申请免费使用一年）、kdevelop
* 面向对象的思想应该把自己想象成几个人在合作写代码，如class类之外的称为用户，实现了封装和分离
* 不同编译器还是有些差异的，VS、gcc编译器有所不同，具体在实现时注意
* P400，动态内存的管理是通过一对运算符来完成的：new，在动态内存中为对象分配空间并返回一个指向该对象的指针，我们可以选择对对象进行初始化，delete，接受一个动态对象的指针，销毁该对象，并释放与之关联的内存。有时会忘记释放内存，会造成内存泄漏，有时在尚有指针引用内存的情况下我们释放了它们，会产生引用非法内存指针。为了更容易使用动态内存，新的标准库提供了两种智能指针，与常规指针重要区别是它负责自动释放所指对象。shared\_ptr允许多个指针指向同一个对象；unique\_ptr则独占所指对象。weak\_ptr是伴随类、弱引用，指向shared\_ptr对象。操作表12.1、12.2。使用智能指针的基本规范P417。
* P403程序使用动态内存出于以下三种原因之一：程序不知道自己需要使用多少对象；程序不知道所需对象的准确类型；程序需要在多个对象间共享数据。
* P415函数的退出有两种可能，正常结束或者发生了异常，无论哪种情况，局部对象都会被销毁。
* P423allocator类允许我们将分配和初始化分离。提供更好的性能和更灵活的内存管理能力。P427有该类介绍，P464有典型例子，使用步骤P428表12.7。
* 未定义的含义就是语法中未定义或没考虑的情况，不知道会发生什么
* exit（int status）：用于退出程序，status一般为1，0，-1，0表示正常退出，1、-1表示异常退出

<https://www.runoob.com/cprogramming/c-function-exit.html>

* 模板的模板定义和模板函数放在同一个文件夹内，不能用直接转到定义，可以使用查找命令
* ORBSLAM2中DBoW2\Scoringobject.h的53行#define 一个scoring类
* ORBSLAM2中DBoW2\templatedvocabulary.h：virtual inline

C/C++关键字 asm、inline、register、virtual、volatile 详解\_askunix-CSDN博客

<https://blog.csdn.net/m0_37925202/article/details/79397883>

* 1

1. 与C不同类型或名词

* 重载：如果在同一作用域内的几个函数名字相同但形参列表不同，我们称之为重载（overloaded）函数。给程序员自己看的，重点时容易理解。
* int main(int argc ,char \*\*aegv，char \* envp)，argc参数个数，包括执行文件，以空格分；argv是输入的参数，是一个数组；envp为环境变量，其参数是命令行或launch文件输入的参数
* cerr输出错误信息，与cout 的区别是不缓冲
* 字面值常量(P37):0x1FF，024（八进制），L‘a’。
* String是一种库类型，表示可变化的字符序列
* 定义在函数体内部的内置类型变量将不会被初始化
* ：：作用运算符，如：std：：cout，类名：：，优先在：：的作用域（std，类内）找函数体出现的变量。P270::使用后作用域限定。
* lvalue、rvalue：左、右值
* 引用（复合类型）：int a=1；int &b=a；int &c=a；可多次引用，其中a、b、c本身不是对象，对象是指a指向的地址上的内容
* void\*指针，能做指针比较、函数输入输出、赋给另外一个void\*指针，不能就直接操作其所指对象，因为无法确定对象类型。
* const限定符：它的值（对象不改变），只能也必须在初始化时赋值。

常量只能赋值给常量，常量引用是指引用绑定的对象不变了，但对象可变

* 类型别名

typedef double wages;//wages是double的同义词

typedef wages base,\*p;//base是double的同义词，p是double\*的同义词

* 结构struct具备了类的功能，称为类

<https://www.cnblogs.com/starfire86/p/5367740.html>

* 多了一些与机器无关的类型，c中类型是与计算机单片机等存储、工作方式直接相关的
* sizeof (type),sizeif expr：计算类或对象所占空间大小
* throw{}：抛出异常；try{}：包含可能触发异常语句；catch{}：捕获异常，并处理异常，可以用来打印错误信息

1. C++11新标

* int \*p=nullptr，可以被转换成任意其他类型指针
* constexpr int sz = size()//只有当size为constexpr函数时，才是一条正确的声明。另外，其需绑定有固定地址的变量，如函数体之外的变量
* 别名声明

using SI = sales\_item;//SI是sales\_item的同义词

* 列表初始化

int a = {d};//使用花括号，当d为double时，会报警

* auto类型说明符：编译器根据运算结果确定变量类型，一般忽略顶层const，保留底层const

auto i=0,\*p=&i; //正确；i、p类型相同

auto sz = 4，pi = 3.14;//错误，两者类型不同

* decltype类型指示符

decltype(f()) sum = x;//sum的类型为f（）的返回类型，并不调用f

decltype（（variable））双层括号的结果肯定是引用，decltype（variable）单层括号的结果只有当variable本身就是一个引用时才是引用

* range for：遍历变量（如字符串变量）中基础元素，可以利用其实现下标合法，不越界。
* 迭代器的两个新函数cbegin和cend返回const\_iterator类型
* begin（数组名）、end（数组名）分别返回数组的首地址的指针和尾元素的下一个地址的指针，可以模仿迭代器
* 商一律向0取整
* sizeof不实际求运算对象的值，故可直接使用作用域符号直接访问类成员，而无需通过对象

1. 类：P227，struct（无说明符默认public）到class（无说明符默认private）的转变P240，默认权限是两者的唯一区别P241。数据抽象-忽略类型实现的细节，封装-保护类不能被随意访问。

* P232编译器处理类分2步：首先编译类成员的声明，然后才轮到成员函数（包括构造和析构函数）的函数体，故成员函数可以不分先后的使用类成员。
* 构造函数：能重载，创建类实例对象时初始化用，类名（形参）：初始化列表{函数体}
* 析构函数：不能重载，实例对象作用域结束时用，~类名（）{函数体}
* public：在整个程序内可被访问，通常成员函数被声明为公有。有效范围为出现下一个说明符之前。
* private：默认状态，可以被类的成员函数访问，但不能被使用该类的代码访问，一般数据成员被声明为私有。
* protected：
* 友元函数：p241，友元函数可以访问非公成员。不是类的成员，不受它所在作用域访问控制级别的约束。友元在类之间不存在传递性。
* 类的静态成员：有时候类需要它的一些成员与类本身直接相关，而不是与类的各个对象保持关联。如银行利率的浮动。存在于任何对象之外，对象中不包含任何与静态数据成员相关的数据。

1. 类型决定了数据所占的比特数，以及该如何解释这些比特内容
2. 字节-8比特，字-32比特，float-字，double-双字负数以补码形式存储，补码=原码按位取反+1。有符号、无符号数相互转换就是直接读取内存中的字节。默认情况下有符号变为无符号。
3. 未经初始化的变量和指针都会引发不可预知的错误。（为什么不由编译器检查呢）
4. 指针的值是地址，指针的对象是所指地址上内容
5. P141隐式转换：小整数类型的运算对象被提升成较大的整数类型（如short int转换成int，有符号转换成无符号），所有运算的对象最终会转换成同一类型（显式也一样）。被转换类型所占空间不应大于转换类型所占空间。尽可能避免损失精度。
6. 整型提升：转换后的类型能容纳原类型的所有值，如：P142，先转为int，否则，转为unsigned int
7. 强制转换：static\_cast:类型转换，不能改变常量属性；const\_cast：改变常量属性，不能做类型转换
8. 迭代语句通常称为循环语句
9. P400我们的程序到目前为止只使用过静态内存或栈内存，它们中的对象由编译器自动创建和销毁。此外还有内存池，又称为自由空间或堆（heap）。程序用堆来存储动态分配的对象。当动态对象不再使用时，我们的代码必须显示的销毁它们。
10. 1