常量 const int number = 100; 在c里面可以通过指针进行修改，但是在C++里面不行，这个和编译器有关系。

1. 引用 int age = 18；int& age2 = age；

#### 引用概念：

* 在C++中新增加了引用的概念。引用可以看作一个已定义变量的别名，引用的语法：Type& name = var; 引用作为函数参数声明时不进行初始化。
* 普通引用在声明时必须用其它的变量进行初始化，引用作为函数参数声明时不进行初始化。
* C++编译器在编译过程中使用常指针作为引用的内部实现，因此引用所占用的空间大小与指针相同。

指针和引用的区别？理解引用的真实含义？

代码分析：main.cpp中的testReference();testReferenceB();testPointerReference();

### **零碎的知识点**

可变参数？看代码可变参数.cpp

Static关键字？C++怎么操作？静态属性在C++中一定要初始化，必须写int Student::age = 23；静态方法不能操作非静态成员。静态可以直接通过类名来操作。非静态可以操作静态。

对象的大小？1.对象的大小与结构体计算方式一致 2.static 静态变量和方法没有计算到类的大小里面3.C++类对象计算需要考虑成员变量大小，内存对齐、是否有虚函数、是否有虚继承等。4.类对象大小没有因为增加了静态成员而变化。因为静态成员是属于类成员共有的，不单独属于任何一个对象，对静态成员的存储不会选择在某个对象空间，而是存在于堆当中，因此不会对对象的大小造成影响。

### **C++继承和派生**

派生类的目的是为了发展，派生类继承了基类的成员，实现了原有代码的重用，这只是一部分，而代码的扩充才是最主要的，只有通过添加新的成员，加入新的功能，类的派生才有实际意义。

以下是两种典型的使用继承的场景：  
1) 当你创建的新类与现有的类相似，只是多出若干成员变量或成员函数时，可以使用继承，这样不但会减少代码量，而且新类会拥有基类的所有功能；

1. 当你需要创建多个类，它们拥有很多相似的成员变量或成员函数时，也可以使用继承；可以将这些类的共同成员提取出来，定义为基类，然后从基类继承，既可以节省代码，也方便后续修改成员；

因为时间问题，我这里不做详细笔记了，大家参考我的代码 整理下。

### **C++多态与虚函数**

C++多态与虚函数，多态（Polymorphism）是面向对象（Object-Oriented，OO）思想”三大特征”之一，其余两个分别是封装（Encapsulation）和继承（Inheritance）–可见多态的重要性；或者说，不懂得什么是多态就不能说懂得面向对象。  
多态是一种机制、一种能力，而非某个关键字；它在类的继承中得以实现，在类的方法调用中得以体现；

### **C++模版**

C++是一门强类型语言，所以无法做到像动态语言（python javascript）那样子，编写一段通用的逻辑，可以把任意类型的变量传进去处理。泛型编程弥补了这个缺点，通过把通用逻辑设计为模板，摆脱了类型的限制，提供了继承机制以外的另一种抽象机制，极大地提升了代码的可重用性。

模板定义本身不参与编译，而是编译器根据模板的用户使用模板时提供的类型参数生成代码，再进行编译，这一过程被称为模板实例化。用户提供不同的类型参数，就会实例化出不同的代码。

普通函数模板：

|  |
| --- |
| template<typename T>  int compare(const T& left, const T& right) {  if (left < right) {  return -1;  }  if (right < left) {  return 1;  }  return 0;  }  compare<int>(1, 2); //使用模板函数 |

成员函数模板：

|  |
| --- |
| class Printer {  public:  template<typename T>  void print(const T& t) {  cout << t <<endl;  }  };  Printer p;  p.print<const char\*>("abc"); //打印abc |

模版类：

template<class T，class B> 定义一个模板类

为类定义一种模式。使得类中的某些数据成员、默写成员函数的參数、某些成员函数的返回值，能够取随意类型

常见的 容器比如 向量 vector <int> 或 vector <string> 就是模板类。

详见代码 Seqlist

### **异常：**

### **C++ STL**

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，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。

|  |  |
| --- | --- |
| **组件** | **描述** |
| 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。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） | 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。 |

在数据存储上，有一种对象类型，它可以持有其它对象或指向其它对像的指针，这种对象类型就叫做容器。很简单，容器就是保存其它对象的对象，当然这是一个朴素的理解，这种“对象”还包含了一系列处理“其它对象”的方法。

容器的分类：

1、顺序容器：是一种各元素之间有顺序关系的线性表，是一种线性结构的可序群集。顺序性容器中的每个元素均有固定的位置，除非用删除或插入的操作改变这个位置。顺序容器的元素排列次序与元素值无关，而是由元素添加到容器里的次序决定。顺序容器包括：vector(向量)、list（列表）、deque（队列）。

  2、关联容器：关联式容器是非线性的树结构，更准确的说是二叉树结构。各元素之间没有严格的物理上的顺序关系，也就是说元素在容器中并没有保存元素置入容器时的逻辑顺序。但是关联式容器提供了另一种根据元素特点排序的功能，这样迭代器就能根据元素的特点“顺序地”获取元素。元素是有序的集合，默认在插入的时候按升序排列。关联容器包括：map（集合）、set（映射）、multimap（多重集合）、multiset（多重映射）。

  3、容器适配器：本质上，适配器是使一种不同的行为类似于另一事物的行为的一种机制。容器适配器让一种已存在的容器类型采用另一种不同的抽象类型的工作方式实现。适配器是容器的接口，它本身不能直接保存元素，它保存元素的机制是调用另一种顺序容器去实现，即可以把适配器看作“它保存一个容器，这个容器再保存所有元素”。STL 中包含三种适配器：栈stack 、队列queue 和优先级队列priority\_queue 。

容器类自动申请和释放内存，因此无需new和delete操作。



① vector

　　（连续的空间存储,可以使用[]操作符）快速的访问随机的元素，快速的在末尾插入元素，但是在序列中间岁间的插入，删除元素要慢，而且如果一开始分配的空间不够的话，有一个重新分配更大空间，然后拷贝的性能开销。

② deque

　　（小片的连续，小片间用链表相连，实际上内部有一个map的指针，因为知道类型，所以还是可以使用[]，只是速度没有vector快）快速的访问随机的元素，快速的在开始和末尾插入元素，随机的插入，删除元素要慢，空间的重新分配要比vector快,重新分配空间后，原有的元素不需要拷贝。对deque的排序操作，可将deque先复制到vector，排序后在复制回deque。

③ list

　　（每个元素间用链表相连）访问随机元素不如vector快，随机的插入元素比vector快，对每个元素分配空间，所以不存在空间不够，重新分配的情况

④ set

　　内部元素唯一，用一棵平衡树结构来存储，因此遍历的时候就排序了，查找也比较快的哦。

⑤ map

　　一对一的映射的结合，key不能重复。

⑥ stack

　　适配器，必须结合其他的容器使用，stl中默认的内部容器是deque。先进后出，只有一个出口，不允许遍历。

⑦ queue

　　是受限制的deque，内部容器一般使用list较简单。先进先出，不允许遍历。

&& non-const reference

Const int&