1. 函数模板
   1. 泛型编程 – 模板技术 特点：类型参数化
   2. template< typename T > 告诉编译器后面紧跟着的函数或者类中出现T，不要报错，T是一个通用的数据类型
   3. 实现通用两个数进行交换函数
   4. 使用
      1. 自动类型推导 必须要推导出一致的T才可以使用
      2. 显示指定类型 mySwap<int>(a,b);
2. 实现对char和 int类型数组进行排序
   1. 利用模板技术 实现对char和int类型数组通用排序函数
3. 函数模板和普通函数的区别以及调用规则
   1. 区别
      1. 如果使用自动类型推导，是不可以发生隐式类型转换的
      2. 普通函数 可以发生隐式类型转换
   2. 调用规则
      1. 如果函数模板和普通函数都可以调用，那么优先调用普通函数
      2. 如果想强制调用函数模板，可以使用空模板参数列表
         1. myPrint<>(a, b);
      3. 函数模板也可以发生函数重载
      4. 如果函数模板能产生更好的匹配，那么优先使用函数模板
4. 模板的实现机制
   1. 编译器并不是把函数模板处理成能够处理任何类型的函数
   2. 函数模板通过具体类型产生不同的函数 --- 通过函数模板产生的函数 称为模板函数
   3. 编译器会对函数模板进行两次编译，在声明的地方对模板代码本身进行编译，在调用的地方对参数替换后的代码进行编译。
5. 模板局限性
   1. 模板并不是真实的通用，对于自定义数据类型，可以使用具体化技术，实现对自定义数据类型特殊使用
   2. **template<>** bool myCompare(**Person** &a, **Person** &b)
6. 类模板
   1. //类模板和函数模板区别：
   2. //1、类模板不可以使用自动类型推导，只能用显示指定类型
   3. //2、类模板中 可以有默认参数
7. 类模板中成员函数创建时机
   1. 类模板中的成员函数 并不是一开始创建的，而是在运行阶段确定出T的数据类型才去创建
8. 类模板做函数参数
   1. 1、指定传入类型
      1. void doWork(Person <string, int>&p)
   2. 2、参数模板化
      1. template<class T1, class T2>
      2. void doWork2(Person <T1, T2>&p)
   3. 3、整个类 模板化
      1. template<class T>
      2. void doWork3( T &p)
   4. 查看T数据类型
      1. typeid(T).name()
9. 类模板碰到继承的问题以及解决
   1. 必须要指定出父类中的T数据类型，才能给子类分配内存
   2. template<class T1 ,class T2>
   3. class Son2 :public Base2<T2>
10. 类模板中的成员函数类外实现
    1. void Person<T1, T2>::showPerson()
11. 类模板的分文件编写问题以及解决
    1. 类模板中的成员函数，不会一开始创建，因此导致分文件编写时连接不到函数的实现，出现无法解析的外部命令错误
    2. 解决方式1：
       1. 直接包含.cpp文件 （不推荐）
    3. 解决方式2：
       1. 将类声明和实现写到同一个文件中，将文件的后缀名改为 .hpp 即可
12. 类模板碰到友元的问题以及解决
    1. 友元类内实现
       1. friend void printPerson(Person<T1, T2> &p)
    2. 友元类外实现
    3. 声明：
       1. friend void printPerson2<>(Person<T1, T2> &p);
    4. 实现：
       1. template<class T1,class T2>
       2. void printPerson2(Person<T1, T2> &p){ 。。。}
    5. template<class T1,class T2>
    6. class Person;
    7. template<class T1,class T2>
    8. void printPerson2(Person<T1, T2> &p);
13. 类模板应用 – 数组类封装
    1. 将类写到 myArray.hpp中
    2. 属性：
       1. T \* pAddress; 指向堆区数组指针
       2. int m\_Capacity 数组容量
       3. int m\_Size ;数组大小
    3. 行为
       1. myArray(int capacity)
       2. myArray(const MyArray & arr)
       3. operator=
       4. operator[]
       5. ~myArray()
       6. getCapacity
       7. getSize
       8. pushback