1. 函数模板
   1. 实现两个整型和浮点型交换，逻辑相似，只是类型不同
   2. 如果想实现出通用的交换代码，可以利用函数模板
   3. 语法：
      1. template< typename T > T 通用的数据类型
      2. mySwap(T&a T&b){ … }
   4. 测试代码
      1. 使用模板两种方式
      2. 自动类型推导 必须推导出一致的T类型，才可以使用
      3. 显示指定类型
      4. 模板必须要确定出T的类型，才可以使用
2. 函数模板案例
   1. 实现对int和char数组排序
   2. 利用函数模板实现
   3. 3个模板 排序 交换 打印
   4. 泛型编程 利用模板技术
   5. 模板技术特点： 将类型参数化
3. 普通函数和函数模板区别以及调用规则
   1. 区别
      1. 普通函数 可以发生隐式类型转换
      2. 函数模板
         1. 自动类型推导，不会发生隐式类型转换
         2. 显示指定类型，可以发生隐式类型转换
   2. 调用规则
      1. 如果普通函数和函数模板都可以匹配，优先使用普通函数
      2. 如果想强制调用函数模板，需要加 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. 如果函数模板可以产生更好的匹配，那么优先使用函数模板
4. 函数模板机制
   1. 函数模板并不是万能，对于自定义类型数据，有些情况不能够正常使用
   2. 通过函数模板产生的函数，我们称为 模板函数
   3. 两次编译，对函数模板进行第一个编译，对替换T之后生成的模板函数 做第二次的编译
5. 模板局限性
   1. 模板并不是真正意义上的万能，对于特殊数据类型，可能运行失败
   2. 解决问题，可以利用具体化实现
   3. 语法：template <> 返回值类型 函数名 (处理数据类型..)
6. 类模板基础语法
   1. template<class T>后面紧跟着是类，类称为类模板
   2. 类模板使用时候，不可以用自动类型推导，只能用显示指定类型
   3. 模板中 也可以使用默认参数
      1. template<class NAMETYPE , class AGETYPE = **int**>
7. 类模板中成员函数创建时机
   1. 类模板中成员函数并不是一开始创建，而是在替换T后，才去生成
8. 类模板做函数参数
   1. 三种方式
      1. 指定传入类型 推荐
      2. 将参数模板化
      3. 将类模板化
   2. 查看T类型
      1. typeid(T).name();
9. 类模板碰到继承的问题以及解决
   1. 子类在继承时，必须确定出父类的T的数据类型，否则无法给T分配内存空间
   2. template<class T1, class T2>
   3. class Son2 :public Base2<T2>
10. 类模板的成员函数类外实现
    1. template<class T1,class T2>
    2. Person<T1,T2>::Person(T1 name, T2 age)
    3. template<class T1, class T2>
    4. void Person<T1,T2>::showPerson()
11. 类模板的成员函数分文件编写
    1. 类模板成员函数并不是一开始创建，导致直接包含.h头文件 ，链接不到
    2. 解决方式1：
       1. 直接包含源文件 （不推荐）
    3. 解决方式2： 将类模板中的成员函数，声明和实现都 写到同一个文件中
       1. 将文件的后缀名改为 .hpp
12. 类模板与友元
    1. 友元函数类内实现
       1. 将全局函数实现体写到类内部
       2. friend void printPerson(Person<*T1*,T2>& p)
    2. 友元函数类外实现
       1. 声明：friend void printPerson<>(Person<T1, T2>& p);
       2. 实现：template<class T1,class T2>
          1. void printPerson(Person<T1, T2>& p)
    3. //先让编译器看到Person类声明
    4. template<class T1, class T2>class Person;
    5. //先让编译器看到函数模板 printPerson的存在
    6. template<class T1, class T2> void printPerson(Person<T1, T2>& p);
13. 类模板案例-数组类封装
    1. 设计类 MyArray
    2. 属性
       1. T \* pAddress
       2. int m\_Capacity
       3. int m\_Size;
    3. 行为
       1. 有参构造
       2. 拷贝构造
       3. 析构
       4. 尾插 （作业：动态插入）
       5. 尾删（作业）
       6. 获取容量
       7. 获取大小
       8. operator =
       9. operator []
    4. 测试
       1. 利用int和Person进行数组测试