作业一

1.什么是程序实体的抽象与封装?

- 对于程序实体而言，抽象是指该程序实体的外部可观察到的行为，不考虑该程序实体的内部是如何实现的;
- 封装是指把该程序实体内部的具体实现细节对使用者隐藏起来，只对外提供一个接口。

2.面向对象程序设计有哪些特征？会带来哪些好处?

- 特征：封装、继承、多态
- 好处
  - 抽象：以数据为中心，强调数据抽象，操作依附于数据，二者联系紧密，控制复杂度。
  - 封装：实现了数据的封装，加强了数据的保护。
  - 模块化：按对象类划分模块，模块边界清晰，有利于组织和管理大型程序。
  - 软件复用：对象类往往具有通用性，再加上继承机制，使得程序容易复用，缩短开发周期。
  - 可维护性：对象类相对稳定，有利于程序维护，延长软件寿命。
  - 软件模型的自然度：基于对象交互的解题方式与问题空间有很好的对应，缩小解题空间与问题空间之间的 语义间隙，实现从问题到解决方案的自然过渡。

3.什么是对象？什么是类？二者之间的关系是怎样的?

- 对象：是有数据及能对其实施的操作所构成的封装体。
- 类：描述了对象的特征，包含什么类型的数据和哪些操作。
- 二者的关系：对象属于值的范畴，是程序运行时刻的实体；类则属于类型的范畴，属于编译时刻的实体。类是针对对象的描述，而对象是类的具体表现。

作业二

1.在 C++中，类成员的访问控制有哪几种？请分别阐述他们的访问级别。

- Public：访问不受限制。
- Private：只能在本类和友元的代码中访问。
- Protected：只能在本类，派生类和友元的代码中访问。

2.在 C++中，this 指针的作用是什么?

- 类定义中的成员函数对该类的所有对象只有一个拷贝，为了区分具体的操作对象，类的每一个成员函数（静态成员函数除外）都有一个隐藏的形参this，其类型为该类对象的指针。通过this指针知道对哪一个对象进行操作，从而实现在成员函数中对类成员的访问。

3.对于类中的常量数据成员和引用数据成员，应该如何进行初始化?

- 在构造函数的函数头和函数体之间加入一个成员初始化表来对常量和引用数据成员进行初始化。

作业三

1.在哪些情况下，会调用类的拷贝构造函数？什么时候需要自定义拷贝构造函数，为什么?

- 以下三种情况会调用类的拷贝构造函数：
  - 用另一个同类型的对象对其进行初始化；
  - 把对象作为实参传给函数时；
  - 把对象作为函数的返回值时。
- 需要自定义拷贝构造函数的时候：当类中的数据成员存在指针时，调用隐式拷贝构造函数会造成两个对象的指针成员指向同一块内存。如果对一个对象操作之后修改了这块空间的内容，则另一个对象将会受到影响，如果不是设计者特意所为，这将是一个隐藏的错误；调用析构函数时会两次释放同一块内存空间造成错误；两个对象中有一个消亡，另一个还没消亡时，则会出现使用已释放内存问题。这种情况应该自定义拷贝构造函数，申请一块新的内存存放相同的值，并且新建一个指针指向这片内存。

2.描述 const 关键字在类中的适用场景和作用。

- 修饰成员函数。防止在一个获取对象状态的成员函数中无意中修改对象的数据成员。常成员函数可以指出对常量对象能实施哪些操作，即只能调用对象类中的常成员函数。

3.描述 static 关键字在类中的适用场景和作用。

- 修饰数据成员，实现同一个类的不同对象之间的数据共享
- 修饰成员函数，使静态成员函数只能访问类的静态成员，没有隐藏的 this 参数。

作业四

1.说一说你对“友元”的理解。

- 为了提高在类的外部对类的数据成员的访问效率，在C++中，可以指定某些与一个类密切相关的、又不适合作为该类成员的程序实体直接访问该类的非public成员，这些程序实体称为该类的友元。友元是数据保护和数据访问效率之间的一种折中方案。 友

元需要在类中用friend显式指出，它们可以是全局函数、其它的类或其它类的某些成员函数。友元不是一个类的成员，友元关系具有不对称性与传递性。

2.在C++中，操作符重载有哪些实现途径?

- 作为一个类的非静态的成员函数（操作符new和delete除外）。
- 作为一个全局（友元）函数。（至少应该有一个参数是类、结构、枚举或它们的引用类型。）

3.在C++中，操作符重载需遵循哪些基本原则?

- 只能重载C++语言中已有的操作符，不可臆造新的操作符。
- 可以重载C++中除下列操作符外的所有操作符：“.”，“.*”，“?:”，“::”，“sizeof”
- 遵循已有操作符的语法：不能改变操作数个数，不改变原操作符的优先级和结合性。
- 尽量遵循已有操作符原来的语义。

作业五

1.什么情况下可以通过重载下标操作符“[]”来实现对其元素的访问?

- 对于由具有线性关系的元素所构成的对象。

2.对操作符new和delete进行重载会带来什么好处？为什么重载new和delete操作符必须要使用静态成员函数?

- 好处：如果是系统提供的new和delete操作，所涉及的空间分配和释放是通过系统的堆区管理系统来进行的，它要考虑各种大小的空间分配与释放，对某个类而言，效率常常不高。通过重载操作符new和delete来实现堆内存的管理，这样可提高堆内存的分配和归还的效率，并且还不会面临“碎片”问题。
- 原因：因为new之前还没有实例化对象，没有this指针；同理delete的整个过程对象也不是一直存在的。

作业六

1.在C++中，protect类成员访问控制的作用是什么?

- 在C++中通过引进protected成员访问控制来缓解继承与数据封装的矛盾：在基类中声明为protected的成员可以被派生类使用，但不能被基类的实例用户使用。这样，一个类就存在两个对外接口，一个接口由类的public成员构成，它提供给实例用户使用；另一个接口由类的public和protected成员构成，该接口提供给派生类使用。

2.派生类从基类那里继承了什么？派生类不能从基类那里继承什么?

- 继承了基类的所有成员（基类的构造函数、析构函数和赋值操作符重载函数除外）

- 不能继承构造函数、析构函数、赋值操作和友元。

3.阐述在C++继承中隐藏的概念。

- 如果派生类中定义了与基类同名的成员，则基类的成员名在派生类的作用域内不直接可见，称为隐藏。访问基类同名成员时要用基类名受限。即使派生类中定义了与基类同名但参数不同的成员函数，基类的同名函数在派生类的作用域中也是不直接可见的，仍然需要用基类名受限方式来使用之。

作业七

1.在 C++中，继承方式的作用是什么？public 继承方式有什么特点?

- 派生类的继承方式和基类成员的访问控制可以共同决定基类成员对派生类的用户具有何种访问控制。 在public继承方式下： 基类的public成员，在派生类中成为private成员； 基类的protected成员，在派生类中成为protected成员； 基类的private成员，在派生类中成为private成员。

2.请阐述 C++中动态绑定和静态绑定的概念，并说明在什么情况下会发生动态绑定。

- 动态绑定：在运行时刻，根据实际引用或指向的对象类型（动态类型）来确定采用哪一个消息处理函数； 静态绑定：在编译时刻，根据对象的静态类型来决定采用哪一个消息处理函数；
- 在基类中用virtual修饰的虚函数，在派生类中有相同构型的成员函数，即重定义，此时通过基类的指针或引用访问基类的虚函数时才进行动态绑定。

3.构造函数和析构函数是否可以是虚函数?

- 构造函数不能是虚函数，而析构函数可以（往往）是虚函数

作业八

1.在多继承中，什么情况下会出现二义性？怎样消除二义性?

- 多个基类包含同名的成员时，会出现名冲突问题，解决方法是基类名受限。当派生类从多个基类派生，而这些直接基类又从同一个基类派生时，会出现重复继承问题，解决方法是采用虚基类。

2.说说你对类之间的整体与部分的关系（聚合及组合）的认识。

- 类之间的整体与部分的关系，即一个类的对象包含了另一个类的对象，可以是聚合或组合关系。
    - 聚合：被包含的对象与包含它的对象独立创建和消亡，被包含的对象可以脱离包含它的对象独立存在。聚合类的 成员对象一般是采用对象指针表示，用于指向被包含的成员对象，而被包含的成员对象是在外部创建，然后加入 进来的。

- 组合：在组合关系中，被包含的对象随包含它的对象创建和消亡，被包含的对象不能脱离包含它的对象独立存 在。组合类的成员对象一般直接是对象，有时也可以采用对象指针表示，但不管是什么表示形式，成员对象一定 是在组合类对象内部创建并随着组合类对象消亡。

3.可以创建抽象类的对象吗？存在纯虚函数是成为抽象类的什么条件（充分必要/充分不必要/必要不充分）？ 抽象类的派生类是否一定要给出纯虚函数的实现？

- 不可以创建抽象类的对象。
- 充分必要条件。
- 不是一定要给出。如果不给出纯虚函数的实现，则该派生类继承了此纯虚函数，自己仍然是抽象类。

作业九

1.std::cin 和 std::cout 相比 scanf 和 printf 的优势是什么？

- 不需要单独指定数据的类型和个数，编译时刻根据数据本身来决定操作的类型和个数，可以避免与类型和个数相关的错误。

2.C++的I/O类库中基本的类有哪些？分别用于什么场景？

- 面向控制台的I/O：从标准输入设备（如键盘）获得数据，把程序结构从标准输出设备（如显示器）
- 输出面向文件的I/O：从外存文件获得数据，把程序结构保存到外存文件中
- 面向字符串变量的I/O：从程序中的字符串变量中获得数据，把程序结果保存到字符串变量中

3.windows下如何使下面的while循环终止？

- 按 Ctrl+Z 组合键后再按回车键。

作业十

1.请简述什么是事件驱动的程序设计。

- 事件驱动的程序设计是一种基于事件（消息）驱动的交互式流程控制结构：程序的任何一个动作都是由某个事件激发的。事件可以是用户的键盘、鼠标、菜单等操，每个事件都会向应用程序发送一些消息。每个应用程序都有一个消息队列。系统会把属于各个应用程序的消息放入各自的消息队列。应用程序不断地从自己的消息队列中获取消息并处理获得的消息。“取消息-处理消息”的过程称为消息循环。当取到某个特定消息（如：WM_QUIT）后，消息循环结束。每个窗口都有一个消息处理函数。大部分的消息都关联到某个窗口。应用程序取到消息后将会去调用相应窗口的消息处理函数。

2.请简述基于 Windows API 的过程式事件驱动程序设计中，主函数 WinMain 的主要功能

- 注册窗口类（定义程序中要创建的窗口类型）：窗口的基本风格、消息处理函数、图标、光标、背景颜色以及菜单等。每类窗口（不是每个窗口）都需要注册。创建应用程序的主窗口（其它窗口等到需要时再创建）。进入消息循环，直到接收到WM_QUIT消息时，消息循环结束。

作业十一

1.请简述什么是“文档-视”结构。

- 文档：用于存储和管理程序中的数据。
- 视：显示文档数据以及实现对文档数据进行操作时与用户的交互功能。
- 文档与视结合构成了“文档-视”结构，它可以实现：数据的内部表示形式和数据的外部展现形式相互独立。一个文档对象可以对应一个或多个视对象，即，对于同一个文档数据可以用不同的方式进行显示和操作。

作业十二

1.为什么需要异常处理？就地处理和异地处理方法分别有哪些?

- 导致程序运行异常的情况可以预料，但无法避免。为了保证程序的鲁棒性，必须在程序中对可能出现的异常进行预见性处理。
- 就地处理：在发现异常错误的地方处理异常。调用C++标准库中的函数exit或abort终止程序执行。
  - abort立即终止程序的执行，不作任何的善后处理工作；
  - exit在终止程序的运行前，会做关闭被程序打开的文件、调用全局对象和static存储类的局部对象的析构函数等工作。
- 异地处理：在其它地方（非异常发现地）处理异常。
  - 通过函数的返回值、指针/引用类型的参数或全局变量把异常情况通知函数的调用者，由调用者处理异常；
  - 通过语言提供的结构化异常处理机制进行处理。

2.简述C++结构化异常处理的实现机制。

- 把有可能遭遇异常的一系列操作（语句或函数调用）构成一个try语句块；
- 如果try语句块中的某个操作在执行中发现了异常，则通过执行一个throw语句抛掷（产生）一个异常对象， 之后的操作不再进行；
- 抛掷的异常对象将由程序中能够处理这个异常的地方通过catch语句块来捕获并处理之。

3.throw和return之间的区别何在?

- return为返回。在return时，当前函数返回给上层函数，上层函数继续执行其他代码部分，直至return才会继续返回给上上层函数。
- throw语句为异常情况下抛掷（产生）异常对象。如果某个try块执行中抛掷了异常对象，则首先在内层try块后寻找对应的catch语句，如果没有，就逐步向外层进行查找，但中间层函数的其他代码部分是无法继续执行的。

4.断言的作用是什么？C++宏assert的实现原理是什么？

- 断言是一个逻辑表达式，描述程序执行到断言处应满足的条件。若条件满足则程序继续执行，否则程序异常终止。断言的作用如下： 帮助理解程序在程序开发阶段，帮助开发者发现程序的错误和进行错误定位。
- 宏assert的实现原理：通过条件编译预处理命令来实现，其实现细节大致如下：

```
//cassert 或 assert.h
......
#ifdef  NDEBUG
#define assert(exp)  ((void)0)
#else
#define assert(exp)  ((exp)?(void)0:<输出诊断信息并调用库函数abort>)
#endif
......
```

作业十三

1.简述C++泛型编程的意义

- 使得一个程序实体能对多种类型的数据进行操作或描述

2.函数模板和类模板的实例化分别有什么方法?

函数模板：

- 隐式实例化(模板实参推导):当模板参数全为类型参数时，直接对函数模板进行调用，编译程序根据函数调用的实参类型，自动地把函数模板实例化为具体的函数
- 显式实例化:当模板参数含有非类型参数时，必须对函数进行显式实例化，给函数模板提供相应的具体类型

类模板：显式实例化

3.为什么尽量要把模板的定义和实现都放在同一个头文件中?

因为使用一个模板之前首先要对其实例化（用一个具体的类型去替代模板的类型参数），而实例化是在编译时刻进行的，它一定要见到相应的源代码，否则无法实例化。

4.下面的调用是否合法，如果合法，T的类型是什么?如果不合法，为什么?

```
template bool compare(const T&,const T&);
compare("hi","world");
```

- 不合法。两个数组长度不同。

作业十四

概念题

1.请列出STL中的主要容器及它们的应用场合。

- vector<元素类型>
  - 用于需要快速定位（访问）任意位置上的元素以及主要在元素序列的尾部增加/删除元素的场合。
  - 在头文件vector中定义，用动态数组实现。
- list<元素类型>
  - 用于经常在元素序列中任意位置上插入/删除元素的场合。
  - 在头文件list中定义，用双向链表实现。
- deque<元素类型>
  - 用于主要在元素序列的两端增加/删除元素以及需要快速定位（访问）任意位置上的元素的场合。
  - 在头文件deque中定义，用分段的连续空间结构实现。
- stack<元素类型>
  - 用于仅在元素序列的尾部增加/删除元素的场合。
  - 在头文件stack中定义，可基于deque、list或vector来实现。
- queue<元素类型>
  - 用于仅在元素序列的尾部增加、头部删除元素的场合。
  - 在头文件queue中定义，可基于deque和list来实现。
- priority_queue<元素类型>
  - 它与queue的操作类似，不同之处在于：每次增加/删除元素之后，它将对元素位置进行调整，使得头部元素总是最大的。也就是说，每次删除操作总是把最大（优先级最高）的元素去掉。
  - 在头文件queue中定义，可基于deque和vector来实现。
- map<关键字类型，值类型>和 multimap<关键字类型，值类型>
  - 用于需要根据关键字来访问元素的场合。容器中每个元素是一个pair结构类型，该结构有两个成员：first和second，关键字对应first，值对应second，元素是根据其关键字排序的。
  - 对于map，不同元素的关键字不能相同；
  - 对于multimap，不同元素的关键字可以相同。
  - 它们在头文件map中定义，常常用某种二叉树来实现。

- set<元素类型>和multiset<元素类型>
  - 它们分别是map和multimap的特例，每个元素只有关键字而没有值，或者说，关键字与值合一了。
  - 在头文件set中定义。
- basic_string<字符类型>
  - 与vector类似，不同之处在于其元素为字符类型，并提供了一系列与字符串相关的操作。
  - string和wstring分别是它的两个实例
  - 在头文件string中定义。

2.请列出STL中基本迭代器的5种类型、说明它们各自可以进行的操作并表示出其相容关系。

- 输出迭代器（output iterator，记为：OutIt）
  - 可以修改它所指向的容器元素
  - 间接访问操作（*）
  - ++操作
- 输入迭代器（input iterator，记为：InIt）
  - 只能读取它所指向的容器元素
  - 间接访问操作（*）和元素成员间接访问（->）
  - ++、==、!=操作。
- 前向迭代器（forward iterator，记为：FwdIt）
  - 可以读取/修改它所指向的容器元素
  - 元素间接访问操作（*）和元素成员间接访问操作（->）
  - ++、==、!=操作
- 双向迭代器（bidirectional iterator，记为：BidIt）
  - 可以读取/修改它所指向的容器元素
  - 元素间接访问操作（*）和元素成员间接访问操作（->）
  - ++、--、==、!=操作
- 随机访问迭代器（random-access iterator，记为：RanIt）
  - 可以读取/修改它所指向的容器元素
  - 元素间接访问操作（*）、元素成员间接访问操作（->）和下标访问元素操作（[]）
  - ++、--、+、-、+=、-=、==、!=、<、>、<=、>=操作

3.容器的成员函数`rbegin()`返回的反向迭代器指向的元素为?

- 容器的尾元素

编程题

1.

```
#include<iostream>
#include<string>
#include<set>
#include<sstream>
using namespace std;
set<string> dict;

int main(){
        string s,buf;
        while(cin>>s){
                for(int i=0;i<s.length();i++)
                        if(isalpha(s[i]))
                                s[i]=tolower(s[i]);
                        else
                                s[i]=' ';
                stringstream ss(s);
                while(ss>>buf)
                        dict.insert(buf);
        }
        for(set<string>::iterator it=dict.begin();it!=dict.end();++it)
                cout<<*it<<"\n";
        return 0;
}
```

2.

```
#include<cstdio>
#include<algorithm>
using namespace std;

int main()
{
    int n,m,kace=0;
    while(scanf("%d%d",&n,&m)==2&&n)
    {
        int a[100];
        for(int i=0; i<n; i++)
        {
            scanf("%d",&a[i]);
        }
        sort(a,a+n);
        while(m--)
        {
```

```
            int key;
            scanf("%d",&key);
            int i=lower_bound(a,a+n,key)-a;
            if(a[i]==key)
            {
                printf("%d found at %d\n",key,i+1);
            }
            else
            {
                printf("%d not found\n",key);
            }
        }
    }
    return 0;
}
```

作业十五

概念题

1.请阐述命令式程序设计和声明式程序设计的区别

命令式程序设计范式（imperative programming）需要对“如何做”进行详细描述，包括操作步骤和状态变化；它们与冯诺依曼体系结构一致，是使用较广泛的程序设计范式，适合于解决大部分的实际应用问题。

声明式程序设计范式（declarative programming）只需要对“做什么”进行描述，不需要给出操作步骤和状态变化；有良好的数学理论支持，易于保证程序的正确性，并且，设计出的程序比较精炼和具有潜在的并行性。

2.请简述函数式程序设计的基本特征

- “纯”函数：
  - 以相同的参数调用一个函数总得到相同的值。（引用透明）
  - 除了产生计算结果，不会改变其他任何东西。（无副作用）
- 没有状态：计算体现为数据之间的映射，它不改变已有数据，而是产生新的数据。（无赋值操作）
- 函数也是值：函数的参数和返回值都可以是函数，可由已有函数生成新的函数。（高阶函数）
- 递归是主要的控制结构：重复操作采用函数的递归调用来实现，而不采用迭代（循环）。

- 表达式的惰性（延迟）求值（Lazy evaluation）：需要用到表达式的值的时候才会去计算它。
- 潜在的并行性：由于程序没有状态以及函数的引用透明和无副作用等特点，因此一些操作可以并行执行。

3.请简述函数式程序设计的基本手段

- 递归和尾递归
- 过滤/映射/规约操作（Filter/Map/Reduce）
- 部分函数应用（Partial Function Application）
- 柯里化（Currying）

4.请简述部分函数应用和柯理化的区别

虽然偏函数应用和柯里化都能实现类似的功能，但它们是有区别的：

- 偏函数应用是通过固定原函数的一些参数值来得到一个参数个数较少的函数，对该函数的调用将得到一个具体的值；
- 柯里化则是把原函数转换成由一系列单参数的函数构成的函数链，对柯里化后的函数调用将得到函数链上的下一个函数，对函数链上最后一个函数的调用才会得到具体的值。
- 偏函数应用可以按任意次序绑定原函数的参数值，而柯里化只能依次绑定原函数的参数值。

编程题

1.

```
#include "leapyear.h"

auto Or = [](bool a, bool b)->bool{return a||b;};
auto And = [](bool a, bool b)->bool{return a&&b;};
auto Not = [](bool a)->bool{return !a;};
auto Aliqot = [](int dividend, int divisor){return dividend%divisor == 0;};
bool LeapYear::IsLeapYear(int year){
    auto aliqot4 = Aliqot(year,4);
    auto notaliquotby100 = Not(Aliqot(year,100));
    auto aliqotby400 = Aliqot(year,400);
    auto leapyear = Or(And(aliqot4,notaliquotby100),aliqotby400);
    return leapyear;
}
```

2.略