T1 C++专业课问答

一、 绪论

1. 程序？

   计算机的工作都是用程序来控制的，没有程序，计算机将一事无成

2. 指令？

   计算机可以识别的命令。也就是由0，1组合的命令，一般是固定的程序是指令的集合

3. 什么是机器语言，汇编语言和高级语言？

   a. 机器语言：由计算机硬件系统可以识别的二进制指令组成的语言成为机器语言，

   b. 汇编语言：将机器指令映射为一些可以被人读懂的助记符，例如add,sub等可以用来表示算术运算中的加减

      机器语言与汇编语言与人类自然语言都相去甚远，而且不同的机器所能识别的汇编语言与机器语言存在一定的差距，程序员在编译程序时需要考虑到大量的机器细节，且编译好的代码不具有泛用性

   c. 高级语言：它屏蔽了机器细节，提高了语言的抽象层次，程序中可以采用具有一定含义的数据命名和容易理解的执行语句。这使得程序书写时可以对应现实中的具体事物

4. 什么是面向对象的语言？

   面向对象的编程语言将客观事物看作具有属性和行为的对象，通过抽象找出同一类对象的共同属性和行为，形成类。通过类的继承和多态可以很方便地实现代码重用，使得软件风格统一。

5. 简要说明一下对象，类和封装的概念

   a. 对象：现实中的对象是现实生活中的一个事物，既可以是有形的也可以是无形的。面向对象中的对象是构成系统的一个基本单位，用来描述客观世界的一个实体，由一组属性和行为构成。动态，具体。

   b. 类：从同类型的对象中抽象出共性，就形成了类。换种方式说，类像一个表头，包含了某一类对象的共同属性和行为，而对象是一个表项，填入了具体的属性值。定义一个数据类型的蓝图。这实际上并没有定义任何数据。静态，抽象。

   c. 封装：是C++的一个核心原则，就是将对象的属性和行为封装成一个整体，在保留操作接口的同时并尽可能的向外部屏蔽细节。

6. 什么是封装？

封装就是将抽象得到的属性和行为结合成一个整体，也就是将数据和操作数据的代码相结合，形成面向对象方法特有的类，在尽可能隐蔽对象内部细节的同时对外提供接口。

7. 为什么几乎所有的计算机都采用二进制？

    a. 便于物理实现，可以用高压电平表示0，1
    b. 二进制运算简单
    c. 机器可靠性高，抗干扰能力强
    d. 通用性强

8. 位，字节，字，机器字长有什么区别？

    a. 位：度量数据的最小单位，表示1位二进制信息
    b. 字节：一个字节有8个二进制数据构成。是信息存储的基本单位
    c. 字：是计算机存储器的基本单位
    d. 机器字长：参与运算的寄存器的二进制的位数

8. 不同二进制编码方法的实质是什么？对负数表示的不同编码

    a. 原码，若机器字长n+1位，原码整数的表示范围：`-(2^n-1) ≤ x ≤ 2^n-1`（关于原点对称）
    b. 反码，若机器字长n+1位，反码整数的表示范围：`-(2^n-1) ≤ x ≤ 2^n-1`（关于原点对称）
    c. 补码，若机器字长n+1位，补码整数的表示范围：`-2^n ≤ x ≤ 2^n-1`（比原码多表示一个-2^n ）
    d. 移码，若机器字长n+1位，移码整数的表示范围：`-2^n ≤ x ≤ 2^n-1`（与补码相同）

	正数	负数
原码	符号位0	符号位1
反码	同原码	符号位1，数值位取反
补码	同原码	反码+1
移码	补码的基础上将符号位取反	补码的基础上将符号位取反

9. 什么是对象？什么是面向对象方法？这种方法有哪些特点？

    a. 对象是构成世界的一个独立单位，它具有自己的静态特征和动态特征。面向对象方法中的对象，是系统中用来描述客观事物的一个实体，它是用来构成系统的一个基本单位，由一组属性和一组行为构成。

b. 面向对象的方法将数据及对数据的操作方法放在一起，作为一个相互依存、不可分离的整体--对象。对同类型对象抽象出其共性，形成类。类中的大多数数据，只能用本类的方法进行处理。类通过一个简单的外部接口，与外界发生关系，对象与对象之间通过消息进行通讯。

10. 面向过程与面向对象编程的区别？

面向过程就是分析出解决问题所需要的步骤，然后用函数把这些步骤一步一步实现，使用的时候一个一个依次调用就可以了；

面向对象是把构成问题事务分解成各个对象，建立对象的目的不是为了完成一个步骤，而是为了描叙某个事物在整个解决问题的步骤中的行为

面向过程

优点：性能比面向对象高，因为类调用时需要实例化，开销比较大，比较消耗资源，比如单片机、嵌入式开发、Linux/Unix等一般采用面向过程开发，性能是最重要的因素。

缺点：不如面向对象易维护、易复用、易扩展

面向对象

优点：易维护、易复用、易扩展，由于面向对象有封装、继承、多态性的特性，可以设计出低耦合的系统，使系统更加灵活、更加易于维护

缺点：性能比面向过程低

11. 面向对象的六大原则？

开闭原则（**Open Close Principle**）

对扩展开放，对修改关闭。在软件的生命周期内，因为变化，升级和维护等原因需要对软件原有代码进行修改，可能会给旧代码引入错误，也有可能会使我们不得不对整个功能进行重构，并且需要原有代码经过重新测试。解决方案：当软件需要变化时，尽量通过扩展软件实体的行为来实现变化，而不是通过修改已有的代码来实现。不过这要求，我们要对需求的变更有前瞻性和预见性。其实只要遵循以下5中设计模式，设计出来的软件就是符合开闭原则的

里氏代换原则（**Liskov Substitution Principle**）

里氏代换原则是面向对象设计的基本原则之一。 所有引用基类的地方必须能透明地使用其子类的对象。在软件中将一个基类对象替换成它的子类对象，程序将不会产生任何错误和异常，反过来则不成立，如果一个软件实体使用的是一个子类对象的话，那么它不一定能够使用基类对象。例如：我喜欢动物，那我一定喜欢狗，因为狗是动物的子类；但是我喜欢狗，不能据此断定我喜欢动物，因为我并不喜欢老鼠，虽然它也是动物。

依赖倒转原则（**Dependence Inversion Principle**）

简单的说就是要求对抽象进行编程，不要对实现进行编程。基类多设置为抽象类，多定义接口。

接口隔离原则（Interface Segregation Principle）

一个类对另一个类的依赖应该建立在**最小的接口**上,通俗的讲就是需要什么就提供什么，不需要的就不要提供。接口中的方法应该尽量少，不要使接口过于臃肿，不要提供很多不相关的逻辑方法。

迪米特法则，最少知道原则（Demeter Principle）

多用组合(has-a)，少用继承(is-a)。如果新对象的某些功能在别的已经创建好的对象里面已经实现，那么应当尽量使用别的对象提供的功能，使之成为新对象的一部分，而不要再重新创建。可以降低类与类之间的耦合程度。

单一职责原则（Single responsibility principle）

一个类应该有且只有一个变化的原因。单一职责原则将不同的职责分离到单独的类，每一个职责都是一个变化的中心。需求变化时，将通过更改职责相关的类来体现。如果一个类拥有多于一个的职责，则多个职责耦合在一起，会有多于一个原因来导致这个类发生变化。一个职责的变化可能会影响到其他的职责，另外，把多个职责耦合在一起，影响复用性。

二、 基本程序概念

1. **常量与符号常量**

    a. 常量：程序运行过程中不可改变的量，直接使用符号（文字）表示的值。包括整型常量，实型常量，字符常量和字符串常量
    b. 符号常量：使用符号为常量命名

2. **符号常量使用之前必须实现声明**

    const 数据类型说明符 常量名=常量值

    ```
    const float PI = 3.141592653589793
    ```

3. **使用关键字const而不是#define语句的好处有哪些？**

    const定义的常量是有类型的，所以在使用它们时编译器可以查错；而且，这些变量在调试时仍然是可见的。

4. **在下面的枚举类型中，Blue的值是多少？**

```
enum COLOR { WHITE,  BLACK = 100,  RED,  BLUE,
GREEN = 300 };
```

Blue 为 102, WHITE 为 0, RED 为 0,

5. 在一个**for**循环中，可以初始化多个变量吗？如何实现？

在for循环设置条件的第一个";"前用,分隔不同的赋值表达式。

```
for (x = 0, y = 10; x < 100; x++,  y++) //不能像C#
第一个;前 int x=0
```

6. 执行完下列语句后，**n**的值为多少？

```
int n;
 for (n = 0; n < 100; n++)
```

n的值为100

7. 什么叫做作用域？什么叫做局部变量？什么叫做全局变量？

作用域是一个标识符在程序正文中有效的区域。

局部变量，一般来讲就是具有块作用域的变量

全局变量，就是具有命名空间作用域的变量。

8. 打印**ASCII**码为**32~127**的字符。

```
#include <iostream>
int main()
{
for (int i = 32; i<128; i++)
    cout << (char) i; //将int转换成char型即可输出数字对应ASCII
码
return 0;
}
```

程序运行输出：

```
!"#$%G'()*+, ./0123456789:;<>?
@ABCDEFGHIJKLMNOP_QRSTUVWXYZ[\]^'abcdefghijklmnopqrstuvwxy
z<|>~s
```

9. 什么叫常量？什么叫变量？

所谓常量是指在程序运行的整个过程中其值始终不可改变的量，除了用文字表示常量外，也可以为常量命名，这就是符号常量；

在程序的执行过程中其值可以变化的量称为变量，变量是需要用名字来标识的。

10. 变量有哪几种存储类型？

变量有以下几种存储类型：

a. auto存储类型：采用堆栈方式分配内存空间，属于一时性存储，其存储空间可以被若干变量多次覆盖使用；

b. register存储类型：存放在通用寄存器中；

c. extern存储类型：在所有函数和程序段中都可引用，**extern 修饰符通常用于当有两个或多个文件共享相同的全局变量或函数的时候**；

d. static存储类型：在内存中是以固定地址存放的，在整个程序运行期间都有效，具有全局生存周期。

以下两种为较少见

e. mutable存储类型：mutable也是为了突破const的限制而设置的。被mutable修饰的变量，将永远处于可变的状态，即使在一个const函数中；

f. thread_local(C++11)存储类型：为线程安全引进的变量声明符；

从 C++ 17 开始，auto 关键字不再是 C++ 存储类说明符，且 register 关键字被弃用。

11. 比较**Break**语句与**Continue**语句的不同用法。

a. Break使程序从循环体和switch语句内跳出，继续执行逻辑上的下一条语句，不能用在别处；

b. continue 语句结束本次循环，接着开始判断决定是否继续执行下一次循环；

12. 逻辑运算——异或

两个操作数对应的每一个位进行异或，若对应位相同，则运算结果为0，若对应位不同，则运算结果为1

13. 枚举类型

enum 枚举类型名 {变量值列表}

```
 enum Weekday {SUN,MON,TUE,WED,THU,FRI,SAT};
//默认值
 enum Weekday {A,B=16,C,D,E=27,F,G};     //合法
 SUN=7; // 非法
 A=1;   // 非法
```

枚举类型不能赋值，但可以在定义是指定值。如果不给出定义值则会按照默认值，上述例子中分别为0，1，2...，6

14. 结构化程序设计的三种基本控制结构是什么？

a. **顺序结构**：按照先后顺序依次执行程序中的语句

b. 选择结构：按照给定条件有选择地执行程序中的语句if else\switch case
c. 循环语句：按照给定规则重复地执行程序中的语句 while\for\do while

15. 什么是宏定义？有什么作用？

宏定义是**替换**，不做计算，也不做表达式求解。简称宏。

格式：#define 标识符 字符串

a. 宏名一般用大写
b. **使用宏可提高程序的通用性和易读性，减少不一致性，**减少输入错误和便于修改。例如：数组大小常用宏定义
c. 提高程序的可维护性

16. **new和malloc的区别？**

a. malloc和new都是在堆上开辟内存的malloc只负责开辟内存，没有初始化功能，需要用户自己初始化；new不但开辟内存，还可以进行初始化。
b. malloc是函数，开辟内存需要传入字节数，malloc的返回值需要强转成指定类型的地址，malloc内存分配成功则是返回void * ，需要通过强制类型转换将void*指针转换成我们需要的类型；
new是运算符，开辟内存需要指定类型，返回指定类型的地址，因此不需要进行强转，故new是符合类型安全性的操作符。使用new操作符申请内存分配时无须指定内存块的大小，编译器会根据类型信息自行计算。而malloc则需要显式地指出所需内存的尺寸。
c. malloc开辟内存失败返回NULL，new开辟内存失败抛出bad_alloc类型的异常
d. malloc开辟的内存永远是通过free来释放的；而new单个元素内存，用的是delete，如果new[]数组，用的是delete[]来释放内存的。

17. **C语言中volatile关键字有什么特点？**

使用volatile关键字声明的变量，系统总是重新从它所在的内存中读取数据，即使它前面的指令刚刚从该处读取过数据，而且读取的数据立刻被保存；相反，若没有使用volatile，编译器可能会做优化处理，可能暂时使用寄存器中的值，而如果该变量由别的程序更新了的话，将会出现不一致的现象！！

总结起来就是：

a. 编译器会禁止对 volatile 修饰的变量做读写优化
b. 每次使用该变量时，系统都会重新从它所在内存中读取数据
c. 这相对于做了读取优化的变量来说，速度当然是慢了一些

18. **假定编译器不做优化，对一个int型变量k，做前缀自加和后缀自加，哪个执行速度更快？**

因为++k运算结束后，k的值和表达式的值相同。 而k++运算结束后，k的值和表达式的值不相同。编译器要开辟一个新的变量来保存k++表达式的值。 所以说：++k更快。

三、 函数基本概念

1. 函数？

能够完成某一特定功能的一段程序，可以被重复使用，使用者不需要关心函数的内部细节，而只需要关注函数的功能。

2. 值调用？引用调用？指针调用？

a. 值调用：当发生函数调用时，首先为形参分配相应的内存空间，然后用将实参的值复制给形参，这是一个参数值单项传递的过程，一旦形参得到了实参的所有值，就与实参脱离关系，相当于是实参的一个副本，对形参的任何操作都不会影响到实参。

b. 引用调用：当发生函数调用时，系统用实参来初始化形参的引用，相当于直接对实参进行操作，形参是实参的一个别名，它们指向内存中的同一块数据单元

c. 指针调用：该方法把参数的地址赋值给形式参数。在函数内，该地址用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响实际参数。

3. 内联函数

用关键字inline声明的函数是内联函数，编译器在编译时将函数体直接嵌入在每一个调用处，节省了参数传递和控制转移的开销。一般是调用频率高且规模较小的函数声明为内联函数。

内联函数的特点：

a. 内联函数不能包含循环和switch等复杂结构
b. 内联函数的声明必须在第一次调用内联函数之前
c. 对内联函数不能进行异常接口声明

4. 什么是函数的重载？重载函数时通过什么来区分？

有两个及两个以上的同名函数，它们的形参类型或者形参的个数不同，编译器在编译时将根据实参和形参的类型以及形参个数找到最佳匹配的函数，这就是函数的重载。

5. C++中的函数是什么？什么叫主调函数，什么叫被调函数，二者之间有什么关系？如何调用一个函数？

一个较为复杂的系统往往需要划分为若干子系统，高级语言中的子程序就是用来实现这种模块划分的。C和C++语言中的子程序就体现为函数。调用其它函数的函数被称为主调函数，被其它函数调用的函数称为被调函数。一个函数很可能既调用别的函数又被另外的函数调用，这样它可能在某一个调用与被调用关系中充当主调函数，而在另一个调用与被调用关系中充当被调函数。

6. 什么叫作嵌套调用？什么叫作递归调用？

函数允许嵌套调用，如果函数1调用了函数2，函数2再调用函数3，便形成了函数的嵌套调用。

递归调用就是函数反复调用自身的过程。

7. **如何实现分治与递归**

    a. 分治是将大问题划分为若干个规模较小、可以直接解决的子问题，然后解决这些子问题，最后将这些子问题的解合并起来，即是原问题的解。分治是一种思想，它不涉及到具体的算法，而大多数情况下，分治都是借由递归来实现的。

    b. 递归是一个很难阐述的概念。从c语言角度来讲，递归就是这个函数反复调用自身，然后将问题一步步地缩小，直到这个问题已经缩小到可以直接解决的程度，然后再一步一步地返回，最终解决原问题。

    c. 递归的逻辑中有两个重要的概念：

        i. 递归边界。递归边界是分解的尽头。
        ii. 递归式。递归式是将问题规模一步步缩小的手段。

8. **函数原型中的参数名与函数定义中的参数名以及函数调用中的参数名必须一致吗？**

    不必一致，所有的参数是根据位置和类型而不是名字来区分的。

四. 类与对象

1. **构造函数？析构函数？有什么作用？**

    a. 构造函数：构造函数就是在对象被创建的时候用一组特定的值来构造对象，将**对象初始化为一个特定状态**，与该类的其他对象进行区分，构造函数将在创建对象时**系统自动调用**。特点：函数名与类名相同，没有返回值，可以重载
    b. 析构函数：析构函数主要在对象被删除前进行一些清理工作，在对象的**生存周期结束时系统自动调用**，释放对象所占用的内存空间。特点：析构函数不接受任何参数。一个类可以有多个构造函数，但是只能由1个析构函数。

2. **拷贝构造函数？何时被调用**

    拷贝构造函数是一种特殊的构造函数，具有一般构造函数的所有特性，其形参是本类对象的引用，用来实现利用一个已经存在的对象，创建一个同类的新对象。

    三种调用：

    a. 用类的一个对象初始化该类的另一个对象
    b. 函数的形参是类对象，函数形参和实参结合时将调用拷贝构造函数(值传递)
    c. 函数的返回值是类对象，函数调用完成返回时需要调用拷贝构造函数。

3. **解释public和private的作用，公有类型成员与私有类型成员有些什么区别？protected关键字有何作用？**

a. 公有类型成员用public关键字声明，公有类型定义了类的外部接口
b. 私有类型的成员用private关键字声明，只允许本类的函数成员来访问，而类外部的任何访问都是非法的，这样，私有的成员就整个隐蔽在类中，在类的外部根本就无法看到，实现了访问权限的有效控制。
c. protected用来声明保护类型的成员，保护类型的性质和私有类型的性质相似，其差别在于继承和派生时派生类的成员函数可以访问基类的保护成员。

4. 拷贝构造函数与赋值运算符(=)有何不同？

a. 赋值运算符(=)作用于一个已存在的对象。赋值运算符是将对象值赋给一个已经存在的实例。
b. 拷贝构造函数会创建一个新的对象。拷贝构造函数使用传入对象的值生成一个新对象的实例。

5. 友元函数，全局函数和成员函数的不同

a. 在定义类的时候，如果定义了一个public访问级的函数，那么这个函数就是这个类的成员函数，当然在类里也可以定义private访问级的函数，这种函数则只能叫工具函数而不能叫成员函数，因为他仅仅是为类的公共成员函数服务的。通过类的实例并不能访问类的工具函数。
b. 全局函数是定义在主函数和任何类定义之外的函数，这种函数在整个程序的任何地方都允许被调用。但过多的全局函数会增加程序的尺寸，使其变得慵忡。
c. 因为通过类的实例并不能访问到类的私用数据成员（即private成员），这时如果在类定义之内定义了友元函数，在类的实例中就可以通过友元函数来访问类的私用数据成员。

6. 简述类与对象的关系

a. 类是一个抽象的概念，它不存在于现实中的时间/空间里，类只是为所有的对象定义了抽象的属性与行为。就好像“Person（人）”这个类，它虽然可以包含很多个体，但它本身不存在于现实世界上。
b. 对象是类的一个具体。它是一个实实在在存在的东西。
c. 类是一个静态的概念，类本身不携带任何数据。当没有为类创建任何对象时，类本身不存在于内存空间中。
d. 对象是一个动态的概念。每一个对象都存在着有别于其它对象的属于自己的独特的属性和行为。对象的属性可以随着它自己的行为而发生改变。

7. C++中编译器所生成的目标文件扩展名是什么？

C++程序编译后生成的文件：

- exe ： 可执行文件，点击即可运行
- .ilk ： 当选定渐增型编译链接时，连接器自动生成ILK文件，记录链接信息
- .obj ： 目标文件，obj文件与cpp文件名字一一对应
- .pch ： prcompiled-header,预编译头文件
- .idb ： 文件保存的信息，使编译器在重新编译的时候只重新编译改动过的函数和最新类定义改动过的源文件，这样可提高变异速度

- .pdb ： Program Database，即程序数据库文件，用来记录调试信息
- .dsp ： Developer Studio Project，配置文件
- .ncb ： No Compile Browser，其中存放了供ClassView、WizardBar和Component Gallery使用的信息，由VC开发环境自动生成
- .plg ： 超文本文件，可以用IE打开，记录build的过程
- .cpp ： C++源文件

8. **this指针存在的目的是什么？**

   在面向对象程序设计中，每个非静态成员函数中都包含一个特殊的指针，指向调用该函数的对象，这个指针称为this指针。this是指向实例化对象本身时候的一个指针，里面存储的是对象本身的地址，通过该地址可以访问内部的成员函数和成员变量。

   什么需要this？因为this作用域是在类的内部，自己声明一个类的时候，还不知道实例化对象的名字，所以用this来使用对象变量的自身。在非静态成员函数中，编译器在编译的时候加上this作为隐含形参，通过this来访问各个成员（即使你没有写上this指针）。

   关于this指针的一个经典回答:当你进入一个房子后，你可以看见桌子、椅子、地板等，但是房子你是看不到全貌了。对于一个类的实例来说，你可以看到它的成员函数、成员变量，但是实例本身呢？this是一个指针，它时时刻刻指向你这个实例本身

9. **静态成员函数为什么没有this指针？**

   静态成员函数并不是针对某个类的实例对象，而是属于整个类的，为所有的对象实例所共有。他在作用域的范围内是全局的，独立于类的对象之外的。他只对类内部的静态成员变量做操作。当实例化一个类的对象时候，里面不存在静态成员的。this指针是相当于一个类的实例的指针，this是用来操作对象实例的内容的，既然静态成员函数和变量都是独立于类的实例对象之外的，他就不能用this指针。也不能操作非静态成员。

10. **什么是动态对象数组？**

    动态对象数组：分配空间给对象数组，对象数组就是数组里的**每个元素存储的都是类的对象**，赋值时先定义对象，然后将对象直接赋值给数组。

11. **const修饰的常数据成员，常对象和常成员函数有什么特点？**

    - 常数据成员： const 类型说明符 数据成员

      ```
      const int year
      ```

    - 常对象：const 类 对象

      ```
      const A a(1,2)
      ```

- 常成员函数：类型说明符 函数名（参数表） const

```
void showDate() const
```

- 常成员函数可以访问常数据成员，也可访问普通数据成员。
- 常数据成员可以被常成员函数访问，也可被普通成员函数访问。
- 如果将一个对象说明为一个常对象，则通过该对象只能调用常成员函数，而不能调用普通的成员函数。
- 常成员函数不能更新对象的数据成员，也不能调用该类中的普通成员函数。
- 常对象和常对象成员只能调用常成员函数
- 普通的对象和对象成员却可以调用普通成员函数和常成员函数

12. **struct和class的区别**

	struct	class
默认继承权限	public	private
默认数据访问控制	public	private
模板参数	不能定义	可以用于定义模板参数

相同点：可以有数据成员，方法，构造函数等。

五.数据的共享与保护

1. **作用域？局部变量？全局变量**

作用域：作用域是一个标识符在程序正文中的有效区域。包括四种作用域：

a. 函数原型作用域，例如函数原型声明时的形参，只在形参的左右括号之间起作用。

```
int func(int i)      //i为参数，作用域类型为函数原型类型
```

b. 局部作用域，函数形参列表中形参的作用域，就是从形参列表的声明处开始，到整个函数的结束为止。函数体内部声明的变量，作用范围就是从声明处开始，一直到这个声明所在的大括号为止。

c. 类作用域，类X的成员m具有类作用域，：

```
Screen::pos ht=24,wd=80;  //使用Screen定义的pos类型

Screen scr(ht,wd,' ');

Screen *p=&scr;

char c=scr.get();  //访问scr对象的get成员

c=p->get();   //访问p所指对象的get成员
```

d. 命名空间作用域。分为三类：

i. 一般命名空间

```
namespace A{
    ...
}
using namespace A;
```

ii. 全局命名空间。全局命名空间是默认的命名空间，在显示声明的命名空间之外声明的标识符都在一个全局命名空间中。

iii. 匿名命名空间。匿名命名空间是一个需要显式声明的没有名字的命名空间。

```
namespace {
    ...
}
```

局部变量：具有局部作用域的变量称为局部变量。

全局变量：具有命名空间作用域的变量称为全局变量，作用范围是整个文件。

2. 静态数据成员？

静态数据成员主要解决一个类的不同对象之间数据共享问题的。静态数据成员用static关键字进行声明，与普通数据成员不同的是，普通数据成员在每一个对象中都有一份拷贝，这也是同类的不同对象之间进行区分的主要特征，而静态数据成员在所有对象中只有一份拷贝，由该类的所有对象共同维护。可以说，静态数据成员不属于任何一个对象，而是属于类的，可以通过类名直接进行访问。

3. 什么叫做静态函数成员？它有何特点？

使用static关键字声明的函数成员是静态的，静态函数成员属于整个类，同一个类的所有对象共同维护，为这些对象所共享。静态函数成员具有以下两个方面的好处，

   a. 一是由于静态成员函数只能直接访问同一个类的静态数据成员，可以保证不会对该类的其余数据成员造成负面影响；
   b. 二是同一个类只维护一个静态函数成员的拷贝，节约了系统的开销，提高程序的运行效率。

4. 如果在类模板的定义中有一个静态数据成员，则在程序运行中会产生多少个相应的静态变量？

这个类模板的每一个实例类都会产生一个相应的静态变量。

5. 友元函数？友元类？

   a. 友元函数是通过关键字friend声明的非成员函数，该函数可以直接调用该类的私有成员和保护成员。
   b. 如果一个类被声明为友元类，则它的全部成员函数都是友元函数。

6. 如果类A是类B的友元，类B是类C的友元，类D是类A的派生类，那么类B是类A的友元吗？类C是类A的友元吗？类D是类B的友元吗？

类B不是类A的友元，友元关系不具有交换性；
类C不是类A的友元，友元关系不具有传递性；
类D不是类B的友元，友元关系不能被继承。

7. 什么叫做可见性？可见性的一般规则是什么？

可见性是标识符是否可以引用的问题；
可见性的一般规则是：

   a. 标识符要声明在前，引用在后
   b. 在同一作用域中，不能声明同名的标识符。
   c. 对于在不同的作用域声明的标识符，遵循的原则是：若有两个或多个具有包含关系的作用域，外层声明的标识符如果在内层没有声明同名标识符时仍可见，如果内层声明了同名标识符则外层标识符不可见。

8. 下面的程序的运行结果是什么，实际运行一下，看看与你的设想有何不同。

```
#include <iostream.h>

void myFunction();

int x = 5,  y = 7;

int main()

{
```

```
    cout << "x from main: " << x << "\n";   //5

    cout << "y from main: " << y << "\n\n"; //7

    myFunction();

    cout << "Back from myFunction!\n\n";

    cout << "x from main: " << x << "\n";   //5

    cout << "y from main: " << y << "\n";   //7

    return 0;

}

void myFunction()

{
    // 全局变量被局部变量隐藏
    int y = 10;

    cout << "x from myFunction: " << x << "\n";
//5

    cout << "y from myFunction: " << y << "\n\n";
//10

}
```

```
x from main: 5

y from main: 7

x from myFunction: 5

y from myFunction: 10

Back from myFunction!

x from main: 5

y from main: 7
```

9. 请写出以下结果

```
// 文件1
#include <iostream.h>
#include "fn1.h"
int n;
```

```
void main()
{
    n = 20;
    fn1();
    cout << "n的值为" <<n;
}
// 文件2 fn1.h文件
extern int n;
void fn1()
{
    n=30;
}
```

程序运行输出：
n的值为30

10. **在函数fn1()中定义一个静态变量n，fn1()中对n的值加1，在主函数中，调用fn1()十次，显示n的值。**

```
#include <iostream.h>
void fn1()
{
    // 静态变量初始化只执行一次
    static int n = 0;
    n++;
    cout << "n的值为" << n <<endl;
}
void main()
{
    for(int i = 0; i < 10; i++) fn1();
}
```

程序运行输出：
n的值为1
n的值为2
n的值为3
n的值为4
n的值为5
n的值为6
n的值为7
n的值为8
n的值为9
n的值为10

六. 数组，指针和字符串

1. **指针型函数和指向函数的指针？**

    a. 指针型函数是函数，函数的返回值是指针类型。

b. 指向函数的指针是指针，指针中存放的地址是函数代码的起始地址，也就是函数名。当函数指针指向某个具体的函数后，就可以使用调用函数的方法调用指针。

2. 深拷贝与浅拷贝？默认拷贝构造函数和自定义拷贝构造函数的区别？

   a. 浅拷贝：也就是默认复制构造函数，只是简单的把两个对象的数据元素一一复制，如果数据元素中存在指针等数据成员，浅拷贝可能会造成两个对象的数据成员指向同一个地址，并没有创造真正的副本，在析构函数释放内存空间时会产生错误。
   b. 深拷贝：为新对象的数据成员开辟新的地址空间，与已有对象的地址空间区分开来。

3. 常量指针与指针常量

   a. 常量指针：指向常量的指针，不能通过指针修改所指向对象的值，但是可以改变指针指向其他区域
   b. 指针常量：指针类型的常量，指针始终指向那一块内存区域，内存区域内的值可以被修改

4. **const int * p1 和 int * const p2的区别是什么？**

   a. const int * p1 声明了一个指向整型常量的指针p1，因此不能通过指针p1来改变它所指向的整型值
   b. int * const p2声明了一个指针型常量，用于存放整型变量的地址，这个指针一旦初始化后，就不能被重新赋值了，始终指向一块地址不能改变。

5. 已知有一个数组名叫**oneArray**，用一条语句求出其元素的个数。

```
nArrayLength =
sizeof(oneArray)/sizeof(oneArray[0]);
```

6. 运算符*和&的作用是什么？

   a. *称为指针运算符，是一个一元操作符，表示指针所指向的对象的值
   b. &称为取地址运算符，也是一个一元操作符，是用来得到一个对象的地址。

7. 什么叫做指针？指针中储存的地址和这个地址中的值有何区别？

   指针是一种数据类型，具有指针类型的变量称为指针变量。指针变量存放的是另外一个对象的地址，这个地址中的值就是另一个对象的内容。

8. 定义一个整型指针，用**new**语句为其分配包含**10**个整型元素的地址空间。

```
int *pInteger = new int[10];
```

9. 在字符串”Hello，world!”中结束符是什么？

   `\0`字符。

10. 引用和指针有何区别？何时只能使用指针而不能使用引用？

    引用变量是一个别名，也就是说，它是某个已存在变量的另一个名字。一旦把引用初始化为某个变量，就可以使用该引用名称或变量名称来指向变量。引用很容易与指针混淆，它们之间有三个主要的不同：

    a. 不存在空引用。引用必须连接到一块合法的内存。
    b. 一旦引用被初始化为一个对象，就不能被指向到另一个对象。指针可以在任何时候指向到另一个对象。
    c. 引用必须在创建时被初始化。指针可以在任何时间被初始化。

11. 程序中定义一个**double**类型变量的指针。分别显示指针占了多少字节和指针所指的变量占了多少字节。

```
  double *counter;

  cout << "\nSize of pointer == "sizeof(counter);
  cout << "\nSize of addressed value == "
<<sizeof(*counter)
```

均为8字节，因为我的电脑64位系统，因此地址为64位

12. 下列程序有何问题，请仔细体会使用指针时应避免出现这个的问题。

```
#include <iostream.h>
int main()
{
    int *p;
    *p= 9;
    cout << "The value at p: " << *p;
    return 0;
}
```

指针p没有初始化，也就是没有指向某个确定的内存单元，它指向内存中的一个随机地址，给这个随机地址赋值是非常危险的。

13. 下列程序有何问题，请改正；仔细体会使用指针时应避免出现的这个问题。

```
#include <iostream.h>
int Fn1();
int main()
{
    int a = Fn1();
    cout << "the value of a is: " << a;
```

```
    return 0;
}
int Fn1()
{
    int * p = new int (5);
    return *p;
}
```

解：

此程序中给*p分配的内存没有被释放掉。

改正：

```
#include <iostream.h>
int* Fn1();
int main()
{
    int *a = Fn1();
    cout << "the value of a is: " << *a;
    delete a;
    return 0;
}
int* Fn1()
{
    int * p = new int (5);
    return p;
}
```

14. 声明一个参数为整型，返回值为长整型的函数指针；声明类A的一个成员函数指针，其参数为整型，返回值长整型。

```
long (* p_fn1)(int);
long ( A::*p_fn2)(int);
```

15. 什么是指针数组和数组指针？

指针数组：指针数组可以说成是”指针的数组”，首先这个变量是一个数组，其次，”指针”修饰这个数组，意思是说这个数组的所有元素都是指针类型，在32位系统中，指针占四个字节。

数组指针：数组指针可以说成是”数组的指针”，首先这个变量是一个指针，其次，”数组”修饰这个指针，意思是说这个指针存放着一个数组的首地址，或者说这个指针指向一个数组的首地址。

```
  const int N=10;
  char *s1[N];  //指针数组，[]优先级高于*，因此先结合为数组
  char (*s2)[N];    //数组指针，()优先级高，先结合为指针
```

16. **malloc，calloc，realloc，free的区别**

    - malloc，calloc，realloc，free属于C函数库，new/delete则是C++函数库；
    - 多个-alloc的比较：
        - alloc：唯一在栈上申请内存的，无需释放；
        - malloc：在堆上申请内存，最常用；
        - calloc：malloc并初始化为0；
        - realloc：将原本申请的内存区域扩容，参数size大小即为扩容后大小，因此此函数要求size大小必须大于ptr内存大小。

17. **在64位系统下，分别定义如下两个变量，请问，sizeof(p)和sizeof (p1)分别值为？**

```
char *p[10];
char(*p1)[10];
```

    重点理解p跟谁结合了，跟[]结合，则p就是一个数组;跟*结合，p就是一个指针。

    首先的优先级一样，均大于*

    - char *p[10]，p与[]结合，所以p就是一个数组，数组的元素比较特殊，是指针，为数组指针，指针大小为8，所以是10*8=80；
    - char(*p1)[10]，与*结合，所以是一个指针，为指针数组，大小为8。

18.

七. 继承与派生

1. **什么是继承？派生？**

    a. 类的继承：新的类从已有类那里得到已有的特性
    b. 类的派生：从一个已有类产生新类

2. **派生类生成过程？**

    吸收基类成员、改造基类成员、增加新的成员

3. 三种继承方式的区别？

    a. public:使得基类中的public和protected成员在派生类中的访问权限不变，但private成员在派生类中不可访问
    b. private：使得基类中的public和protected成员在派生类中变为private成员，而基类中的private成员在派生类中不可访问
    c. protected：使得基类中的public和protected成员在派生类中变为protected成员，而基类中的private成员在派生类中不可访问

    private与protected的主要区别在于：通过private方式继承的派生类，如果继续派生，那么新产生的派生类不能访问间接基类的任何数据元素。而protected方式继承的派生类，如果继续派生，则新产生的派生类有可能访问间接基类中的public和protected成员

4. 派生类构造函数的执行顺序

    a. 如果该类由直接或者间接的虚基类，则先执行虚基类的构造函数
    b. 该类还有其他基类，则按照这些基类在继承声明的顺序依次调用构造函数
    c. 按照类定义中出现的顺序，对新增数据成员对象进行初始化
    d. 执行构造函数的函数体

5. **如果在派生类B已经重载了基类A的一个成员函数fn1()，没有重载成员函数fn2()，如何调用基类的成员函数fn1()、fn2()？**

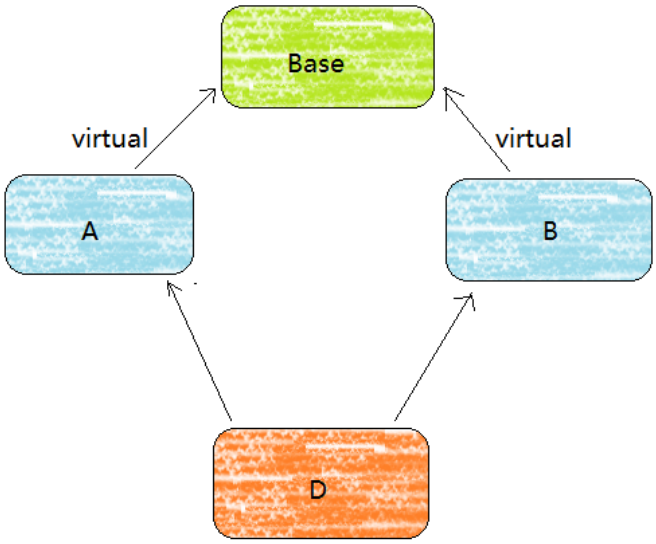
调用方法为：

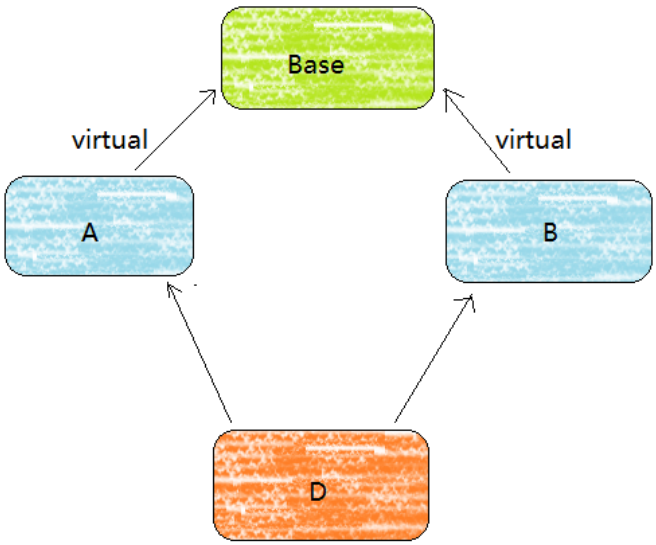
```
A::fn1();
fn2();
```

6. 虚基类？有什么作用

    当某类的部分或者全部基类都是从另一个基类中继承而来时，这些直接基类中，从上一层基类中继承的成员就具有相同的名称，派生类的对象的同名成员在内存中拥有多个拷贝，此时，可以使用作用域分辨符来唯一标识它们。也可以将共同基类声明为虚基类，这样虚基类的成员在内存中就只有一个副本，解决了同名成员的唯一标识问题。

    菱形继承





7. 组合和继承有什么共同点和差异？

a. 组合相当于一个类的数据成员是另一个类的对象，是整体与部分之间的关系。例如一辆汽车由轮子和发动机组成，汽车是整体，而轮子和发动机都是汽车的一部分。
b. 继承是从一般到特殊的过程，一般来说，越底层的类越抽象，包含了某类物体的共同特征，比如学生是一个抽象类，而高中生，大学生是具体的类，这就是从一般到特殊的过程。

8. 派生类不能继承基类的哪些方法？

一个派生类继承了所有的基类方法，但下列情况除外：

- 基类的构造函数、析构函数和拷贝构造函数。
- 基类的重载运算符。
- 基类的友元函数。

八、多态性

1. 多态？

同样的消息被不同对象接受后产生不同的行为。简单来说，就是同一段程序代码所传递的实参不同，最后得到的结果也不同

2. 多态的类型？

a. 重载多态：普通函数以及类的成员函数的重载都属于重载多态。运算符重载也属于函数重载，是重载多态
b. 强制多态：将一个变量的类型强制改变，以满足程序或者操作的要求
c. 包含多态：定义于不同类的同名函数的多态行为，主要通过虚函数来实现。
d. 参数多态：程序所处理的对象的类型参数化，使得同一段程序可以处理不同类型的对象

3. 从实现的角度，多态可以分为哪几类？

a. 编译时的多态：在程序编译的过程中就已经确定了同名函数的具体操作对象
b. 运行时的多态：在程序运行过程中才动态确定了函数的具体操作对象

4. 为什么要使用多态？

- 可替换性：多态对已存在的代码具有可替换性
- 可扩充性：增加新的子类并不影响已存在类的多态性、继承性以及其他特性的运行和操作
- 接口性：多态是父类（超类）通过方法签名，向子类提供了共同的接口，子类可以通过覆写完善或者覆盖这个接口。
- 灵活性：在应用中体现了灵活多样的操作，提高了使用效率

5. 静态绑定？动态绑定？

a. 绑定：计算机程序自身彼此关联的过程，也是一条消息和一个对象的方法相结合的过程。
b. 静态绑定：绑定工作在编译连接阶段就完成的情况称之为静态绑定。即在编译、链接的过程中，系统根据类型匹配及最佳个数匹配等原则确定某一个同名标识符到底调用哪一段代码。例如重载、强制和参数多态都属于静态绑定
c. 动态绑定：绑定工作在程序运行阶段才完成的情况就是动态绑定。编译时无法解决的绑定问题，等到程序运行之后才能明确具体的操作对象。典型的动态绑定是包含多态对象的确定，主要是通过虚函数来实现的。包含多态的操作对象的确定就是通过动态绑定完成的。

6. 运算符重载？

对已有运算符赋予新的含义，使得同一个运算符面对不同类型的对象时产生不同的结果。

7. 运算符重载的形式？

a. 重载为类的非静态成员函数。如果是重载为成员函数，则函数的参数要比原来的操作数的个数少一个，因为函数体把该类对象作为第一个操作数，隐藏了。后置运算符的形参列表有一个整型形参，用来区分前置++，--与后置的++，--
b. 重载为类的非成员函数，如果需要访问类的私有成员，可以把该函数声明为友元函数。例如重载输出流运算符，就必须将该重载函数声明为友元函数，因为该函数的左操作数是输出流对象，而不是本类的对象。

8. 运算符重载需要注意的要点。

a. 不是所有的运算符都能被重载。
b. 重载不能改变运算符的优先级和结合性
c. 重载不会改变运算符的用法，原有有几个操作数、操作数在左边还是在右边，这些都不会改变
d. 运算符重载函数不能有默认的参数
e. 运算符重载函数既可以作为类的成员函数，也可以作为全局函数
f. 箭头运算符->、下标运算符[ ]、函数调用运算符( )、赋值运算符=只能以成员函数的形式重载

9. C++中哪些运算符不能重载？

不能被重载的运算符有：

- 作用域操作符： ::
- 条件操作符： ?:
- 点操作符： .
- 指向成员操作的指针操作符：->*　.*
- 预处理符号：#

10. 虚函数的实现方式？

根据赋值兼容的规则，可以用基类类型的指针指向派生类的对象，如果派生类中有与基类同名的函数，那么通过基类类型的指针访问该同名函数，只能访问到基类的函数。但是如果将基类的同名函数设置为虚函数，就可以使用基类类型的指针就可以访问到指针指向的派生类对象的同名函数。因此，通过基类指针指向不同的派生类，就可以实现不同的操作，从而实现了动态绑定。

1. 类之间满足赋值兼容原则
2. 声明虚函数
3. 由成员函数调用或者通过指针、引用来访问虚函数

11. 抽象类？纯虚函数？抽象类的派生类是否一定要给出纯虚函数的实现？

a. 纯虚函数：在基类中声明的虚函数，它没有定义具体的操作内容，需要派生类根据实际需求自己给出定义。纯虚函数的函数体由派生类给出。

b. 抽象类：带有纯虚函数的类是抽象类。抽象类的作用是它为一个类群的派生类提供一个公共的接口，使它们能够有效的发挥多态特性。
c. 但抽象类的派生类并非一定要给出纯虚函数的实现，如果派生类没有给出纯虚函数的实现，这个派生类仍然是一个抽象类。

11. 为什么构造函数不能是虚函数？而析构函数可以

a. 虚函数是动态绑定的基础，主要是针对对象的，而构造函数是在创建对象之前调用的，所以虚构造函数没有存在的意义。
b. 但是析构函数是在对象消亡前做一些清理工作，如果一个类的析构函数是虚函数，那么由这个类派生的所有类的构造函数也都是虚函数，这样就能保证使用基类指针指向不同的派生类对象进行不同的清理工作。

12. 重载，覆盖，隐藏的区别？

成员函数被重载的特征

a. 相同的范围（在同一个类中）；
b. 函数名字相同；
c. 参数不同；
d. virtual 关键字可有可无。

覆盖是指派生类函数覆盖基类函数

a. 不同的范围（分别位于派生类与基类）；
b. 函数名字相同；
c. 参数相同；
d. 基类函数必须有virtual 关键字。

隐藏是指派生类的函数屏蔽了与其同名的基类函数

a. 如果派生类的函数与基类的函数同名，但是参数不同。此时，不论有无virtual关键字，基类的函数将被隐藏（注意别与重载混淆）。
b. 如果派生类的函数与基类的函数同名，并且参数也相同，但是基类函数没有virtual 关键字。此时，基类的函数被隐藏（注意别与覆盖混淆）

九、模板

1. 模板？

采用类型作为参数的程序设计方式，主要包括函数模板和类模板，用户可以编写与类型无关的代码，更好的实现代码重用。

2. 模板函数和函数模板、类函数和模板类的区别？

在 C++ 中，模板分为函数模板和类模板两种。函数模板是用于生成函数的，类模板则是用于生成类的。

a. 函数模板是模板，模板函数时具体的函数
b. 类模板是模板，模板类时具体的类
c. 由函数模板实例化而得到的函数称为模板函数
d. 由类模板实例化得到的类叫模板类

3. 什么是类模板

类模板是对不同类的公共性质的进一步抽象，使得类中某些数据成员，函数成员的参数、返回值或者局部变量可以取任意的数据类型，是一种参数化类，只有对类模板实例化生成类之后，才能创建具体的对象。

4. 什么是函数模板？

十、STL及相关概念

1. 什么是**STL**

支持C++泛型编程的模板库，可以利用已经编译好的模板实现自定义程序功能

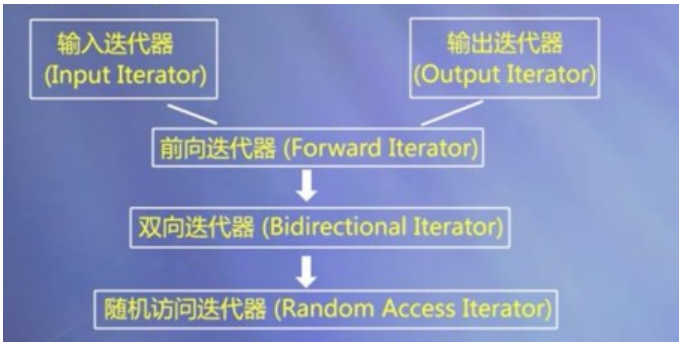
2. 什么是容器

存储一些列数据元素的对象

3. 什么是迭代器

迭代器是一种泛化的指针，可以对容器中的元素序列进行遍历，而不需要知道这个容器中的元素在内存中的具体位置迭代器的分类：

a. 输入迭代器：可以从序列中读取数据
b. 输出迭代器：可以向序列中写入数据
c. 前向迭代器：既包含输入迭代器，有包含输出迭代器，但是只允许对序列进行单向遍历
d. 双向迭代器：在前向迭代器的基础上，可以对序列进行双向遍历
e. 随机迭代器：在双向迭代器的基础上，支持向前或者向后移动n个数据元素，可以做到和指针一样的功能



4. 关联容器

关联容器每个元素都有一个键，可以根据键高效的查找对应的元素按照关键值是否唯一:单重关联容器与多重关联容器

按照键和元素之间的关系：简单关联容器与二元关联容器

5. 集合

集合可以用来存储没有重复的元素，而且这些元素在集合中是有序排列的，可以高效的查找到某个指定的元素，同时，也可以查找到指定范围大小的元素所处的区间

6. 有哪些方法可以让函数返回多个值？请举例说明

    a. 用引用传参的方式将需要修改的变量作为实参传递给函数的形参，函数内部对形参的任何操作都将直接作用于实参。
    b. 通过全局变量的方式，函数共享处于文件作用域的全局变量，函数体内对全局变量的修改是有效的
    c. 通过数组指针。使用数组指针作为函数的形参，函数体内对数组的任何操作都将直接作用于内存的数组空间
    d. 将需要返回的多个值组合成数组或者字符串，函数的返回值设置为数组或字符串

7. 什么是容器

容器时包含一组元素的对象。主要分为两种基本类型，一个是顺序容器，它是将一组相同类型的元素以线性的方式组织起来。另一个是关联容器，每个元素都有对应的关键字，可以通过关键字迅速提取相应的元素。

8. 什么是函数对象

函数对象是一种泛化的函数，可以没有参数，也可以有多个参数，任何普通的函数或者是重载了调用运算符operator( )的类的对象都可以视为函数对象，它的作用是获取一个值或者是改变操作的状态。

9. 什么是STL中的算法？

算法本身就是一种函数模板，能够完成一定的功能。大致可分为不可变序列算法、可变序列算法、排序和搜索算法、数值算法。

10. 什么是函数适配器？

函数适配器实现了，将一种函数对象转化为另一种符合要求的函数对象。

11. vector的特点？

> Vector是STL的常见容器之一，vector是表示可变大小数组的序列容器。vector采用的连续存储空间来存储元素。因此，它可以像数组一样用下标对元素进行操作，但它与数组最大的不同就是，它的**大小可以改变**，并且是自主控制容量的变化。与array相似，但array是静态空间，一旦配置了就不能改变；vector是动态空间，随着元素的加入，它的内部机制会自行扩充空间以容纳新元素，比数组高效很多。 Vector 所有的方法都使用了 synchronized 修饰符，即：Vector 线程安全但是性能较低，适用于多线程环境。

12. **Set和Vctor的区别**

SET

> Set是一种 关联容器 ，它用于存储数据，并且能从一个数据集合中取出数据。它的每个元素的值必须唯一，而且系统会根据该值来自动将数据排序 。每个元素的值不能直接被改变。 **内部结构采用红黑树的平衡二叉树** 。multiset 跟set 类似，唯一的区别是允许键值重复。

VECTOR

vector就是动态数组。它也是在堆中分配内存,元素连续存放,有保留内存,如果减少大小后，内存也不会释放.如果新值大于当前大小时才会再分配内存。
它拥有一段连续的内存空间，并且起始地址不变，因此它能非常好的支持随即存取，即[]操作符，但由于它的内存空间是连续的，所以在中间进行插入和删除会造成内存块的拷贝，另外，当该数组后的内存空间不够时，需要重新申请一块足够大的内存并进行内存的拷贝。这些都大大影响了vector的效率。

> *红黑树的操作时间跟二叉查找树的时间复杂度是一样的，执行查找、插入、删除等操作的时间复杂度为O（logn）。*

> *红黑树是特殊的AVL树，遵循红定理和黑定理*
> *红定理：不能有两个相连的红节点*
> *黑定理：根节点必须是黑节点，而且所有节点通向NULL的路径上，所经过的黑节点的个数必须相等*

十一、 流类库与输入输出

1. **什么叫做流？流的提取和插入是指什么？I/O流在C++中起着怎样的作用？**

流是一种抽象，它负责在数据的生产者和数据的消费者之间建立联系，并管理数据的流动，一般意义下的读操作在流数据抽象中被称为（从流中）提取，写操作被称为（向流中）插入。操作系统是将键盘、屏幕、打印机和通信端口作为扩充文件来处理的，I/O流类就是用来与这些扩充文件进行交互，实现数据的输入与输出。

2. **cerr、clog和cout有何区别？**

    a. cerr 标准错误输出，没有缓冲，发送给它的内容立即被输出，适用于立即向屏幕输出的错误信息；
    b. clog 类似于cerr，但是有缓冲，缓冲区满时被输出，在向磁盘输出时效率更高。
    c. cout是标准输出流。

3. **使用I/O流以文本方式建立一个文件test1.txt，写入字符“已成功写入文件！”，用其它字处理程序（例如windows的记事本程序Notepad）打开，看看是否正确写入。**

```
#include <fstream.h>

void main()

{

ofstream file1("test.txt");

file1 << "已成功写入文件！";

file1.close();

}
```

4. **使用I/O流以文本方式打开上一题建立的文件test1.txt，读出其内容显示出来，看看是否正确。**

```
#include <fstream.h>

void main()

{

char ch;

ifstream file2("test.txt");

while (file2.get(ch))

cout << ch;
```

```
file2.close();

}
```

5. **iostream.h、stdio.h、fstream.h，stdlib.h这几个头文件的作用是什么？**

c/c++的常用头文件

```
//标准C的头文件

#include <ctype.h>//字符处理

#include <errno.h>//定义错误码

#include <float.h>//浮点数处理

#include <fstream.h>//文件输入/输出

#include <iomanip.h>//参数化输入/输出

#include <iostream.h> //数据流输入/输出

#include <limits.h> //定义各种数据类型最值常量

#include <locale.h> //定义本地化函数

#include <math.h> //定义数学函数

#include <stdio.h> //定义输入/输出函数

#include <stdlib.h> //定义杂项函数及内存分配函数

#include <string.h> //字符串处理

#include <strstrea.h> //基于数组的输入/输出

#include <time.h> //定义关于时间的函数

#include <wchar.h> //宽字符处理及输入/输出

#include <wctype.h> //宽字符分类
```

6. **输入输出格式控制的方法有哪些？**

使用控制符控制输出格式：

```
#include <iomanip>
```

```
cout<<"dec:"<<dec<<a<<endl; //以十进制形式输出整数
cout<<"hex:"<<hex<<a<<endl; //以十六进制形式输出整数a
cout<<"oct:"<<setbase(8)<<a<<endl; //以八进制形式输出整数a
cout<<setw(10)<<pt<<endl; //指定域宽为,输出字符串
cout<<setfill('*')<<setw(10)<<pt<<endl; //指定域宽,输出字符串,空白处以'*'填充
```

用流对象的成员函数控制输出格式：例：

```
cout.setf(ios::hex); //设置以十六进制输出的状态
cout<<"hex:"<<a<<endl; //以十六进制形式输出a
cout.unsetf(ios::hex); //终止十六进制的格式设置
cout.width(10); //指定域宽为
cout.fill('*'); //指定空白处以'*'填充
cout<<pt<<endl; //输出字符串
```

7. I/O流类可以大致分为哪几类？

   a. 通用输入输出流类
   b. 文件输入输出流类
   c. 字符串输入输出流类

十二、 异常处理

1. 什么叫做异常？什么叫做异常处理？

当一个函数在执行的过程中出现了一些不平常的情况，或运行结果无法定义的情况，使得操作不得不被中断时，我们说出现了异常。异常通常是用throw关键字产生的一个对象，用来表明出现了一些意外的情况。我们在设计程序时，就要充分考虑到各种意外情况，并给与恰当的处理。这就是我们所说的异常处理。

2. C++的异常处理机制有何优点？

C++的异常处理机制使得异常的引发和处理不必在同一函数中，这样底层的函数可以着重解决具体问题，而不必过多地考虑对异常的处理。上层调用者可以在适当的位置设计对不同类型异常的处理。

3. 举例throw 、try、catch语句的用法？

throw 语句用来引发异常，用法为：

```
throw 表达式;
```

例如： throw 1.0E-10;

catch语句用来处理某中类型的异常，它跟在一个try程序块后面处理这个try程序块产生的异常，如果一个函数要调用一个可能会引发异常的函数， 并且想在异常真的出现后处理异常，就必须使用try语句来捕获异常。

```
try{

语句 //可能会引发多种异常

}

catch（参数声明1）

{

语句 //异常处理程序

}
```

4. C++异常处理中的构造与析构？

   找到一个匹配的catch异常处理后，如果catch子句的异常声明是一个值参数，则其初始化方式是拷贝构造函数，如果是引用，则使用该引用指向异常对象。

   C++具有在异常抛掷前构造的所有局部对象自动调用析构函数的能力。

   异常抛出后，从进入try块起，到异常抛掷前，这期间在栈上构造的所有对象都会被自动析构，析构的顺序与构造顺序相反。这一过程被称为栈的解旋（unwinding）。

十三、 MFC类库与Windows程序开发简介

1. **MS-DOS环境下的C++程序中，main()函数必不可少，在Windows程序中，什么函数代替了main()函数，它有何特点？**

   Windows程序中替代main()函数是WinMain()函数，每一个Windows程序都需要有一个WinMain()函数，该函数主要是建立应用程序的主窗口。与MS-DOS程序的根本差别在于：MS-DOS程序是通过调用操作系统的功能来获得用户输入的，而Windows程序则是通过操作系统发送的消息来处理用户输入的，程序的主窗口中需要包含处理Windows所发送消息的代码。

2. **什么叫做类库？**

   类库是一个可以在应用程序中使用的相互关联的C++类的集合。

3. **当我们用应用程序向导生成MFC应用程序时，在源代码中找不到WinMain()函数，这是为什么？**

当使用应用程序向导生成MFC应用程序时，WinMain()函数已被封装在MFC类库中了。对于大多数Windows程序，都必须从CwinApp类派生出自己的应用程序类，WinMain()就封装在CwinApp类里。

T2 C语言程序设计

一、 往年真题

1. （2011）水仙花数。

```c
int isNarcNum(int n) {
    if (n < 100 || n>999) return 0;
    int temp = n;
    int sum = 0;
    while (temp != 0) {
        int rem = temp % 10;
        sum += pow(rem, 3);
        temp /= 10;
    }
    return sum == n ? 1 : 0;
}
```

2. （2012）对称数或回文数？

```c
int isHui(int num){
    int copy_num=num;
    int sum=0;
    while(copy_num!=0){
        int tmp=copy_num%10;
        sum=sum*10+tmp;
        copy_num=copy_num/10;
    }
    return num==sum?True:False;
}
```

3. （2012）质数？

```
bool isPrime(int num){
    for(int i=2;i<=sqrt(num);i++){
        if(num%i==0){
            return false;
        }
    }
    return true;
}
```

4. （**2012**）因式分解？

```
void devideFactor(int num){
    cout<<num<<"=";
    for(int i=2;i<=num;i++){
        while(num%i==0){
            cout<<i;
            num/=i;
            if(num!=1){
                cout<<"*";
            }
        }
    }
    cout<<endl;

}
```

5. （**2012**）统计词频

```
const int N=128;
int* strFrequency(char s[]){
    int* count=(int *)malloc(sizeof(int)*N);
    memset(count,0,sizeof(int)*N);
    for(int i=0;i<strlen(s);i++){
        count[s[i]]++;
    }
    return count;
}
```

6. （**2012**）二进制转十进制

```
#include<iostream>
```

```cpp
#include<string>
using namespace std;
int main(){
    int num=0;
    cout<<"请输入一个整数：";
    cin>>num;
    string line;
    while(num!=0){
        int tmp=num%2;
        // 头插法，逆序生成，先出来的是低位
        line.insert(0,tmp?"1":"0");
        num/=2;
    }
    cout<<line<<endl;
}
```

```cpp
template<typename T>
void Reverse(vector<T> &v){
    int left=0,right=v.size()-1;
    T tmp;
    while(left<right){
        tmp=v[left];
        v[left++]=v[right];
        v[right--]=tmp;
    }
}

char Int2Char(int num){
    if(num>=0 && num<=9){
        return num+'0';
    }
    else{
        return num-10+'A';
    }
}
// 十进制转化为N进制
void ConvertT2N(int num,int k){
    vector<char> ans;
    if(num==0){
        ans.push_back('0');
    }
    while(num!=0){
        // 先出来的是低位
        ans.push_back(Int2Char(num%k));
        num/=k;
    }
    Reverse(ans);   //反转
    for(int i=0;i<ans.size();i++){
        cout<<ans[i];
    }
```

```
//  for(int i=ans.size()-1;i>=0;i--){
//      cout<<ans[i];
//  }
}
```

```
int Char2Int(char t){
    if(t>='0' && t<='9'){
        return t-'0';
    }
    else{
        return t-'A'+10;
    }
}
// M进制转化为十进制
int ConvertM2T(string num,int k){
    int sum=0;
    //num[num.size()-1]为个位
    for(int i=0;i<num.size();i++){
        sum=k*sum+Char2Int(num[i]);
    }
    return sum;
}
```

7. （2013）砝码问题

```
#include<iostream>
using namespace std;

#define MAX_SIZE 30
int fama[MAX_SIZE]={0};
int* count=0;
int main(){
    int MAX=0;
    fama[1]=5,fama[2]=3,fama[3]=2;
    fama[5]=2,fama[10]=1,fama[20]=1;
    for(int i=1;i<MAX_SIZE;i++){
        MAX+=fama[i]*i;
    }
    count=new int[MAX];
    for(int i=0;i<MAX;i++){
        count[i]=0;
    }
    for(int i=0;i<=fama[1];i++){
        for(int j=0;j<=fama[2];j++){
            for(int k=0;k<=fama[3];k++){
                for(int l=0;l<=fama[5];l++){
                    for(int m=0;m<=fama[10];m++){
```

```
                        for(int
n=0;n<=fama[20];n++){
                            int
tmp=1*i+2*j+3*k+5*l+10*m+20*n;
                            if(tmp>0){
                                count[tmp]++;
                            }
                        }
                    }
                }
            }
        }
    }


    for(int i=1;i<MAX;i++){
        if(count[i]>0){
            cout<<i<<"有"<<count[i]<<"种组合"
<<endl;
        }
        cout<<i<<"有"<<count[i]<<"种组合"<<endl;
    }
    return 0;

}
```

8. （2013）字符串比较，输入n个由'0'和'1'组成的字符串排序输出，规定比较规则如下：

a. 字符串长的比较大
b.字符串等长的含'0'多比较小
c.字符串等长，晗1相等，该字符串相等

```
#include<iostream>
#include<cstring>
using namespace std;
// 获得0的数量
int getZeroNumbs(char *a){
    int count=0;
    for(int i=0;i<strlen(a);i++){
        if(a[i]=='0'){
            count++;
        }
    }
    return count;
}
//字符串比较
int StringCompare(char *a,char *b){
    int len_a=strlen(a),len_b=strlen(b);
    if(len_a==len_b){
```

```
        int
zero_a=getZeroNumbs(a),zero_b=getZeroNumbs(b);
        if(zero_a==zero_b){
            return 0;   //字符串相等
        }
        else{
            return zero_a>zero_b?-1:1;  // 0多则字
符串小
        }
    }
    else{
        return len_a>len_b?1:-1;    // 长度大，则字
符串大
    }
}
//冒泡排序
void BubbleSort(char *a[],int n){
    char *tmp;
    for(int i=0;i<n;i++){
        for(int j=n-1;j>i;j--){
            if(StringCompare(a[j],a[j-1])>0){
                tmp=a[j];
                a[j]=a[j-1];
                a[j-1]=tmp;
            }
        }
    }
}

int main(){
    char *a[5]=
{"011100","1100101","111","000000","0001100000"};
    BubbleSort(a,5);
    for(int i=0;i<5;i++){
        cout<<a[i]<<endl;
    }

}
```

9. （2014）杨辉三角

```
#include <iostream>
using namespace std;

int main(int argc, char *argv[])
{
    int a[100] = {0,1}; //初始化数组 次序0，1分别初始
化为0，1
```

```
    int n=10;   //层数
    int l, r;   //存放上一层左边的数和右边的数
    for(int i = 1; i <= n; i++)
    {
        // 每一行都要初始化为0
        l = 0;
        for(int j = 1; j <= i ; j++)
        {
            r = a[j];                //面试题空白处
            a[j] = l + r;            //面试题空白处
            cout<<a[j]<<" ";
            l = r;
        }
        cout<<endl;
    }
}
```

10. （**2014**）大数相加

```
string add(string a, string b){
    string res;
    //个位在a[a_len-1]
    int a_len=a.size();
    int b_len=b.size();
    int max_len=max(a_len,b_len);
    int flag=0; // 进位
    int num1,num2;
    for(int i=0;i<max_len;i++){
        // 取出对应位的数
        num1=a_len-1-i>=0?a[a_len-1-i]-'0':0;
        num2=b_len-1-i>=0?b[b_len-1-i]-'0':0;
        int tmp=num1+num2+flag;
        int val=tmp%10;
        flag=tmp/10;
        // 头插反向
        res.insert(0,1,val+'0');
    }
    if(flag==1){
        res.insert(0,1,flag+'0');
    }
    return res;
}
```

```
// 推荐
void add(string a, string b, string &res){
```

```
    // 指向个位
    int pa=a.size()-1, pb=b.size()-1;
    // 中间变量
    int numa,numb,tmp,val;
    // 进位
    int carry=0;
    while(pa>=0 || pb>=0){
        // 如果pa>=0则赋值对应的数值，否则赋值0
        numa=pa>=0?a[pa--]-'0':0;
        numb=pb>=0?b[pb--]-'0':0;
        // 求和
        tmp=numa+numb+carry;
        // 得到个位
        val=tmp%10;
        carry=tmp/10;
        // 对res头插，位置0处插入1个val+'0'
        res.insert(0,1,val+'0');

    }
    // 如果还有进位，则要再添加
    while(carry>0){
        val=carry%10;
        carry/=10;
        res.insert(0,1,val+'0');
    }
}
```

11. （2017）随机生成10个不重复的数

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;

#define MAX_SIZE 100

bool array[MAX_SIZE]={false};   //用于判断该数是否已经使用

int main(){
    srand(time(0));
    int n=10;
    for(int i=0;i<n;i++){
        int index=rand()%MAX_SIZE;
        if(array[index]==false){
            array[index]=true;
            cout<<index<<" ";
        }
```

```
        else{
            i--;
        }
    }
    return 0;

}
```

```
// 更好
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;

// 生成0-100的正整数
const int N = 100;
// 生成10个
const int nums=10;

int record[nums];

bool isInRecord(int n,int len){
    for(int i=0;i<len;i++){
        if(record[i]==n){
            return true;
        }
    }
    return false;
}

int main(){
    srand(time(0));
    int count=0,val;
    while(count<nums){
        val=rand()%N;
        // 如果不在record ，则添加数据
        if(!isInRecord(val,count)){
            record[count++]=val;
        }
    }
    // 输出
    for(int i=0;i<nums;i++){

        cout<<record[i]<<"\t";
    }
    return 0;

}
```

12. （2019）99可由多少组（100以内）相加而得。

```
#include<iostream>
using namespace std;

int main(){
    int n=99;
    int count=0;
    // 注意i==j也要包括
    for(int i=0,j=n;i<=j;i++,j--){
        count++;
    }
    cout<<count;


    return 0;

}
```

13. （2019）汉诺塔

```
#include<iostream>

using namespace std;

void hanoi(int n,char a,char b,char c);
inline void move(char a,char b);

int main(){
    int n;
    cout<<"请输入盘子个数：";
    cin>>n;
    hanoi(n,'A','B','C');
    return 0;

}

void hanoi(int n,char a,char b,char c){
    if(n==1) move(a,c);
    else{
        hanoi(n-1,a,c,b);
        move(a,c);
        hanoi(n-1,b,a,c);
    }
}
```

```
inline void move(char a,char b){
    cout<<a<<"->"<<b<<endl;
}
```

14. **(2019)**生成密码串

```
#include<iostream>
#include<ctime>
#include<cstdlib>
using namespace std;

int main(){
    int len=6;
    int count[3]={0};
    srand(time(NULL));

    count[0]=rand()%4+1;    //小写字母个数
    count[1]=rand()%(len-count[0]-1)+1;     //大写
字母个数
    count[2]=len-count[0]-count[1]; //数字个数

//  cout<<count[0]<<" "<<count[1]<<" "<<count[2];
    while(true){
        int tmp=rand()%128;
        if('a'<tmp && 'z'>tmp &&count[0]>0 ){
            cout<<char(tmp);
            count[0]--;
        }
        else if('A'<tmp && 'Z'>tmp &&count[1]>0 )
{
            cout<<char(tmp);
            count[1]--;
        }
        else if('0'<tmp && '9'>tmp &&count[2]>0 )
{
            cout<<char(tmp);
            count[2]--;
        }

        if(count[0]+count[1]+count[2]==0){
            break;
        }
    }
    return 0;

}
```

二、 C++程序设计课后题

1. 编写函数求两个整数的最大公约数和最小公倍数。

```
int GDC(int a,int b){
    int tmp;
    while(b!=0){
        tmp=a%b;
        a=b;
        b=tmp;
    }
    return a;
}
```

```
int GDC(int a,int b){
    if(b==0)
        return a;
    else
        return GDC(b,a%b);
}
```

```
int LCM(int a,int b){
    return a*b/GDC(a,b);
}
```

2. 在主程序中提示输入整数**n**，编写函数用递归的方法求**1 + 2 + … + n**的值。

```
int sumN(int n){
    int sum=0;
    for(int i=1;i<=n;i++){
        sum+=i;
    }
    return sum;
}
```

```
int sumN(int n){
    if(n==1){
        return 1;
    }
    else{
        return sumN(n-1)+n;
    }
}
```

3. 编写递归函数`GetPower(int x, int y)`计算x的y次幂，在主程序中实现输入输出。

```
int GetPower(int x,int y){
    int pow=1;
    for(int i=0;i<y;i++){
        pow*=x;
    }
    return x;
}
```

```
int QuickPower(int x,int y){
    inr res=1;
    while(y!=0){
        // 先出来的为个位
        if(y%2==1){
            res*=x;
        }
        x*=x;
        y/=2;
    }
    return res;
}
```

4. 用递归的方法编写函数求Fibonacci 级数，公式为

$$fib(n) = fib(n-1) + fib(n-2),\ n > 2; fib(1) = fib(2) = 1$$

```
int Fib(int num){
    if(num==1 || num==2){
        return 1;
    }
    int left=1,right=1,tmp=0;
    for(int i=1;i<=num;i++){
        tmp=left+right;
        left=right;
        right=tmp;
    }
    return right;
}
```

```
int Fib(int n){
    if(n==1 || n==0){
        return 1;
    }
    else{
        return Fib(n-1)+Fib(n-2);
    }
}
```

5. 用递归的方法编写函数求n阶勒让德多项式的值，在主程序中实现输入、输出

$$P_n(x)=\begin{cases}1 & (n=0)\\ x & (n=1)\\ \left((2n-1)\times x\times p_{n-1}(x)-(n-1)\times P_{n-2}(x)\right)/n & (n\geqslant 1)\end{cases}$$

```
double P(double x, double n) {
    if (n == 0)
        return 1;
    else if (n == 1)
        return x;
    else
        return ((2 * n - 1) * x * P(x, n - 1) -
(n - 1) * P(x, n - 2)) / n;
}
```

6. 使用模板函数实现`Swap( x, y )`，函数功能为交换x、y的值。

```
template<typename T>
void swap(T &x,T &y){
    T tmp=x;
    x=y;
    y=tmp;
}
```

7. 字符串倒置

```
void reverse(char *s){
    int left=0,right=strlen(s);
    while(right<left){
        char tmp=s[right];
        s[right--]=s[left];
        s[left++]=tmp;
    }
}
```

8. 矩阵转置

```
void transpose(matrix &A){
    int tmp;
    for(int i=1;i<A.row;i++){
        for(int j=0;j<i;j++){
            tmp=A.val[i][j];
            A.val[i][j]=A.val[j][i];
            A.val[j][i]=tmp;
        }
    }
}
```

9. 矩阵乘法

```
void MatMul(const Matrix &A,const Matrix
&B,Matrix &C){
    C.row=A.row;
    C.col=B.col;
    for(int i=0;i<A.row;i++){
        for(int j=0;j<B.col;j++){
            for(int k=0;k<A.col;k++){
                C.val[i][j]+=A.val[i][k]*B.val[k]
[j];
            }
        }

    }

}
```

10. 字符串匹配

```
#include<iostream>
#include<cstring>
using namespace std;

void get_next(int *next,char *T,int len);
int KMP(char *s,int len,char *p,int plen);

int main(){
    char a[] =
"bacbababadababacambabacaddababacasdsd";
    char b[] = "ababaca";
    int m = KMP(a,strlen(a),b,strlen(b));
    cout<<m<<endl;
    return 0;

}

void get_next(int *next,char *T,int len){
    next[0]=-1;
    int k=-1;
    for(int q=1;q<=len;q++){
        while(k>-1 && T[k+1]!=T[q]){
            k=next[k];
        }
        if(T[k+1]==T[q]){
            k++;
        }
        next[q]=k;
    }
}
```

```
int KMP(char *s,int len,char *p,int plen){
    int *next=new int(plen);
    get_next(next,p,plen);

    int k=-1;
    int count=0;
    for(int i=0;i<len;i++){
        while(k>-1 && p[k+1]!=s[i]){
            k=next[k];
        }
        if(p[k+1]==s[i]){
            k++;
        }
        if(k==plen-1){
            count++;
        }
    }

    return count;

}
```

11. 计算天数

```
#include<iostream>
using namespace std;
int dayNums[13]=
{0,31,28,31,30,31,30,31,31,30,31,30,31};

int calDayNums(int year,int month,int day);
bool isLeap(int year);

int main(){
    int year,month,day;
    cout<<"请输入 年 月 日: ";
    cin>>year,cin>>month,cin>>day;
    cout<<"为该年的第"<<calDayNums(year,month,day)
<<"天"<<endl;

}

int calDayNums(int year,int month,int day){
    int sum=0;
    for(int i=1;i<=month;i++){
        if(i==2 && isLeap(year) ){
            sum++;
        }
```

```
        sum+=dayNums[i];
    }
    sum+=day;
    return sum;
}

bool isLeap(int year){
    return year%4==0 && year%100!=0 ||
year%400==0;
}
```

12. 如果一个数等于它的因子之和，则称该数为“完数”（或“完全数”)。例如，6的因子为1、2、3，而 6=1+2+3，因此6是“完数”。

```
bool isPerfectNumber(int num){
    int sum=0;
    // i超过了该数的一半，则肯定不会是该数的因子
    for(int i=1;i<=num/2;i++){
        if(num%i==0){
            sum+=i;
        }
    }
    return sum==num;
}
```

三、 面试算法真题

1. 给n 个整数x1...xn，找出最小的整数。请用二分递归法来实现-分成两半递归，各解出最小值然后比较。

```
int BinSearch(int A[],int left,int right){
    if(left==right){
        return A[left];      // 递归终止条件
    }

    int middle=left+(right-left)/2; //查中间值
    int a=BinSearch(A,left,middle);
    int b=BinSearch(A,middle+1,right);
    return  min(a,b);
}
```

2. 编程计算n条直线可以将1个平面分成多少个子平面？

a. 设n直线可分为f(n)个平面;第n-1直线可分为f(n-1)个平面。
b. n-1直线增加第n直线,可分为f(n)个平面。若要增加的第n直线划分的子平面最
多:第n条直线和原n-1条直线需产生最大n-1个交点。
c. 这些交点将第n条直线划分出: 2 条剝线+ n-2条线段。这些射线和线段，每一条都会将平
移划分出一个子平面(可画图辅助理解),即f(n) - f(n-1) = n。

$$f(n) = n + f(n-1)$$
$$f(1) = 2$$

```
int f(int n){
    if(n==1){
        return 2;
    }
    return n+f(n-1);
}
```

3. 已升序有序的数组A[N],指定正整数M。使用二分查找法，查找在A[N]中和M 相等数的位置，返回其下标k;如果没有找到则返回A 中最大的<M 数的下标k。如：A[5] = {1 , 3 , 15 ,70 , 108} ,查找M=20，返回k = 2。

输入： 20 输出： 2

```
#include<iostream>
#include<cmath>

using namespace std;
const int MAX_SIZE=200;
// 寻找上界
int BinSearch(int A[],int left,int right,int k){
    if(left<right){
        int middle=left+(right-left)/2;
        // 找左边
        if(A[middle]>k) {
            return BinSearch(A,left,middle-1,k);
        }
        // 找右边
        else if(A[middle]<k){
            return BinSearch(A,middle+1,right,k);
        }
        // 相等
        else{
            return middle;
        }
    }
    // 如果已经出界了，则返回
    else{
        // 需要判断出界
        // k位于A[right-1]的左边
        if( right-1>=0 && A[right-1]>k){
            return right-1;
```

```
        }
        // k位于A[right-1]的右边，A[right]的左边
        else if(right>=0 && A[right]>k){
            return right;
        }
        // k位于A[right]的右边，A[right+1]的左边
        else if( right+1<=sizeof(A)/sizeof(int)
&& A[right+1]>k){
            return right+1;
        }
        else{
        return -1;
        }

    }
}


int main(){
    int A[MAX_SIZE];
    int k=-20;
    for(int i=0;i<MAX_SIZE;i++){
        A[i]=i*2+1;
    }
    int index=BinSearch(A,0,MAX_SIZE-1,k);
    cout<<"搜索到k="<<k<<"的位置为："<<index<<endl;
    cout<<"位于"<<A[index-1]<<"与"<<A[index]<<"之
间";
}
```

```
// upper bound 寻找上界
int BinSearch2(int A[],int left,int right,int k){
    while(left < right){
        int middle = left +(right-left)/2;
        if(A[middle] < k) {
            left = middle+1;
        }
        // A[middle]>=k,
        else {
            right = middle;
        }
    }
    // 同样也需要判断，是否为上边界
    if( A[left]>=k){
        return left;
    }
    else{
        return -1;
    }
}
```

```
int main(){
    int A[MAX_SIZE];
    int k=24;
    for(int i=0;i<MAX_SIZE;i++){
        A[i]=i*2+5;
        cout<<A[i]<<" ";
    }
    int index=BinSearch2(A,0,MAX_SIZE-1,k);
    cout<<"搜索到k="<<k<<"的位置为："<<index<<endl;
    cout<<"位于"<<A[index-1]<<"与"<<A[index]<<"之
间";
}
```

四、 模拟题

1. 有两个磁盘文件**A** 和**B,**各存放一行字母，要求把这两个文件中的信息合并（按字母顺序排列），输出到一个新文件**C** 中。

输入文件： 文件1：abAB 文件2： cdCD
输出文件： ABCDabcd

```
#include<iostream>
#include<string>
#include<fstream>
using namespace std;

#define MAX_SIZE 100
string a,temp_s;

void Bubble(string &a){
    char tmp;
    bool isSwap;
    for(int i=0;i<a.size();i++){
        isSwap=false;
        for(int j=a.size()-1;j>i;j--){
            if(a[j]<a[j-1]){
                tmp=a[j];
                a[j]=a[j-1];
                a[j-1]=tmp;
                isSwap=true;
            }
        }
        if(isSwap==false){
            break;
```

```
        }
    }
}

int main(){

    ifstream in1("./Data/m99/file_a.txt");
    ifstream in2("./Data/m99/file_b.txt");
    if(!in1.is_open() || ! in2.is_open()){
        cout<<"文件打开失败"<<endl;
        return 0;
    }
    char tmp[MAX_SIZE];
    in1.getline(tmp,MAX_SIZE);
    a=(string)tmp;      // 读取第一个文件字符串
    in2.getline(tmp,MAX_SIZE);
    a=a+(string)tmp;    // 读取第一个文件字符串
    cout<<a<<endl;
    Bubble(a);          // 冒泡排序
    cout<<a<<endl;
    in1.close();
    in2.close();
}
```

```
// 方法2采用归并法,a b均为有序
void Combine(char *a,char *b,char *c){
    int len_a=strlen(a),len_b=strlen(b);
    int p_a=0,p_b=0,p_c=0;
    while(p_a<len_a && p_b<len_b){
        if(a[p_a]<=b[p_b]){
            c[p_c++]=a[p_a++];
        }
        else{
            c[p_c++]=b[p_b++];
        }
    }
    while(p_a<len_a){
        c[p_c++]=a[p_a++];
    }
    while(p_b<len_b){
        c[p_c++]=b[p_b++];
    }
}
```

2. 输入一串主串，一串子串，计算子串在主串中出现的频率。如：

输入：ABCCDFEFCDCDFFCD 输出：2
CDF

```
#include<iostream>
#include<string>
using namespace std;
// 暴力匹配
int Match(string a,string b){
    int i,j,k,count=0;
    for(i=0;i<a.size();i++){
        //首先匹配第一个字符
        if(a[i]==b[0]){
                //接着继续匹配到串结束
                for(j=1,k=i+1;a[k]==b[j] &&
j<b.size() && k<a.size();j++,k++);
            }
        if(j==b.size()){
            count++;
        }
    }
    return count;
}

int main(){
    string a="asasadasazccaszwwdasa";
    string b="asa";
    cout<<Match(a,b);
}
```

3. 请回答以下输出结果

```
#include<iostream>
using namespace std;
struct student
{
    int x;
    char c;
} a;
int main()
{
    a.x=3;
    a.c='a' ;
    f(a);
    cout<<a.x<<","<<a.c;
}
f( struct student b )
    b. x=20;
    b.c='y';
}
```

结果为：3,a; 采用值传递

4. 填空

```
#include<iostream>
using namespace std;

int main(){
    char *s[]=
{"man","woman","girl","boy","sister"};
    char **q;
    for(int k=0;k<5;k++){
        ____?_____
        cout<<*q<<endl;
    }
    return 0;
}
```

应填入：

```
q=&s[k];
```

这里不把`char **q`理解为是二维数组，而是指向指针的指针。如果q是字符指针`char *q`那么直接输出q就行。这里*q，说明**q是指针并指向了一个字符指针**。那么如何指向字符指针？

```
int *p = &a;     // 此处p是指向整型变量
int A[5];
int *q = A; // 此处q是指向整型数组
```

5. **利用指针函数。编写一个函数，输入n为偶数时，调用函数求1/2+1/4+...+1/n,当输入n为奇数时，调用函数1/1+1/3+...+1/n。**

输入： 4 输出：0.75
输入： 5 输出：1.533333

```
double *ptr;           //普通指针
double (*ptr)(int a);      //函数指针
double *func(int a);  //指针函数
double *p[2];          // 指针数组
double (*p)[2];            // 指向数组的指针
```

```
#include<iostream>
using namespace std;
```

```
// 计算偶数
double calEven(int num){
  double sum=0;
  for(int i=2;i<=num;i=i+2){
      sum+=1.0/i;
  }
  return sum;
}
// 计算奇数
double calOdd(int num){
  double sum=0;
  for(int i=1;i<=num;i=i+2){
      sum+=1.0/i;
  }
  return sum;
}


int main(){
  int n=65;
  double res=0;
  double (*pfunc)(int);   //函数指针
  if(n%2==0){
      pfunc=calEven;  //注意函数指针赋值，函数名就是地址
  }
  else{
      pfunc=calOdd;
  }
    // 函数指针的函数调用与声明时的结构类似
  res=(*pfunc)(n);
  cout<<res;
}
```

6. 有**n** 个整数，使其前面各数顺序向后移**m** 个位置，最后**m** 个数变成最前面的**m** 个数。

```
// 三次反转即可
#include<iostream>
using namespace std;

void reverse(int A[],int left,int right){
    while(left<right){
        int tmp=A[left];
        A[left++]=A[right];
        A[right--]=tmp;
    }
}
```

```
int main(){
    int a[]=
{1,2,4,5,6,6,8,8,9,1,2,2,3,3,3,3,4,5,8,8,9,9};
    const int N1=9;
    const int N2=13;
    //各自反转
    reverse(a,0,N1);
    //各自反转
    reverse(a,N1,N1+N2);
    //整体反转
    reverse(a,0,N1+N2);
    for(int i=0;i<N1+N2;i++){
        cout<<a[i]<<" ";
    }
    cout<<endl;
    return 0;
}
```

7. 用户输入指定**n**，求**1+2!+3!+...+n!**的和。

如：

输入：4 输出：33

```
#include<iostream>
using namespace std;

int main(){
    int n=5,acc=1,sum=0;
    for(int i=1;i<=n;i++){
        acc*=i;
        sum+=acc;
    }
    cout<<sum;
}
```

8. 有**1**、**2**、**3**、**4** 个数字，能组成多少个千百十位互不相同无重复数字的三位数？都是多少？

输出： 123 124 ...432

```
#include<iostream>
using namespace std;
// 砝码问题
int main(){
    for(int i=1;i<=4;i++){
        for(int j=1;j<=4;j++){
            for(int k=1;k<=4;k++){
                if(i!=j && i!=k){
```

```
                    cout<<i*100+j*10+k<<endl;
                }
            }
        }
    }
}
```

五、 谭浩强C习题

1. **有一行电文译文下面规律译成密码： A->Z a->z B->Y b->y C->X c->x ... 即第一个字母变成第26 个字母,第i 个字母变成第26-i-1 个字母。**

   例如有一行电文译文下面规律译成密码：
   A->Z B->Y C->X a->z b->y c->x ...
   非字母字符不变,要求编程序将密码回原文,并打印出密码和原文.

```
#include<iostream>
#include<cstring>
using namespace std;

int main(){
    char s[]="ABCabcDSA";
    cout<<"源字符串："<<s<<endl;
    for(int i=0;i<strlen(s);i++){
        int index;
        // 分大小写字母
        if(islower(s[i])){
            s[i]=26-(s[i]-'a')-1+'a';
        }
        else{
            s[i]=26-(s[i]-'A')-1+'A';
        }
    }
    cout<<"加密后字符串："<<s<<endl;
}
```

2. **输出一串字符中最长的单词。**

   如:
   输入： i am huang wang hui 23 years old 输出：huang

```
#include<iostream>
#include<cstring>
using namespace std;

int main(){
    char c[]="I am Huang Wanghui 23 yearsold";
```

```
    // pos为一个字符开始位置
    // len为一个字符的的长度
    int pos=0,len=0;
    // max_pos为最大字符开始位置
    // max为最大字符的的长度
    int max_pos=0,max=0;
    cout<<c<<endl;
    for(int i=0;i<strlen(c);i++){
        if(c[i]!=' '){
            len++;
        }
        else{
            if(max<len){
                max_pos=pos,max=len;
            }
            pos=i;
            len=0;
        }
    }
    // 如果最后一个字符不是' '，则需要判断
    if(max<len){
        max_pos=pos,max=len;
    }
    // 输出最大字符
    for(int i=pos+1;i<=max+pos;i++){
        cout<<c[i];
    }
    cout<<endl<<"长度为："<<max<<endl;
    return 0;
}
```

3. 实现计算`sinx`， `cosx`， `e^x` ,在**[a,b]**积分的通用函数

```
#include<iostream>
#include<cmath>
using namespace std;
const double e = 2.71828;
const double pi = 3.1415926;

// 积分函数，采用函数指针
double acc(double a,double b, double (*f)
(double)){
    // 将区间分为n个
    int n=1000000;
    double delta=(b-a)/n,sum=0;
    for(int i=0;i<n;i++){
        // 注意函数指针的函数调用
        sum+=delta*(*f)(a+i*delta);
    }
    return sum;
```

```
}

double ex(double x){
    return pow(e,x);
}

double x2(double x){
    return pow(x,2);
}

int main(){
    double a=2,b=4;
    cout<<"sinx "<<a<<"到"<<b<<"的积分"
<<acc(a,b,sin)<<endl;
    cout<<"cosx "<<a<<"到"<<b<<"的积分"
<<acc(a,b,cos)<<endl;
    cout<<"e^x "<<a<<"到"<<b<<"的积分"<<acc(a,b,ex)
<<endl;
    cout<<"x^2 "<<a<<"到"<<b<<"的积分"<<acc(a,b,x2)
<<endl;
}
```

4. 输入一个字符串，内有数字和非数字字符。将其中连续的数字作为一个整数，依次存放到一数组**a** 中。

例如，123 放在a[0]，456 放在a[1]..统计共有多少个整数，并输出这些数：
输入：A123x456 17960? 302tab5876 输出：123 456 17960 302 5876

```
// 与2类似
#include<iostream>
#include<cstring>

using namespace std;
const int MAX_SIZE = 100;
// 用于存储数字
int a[MAX_SIZE];
// 内联函数用于判断字符是不是数字
inline bool isnumber(char c){
    return c>='0' && c<='9';
}

int main(){
    // 将数组初始化
    // 注意memset只针对char大小，8位的数据
    // 如果赋值超过了8位就会报错
    memset(a,-1,MAX_SIZE*sizeof(int));
    char s[]="A123x456 17960?302tab5876";
    int sum=0,count=0;
    //用于标记是数字，方便结束时保存数据到数组
    bool isnum=false;
    for(int i=0;i<strlen(s);i++){
```

```
        if(isnumber(s[i])){
            sum=sum*10+(s[i]-'0');
            isnum=true;
        }
        else{
            //如果是数字，则要存储并重置
            if(isnum){
                a[count++]=sum;
                sum=0;
                isnum=false;
            }
        }
    }
    // 可能最后一段字符串为数值
    //还没记录就已经结束
    if(isnum){
        a[count++]=sum;
        sum=0;
        isnum=false;
    }
    // 输出
    for(int i=0;i<count;i++){
        cout<<a[i]<<endl;
    }
}
```

5. 实现`strcmp`函数

```
int getLength(char s[]){
    int count =0;
    while(s[count++]!='\0');
    return count;
}

int strcmp(char s1[],char s2[]){
    int len_s1=getLength(s1);
    int len_s2=getLength(s2);
    // 判断字符串长度
    if(len_s1==len_s2){
        // 长度相等则一位一位比较
        for(int i=0;i<len_s1;i++){
            if(s1[i]!=s2[i]){
                return s1[i]>s2[i]?1:-1;
            }
        }
        return 0;
    }
    //如果不等，则越长越大
    else{
        return len_s1>len_s2?1:-1;
    }
```

```
}
```

六、选做题

1. （**HUSTOJ**）高精度计算，求**n**（**<100**）的阶乘。

如：

输入： 19 输出：121645100408832000

```
#include<iostream>
#include<cstring>
using namespace std;

#define MAX_SIZE 1000
int Num[MAX_SIZE];
// 逆序存放 ，即个位为A[0]，十位为A[1]...
int getLength(){
    int len=MAX_SIZE;
    for(int i=MAX_SIZE-1;i>=0;i--){
        if(Num[i]==0){
            len--;
        }
        else{
            break;
        }
    }
    return len;
}
//标准化，将进位前推移
//如A[2]=234,A[3]=0,A[4]=0
//则变为A[2]=4,A[3]=23
//进一步变为A[2]=4,A[3]=3,A[4]=2
//使得每一位都不超过9
void Formal(){
    int carry=0;    //进位，初始化为0
    for(int i=0;i<MAX_SIZE;i++){
        Num[i]+=carry;  //加上上一轮的进位
        carry=0;
        // 如果超过了9，则向前推
        if(Num[i]>=10){
            int tmp=Num[i];
            Num[i]=Num[i]%10;
            carry=tmp/10;
        }
        if(Num[i]==-1){
            break;
        }
    }
}
```

```
void multiply(int n,int len){
    //将每位与n相乘，然后标准化
    for(int i=0;i<len;i++){
            Num[i]=Num[i]*n;
    }
    Formal();

}

void printA(int len){
    for(int i=len-1;i>=0;i--){
        cout<<Num[i];
    }
    cout<<"\n";
}

int main(){
    int n=30;
    memset(Num,0,sizeof(int)*MAX_SIZE);
    Num[0]=1;
    for(int i=1;i<=n;i++){
        multiply(i,getLength());
        printA(getLength());
    }

}
```

2. 输入正整数**n**,按从小到大的顺序输出所有形如 `abcde/fghij=n` 的表达式，其中**a~j** 恰好为数字**0~9** 的一个排列，**2<=n<=79**。

输入：62　输出：79546/01283=62
94736/01528=62

```
#include<iostream>
using namespace std;

// 判断各个位是否有重复的
bool isDiff(int abcde,int fghij){
    // 使用布尔数组标记数字是否出现
    bool isHappen[10]={false};
    if(abcde<10000){
        isHappen[0]=true;
    }
    while(abcde!=0){
        int tmp=abcde%10;
        if(isHappen[tmp]){
            return false;
```

```
        }
        // 如果数字出现过，则标记为true
        isHappen[tmp]=true;
        abcde/=10;
    }
    // 都小于10000，则必有2个0
    if(fghij<10000){
        if(isHappen[0]){
            return false;
        }
        else{
            isHappen[0]=true;
        }
    }
    while(fghij!=0){
        int tmp=fghij%10;
        if(isHappen[tmp]==true){
            return false;
        }
        isHappen[tmp]=true;
        fghij/=10;
    }
    return true;
}

int main(){
    int abcde,fghij,n=62;
    for(abcde=1234;abcde<=98765/n+1;abcde++){
        fghij=abcde*n;
        if(fghij<=98765){
            if(isDiff(abcde,fghij)){
                cout<<fghij<<"/"<<abcde<<"="
<<n<<endl;
            }
        }
    }
}
```

3.