# C++11

## 1、新增数据类型

long long和unsigned long long，支持64位的整型。（输出：%lld 和%llu）

## 统一初始化：

初始化可添加等号，也可不添加 double y {2.75};

初始化列表可用于new表达式 int \*a= new int [4] {1,2,3,4};

## auto关键字：

自动类型推断，要求进行显示的初始化，能让变量的类型设置为初始值的类型 auto maton=112; auto ptr=&maton;

decltype关键字：decltype (x) y 让y和表达式x的类型一致 decltype(&x) y

nullptr:解决了Null的二义性的问题，原来Null既是指针常量，又是整型常量。

## 4、智能指针：

（1）unique\_ptr,shared\_ptr,weak\_ptr;智能指针过期时，内存自动释放

auto\_ptr<int> a (new int (5)) 添加头文件memory

（2）摒弃auto\_ptr:当两个常规指针指向同一个对象时，程序试图删除一个对象两次，就会导致内存崩溃。

解决办法：用unique\_ptr,程序不会在运行阶段崩溃，会在编译阶段就崩溃

用shared\_ptr，进行指针计数，生成加1，销毁-1，最后一个指针过期，才调用delete。

（3）使用多个指针调用同一对象时，用shared\_ptr,如果函数用new分配内存，返回指向改内存的指针，将返回类型声明为unique\_ptr是不错的选择。**对象之间“共享数据”，对象创建与销毁“分离”** **放入容器中的动态对象，用** shared\_ptr  
e.g shared\_ptr<Parent> pA(new Parent);

shared\_ptr<Child> pB(new Child);

pA->child = pB;

pB->parent = pA;

在Parent类中存储了指向Child类对象的智能指针成员变量,而在Child类中也存储了指向Parent类对象的智能指针成员变量,如此就会造成环形引用。**一般都是在可能出现环形引用的地方使用weak\_ptr来代替shared\_ptr**

# 对面向对象的理解

答：面向对象主要包括三个特性：继承、封装和多态。通过继承，子类可以拥有父类所有成员。封装是使用方法把类的数据成员隐藏起来。多态包括运行多态和编译多态。编译多态主要是函数重载。函数重载是指使用同一个函数名。可以通过形参列表的不同调用到需要的函数。运行多态主要是通过虚函数实现的，一个基类指针操纵多个派生类。

1. 封装

1.保护数据成员，不让类以外的程序直接访问或修改，只能通过提供的公共的接口访问 ==>数据封装。

　　2. 方法的细节对用户是隐藏的，只要接口不变，内容的修改不会影响到外部的调用者 ==>方法封装。

      3. 当对象含有完整的属性和与之对应的方法时称为封装。

　　4. 从对象外面不能直接访问对象的属性，只能通过和该属性对应的方法访问。

　　5. 对象的方法可以接收对象外面的消息。

1. 继承

public（公有继承）：继承时保持基类中各成员属性不变，并且基类中private成员被隐藏。派生类的成员只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象只能访问基类中的public成员。  
 private（[私有继承](https://www.baidu.com/s?wd=%E7%A7%81%E6%9C%89%E7%BB%A7%E6%89%BF&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)）：继承时基类中各成员属性均变为private，并且基类中private成员被隐藏。派生类的成员也只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象不能访问基类中的任何的成员。   
 protected（保护性继承）：继承时基类中各成员属性均变为protected，并且基类中private成员被隐藏。派生类的成员只能访问基类中的public/protected成员，而不能访问private成员；派生类的对象不能访问基类中的任何的成员。

1. 多态

同一操作作用于不同的对象，可以有不同的解释，产生不同的执行结果。在运行时，可以通过指向基类的指针，来调用实现派生类中的方法。 意味着一个对象有着多重特征，可以在特定的情况下，表现不同的状态，从而对应着不同的属性和方法

# 继承模型

## 普通继承（包含虚函数）

### 单继承：

class Base{

public:

Base (int a = 1):base(a){}

void fun0(){cout << base << endl;}

int base;

};

class Derive:public Base{

public:

Derive (int a = 2):derive(a){}

void fun1(){cout << base1 << endl;}

int derive;

};



### 多继承

class Base1{

public:

Base1 (int a = 2):base1(a){}

void fun1(){cout << base1 << endl;}

int base1;

};

class Base2{

public:

Base2 (int a = 3):base2(a){}

void fun2(){cout << base2 << endl;}

int base2;

};

class Derive: public Base1, public Base2{

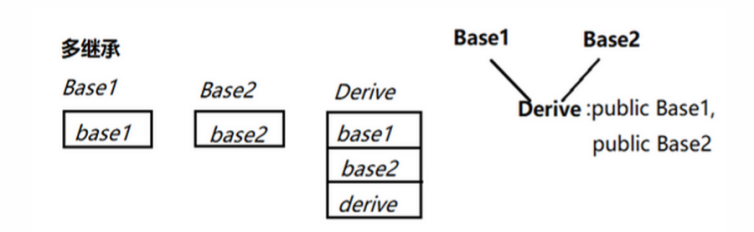
public:

Derive (int value = 4):derive (value){}

void fun3(){cout << derive << endl;}

int derive;

};



### 菱形继承

class Base{

public:

Base (int a = 1):base(a){}

void fun0(){cout << base << endl;}

int base;

};

class Base1:public Base{

public:

Base1 (int a = 2):base1(a){}

void fun1(){cout << base1 << endl;}

int base1;

};

class Base2:public Base{

public:

Base2 (int a = 3):base2(a){}

void fun2(){cout << base2 << endl;}

int base2;

};

class Derive: public Base1, public Base2{

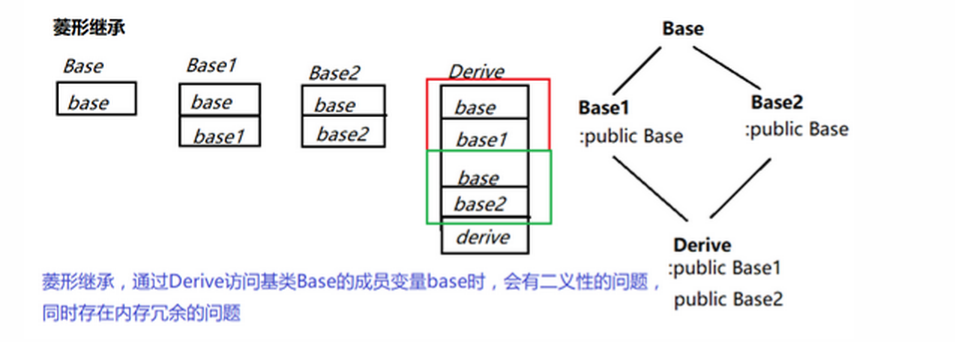
public:

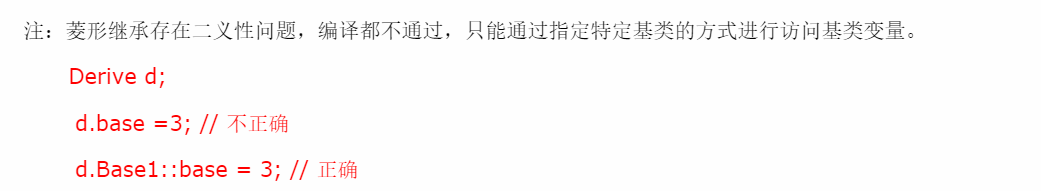
Derive (int value = 4):derive (value){}

void fun3(){cout << derive << endl;}

int derive;

};





## 普通继承（包含虚函数）

### 单继承

class Base{

public:

Base (int a = 1):base(a){}

virtual void fun0(){cout << base << endl;}

int base;

};

class Derive:public Base{

public:

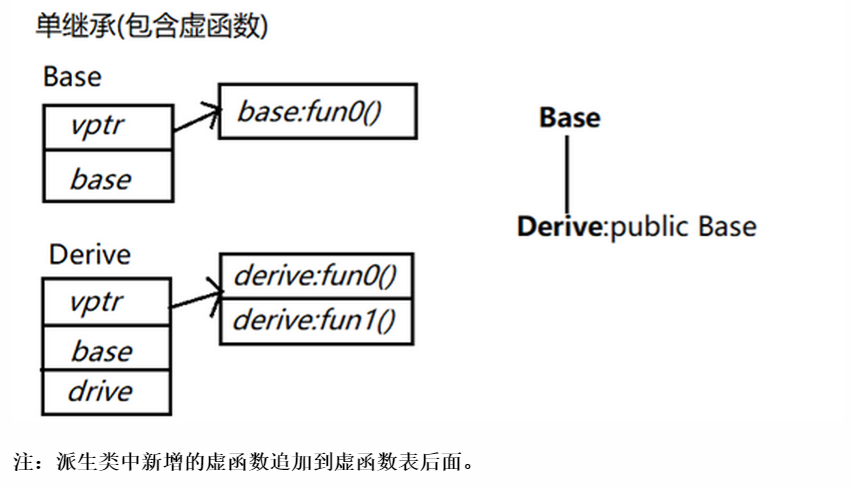
Derive (int a = 2):derive(a){}

virtual void fun0(){};

virtual void fun1(){cout << derive << endl;}

int derive;

};



### 多继承

class Base1{

public:

Base1 (int a = 2):base1(a){}

virtual void fun1(){cout << base1 << endl;}

int base1;

};

class Base2{

public:

Base2 (int a = 3):base2(a){}

virtual void fun2(){cout << base2 << endl;}

int base2;

};

class Derive: public Base1, public Base2{

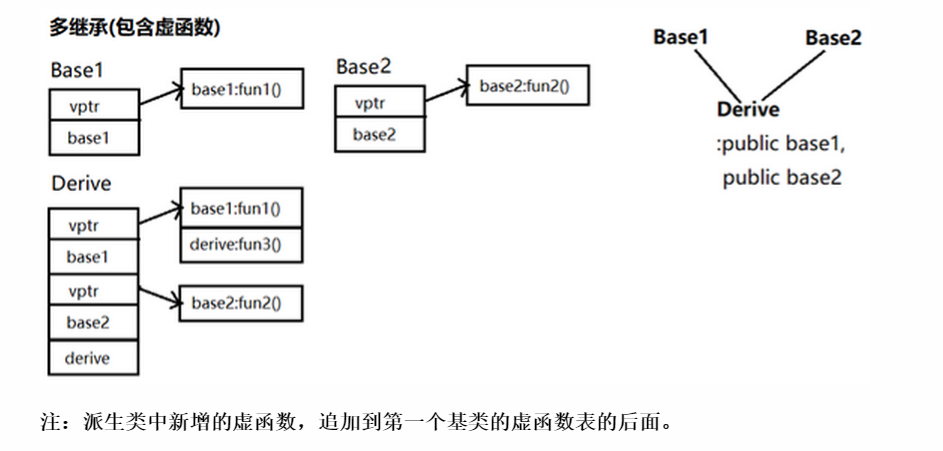
public:

Derive (int value = 4):derive (value){}

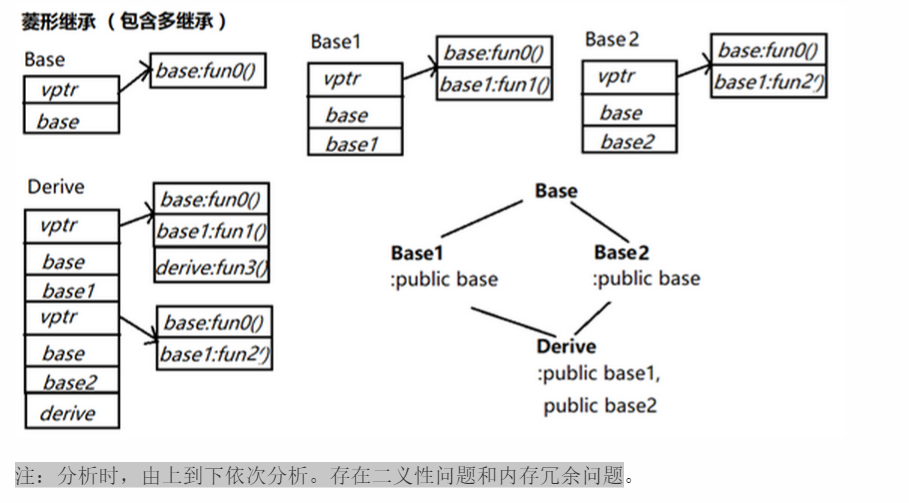
virtual void fun3(){cout << derive << endl;}

int derive;

};



### 菱形继承



## 虚继承（不含虚函数）

新增虚基类指针，指向虚基类表，虚基类表中首项存储虚基类指针的偏移量，接下来依次存储虚基类的偏移量（偏移量是相对于虚基类表指针的存储地址）

### 单虚继承

class Base{

public:

Base (int a = 1):base(a){}

void fun0(){cout << base << endl;}

int base;

};

class Base1:virtual public Base{

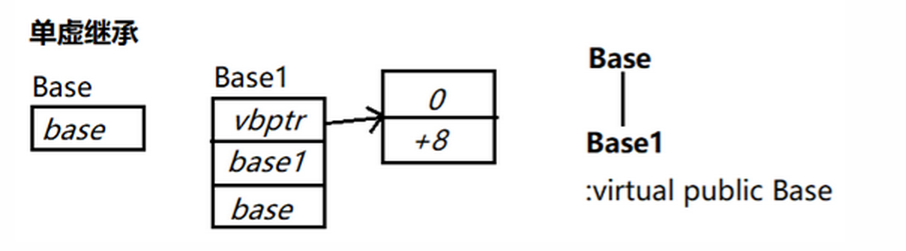
public:

Base1 (int a = 2):base1(a){}

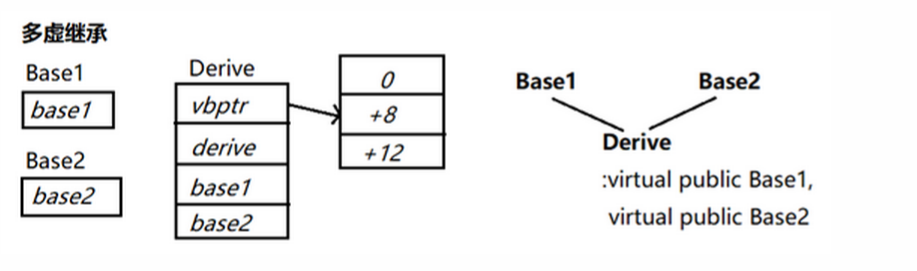
void fun1(){cout << base1 << endl;}

int base1;

};

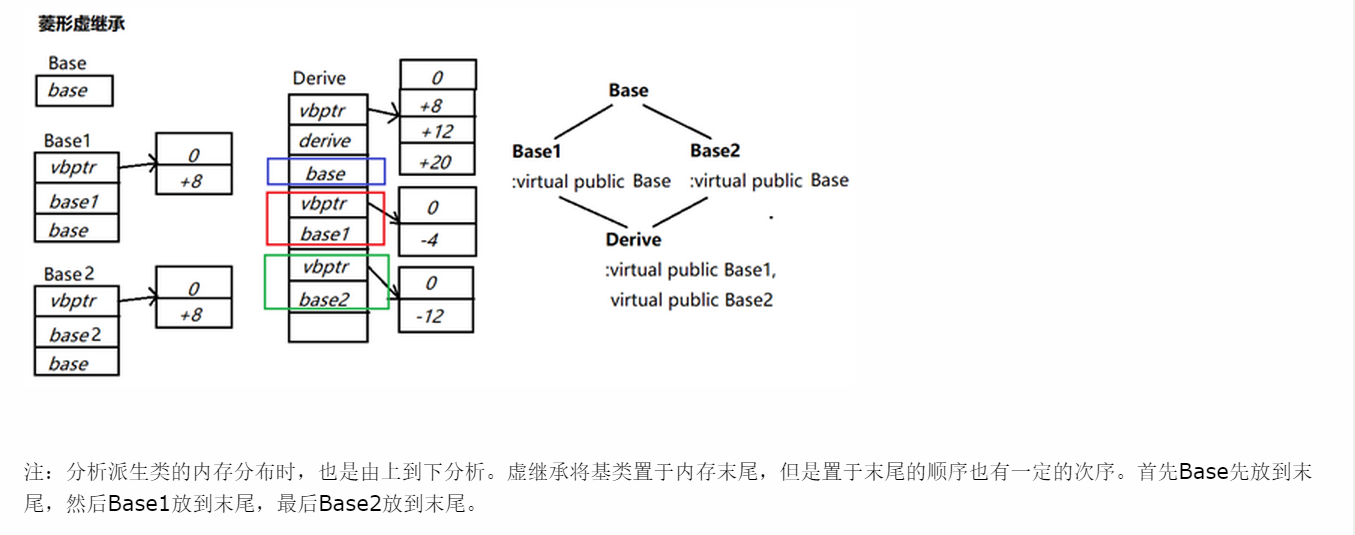


### 多虚继承

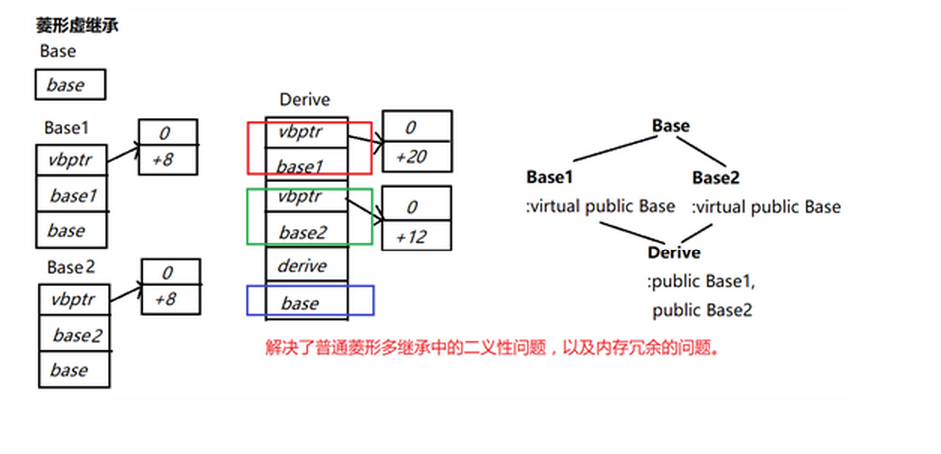


1. 菱形虚继承

第一种形式：

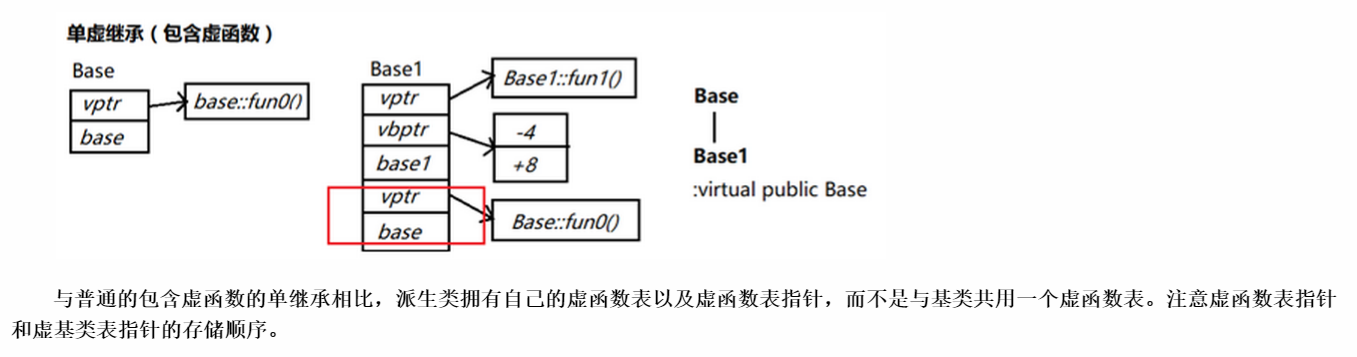


第二种形式：

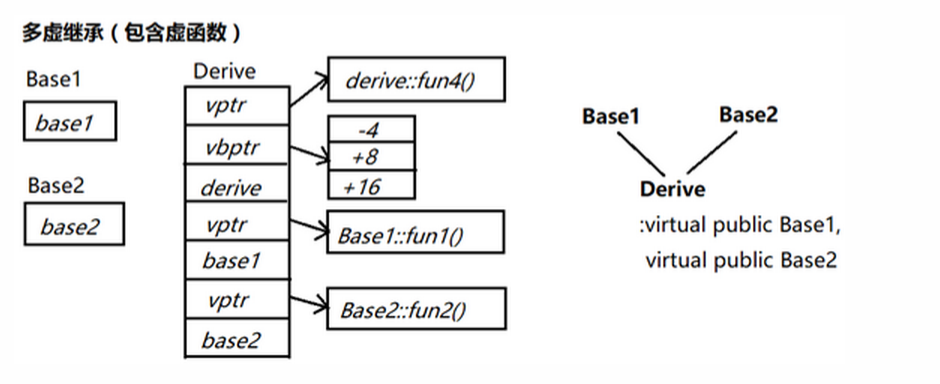


## 虚继承（含虚函数）

### 单虚继承

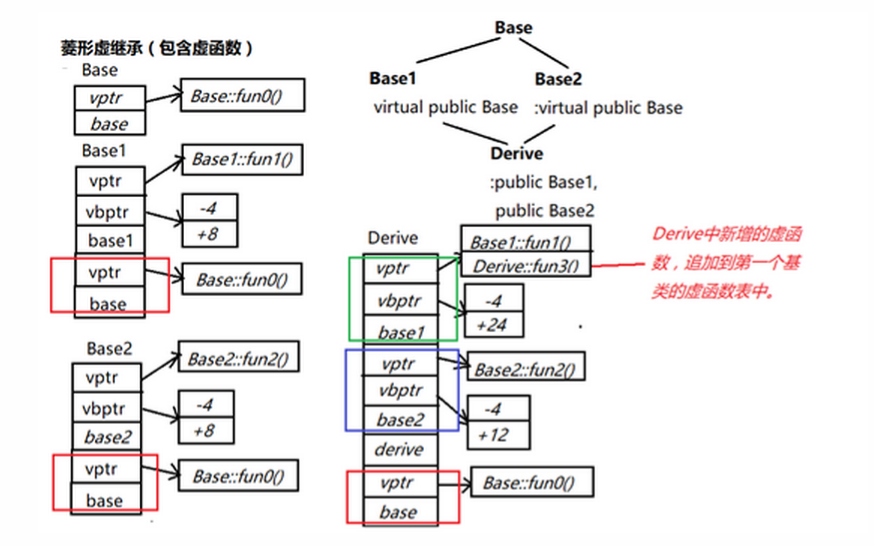


### 多虚函数



### 菱形虚继承

第一种形式：



# 四、TCP协议

## 1、三次握手

* 第一次握手:建立连接时，客户端发送[syn](https://baike.so.com/doc/5252195-5485447.html" \t "_blank)包到[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "_blank)，并进入SYN\_SEND状态 。
* 第二次握手：[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "_blank)收到[syn](https://baike.so.com/doc/5252195-5485447.html" \t "_blank)包，必须确认客户的SYN([ack](https://baike.so.com/doc/5444927-5683287.html" \t "_blank)=j+1)，同时自己也发送一个SYN包(syn=k)，即SYN+ACK包，此时服务器进入SYN\_RECV状态
* 第三次握手：客户端收到服务器的SYN+ACK包，向[服务器](https://baike.so.com/doc/4487696-4696885.html" \t "_blank)发送确认包ACK([ack](https://baike.so.com/doc/5444927-5683287.html" \t "_blank)=k+1)，此包发送完毕，客户端和服务器进入ESTABLISHED(TCP连接成功)状态，完成三次握手

（2）如果两次，那么B无法确定B的信息A是否能收到，所以如果B先说话，可能后面的A都收不到，会出现问题。

或（2）主机A发出的连接请求没有收到主机B的确认，于是经过一段时间之后，主机A重新向主机B发送连接请求，且建立成功，顺序完成数据传输。考虑这样一种特殊情况，主机A第一次发送的连接请求并没有丢失，而是因为网络节点导致延迟到达主机B，主机B以为是主机A又发起的新连接，并向主机A发回确认，但是此时主机A根本不会理会，主机B就一直等待主机A发送数据，导致主机B的资源浪费。

1. 如果四次，那么就造成了浪费，因为三次结束之后，就已经可以保证A可以给B发消息，A也可以收到B的消息；B可以给A发信息，B也可以收到A的信息。



## 2、backlog

backlog其实是一个连接队列， backlog大小包括半连接状态和全连接状态两种队列大小。

半连接状态为：服务器处于Listen状态时收到客户端SYN报文时放入半连接队列中，即SYN queue（服务器端口状态为：SYN\_RCVD）。

### 全连接状态为：TCP的连接状态从服务器（SYN+ACK）响应客户端后，到客户端的ACK报文到达服务器之前，则一直保留在半连接状态中；当服务器接收到客户端的ACK报文后，该条目将从半连接队列搬到全连接队列尾部，即 accept queue

## 3、TCP洪水攻击

SYN Flood是当前最流行的DoS（拒绝服务攻击）与DDoS（分布式拒绝服务攻击）的方式之一，这是一种利用TCP协议缺陷，发送大量伪造的TCP连接请求，常用假冒的IP或IP号段发来海量的请求连接的第一个握手包（SYN包），被攻击服务器回应第二个握手包（SYN+ACK包），因为对方是假冒IP，对方永远收不到包且不会回应第三个握手包。导致被攻击服务器保持大量SYN\_RECV状态的“半连接”，并且会重试默认5次回应第二个握手包，塞满TCP等待连接队列，资源耗尽（CPU满负荷或内存不足），让正常的业务请求连接不进来。

应急处理：tcp\_synack\_retries = 0是关键，表示回应第二个握手包（SYN+ACK包）给客户端IP后，如果收不到第三次握手包（ACK包）后，不进行重试，加快回收“半连接”，不要耗光资源。

## 4、关闭TCP(四次挥手)

答：第一步，当主机A的应用程序通知TCP数据已经发送完毕时，TCP向主机B发送一个带有FIN附加标记的报文段(FIN表示英文finish)。

第二步，主机B收到这个FIN报文段之后，并不立即用FIN报文段回复主机A，而是先向主机A发送一个确认序号ACK，同时通知自己相应的应用程序:对方要求关闭连接(先发送ACK的目的是为了防止在这段时间内，对方重传FIN报文段)。

第三步，主机B的应用程序告诉TCP:我要彻底的关闭连接，TCP向主机A送一个FIN报文段。

第四步，主机A收到这个FIN报文段后，向主机B发送一个ACK表示连接彻底释放

## 5、如何保证tcp的稳定性

1、确认机制（接收方每次接收到数据时都会告知发送方已经正确接收了数据段以及期望收到的下一个数据段的序号）

2、校验和（校验数据是否损坏）

3、超时重传（一定时间内还没有收到分组就会重传，定时器重传/快速重传）

4、序列号（用于检测接受的数据段是否是当前连接的数据段，防止上一次连接的延迟包到达导致的错误）

5、按序到达（tcp包头中有用于片分的字段，其中有13位的一个标识分片的字段，用来进行重组的）

6、重置（当接受到错误的数据段时会重置连接，错误的数据段指该的序列号非正确序列号）

## 6、滑动窗口机制：

传输层进行流量控制的一种措施，接收方通过通告发送方自己的窗口大小，从而控制发送方的发送速度，从而达到防止发送方发送速度过快而导致自己被淹没的目的，并且滑动窗口分为接收窗口和发送窗口。

## 7、TCP socket的接收和发送缓冲区

每个TCP Socket在内核中都有一个发送缓冲区和一个接收缓冲区， TCP的全双工工作模式以及TCP的滑动窗口就是依赖这两个独立的buffer以及buffer的填充状态。

****应用程序调用write()或send()时，仅仅是把应用程序buffer中的数据copy到socket的发送缓冲区中（write()或send()返回时，data并不一定已经发送到对端了），在适当的时机，内核会把socket发送缓冲区的数据发送到接收方的socket接受缓冲区。****

****接收方的应用程序调用read()或receve()时，仅仅是从接收方的socket接受缓冲区中把数据copy到应用程序的buffer中。如果应用程序一直没有调用read()读取的话，此数据会一直缓存在相应的socket的接收缓冲区。****

如果应用进程一直没有读取，buffer满了之后，发生的动作是：通知对端TCP协议中的窗口关闭。这个便是滑动窗口的实现。保证TCP套接口接收缓冲区不会溢出，从而保证了TCP是可靠传输。因为对方不允许发出超过所通告窗口大小的数据。 这就是TCP的流量控制，如果对方无视窗口大小而发出了超过窗口大小的数据，则接收方TCP将丢弃它

## TCP的拥塞控制

1. **拥塞**：最初的TCP在连接建立成功后会向网络中发送大量的数据包，这样很容易导致网络中路由器缓存空间耗尽，从而发生拥塞。
2. **如何检测拥塞**：发送方发送一些报文段时，如果发送方没有在时间间隔内收到接收方的确认报文段，则就可以人为网络出现了拥塞。
3. **慢启动：**在主机刚开始发送数据报的时候先探测一下网络的状况，如果网络状况良好，发送方每发送一次文段都能正确的接受确认报文段。那么就从小到大的增加拥塞窗口的大小，按指数增加。
4. **拥塞避免：**慢启动时，cwnd会很快增长，cwnd不能一直这样无限增长下去，设置慢启动门限ssthresh，当cwnd超过该值后，慢启动过程结束，进入拥塞避免阶段。Cwnd线性增长，每次接收方确认成功接收，cwnd增加一。
5. **解决拥塞：**

1、把ssthresh降低为cwnd值的一半   
   2.把cwnd重新设置为1   
    3.重新进入慢启动过程。   
从整体上来讲，TCP拥塞控制窗口变化的原则是AIMD原则，即加法增大、乘法减小。可以看出TCP的该原则可以较好地保证流之间的公平性，因为一旦出现丢包，那么立即减半退避，可以给其他新建的流留有足够的空间，从而保证整个的公平性。

1. **AIMD(加法增大乘法减小)**

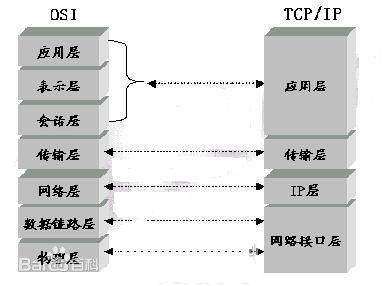
          1. 乘法减小：无论在慢启动阶段还是在拥塞控制阶段，只要网络出现超时，就是将**cwnd置为1**，ssthresh置为cwnd的一半，然后开始执行慢启动算法（cwnd<ssthresh）。

         2. 加法增大：当网络频发出现超时情况时，ssthresh就下降的很快，为了减少注入到网络中的分组数，而加法增大是指执行拥塞避免算法后，是拥塞窗口**缓慢的增大**，以防止网络过早出现拥塞。

# 五、UDP协议

Udp：面向无连接，不可靠

## **1、UDP的接收缓冲区** 每个UDP Socket都有一个接收缓冲区，没有发送缓冲区。有数据就直接发送，不管对方是否能够正确接收，也不管对端接收缓冲区是否已经满了。 UDP是没有流量控制的；快的发送者可以很容易地就淹没慢的接收者，导致接收方的UDP丢弃数据包。



## 2、Udp什么时候丢包

1、接收端处理时间过长导致丢包：调用recv方法接收端收到数据后，处理数据花了一些时间，处理完后再次调用recv方法，在这二次调用间隔里，发过来的包可能丢失。对于这种情况可以修改接收端，将包接收后存入一个缓冲区，然后迅速返回继续recv.

2、发送的包巨大丢包：虽然send方法会帮你做大包切割成小包发送的事情，但包太大也不行。例如超过50K的一个udp包，不切割直接通过send方法发送也会导致这个包丢失。这种情况需要切割成小包再逐个send.

3、发送的包较大，超过接受者缓存导致丢包：包超过mtu size数倍，几个大的udp包可能会超过接收者的缓冲，导致丢包。这种情况可以设置socket接收缓冲。以前遇到过这种问题，我把接收缓冲设置成64K就解决了。

4、发送的包频率太快：虽然每个包的大小都小于mtu size 但是频率太快，例如40多个mut size的包连续发送中间不sleep,也有可能导致丢包。这种情况也有时可以通过设置socket接收缓冲解决，但有时解决不了。所以在发送频率过快的时候还是考虑sleep一下吧。

5、局域网内不丢包，公网上丢包。这个问题我也是通过切割小包并sleep发送解决的。如果流量太大，这个办法也不灵了。总之udp丢包总是会有的，如果出现了用我的方法解决不了，还有这个几个方法： 要么减小流量，要么换tcp[协议](http://cisco.chinaitlab.com/List_11.html" \t "https://blog.csdn.net/chen_fly2011/article/details/_blank)传输，要么做丢包重传的工作。

## 3、TCP和UDP的区别

1. TCP是基于数据流传输，UDP基于包传输
2. TCP面向连接（如打电话要先拨号建立连接）;UDP是无连接的，即发送数据之前不需要建立连接
3. TCP提供可靠的服务。也就是说，通过TCP连接传送的数据，无差错，不丢失，不重复，且按序到达;UDP尽最大努力交付，即不保证可靠交付
4. Tcp通过校验和，重传控制，序号标识，滑动窗口、确认应答实现可靠传输。如丢包时的重发控制，还可以对次序乱掉的分包进行顺序控制。
5. UDP具有较好的实时性，工作效率比TCP高，适用于对高速传输和实时性有较高要求的通信或广播通信。
6. 每一条TCP连接只能是点到点的;UDP支持一对一，一对多，多对一和多对多的交互通信
7. TCP对系统资源要求较多，UDP对系统资源要求较少。

## 基于TCP和UDP的协议

（1）HTTP(HyperText Transfer Protocol，超文本传输协议）运行于TCP协议上。  
（2）TELNET (Teletype over the Network, 网络电传) ，通过一个终端(terminal)登陆到网络(运行在TCP协议上)。  
（3）FTP (File Transfer Protocol, 文件传输协议) ，由名知义(运行在TCP协议上) 。  
（4）SMTP (Simple Mail Transfer Protocol，简单邮件传输协议) ，用来发送电子邮件(运行在TCP协议上) 。  
（5）DNS (Domain Name Service，域名服务) ，用于完成地址查找，邮件转发等工作(运行在TCP和UDP协议上) 。  
（6）ECHO (Echo Protocol, 回绕协议) ，用于查错及测量应答时间(运行在TCP和UDP协议上) 。  
（7）NTP (Network Time Protocol，网络时间协议) ，用于网络同步(运行在UDP协议上) 。  
（8）SNMP (Simple Network Management Protocol, 简单网络管理协议) ，用于网络信息的收集和网络管理。(运行于UDP协议上）  
（9）BOOTP (Boot Protocol，启动协议) ，应用于无盘设备(运行在UDP协议上)。

# 六、http协议

超文本传输协议，HTTP是一个基于TCP/IP通信协议来传递数据（HTML 文件, 图片文件, 查询结果等）。

HTTP协议工作于客户端-服务端（C/S）架构为上。浏览器作为HTTP客户端通过URL向HTTP服务端即WEB服务器发送所有请求。Web服务器根据接收到的请求后，向客户端发送响应信息。

## 原理：

### 一次HTTP操作称为一个事务，其工作过程可分为四步：

### 1）**首先客户机与服务器需要建立连接**。只要单击某个超级链接，HTTP的工作开始。

### 2）**建立连接后，客户机发送一个请求给服务器**，请求方式的格式为：统一资源标识符（URL）、协议版本号，后边是MIME信息包括请求修饰符、客户机信息和可能的内容。

### 3）**服务器接到请求后，给予相应的响应信息**，其格式为一个状态行，包括信息的协议版本号、一个成功或错误的代码，后边是MIME信息包括服务器信息、实体信息和可能的内容。

### 4）**客户端接收服务器所返回的信息通过浏览器显示在用户的显示屏上，然后客户机与服务器断开连接**

## 主要特点：

### 1、简单快速：客户向服务器请求服务时，只需传送请求方法和路径。请求方法常用的有GET、HEAD、POST。每种方法规定了客户与服务器联系的类型不同。由于HTTP协议简单，使得HTTP服务器的程序规模小，因而通信速度很快。

### 2、灵活：HTTP允许传输任意类型的数据对象。正在传输的类型由Content-Type加以标记。

### 3、无连接：无连接的含义是限制每次连接只处理一个请求。服务器处理完客户的请求，并收到客户的应答后，即断开连接。采用这种方式可以节省传输时间。

### 4、无状态：HTTP协议是无状态协议。无状态是指协议**对于事务处理没有记忆能力**。缺少状态意味着如果后续处理需要前面的信息，则它必须重传，这样可能导致每次连接传送的数据量增大。另一方面，在服务器不需要先前信息时它的应答就较快。

### 5、支持B/S及C/S模式。

HTTP使用统一资源标识符（Uniform Resource Identifiers, URI）来传输数据和建立连接。URL是一种特殊类型的URI，包含了用于查找某个资源的足够的信息（不仅可以标识，而且可以定位资源）

URL,全称是UniformResourceLocator, 中文叫统一资源定位符,是互联网上用来标识某一处资源的地址

## http状态码：

状态代码有三位数字组成，第一个数字定义了响应的类别，共分五种类别:

1xx：指示信息--表示请求已接收，继续处理

2xx：成功--表示请求已被成功接收、理解、接受

3xx：重定向--要完成请求必须进行更进一步的操作

4xx：客户端错误--请求有语法错误或请求无法实现

5xx：服务器端错误--服务器未能实现合法的请求

常见状态码：

200 OK //客户端请求成功

400 Bad Request //客户端请求有语法错误，不能被服务器所理解

401 Unauthorized //请求未经授权，这个状态代码必须和WWW-Authenticate报头域一起使用

403 Forbidden //服务器收到请求，但是拒绝提供服务

404 Not Found //请求资源不存在，eg：输入了错误的URL

500 Internal Server Error //服务器发生不可预期的错误

503 Server Unavailable //服务器当前不能处理客户端的请求，一段时间后可能恢复正常

## 1、get 和post区别

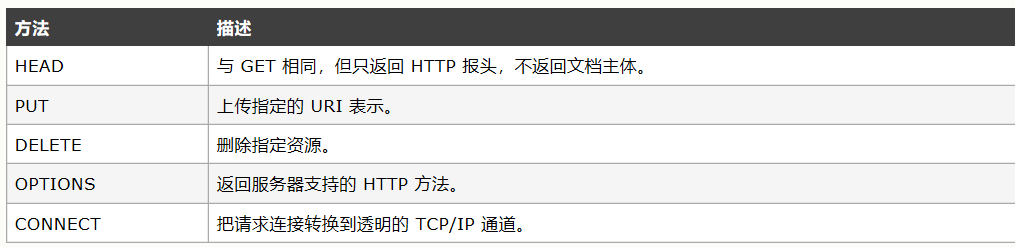


get产生一个TCP数据包；post产生两个TCP数据包。

对于get方式的请求，浏览器会把http header和data一并发送出去，服务器响应200（返回数据);

而对于POST，浏览器先发送header，服务器响应100 continue，浏览器再发送data，服务器响应200 ok（返回数据）。

## 其他http请求方法



## 3、总结

http:源地址发起请求得到目的地址，确定是哪两个进程间相互通信。

Tcp：不同进程间建立连接通信。

Ip：这两个进程所在的计算机相互连接通信。

这些协议对应的客户端和服务器都是同样的，一次通信全部协议都要用到。

## 4、tcp和http联系和区别

 http是要基于TCP连接基础上的，简单的说，TCP就是单纯建立连接，不涉及任何我们需要请求的实际数据，简单的传输。http是用来收发数据，即实际应用上来的。

1. 首先是tcp连接：三次握手
2. 前面客户端和应用服务器建立TCP连接之后，就需要用http协议来传送数据了，HTTP协议简单来说，还是请求，确认，连接。

 第一：发送一个http请求给客户端，这个请求包括请求头和请求内容：

request header：

1、请求的方法是POST/GET,请求的URL，http协议版本

2、请求的数据，和编码方式

3、是否有cookies和sessions，是否缓存等。

request body：即请求的内容.

第二：S收到了http请求，然后根据请求头，返回http响应。

response header：

1. cookies或者sessions
2. 状态码
3. 内容大小等

response body：即响应的内容，包括，JS什么的。

      第三：C收到了以后，就由浏览器完成一系列的渲染，包括执行JS脚本等。

## 5、cookies和session

**cookie机制采用的是在客户端保持状态的方案，而session机制采用的是在服务器端保持状态的方案。**

### 1、Cookie：

**e.g.如我们在网上购物的时候， 开始的时候先登录，然后选中商品加入到自己的购物车-其在客户单与服务器端的动作如下：   
客户端—–（request:包含登录信息）——->服务器端   
客户端<—（response:登录成功与否）——-服务器端   
客户端—–（request:购物车里的信息）—–>服务器端   
客户端<—（response:添加成功与否）—— -服务器端   
这里关键是在第二次请求加入购物车的时候，因为客户端请求服务端是一种无状态的连接,那么服务器端怎么知道是谁，以及加入到谁的购物车。   
解决的方式就是request到达服务器端的时候，服务器给在response中加一个“小饼干”,这个“小饼干”中就包含用户登录时候的一些基本的信息。当第二次客户端发起请求的时候，服务端检查到这个“小饼干”,据此识别出客户端，找到相应缓存，缓存里面存放着用户名，密码和一些用户设置项。。而这个所谓的“小饼干”就是Cookie,持有这个小饼干的是客户端-即客户端保存中数据。**

### 2、session：

Session中也有Cookie,只是其中保存的不再是直接的数据，而是一个ID，在服务器端，服务器根据客户端的JSESSIONID来判断。 服务器端是以键值对的方式保存数据的，这里的key就是JSESSIONID，根据key来获取value,如果没有这个JSESSIONID，就新创建加入Map中。

### 3、两者比较：

相同点：   
cookie与session都是用来跟踪浏览器用户身份的会话方式。

不同点：   
总的来说，cookie是采取的客户端保状态的会话方式，而session采取的是服务器保持状态的会话方式。   
采用session的会话方式，用户量大时，因为数据保存在服务器，其服务器压力毫无疑问会比较大。还有其他的一些不同，如二者的存取方式等

## 6、time\_wait原理

### time\_wait状态

首先调用close()发起主动关闭的一方，在发送最后一个ACK之后会进入time-wait状态，也就是说该发送方会保持2MSL事件之后才会回到初始状态。产生这种结果使得这个TCP连接在2MSL连接等待期间，定义的这个连接的四元数组（local\_ip, local\_port, remote\_ip,remote\_port）不能被使用。

### 2、time\_wait状态原因

1）可靠地实现TCP全双工连接的终止

TCP协议在关闭连接的四次握手过程中，最终的ACK是由主动关闭连接的一端（后面统称A端）发出的，如果这个ACK丢失，对方（后面统称B端）将重发出最终的FIN，因此A端必须维护状态信息（TIME\_WAIT）允许它重发最终的ACK。如果A端不维持TIME\_WAIT状态，而是处于CLOSED 状态，主动A的TCP传输层会用RST包（复位因某种原因引起出现的错误连接，也用来拒绝非法数据和请求）响应对方，这会被对方认为是有错误发生，然而这些事实上只是正常的关闭连接的过程，并非异常。

因而，要实现TCP全双工连接的正常终止，必须处理终止过程中四个分节任何一个分节的丢失情况，主动关闭连接的A端必须维持TIME\_WAIT状态 。

2）允许老的重复分节在网络中消逝

TCP分节可能由于路由器异常而“迷途”，在迷途期间，TCP发送端可能因确认超时而重发这个分节，迷途的分节在路由器修复后也会被送到最终目的地，这个迟到的迷途分节到达时可能会引起问题。在关闭“前一个连接”之后，马上又重新建立起一个相同的IP和端口之间的“新连接”，“前一个连接”的迷途重复分组在“前一个连接”终止后到达，而被“新连接”收到了。为了避免这个情况，TCP协议不允许处于TIME\_WAIT状态的连接启动一个新的可用连接，因为TIME\_WAIT状态持续2MSL（数据包在网络中的最大生存时间），就可以保证当成功建立一个新TCP连接的时候，来自旧连接重复分组已经在网络中消逝。

## 7、从输入URL到浏览器界面显示的完整过程

（1）查看缓存：无缓存或者缓存已经过期，发起请求；有缓存且可用，直接解析转码提供给客户端。

（2）如果查不到缓存，浏览器会做一个系统调用，查看hosts文件，找到返回IP；

（3）进行了前两步查询无果，根据路由器的DNS缓存，ISP的DNS缓存查找IP，找到返回IP；

（4）还找不到，就要进行DNS递归解析，从根域名服务器递归搜索，得到IP；

（5）浏览器和网站进行TCP连接，三次握手，建立连接。

（6）浏览器向主机发送一个http请求报文，包括URL，请求的方法，遵循的协议，返回的信息是否需要缓存，客户端是否发送cookies等

（7）服务接收到客户端发送的HTTP请求后，查找客户端请求的资源，并返回响应报文

（8）浏览器接受到http报文，根据情况关闭tcp连接或者重用，对响应进行解码，根据资源类型决定如何处理

（9）进行页面渲染，绘制像素点，显示在用户屏幕上

# 七、https

HTTP协议传输的数据都是未加密的，也就是明文的，因此使用HTTP协议传输隐私信息非常不安全，为了保证这些隐私数据能加密传输，HTTP下加入SSL层,对HTTP协议传输的数据进行加密，从而就诞生了HTTPS。简单来说，HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全。

## 1、HTTPS和HTTP的区别

　　1、https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。

　　2、http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。

　　3、http和https使用的是完全不同的连接方式，用的端口也不一样，前者是80，后者是443。

4、http的连接很简单，是无状态的；HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，比http协议安全。

## 2、https的工作原理

（1）客户使用https的URL访问Web服务器，要求与Web服务器建立SSL连接。

　　（2）Web服务器收到客户端请求后，会将网站的证书信息（证书中包含公钥）传送一份给客户端。

　　（3）客户端的浏览器与Web服务器开始协商SSL连接的安全等级，也就是信息加密的等级。

　　（4）客户端的浏览器根据双方同意的安全等级，建立会话密钥，然后利用网站的公钥将会话密钥加密，并传送给网站。

　　（5）Web服务器利用自己的私钥解密出会话密钥。

（6）Web服务器利用会话密钥加密与客户端之间的通信。

## 3、https的优点

（1）使用HTTPS协议可认证用户和服务器，确保数据发送到正确的客户机和服务器；

　　（2）HTTPS协议是由SSL+HTTP协议构建的可进行加密传输、身份认证的网络协议，要比http协议安全，可防止数据在传输过程中不被窃取、改变，确保数据的完整性。

　　（3）HTTPS是现行架构下最安全的解决方案，虽然不是绝对安全，但它大幅增加了中间人攻击的成本。

　　（4）谷歌曾在2014年8月份调整搜索引擎算法，并称“比起同等HTTP网站，采用HTTPS加密的网站在搜索结果中的排名将会更高”。

## 4、https的缺点

（1）HTTPS协议握手阶段比较费时，会使页面的加载时间延长近50%，增加10%到20%的耗电；

　　（2）HTTPS连接缓存不如HTTP高效，会增加数据开销和功耗，甚至已有的安全措施也会因此而受到影响；

　　（3）SSL证书需要钱，功能越强大的证书费用越高，个人网站、小网站没有必要一般不会用。

　  （4）SSL证书通常需要绑定IP，不能在同一IP上绑定多个域名，IPv4资源不可能支撑这个消耗。

（5）HTTPS协议的加密范围也比较有限，在黑客攻击、拒绝服务攻击、服务器劫持等方面几乎起不到什么作用。最关键的，SSL证书的信用链体系并不安全，特别是在某些国家可以控制CA根证书的情况下，中间人攻击一样可行。

# 八、Socket开发：（socket结构体包括so\_type,so\_pcb）

**so\_type：**

SOCK\_STREAM 提供有序的、可靠的、双向的和基于连接的字节流服务，当使用Internet地址族时使用TCP。

SOCK\_DGRAM 支持无连接的、不可靠的和使用固定大小（通常很小）缓冲区的数据报服务，当使用Internet地址族使用UDP。

SOCK\_RAW 原始套接字，允许对底层协议如IP或ICMP进行直接访问，可以用于自定义协议的开发。

**so\_pcb：**

表示socket控制块，其又指向一个结构体，该结构体包含了当前主机的ip地址(inp\_laddr)，当前主机进程的端口号(inp\_lport)，发送端主机的ip地址(inp\_faddr)，发送端主体进程的端口号

一个socket有一个so\_pcb，描述了该socket的所有信息，而每个socket有一个编号，这个编号就是socket描述符

如何实例化一个socket：so\_type+ip地址+端口号

## （1）Socket整个流程：

服务器端在调用listen之后，内核会建立两个队列，SYN队列和ACCEPT队列，其中ACCPET队列的长度由backlog指定。

服务器端在调用accept之后，将阻塞，等待ACCPET队列有元素。

客户端在调用connect之后，将开始发起SYN请求，请求与服务器建立连接，此时称为第一次握手。

服务器端在接受到SYN请求之后，把请求方放入SYN队列中，并给客户端回复一个确认帧ACK，此帧还会携带一个请求与客户端建立连接的请求标志，也就是SYN，这称为第二次握手

客户端收到SYN+ACK帧后，connect返回，并发送确认建立连接帧ACK给服务器端。这称为第三次握手

服务器端收到ACK帧后，会把请求方从SYN队列中移出，放至ACCEPT队列中，而accept函数也等到了自己的资源，从阻塞中唤醒，从ACCEPT队列中取出请求方，重新建立一个新的sockfd，并返回。

这就是listen,accept,connect这三个函数的工作流程及原理。从这个过程可以看到，在connect函数中发生了两次握手。

Accept()返回新的套接字，表示现在有空闲的连接。取的是accpet队列的第一个连接

## （2）阻塞的几种情况：

1、accept queue为空，比如三次握手还没建立完成

2、连接成功后，客户端和服务器迟迟不发送数据

3、单线程多客户端的情况，需要排队

# 九、STL容器

顺序容器和关联容器的本质区别：关联容器是通过键(key)存储和读取元素的，而顺序容器则通过元素在容器中的位置顺序存储和访问元素。

## 1、顺序容器：vector、list、deque

（1）vector：一段连续的内存地址，基于数组的实现

（2）list：一段不连续的内存，基于链表的实现

（3）deque：和vector一样，但是提供首元素插入和删除的双向支持（push\_front）

list和vector相比，允许快速插入和删除，但是随机访问比较慢

## 关联容器：map、set、multimap、multiset(允许一个键对应多个值)

### （1）Map

key-value形式（键值唯一），内部存储结构为红黑树，适合进行记录的存储和查询

### （2）Set

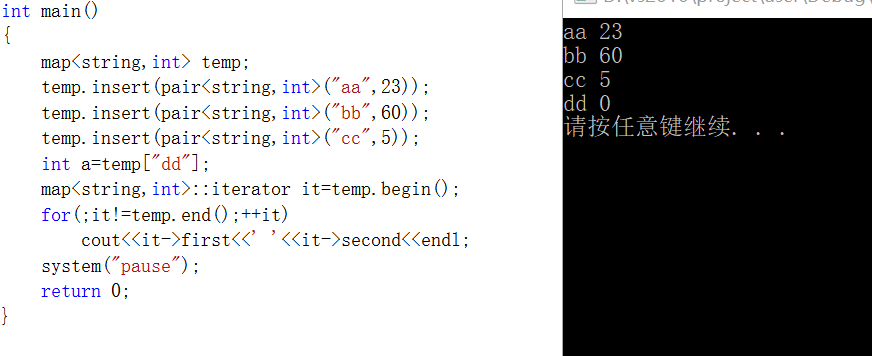
单值形式，键值和值相等。Set没有办法改值（因为改值就是改键值），而且值不重复。

问：如果vector大小只有10，访问11，会有什么结果？

实测：不一定会崩溃，视编译器而定。

问：如果访问map不存在的键值，会有什么结果？

答：不会崩溃，会插入这个键值，值为默认值。



1. Unordermap:通过hash值找到对应的value,基本上和hashmap一样，但是hashmap是非正规容器，unorderemap是c++规范化的，是无序的。
2. Multimap：结构与map相同，元素可以重复

## 3、Vector内存管理：resize和reserve

### （1）Reserve:增加了容器的预留空间capacity，但是没有增加size，在空间内不真正创建元素对象，所以在没有添加新的对象之前，不能引用容器内的元素。加入新的元素时，要调用push\_back/insert函数

### （2）Resize：改变容器的大小，且在创建对象，因此调用这个函数后，就可以 引用容器内的对象了，因此当加入新的元素时，用[]操作符，或者用迭代器来引用元素对象。此时再调用push\_back()函数，是加在这个新的空间后面的。

# 十、set底层为什么用红黑树？

map,set底层都提供了排序功能，且查找速度快。红黑树实际上是AVL的一种变形，但是其比AVL(平衡二叉搜索树)具有更高的插入效率，当然查找效率会平衡二叉树稍微低一点点，毕竟平衡二叉树太完美了。但是这种查找效率的损失是非常值得的。它的操作有着良好的最坏情况运行时间，并且在实践中是高效的: 它可以在O(log n)时间内做查找，插入和删除，这里的n是树中元素的数目。

## （2）红黑树的特点

# 十一、map和hashmap

## 1、hash\_map原理：

首先分配一大片内存，形成许多桶。是利用hash函数，对key进行映射到不同区域（桶）进行保存。其插入过程是：

（1）得到key

（2）通过hash函数得到hash值

（3）得到桶号(一般都为hash值对桶数求模)

（4）存放key和value在桶内。

其取值过程是:

（1）得到key

（2）通过hash函数得到hash值

（3）得到桶号(一般都为hash值对桶数求模)

（4）比较桶的内部元素是否与key相等，若都不相等，则没有找到。

（5）取出相等的记录的value

## **2、hash\_map和map的区别**

### 构造函数。hash\_map需要hash函数，等于函数；map只需要比较函数(小于函数).

### 存储结构。hash\_map采用hash表存储，map一般采用红黑树(RB Tree)实现。因此其memory数据结构是不一样的。

频繁的插入和删除数据，用map更快。频繁在大量数据中查询，用hash\_map更快，几乎是常数时间，代价仅仅是消耗比较大的内存。

## 3、什么时候需要用hash\_map，什么时候需要用map?

总体来说，hash\_map 查找速度会比map快，而且查找速度基本和数据数据量大小，属于常数级别;而map的查找速度是log(n)级别。并不一定常数就比log(n)小，hash还有hash函数的耗时，如果你考虑效率，特别是在元素达到一定数量级时，可以考虑使用hash\_map。但若你对内存使用特别严格，希望程序尽可能少消耗内存，那么一定要小心，hash\_map可能会让你陷入尴尬，特别是当你的hash\_map对象特别多时，你就更无法控制了，而且hash\_map的构造速度较慢。

**权衡三个因素: 查找速度, 数据量, 内存使用**

## 4、hashmap是不是线程安全

答:是线程不安全的，原因为hash碰撞和扩容

1. hash碰撞：如果多个线程，在某一时刻同时操作HashMap并执行put操作，而有大于两个key的hash值相同，这个时候需要解决碰撞冲突，通过拉链法解决冲突，暂且不讨论是从链表头部插入还是从尾部初入，这个时候两个线程如果恰好都取到了对应位置的头结点e1，而最终的结果可想而知，a1、a2两个数据中势必会有一个会丢失
2. 扩容：当多个线程同时检测到总数量超过门限值（hashmap的初始容量）的时候就会同时调用resize操作，各自生成新的数组并rehash后赋给该map底层的数组table，结果最终只有最后一个线程生成的新数组被赋给table变量，其他线程的均会丢失。而且当某些线程已经完成赋值而其他线程刚开始的时候，就会用已经被赋值的table作为原始数组，这样也会有问题。

# 十二、迭代器

迭代器iterator 提供了一种一般化的方法对顺序或关联容器类型中的每个元素进行连续访问

begin()返回一个iterator 它指向容器的第一个元素

end()返回一个iterator 它指向容器的末尾元素的**下一个**位置

使用方法：

e.g.

vector<int> ::iterator it;

For(int it=vec.begin();it!=vec.end();++it){}; 跳出循环后，it指向末尾元素的下一个位置

**1、在定义vector的迭代器it后，如果it指向末尾元素，此时删除末尾元素，cout<<\*it会报错。**

**2、定义list的迭代器it后，如果it指向末尾元素，此时删除末尾元素，cout<<\*it不会报错，会返回此时的末尾元素**

**顺序容器的迭代器：删除一个元素后，之后的所有元素的迭代器全部失效**

**关联容器的迭代器：删除一个元素后，只有删除元素的迭代器失效，其他的元素仍然可以正常输出**

# 十三、拷贝构造函数

## 1、什么时候使用拷贝构造函数

### （1）对象作为值传递

### （2）对象作为函数返回值

### （3）用一个对象给另一个对象初始化

## 2、为什么拷贝构造函数的参数只能是引用？

答、**防止无限递归调用**。当 一个对象需要以值方式传递时，编译器会生成代码调用它的拷贝构造函数以生成一个复本。如果类A的拷贝构造函数是以值方式传递一个类A对象作为参数的话，当 需要调用类A的拷贝构造函数时，需要以值方式传进一个A的对象作为实参； 而以值方式传递需要调用类A的拷贝构造函数；结果就是调用类A的拷贝构造函数导 致又一次调用类A的拷贝构造函数，这就是一个无限递归。

## 深拷贝和浅拷贝

深拷贝：复制对象的同时，重新分配资源

浅拷贝：只是复制了对象，依然指向同一块内存，如果通过原来的对象释放了内存，那么就导致新的对象变成野指针。

## 引用和指针的区别

1. 不能有空引用，但是可以有空指针，且引用要在定义的时候初始化
2. 引用只是一个别名，而指针是一个变量，存储的是地址，指向内存的一个存储单元。
3. 指针的值在初始化后可以改变，即指向其它的存储单元，而引用在进行初始化后就不会再改变了
4. Sizeof(指针)得到的是指针本身的大小,sizeof（引用）得到的是所指变量的大小
5. 有多级指针，但是没有多级引用，只能有一级引用。
6. 引用访问一个变量是直接访问，而指针访问一个变量是间接访问
7. 自增运算意义不一样（指针是指向下一个空间，引用时引用的变量值加1）

# 十四、线程部分

## 1、线程和进程的区别

1. 进程是资源分配和调度的独立单位，线程是cpu调度的基本单位
2. 同一个进程可以包含多个线程，多个线程共享进程的资源，一个进程至少包含一个线程。
3. （进程的创建调用fork或者vfork，而线程的创建调用pthread\_create，指的是linux）进程结束后它拥有的所有线程都将销毁，而线程的结束不会影响同个进程中的其他线程的结束（但是一个线程崩溃，所在的进程就崩溃了）
4. 线程是轻量级的进程，它的创建和销毁所需要的时间比进程小很多，所有操作系统中的执行功能都是创建线程去完成的
5. 线程中执行时一般都要进行同步和互斥，因为他们共享同一进程的所有资源
6. 线程的cpu利用率比较高

## 2、线程的几种状态

* 1. 新建：新创建了一个线程对象。
  2. 就绪：该状态的线程位于“**可运行线程池**”中，变得可运行，只**等待获取[CPU](http://product.it168.com/list/b/0217_1.shtml" \t "_blank" \o "CPU)的使用权**。**即在就绪状态的进程除CPU之外，其它的运行所需资源都已全部获得。**
  3. 运行：就绪状态的线程获取了CPU，执行程序代码。
  4. 阻塞：线程因为某种原因放弃CPU使用权，暂时停止运行

## 3、如何创建线程？

1. C++底部的，并不是Windows标准API，创建线程函数，该函底层调用CreateThread。

e.g DWORD WINAPI ThreadProcess(LPVOID LpParam)

{

…;

}

void main()

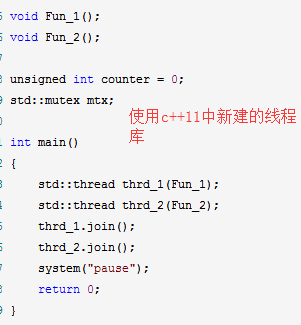
{

creatThread(Null,0, ThreadProcess,Null,0,Null);

}

首先DWORD是数据类型，在这里是返回值，WINAPI是函数的调用形式，最终就是\_stdcall形式，就被调用函数来清理栈中的参数，这种形式是不能实现可变参数的。

1. 使用C++新增的线程库，加thread头文件：



## 4、如何终止线程：ExitThread()

Linux环境下用Thread\_create创建线程，pthread\_exit(NULL)终止线程

## 5、多线程的同步和互斥问题

### 线程同步是指同一进程中的多个线程互相协调工作从而达到一致性。之所以需要线程同步，是因为多个线程同时对一个数据对象进行修改操作时，可能会对数据造成破坏

### （1）使用C++11标准库的thread、mutex头文件：(线程1和2访问的次数不固定)

在线程关联的地方加lock\_guard< mutex> mtx\_locker(mtx);



### （2）使用windows API的临界区对象：(线程1和2访问的次数不固定)

临界区是指一个**访问共享资源的代码段**，临界区对象则是指当用户使用某个线程访问共享资源时，必须使代码段独占该资源，不允许其他线程访问该资源。在该线程访问完资源后，其他线程才能对资源进行访问

EnterCriticalSection，该函数的作用是使调用该函数的线程进入已经初始化的临界区，并拥有该临界区的所有权

LeaveCriticalSection，该函数的作用是使调用该函数的线程离开临界区并释放对该临界区的所有权，以便让其他线程也获得访问该共享资源的机会。

### （3）使用Windows API的事件对象(线程1和2依次访问)

事件对象是一种**内核对象**，用户在程序中使用内核对象的有无信号状态来实现线程的同步

主函数调用：HANDLE h\_event = CreateEvent(nullptr, true, false, nullptr);

ResetEvent(h\_event);

线程里：WaitForSingleObject(h\_event, INFINITE);

用完：SetEvent(h\_event);

### （4）使用Windows API的互斥对象(线程1和2依次访问)

主函数：h\_mutex = CreateMutex(nullptr, false, nullptr);

线程：WaitForSingleObject(h\_mutex, INFINITE);

用完：ReleaseMutex(h\_mutex);

### （5）信号量

信号量是维护0到指定最大值之间的同步对象。信号量状态在其计数大于0时是有信号的，而其计数是0时是无信号的。信号量对象在控制上可以支持有限数量共享资源的访问。

信号量的特点和用途可用下列几句话定义：

1. 如果当前资源的数量大于0，则信号量有效；  
   （2）如果当前资源数量是0，则信号量无效；  
   （3）系统决不允许当前资源的数量为负值；  
   （4）当前资源数量决不能大于最大资源数量。

## 6、并发和并行

并行：同一时刻同时做多件事情,

并发：同一时间间隔内做多件事情

我们通常使用的计算机中只有一个CPU，也就是说只有一颗心，要让它一心多用，同时运行多个进程，就必须使用并发技术。实现并发技术相当复杂，最容易理解的是“时间片轮转进程调度算法”，它的思想简单介绍如下：在操作系统的管理下，所有正在运行的进程轮流使用CPU，每个进程允许占用CPU的时间非常短(比如10毫秒)，这样用户根本感觉不出来CPU是在轮流为多个进程服务，就好象所有的进程都在不间断地运行一样。但实际上在任何一个时间内有且仅有一个进程占有CPU。如果一台计算机有多个CPU，情况就不同了，如果进程数小于CPU数，则不同的进程可以分配给不同的CPU来运行，这样，多个进程就是真正同时运行的，这便是并行。但如果进程数大于CPU数，则仍然需要使用并发技术。

## 7、线程共享进程的哪些资源

### （1）线程共享的环境包括：

### 1.进程代码段

### 2.进程的公有数据(利用这些共享的数据，线程很容易的实现相互之间的通讯)

### 3.进程打开的文件描述符、信号的处理器、进程的当前目录和进程用户ID与进程组ID。

### （2）线程独立的资源包括：

### 1.线程ID

每个线程都有自己的线程ID，这个ID在本进程中是唯一的。进程用此来标识线程。

2.寄存器组的值

由于线程间是并发运行的，每个线程有自己不同的运行线索，当从一个线程切换到另一个线程上 时，必须将原有的线程的寄存器集合的状态保存，以便将来该线程在被重新切换到时能得以恢复。

3.线程的堆栈

堆栈是保证线程独立运行所必须的。线程函数可以调用函数，而被调用函数中又是可以层层嵌套的，所以线程必须拥有自己的函数堆栈， 使得函数调用可以正常执行，不受其他线程的影响。

4.错误返回码

由于同一个进程中有很多个线程在同时运行，可能某个线程进行系统调用后设置了errno值，而在该 线程还没有处理这个错误，另外一个线程就在此时被调度器投入运行，这样错误值就有可能被修改。所以，不同的线程应该拥有自己的错误返回码变量。

5.线程的信号屏蔽码

由于每个线程所感兴趣的信号不同，所以线程的信号屏蔽码应该由线程自己管理。但所有的线程都 共享同样的信号处理器。

6.线程的优先级

由于线程需要像进程那样能够被调度，那么就必须要有可供调度使用的参数，这个参数就是线程的优先级。

## 8、信号量的PV操作

对于信号量，可以认为是一个仓库，有两个概念，容量和当前的货物个数。

P操作从仓库拿货，如果仓库中没有货，线程一直等待，直到V操作，往仓库里添加了货物，为了避免P操作一直等待下去，会有一个超时时间。

V操作往仓库送货，如果仓库满了，线程等待，直到有P操作，从仓库中拿走货物，有空的位置。

创建信号量，设置容量，先有V操作，才能P操作。

P操作：货物个数减1，减过之后，货物个数大于等于0，说明已经拿到货物，线程继续。否者线程阻塞。**（即有线程要访问使用资源）**

V操作：货物个数加1，加过之后，货物个数小于等于容量，说明添加成功，线程继续。否者线程阻塞。**（线程要释放资源）**

信号量：0≤ 信号量≤容量 ，取值 表示当前可以使用的货物；

　　      信号量<0 ，  取值 表示当前等待使用货物的线程；

            信号量>容量 ，  信号量-容量 表示当前等待添加货物的线程。

通常，信号量的容量设置很大，可以一直V操作，不会阻塞，但是P操作的时候，很可能阻塞。

信号量如何做到线程同步

可以认为信号量关联一组线程，保存一个指针，指向线程数组的首地址。比如当前信号量为-1，进行P操作，信号量为-2，说明没有拿到货物，线程等待，取值为-2，说明有两个线程等待那货物。这个时候，其他线程进行V操作，信号量加1，为-1，信号量通知等待的线程中，第一个线程继续执行，第二个线程继续等待。

也就是说，P操作等待的情况是减1后，信号量小于0。 P操作继续执行的情况有两种：a、减1后，信号量大于等于0，不需等待，直接执行；b、信号量小于0，等待中，其他人进行了V操作，通知这个线程，继续执行。

总结：信号量为n时，表示有n个可用资源；信号量为-n时，表示有n个等待进程。因此，只有信号量大于0时，线程才能访问资源，且访问了资源后，信号量减1。

# 十五、进程部分

## 1、进程定义：

系统进行**资源分配和调度的基本单位**，进程是程序的基本执行实体

如何创建进程：

BOOL ret = CreateProcess(NULL, chCommandLine, NULL, NULL, FALSE, 0, NULL, NULL, &si, &pi);

关闭进程：CloseHandle(pi.hThread);

## 进程间通信（IPC）

### 管道：半双工通信方式，具有亲缘关系的进程之间的通信。

### 命名管道（FIFO）：FIFO可以在无关的进程之间交换数据，与无名管道不同

### 消息队列：是消息的链接表，存放在内核中，一个消息队列由一个标识符来标识。

### 信号量：它是一个计数器。信号量用于实现进程间的互斥与同步，而不是用于存储进程间通信数据。

### 5、共享内存（Shared Memory），指两个或多个进程共享一个给定的存储区。

特点：

（1）**共享内存是最快的一种 IPC，因为进程是直接对内存进行存取。**

（2）因为多个进程可以同时操作，所以需要进行同步。

（3）信号量+共享内存通常结合在一起使用，信号量用来同步对共享内存的访问。

### 6、套接字

## 3、多线程和多进程

### 区别

1. 多进程之间的资源是相互独立的，多线程之间共享一个进程的资源
2. 通信：多线程之间可以通过直接读写进程数据段（如全局变量来进行通信）--需要进程同步和互斥手段的辅助，以保证数据的一致性；而多进程间的通信是通过ipc实现
3. 多线程之间的切换比多进程之间的切换要快。

### 优缺点

多进程优点：

　　1、每个进程互相独立，不影响主程序的稳定性，子进程崩溃没关系；

　　2、通过增加CPU，就可以容易扩充性能；

　　3、可以尽量减少线程加锁/解锁的影响，极大提高性能，就算是线程运行的模块算法效率低也没关系；

　　4、每个子进程都有2GB地址空间和相关资源，总体能够达到的性能上限非常大。

　多进程缺点：

　　1、逻辑控制复杂，需要和主程序交互；

　　2、需要跨进程边界，如果有大数据量传送，就不太好，适合小数据量传送、密集运算 多进程调度开销比较大；

　　3、最好是多进程和多线程结合，即根据实际的需要，每个CPU开启一个子进程，这个子进程开启多线程可以为若干同类型的数据进行处理。当然你也可以利用多线程+多CPU+轮询方式来解决问题……

　　4、方法和手段是多样的，关键是自己看起来实现方便有能够满足要求，代价也合适。

　　多线程：

　多线程的优点：

　　1、无需跨进程边界；

　　2、程序逻辑和控制方式简单；

　　3、所有线程可以直接共享内存和变量等；

　　4、线程方式消耗的总资源比进程方式好。

　多线程缺点：

　　1、每个线程与主程序共用地址空间，受限于2GB地址空间；

　　2、线程之间的同步和加锁控制比较麻烦；

　　3、一个线程的崩溃可能影响到整个程序的稳定性；

　　4、到达一定的线程数程度后，即使再增加CPU也无法提高性能，例如Windows Server 2003，大约是1500个左右的线程数就快到极限了（线程堆栈设定为1M），如果设定线程堆栈为2M，还达不到1500个线程总数；

5、线程能够提高的总性能有限，而且线程多了之后，线程本身的调度也是一个麻烦事儿，需要消耗较多的CPU。

# 十六、I/O阻塞和非阻塞、同步和异步

## 阻塞和非阻塞：

如果A等待的过程中挂起，称为阻塞

如果A等待的过程中不会挂起，称为非阻塞

## 同步和异步：

如果A等待B的结果，称为同步

如果A不等待B的结果。称为异步

## 异步实现方式

状态：任务A去查询任务B的结果如何

通知：任务B执行完成后通知任务A来实现

回调：任务A定义一个回调函数，任务B结束后自动调用回调函数

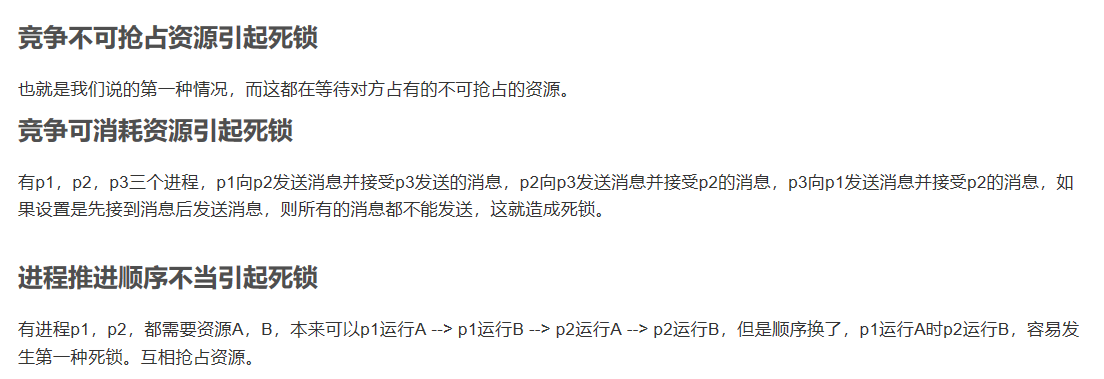
# 十七、死锁的原因，如何避免死锁

## 1、死锁定义：

死锁是指两个或两个以上的进程在执行过程中，由于竞争资源或者由于彼此通信而造成的一种阻塞的现象，若无外力作用，它们都将无法推进下去。此时称系统处于死锁状态或系统产生了死锁，这些永远在互相等待的进程称为死锁进程。

## 2、产生死锁的四个必要条件：

1. 互斥条件：一个资源每次只能被一个进程使用。  
   （2） 请求与保持条件：一个进程因请求资源而阻塞时，对已获得的资源保持不放。  
   （3） 不剥夺条件:进程已获得的资源，在未使用完之前，不能强行剥夺。  
   （4） 循环等待条件:若干进程之间形成一种头尾相接的循环等待资源关系。



## 3、处理死锁的方法

（1）预防死锁。破坏死锁产生的必要条件中的一个或多个。注意，互斥条件不能被破坏，否则会造成结果的不可再现性。   
（2）避免死锁。在资源分匹配过程中，防止系统进入不安全区域。 （银行家算法、安全性算法）  
（3）检测死锁。通过检测机构检测死锁的发生，然后采取适当措施解除死锁。（画锁的关系图）   
（4）解除死锁。在检测机构检测死锁发生后，采取适当措施解除死锁。

## 4、预防死锁方法

### （1）破坏请求和保持条件

**协议1**

所有进程开始前，必须一次性地申请所需的所有资源，这样运行期间就不会再提出资源要求，破坏了请求条件，即使有一种资源不能满足需求，也不会给它分配正在空闲的资源，这样它就没有资源，就破坏了保持条件，从而预防死锁的发生。

**协议2**

允许一个进程只获得初期的资源就开始运行，然后再把运行完的资源释放出来。然后再请求新的资源。

### （2）破坏不剥夺条件

当一个已经保持了某种不可抢占资源的进程，提出新资源请求不能被满足时，它必须释放已经保持的所有资源，以后需要时再重新申请。

### （3）破坏循环等待条件

对系统中的所有资源类型进行线性排序，然后规定每个进程必须按序列号递增的顺序请求资源。假如进程请求到了一些序列号较高的资源，然后有请求一个序列较低的资源时，必须先释放相同和更高序号的资源后才能申请低序号的资源。多个同类资源必须一起请求

## 线程避免死锁：

### 加锁顺序（线程按照一定的顺序加锁）

### 加锁时限（线程尝试获取锁的时候加上一定的时限，超过时限则放弃对该锁的请求，并释放自己占有的锁）

### 死锁检测

每当一个线程获得了锁，会在线程和锁相关的数据结构中（map、graph等等）将其记下。除此之外，每当有线程请求锁，也需要记录在这个数据结构中。当一个线程请求锁失败时，这个线程可以遍历锁的关系图看看是否有死锁发生

# 十八、虚析构函数作用：

（1）派生类指针操作派生类对象，基类不是虚析构函数：先析构派生类，再析构基类

（2）基类类指针操作派生类对象，基类不是虚析构函数：只析构基类

（3）基类指针操作派生类对象，基类是虚析构函数：先析构派生类，再析构基类

原因是基类指针指向了派生类对象，而基类中的析构函数却是非virtual的，之前讲过，**虚函数是动态绑定的基础**。现在析构函数不是virtual的，因此不会发生动态绑定，而是静态绑定，指针的静态类型为基类指针，因此在delete时候只会调用基类的析构函数，而不会调用派生类的析构函数。（A \*p=new B();）

# 十九、虚函数和纯虚函数

## 1、虚函数

1、虚函数：运行多态的一种机制

2、虚函数只能用于修饰类的普通成员函数，不可以是全局函数，否则会编译错误。

3、每个含有虚函数的类有一张虚函数表，表中每一项是一个虚函数的地址， 也就是说，虚函数表的每一项是一个虚函数的指针

4、为什么构造函数不能是虚函数：虚函数对应一个vtable，这大家都知道，可是这个vtable其实是存储在对象的内存空间的。问题出来了，如果构造函数是虚的，就需要通过 vtable来调用，可是对象还没有实例化，也就是内存空间还没有，无法找到vtable，所以构造函数不能是虚函数。

5、友元函数（不能被继承）、普通函数（不能被继承）、静态成员函数（整个类共有，不属于对象）、内联函数（编译时期绑定）、构造函数（还没生成对象）都不能是虚函数

## 2、函数指针

指向函数的指针，函数的本质就是一段二进制代码，我们可以通过指针指向这段代码的开头，计算机就会从这个开头一直往下执行，直到函数结束，并且通过指令返回回来。函数的指针与普通的指针本质上是一样的，也是由四个基本的内存单元组成，存储着内存的地址，这个地址就是函数的首地址。

e.g int (\*fun)() 定义了一个返回值为int的函数的指针。

## 3、虚函数实现原理

虚函数表指针：类中除了定义的函数成员，还有一个成员是虚函数表指针（占四个基本内存单元），这个指针指向一个虚函数表的起始位置，这个表会与类的定义同时出现，这个表存放着该类的虚函数指针，调用的时候可以找到该类的虚函数表指针，通过虚函数表指针找到虚函数表，通过虚函数表的偏移找到函数的入口地址，从而找到要使用的虚函数。

1. 实例化子类对象，子类没有定义虚函数：从父类中继承了虚函数，所以在实例化该类子类对象的时候也会产生一个虚函数表，这个虚函数表是子类的虚函数表，但是记录的子类的虚函数地址却是与父类的是一样的。所以通过子类对象的虚函数表指针找到自己的虚函数表，在自己的虚函数表找到的要执行的函数指针也是父类的相应函数入口的地址。
2. 实例化子类对象，子类定义了自己相应的函数：虚函数表当中关于这个函数的指针就会覆盖掉原有的指向父类函数的指针的值，换句话说就是指向了自己定义的相应函数，这样如果用父类的指针，指向子类的对象，就会通过子类对象当中的虚函数表指针找到子类的虚函数表，从而通过子类的虚函数表找到子类的相应虚函数地址，而此时的地址已经是该函数自己定义的虚函数入口地址，而不是父类的相应虚函数入口地址，所以执行的将会是子类当中的虚函数

## 2、纯虚函数

纯虚函数的定义：

virtual 函数类型 函数名（参数表列）=0；

特点：纯虚函数没有函数体；

最后面的“=0”只是形式，通知编译系统这是一个纯虚函数，没有任何别的意义；

比如：

virtual float area()const{return 0;}这只是一个虚函数可不是纯虚函数，因为它已经将函数实现了，有函数体

virtual float area()=0这是一个纯虚函数

如果在一个类中声明了纯虚函数，而在其派生类中没有对该函数定义，则该虚函数在派生类中依然为纯虚函数。也只有在整个派生类的某一级被定义实现，它就不是纯虚函数了。

拥有纯虚函数的类（即抽象类）不能实例化，但是可以创建其子类的对象，基类的引用和指针可以引用子类的对象，所以拥有纯虚函数的类可以定义指针和引用。

作用：在[基类](https://baike.baidu.com/item/%E5%9F%BA%E7%B1%BB/9589663" \t "https://baike.baidu.com/item/%E7%BA%AF%E8%99%9A%E5%87%BD%E6%95%B0/_blank)中不能对虚函数给出有意义的[实现](https://baike.baidu.com/item/%E5%AE%9E%E7%8E%B0/10801654" \t "https://baike.baidu.com/item/%E7%BA%AF%E8%99%9A%E5%87%BD%E6%95%B0/_blank)，而把它声明为纯虚函数，它的实现留给该基类的[派生类](https://baike.baidu.com/item/%E6%B4%BE%E7%94%9F%E7%B1%BB/9589520" \t "https://baike.baidu.com/item/%E7%BA%AF%E8%99%9A%E5%87%BD%E6%95%B0/_blank)去做；纯虚函数**相当于接口**，不能直接实例化，需要派生类来实现函数定义

# 二十、抽象类和接口

## 抽象类：是一些不能用来定义对象，而只作为基类被继承的类。含有纯虚函数的类被称为抽象类。哪怕只有一个纯虚函数，那么这个类也是一个抽象类，纯虚函数没有函数体，所以抽象类不允许实例化对象，抽象类的子类也可以是一个抽象类。抽象类子类只有把抽象类当中的所有的纯虚函数都做了实现才可以实例化对象。

## 1、抽象类与具体类：

### （1）凡是包含纯虚函数的类都是抽象类

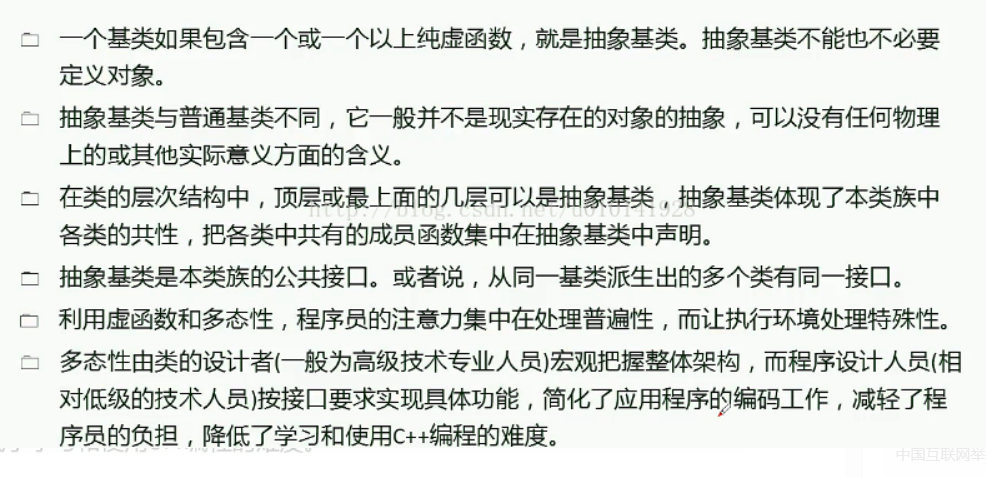
### （2）纯虚函数不用实现，所以不能被调用，抽象类无法建立对象；

### **（3）抽象类的作用是作为一个类族的共同基类，即为一个类族提供给一个公共接口**。

### （4）如果在抽象类所派生出的新类中对基类的所有纯虚函数进行定义，那么这些函数就可以被调用，这个派生类就不是抽象类，而是可以用来定义对象的具体类。

### （5）如果在派生类中没有对所有纯虚函数进行定义，则次派生类仍然是抽象类，不能用来定义对象。

### （6）并不是抽象类中的成员函数都是纯虚函数



## 2、接口：

如果在抽象类当中仅含有纯虚函数而不含其他任何东西，我们称之为接口类。

### 没有任何数据成员

### 仅有成员函数

### 成员函数都是纯虚函数

实际的工作中接口类更多的表达一种能力或协议

## 3、抽象类和接口的联系和区别：

### 联系

1、不能实例化；

2、包含未实现的方法声明；

3、派生类必须实现未实现的方法，抽象类是抽象方法，接口则是所有成员（不仅是方法包括其他成员）。

### 区别

1、类可以实现无限个接口，但仅能从一个抽象（或任何其他类型）类继承，从抽象类派生的类仍可实现接口，从而得出接口是用来解决[多重继承](https://baike.baidu.com/item/%E5%A4%9A%E9%87%8D%E7%BB%A7%E6%89%BF" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E7%B1%BB/_blank)问题的。

2、抽象类当中可以存在非抽象的方法，可接口不能，且它里面的方法只是一个声明必须用public来修饰没有具体实现的方法。

3、抽象类中的[成员变量](https://baike.baidu.com/item/%E6%88%90%E5%91%98%E5%8F%98%E9%87%8F" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E7%B1%BB/_blank)可以被不同的[修饰符](https://baike.baidu.com/item/%E4%BF%AE%E9%A5%B0%E7%AC%A6" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E7%B1%BB/_blank)来修饰，可接口中的成员变量默认的都是静态[常量](https://baike.baidu.com/item/%E5%B8%B8%E9%87%8F" \t "https://baike.baidu.com/item/%E6%8A%BD%E8%B1%A1%E7%B1%BB/_blank)（static final）。

4、抽象类是对象的抽象，然而接口是一种行为规范。

# 二十一、虚继承

虚继承是解决C++多重继承问题的一种手段，从不同途径继承来的同一基类，会在子类中存在多份拷贝。这将存在两个问题：其一，浪费存储空间；第二，存在二义性问题，通常可以将派生类对象的地址赋值给基类对象，实现的具体方式是，将基类指针指向继承类（继承类有基类的拷贝）中的基类对象的地址，但是多重继承可能存在一个基类的多份拷贝，这就出现了二义性。

虚继承底层实现原理与编译器相关，一般通过虚基类指针和虚基类表实现，每个虚继承的子类都有一个虚基类指针（占用一个指针的存储空间，4字节）和虚基类表（不占用类对象的存储空间）（需要强调的是，虚基类依旧会在子类里面存在拷贝，只是仅仅最多存在一份而已，并不是不在子类里面了）；当虚继承的子类被当做父类继承时，虚基类指针也会被继承。

实际上，vbptr指的是虚基类表指针（virtual base table pointer），该指针指向了一个虚基类表（virtual table），虚表中记录了虚基类与本类的偏移地址；通过偏移地址，这样就找到了虚基类成员，而虚继承也不用像普通多继承那样维持着公共基类（虚基类）的两份同样的拷贝，节省了存储空间。

# 二十二、常用的设计模式

## 单例模式：见二十三

## 简单工厂模式

定义一个类来负责创建其他类的实例，比如四则运算类，不同情况调用加减乘除不同类。

## 工厂模式

定义一个类的接口，让子类决定实例化哪一个类

## 生产者-消费者模式：

e.g 信号量的pv操作

# 二十三、单例模式：

## 优点：

在内存里只有一个实例，**减少了内存的开销**，尤其是频繁的创建和销毁实例（比如管理学院首页页面缓存）； 避免对资源的多重占用（比如写文件操作）。

如果不使用机制对窗口对象进行唯一化，将弹出多个窗口，如果这些窗口显示的内容完全一致，则是重复对象，浪费内存资源；如果这些窗口显示的内容不一致，则意味着在某一瞬间系统有多个状态，与实际不符，也会给用户带来误解，不知道哪一个才是真实的状态。因此有时确保系统中某个对象的唯一性即一个类只能有一个实例非常重要。

## 单例模式的特点：

1. 单例模式只能有一个实例
2. 单例模式必须自己创建自己的唯一实例
3. 单例模式必须给其他对象提供这一实例

## 单例模式的实现方式：

1. 私有的构造方法（构造函数是私有的）
2. 指向自己实例的私有引用（创建对象时静态的私有的）
3. 以自己实例为返回值的静态的公有的方法

### （1）懒汉模式（线程不安全）

这种方式是最基本的实现方式，这种实现最大的问题就是不支持多线程。

public class Singleton {

private static Singleton instance;

private Singleton (){}

public static Singleton getInstance() {

if (instance == null) {

instance = new Singleton();

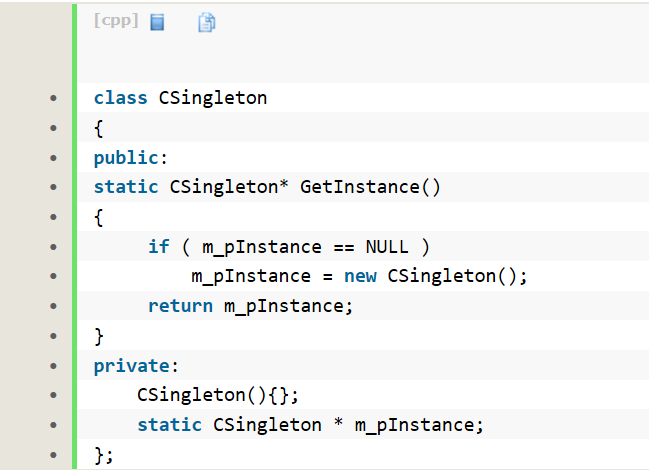
}

return instance;

}

}

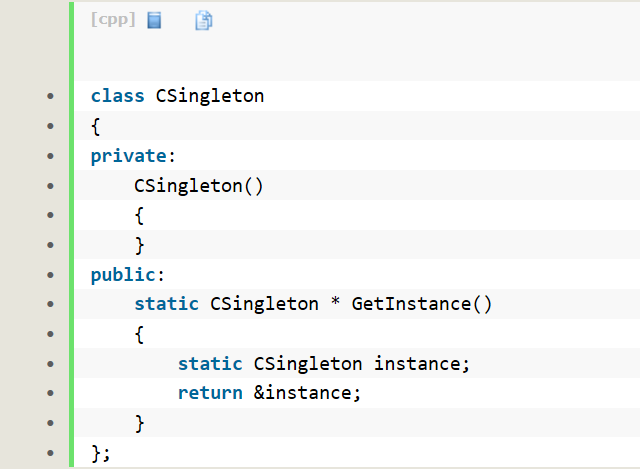
C++实现代码：



### （2）饿汉式：

优点：没有加锁，执行效率会提高。  
缺点：**类加载时就初始化，**浪费内存。第一次加载类的时候会连带着创建Singleton实例，这样的结果与我们所期望的不同，因为创建实例的时候可能并不是我们需要这个实例的时候。同时如果这个Singleton实例的创建非常消耗系统资源，而应用始终都没有使用Singleton实例，那么创建Singleton消耗的系统资源就被白白浪费了。

C++实现代码：



### （3）双检锁/双重校验锁：懒汉式

C++实现代码：



### （4）双重锁的作用：

**在加锁的的前后都进行singleton==null的判断。**

**第一个判断的作用：由于加锁影响效率，因此就加第一重判断，只有在实例还没生成的时候，才进行lock（）语句**

**第二个判断的作用：假设有两个线程A和B同时进入第一重判断后，A进入lock（）语句后，生成实例，然后退出lock。然后接下来B进去lock语句，如果没有第二重判断，就又生成了一个实例。违背了单例的原则。**

# 二十四、堆和栈

## 1、堆和栈的区别：

1、堆是由程序员分配，栈是由系统分配

2、对于堆来讲，生长方向是向上的，也就是向着内存地址增加的方向；对于栈来讲，它的生长方向是向下的，是向着内存地址减小的方向增长。

3、栈的资源由系统释放，速度比较快，堆由程序员手动释放，容易产生碎片。

4、栈的内存大小是有限的，堆的内存大小比较大（32位系统下可以达到4G）。

## 2、New和malloc的区别

1、new不用指明需要分配的内存大小，malloc需要说明分配内存的大小

2、new是返回指针对象类型，malloc需要

3、new可以重载，malloc不能重载

4、new是在自由存储区进行的操作，而malloc是在堆上

5、new是关键字，malloc是标准库函数

6、new失败时抛出异常，malloc返回null

7、自定义类型只能用new

## 3、内存管理

1、栈区（stack）— 由编译器自动分配释放 ，存放为运行函数而分配的局部变量、函数参数、返回数据、返回地址等。  
2、堆区（heap） — 一般由程序员分配释放，new, malloc之类的，若程序员不释放，程序结束时可能由OS回收 。

3、全局区（静态区）（static）— 存放全局变量、静态数据、常量。程序结束后由系统释放。  
4、文字常量区 — 常量字符串就是放在这里的。程序结束后由系统释放。

## 4、堆、栈溢出问题

## [堆栈溢出](https://baike.baidu.com/item/%E5%A0%86%E6%A0%88%E6%BA%A2%E5%87%BA" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88%E6%BA%A2%E5%87%BA/_blank)就是不顾堆栈中分配的局部[数据块](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%9D%97" \t "https://baike.baidu.com/item/%E5%A0%86%E6%A0%88%E6%BA%A2%E5%87%BA/_blank)大小，向该数据块写入了过多的数据，导致数据越界，结果覆盖了别的数据。溢出是指这个数据结构满溢，不能存放更多数据

堆溢出：e.g不断创建新对象，用完没有销毁

栈溢出：e.g递归层数太多

# 二十五、内存泄漏问题

## 1、内存泄漏原因

1. **正确的使用new和delete运算符，需要注意的是new和delete要匹配使用，对于初学者这种情况是最常出现的。一般出错的地方像如下的例子，在指针p的值被另一个函数所使用。**
2. **释放对象数组时，没有使用delete[]**

方括号是告诉编译器这个指针指向的是一个对象数组，同时也告诉编译器正确的对象地址值并调用对象的析构函数，如果没有方括号，那么这个指针就被默认为只指向一个对象，对象数组中的其他对象的析构函数就不会被调用，结果造成了内存泄露。如果在方括号中间放了一个比对象数组大小还大的数字，那么编译器就会调用无效对象（内存溢出）的析构函数，会造成堆的奔溃。如果方括号中间的数字值比对象数组的大小小的话，编译器就不能调用足够多个析构函数，结果会造成内存泄露。

1. **缺少拷贝构造函数。在类里存在成员变量是指针时，在进行赋值=运算和按值传参时，必须重载拷贝构造函数，重新实现其指针拷贝的部分.**

这是由于编译系统在我们没有自己定义拷贝构造函数时，会在拷贝对象时调用默认拷贝构造函数，进行的是浅拷贝！即对指针name拷贝后会出现两个指针指向同一个内存空间。

1. **没有将基类的析构函数定义为虚函数。当基类指针指向子类对象时，如果基类的析构函数不是virtual，那么子类的析构函数将不会被调用，子类的资源没有正确释放，因此造成内存泄露。**

## 2、如何检测内存泄漏：

在 [MFC](https://www.baidu.com/s?wd=MFC&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank) 中，可以使用 DEBUG\_NEW 宏代替 new 运算符来帮助定位内存泄漏。 如果不想重写整个程序来使用 DEBUG\_NEW代替new，则可以在源文件中定义下面的宏： [#define](https://www.baidu.com/s?wd=%23define&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank) new DEBUG\_NEW 当进行对象转储时，用 DEBUG\_NEW分配的每个对象均将显示被分配 到的文件和行号，可以查明内存泄漏源

# 二十六、内存溢出和内存泄漏区别

内存溢出：申请的内存空间不足

内存泄漏：申请内存空间，用完后没有回收，内存泄漏堆积后的后果就是内存溢出。

# 二十七、变量的定义和声明有什么区别？声明的关键字？

## 1、定义：

**需要建立存储空间**的。例如：int a 在声明的时候就已经建立了存储空间

## 声明：

**不需要建立存储空间**。例如：extern int a 其中变量a是在别的文件中定义的。

我们声明的最终目的是为了**提前使用**，即在定义之前使用，如果不需要提前使用就没有单独声明的必要，变量是如此，函数也是如此，所以声明不会分配存储空间，只有定义时才会分配存储空间。

## static关键字作用

(1)对于局部变量用static声明，则是为该变量分配的空间在整个程序的执行期内都始终存在。

(2)外部变量用static来声明，则该变量的作用只限于本文件模块。

# 二十八、类的关系

## 纵向

### 继承：一个类继承另外的一个类的功能，并可以增加它自己的新功能的能力（空心三角箭头 实线子类指向父类 ）

### 实现：一个类实现一个接口（空心三角箭头虚线，类指向接口）

## 横向：

### 依赖 ：一个类使用到另一个类。关系是临时的偶然的，但类B的变化会影响到A（虚线箭头），比如过河借船。A中的某个方法把B的对象作为参数

### 关联：两个类关系是长期的，双方平等的（实线箭头），比如我和朋友。 A中包含B的指针或引用

### 聚合：整体与部分，可以分离，有各自的生命周期，部分可以属于多个整体，也可以为多个整体共享（空心菱形加实线箭头），比如公司和员工。 A中包含B的指针或引用

### 组合：整体与部分，不可分，整体的生命周期结束，部分的也结束（实心菱形加实线箭头），比如人和人的大脑。A中包含B的成员变量

# 二十九、结构体和联合体

## 结构体

使不同的数据类型整合成一个整体--自定义的数据类型

结构体变量中，每个成员都占有自己的内存空间，它们是同时存在的。结构体变量所占内存长度是各成员的内存长度的总和

## 联合体

使几个不同类型的变量共占一段内存（相互覆盖）

联合体变量所占内存长度是各最长的成员占的内存长度

对联合体变量的不同成员赋值，将会对其他成员重写，原来的成员的值就不存在了

# 三十、数据库

## 事务

## 作为单个逻辑工作单元执行的一系列操作（事务就是被绑定在一起作为一个逻辑工作单元的SQL语句）

## **⑴ 原子性（Atomicity）**

　　原子性是指事务是一个不可再分割的工作单位，事务包含的所有操作要么全部成功，要么全部失败回滚

## **⑵ 一致性（Consistency）**

　　一致性是指事务必须使数据库从一个一致性状态变换到另一个一致性状态，也就是说一个事务执行之前和执行之后都必须处于一致性状态。

　　拿转账来说，假设用户A和用户B两者的钱加起来一共是5000，那么不管A和B之间如何转账，转几次账，事务结束后两个用户的钱相加起来应该还得是5000，这就是事务的一致性。

## **⑶ 隔离性（Isolation）**

　　隔离性是当多个用户并发访问数据库时，比如操作同一张表时，数据库为每一个用户开启的事务，不能被其他事务的操作所干扰，多个并发事务之间要相互隔离。

　　即要达到这么一种效果：对于任意两个并发的事务T1和T2，在事务T1看来，T2要么在T1开始之前就已经结束，要么在T1结束之后才开始，这样每个事务都感觉不到有其他事务在并发地执行。

## **⑷ 持久性（Durability）**

持久性是指一个事务一旦被提交了，那么对数据库中的数据的改变就是永久性的，即便是在数据库系统遇到故障的情况下也不会丢失提交事务的操作。

### 1，脏读

　　脏读是指在一个事务处理过程里读取了另一个未提交的事务中的数据。

### 2，不可重复读

　　不可重复读是指在对于数据库中的某个数据，一个事务范围内多次查询却返回了不同的数据值，这是由于在查询间隔，被另一个事务修改并提交了。

不可重复读和脏读的区别是，脏读是某一事务读取了另一个事务未提交的脏数据，而不可重复读则是读取了前一事务提交的数据。

### 3，虚读(幻读)

　　幻读是事务非独立执行时发生的一种现象。例如事务T1对一个表中所有的行的某个数据项做了从“1”修改为“2”的操作，这时事务T2又对这个表中插入了一行数据项，而这个数据项的数值还是为“1”并且提交给数据库。而操作事务T1的用户如果再查看刚刚修改的数据，会发现还有一行没有修改，其实这行是从事务T2中添加的，就好像产生幻觉一样，这就是发生了幻读。

　　幻读和不可重复读都是读取了另一条已经提交的事务（这点就脏读不同），所不同的是不可重复读查询的都是同一个数据项，而幻读针对的是一批数据整体（比如数据的个数）

## 主键

表通常具有包含唯一标识表中每一行的值的一列或一组列。这样的一列或多列称为表的主键 (PK)。

## 索引：

唯一、不为空、经常被查询的字段适合建为索引

对表中一列或者多列的值进行排序的一种结构，他可以快速访问数据库表的特定信息。

聚合索引和非聚合索引：

例如字典：拼音就是聚合索引，偏旁就是非聚合索引

每个表只能有一个聚集索引 ，因为目录只能按照一种方法进行排序。（主键）

### 什么情况不适合用索引

1.对于在查询过程中很少使用或参考的列，不应该创建索引。

2.对于那些只有很少数据值的列，不应该创建索引。

3.对于那些定义为image，text和bit数据类型的列，不应该创建索引。

4.当修改性能远大于检索性能，不应该建立索引。



索引类型包括：

单列索引(普通索引，唯一索引，主键索引)、组合索引、全文索引、空间索引

1、单列索引：一个索引只包含单个列

1.1普通索引：MySQL中基本索引类型，没有什么限制，允许在定义索引的列中插入重复值和空值，纯粹为了查询数据更快一点。

 1.2唯一索引：索引列中的值必须是唯一的，但是允许为空值，

1.3主键索引：是一种特殊的唯一索引，不允许有空值。（主键约束，就是一个主键索引）

1. 组合索引

在表中的多个字段组合上创建的索引，只有在查询条件中使用了这些字段的左边字段时，索引才会被使用，使用组合索引时遵循最左前缀集合。例如，这里由id、name和age3个字段构成的索引，索引行中就按id/name/age的顺序存放，索引可以索引下面字段组合(id，name，age)、(id，name)或者(id)。

1. 全文索引：

只有在MyISAM引擎上才能使用，只能在CHAR,VARCHAR,TEXT类型字段上使用全文索引，介绍了要求，说说什么是全文索引，就是在一堆文字中，通过其中的某个关键字等，就能找到该字段所属的记录行

1. 空间索引：只有在MyISAM引擎上才能使用，针对空间数据类型。

## 外键

如果一个公共关键字在一个关系中是主关键字，那么它在另一个关系中称为外关键字。外键表示了两个关系之间的相关联系。

## **主键和外键的区别？**

主键在本表中是唯一的、不可唯空的，外键可以重复可以唯空；外键和另一张表的主键关联，不能创建对应表中不存在的外键。

## 4、视图

是一个虚拟表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，视图并不在数据库中以存储的数据值集形式存在。行和列数据来自由定义视图的查询所引用的表，并且在引用视图时动态生成。这样程序员只能看到视图定义的数据，看不到视图引用表的数据

视图的特点：

1. 列可以来自不同的表
2. 视图是由基本表(实表)产生的表（虚表）
3. 视图的建立和删除不会影响基本表
4. 视图的更新直接影响基本表
5. 当视图来自不同的表时，不允许添加和删除数据

**从[数据库系统](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E5%BA%93%E7%B3%BB%E7%BB%9F" \t "https://baike.baidu.com/item/%E8%A7%86%E5%9B%BE/_blank)内部来看，视图是由一张或多张表中的数据组成的，从数据库系统外部来看，视图就如同一张表一样，对表能够进行的一般操作都可以应用于视图，例如查询，插入，修改，删除操作等**

## 5、连接查询（多表查询）

Left join...on(左连接)：输出左表的全部和右表满足条件的部分

Right join....on（右连接）：输出右表的全部和左表满足条件的 部分

Full join...on（全外连接）：左表全部+右表全部

Inner join...on（内连接）：左表满足条件的部分+右表满足条件的部分

Union all:记录联合

### **内连接：**

在表关系的笛卡儿积关系中，保留表关系中所有匹配的数据记录，舍弃不匹配的所有记录。分为自然连接，等值连接和不等值连接。

自然连接：根据表关系相同名称的字段进行记录匹配，然后去掉重复的字段。

等值连接：选择字段值相等的数据记录

不等值连接：选择字段值不等的数据记录

### **外连接：**

保留表关系匹配的数据记录，也会保留部分不匹配的记录，这部分记录来源包括左外连接，右外连接和全外连接。

左外连接：包含左边表的不匹配记录

右外连接：包含右边表的不匹配记录

全外连接：包含关联左右两边表的不匹配的记录

**Union 和union all:union去掉重复数据记录，union all没有；**

## 触发器

**由事件触发，当对一个表进行操作时，就激活它执行。保证数据完整性的一种方法。**

## 约束操作

**//表已建立时**

**去掉unique属性**

找到约束名id（show create table）

Alter table ...drop key id;

**去掉主键属性**

Alter table ...drop primary key;

**去掉非空属性**

Alter table ...modify 属性 数据类型null;/change 属性 属性 数据类型 null；

**去掉自增属性**

Alter table...modify 属性 数据类型；/change 属性 属性 数据类型；

**去掉外键属性**

找到约束名id

Alter table...drop foreign key id;

**//表已建立时**

**增加unique属性**

Alter table...modify 属性 数据类型 unique;(change)

**增加主键属性**

Alter table... modify 属性 数据类型 primary key;(change)

**增加非空属性**

Alter table...modify 属性 数据类型 not full;(change)

**增加自增属性（只能设置主键自增，且设置后，不管插入是否成功，主键都自增）**

Alter table...modify 属性 数据类型 primary key auto\_increment;

**增加外键属性**

Alter table... add foreign key(外键属性) references 主表(主键属性)；

**//表定义时**

**增加unique属性**

属性 数据类型 unqiue；/属性 数据类型，constraint 约束名 unique(属性)

**增加主键属性**

属性 数据类型 primary key;/属性 数据类型，constraint 约束名 primary key(属性)

**增加非空属性**

属性 数据类型 not full;

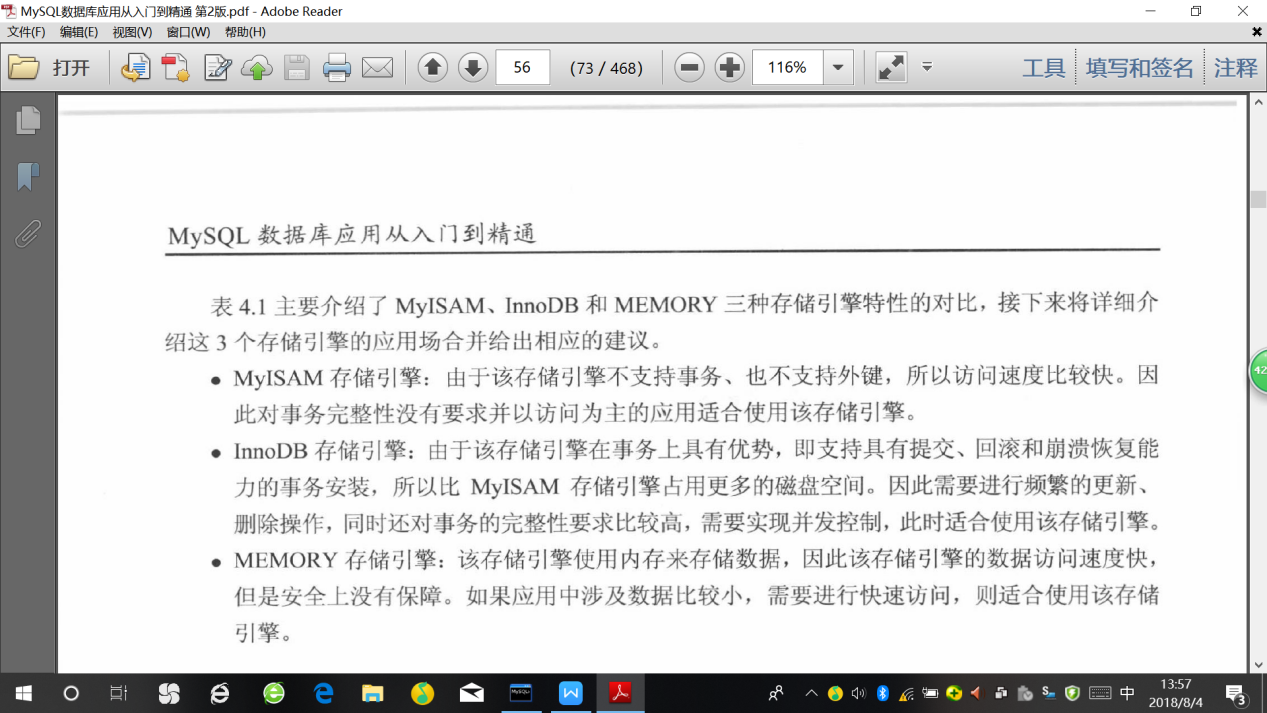
**增加自增属性**

属性 数据类型 primary key auto\_increment;

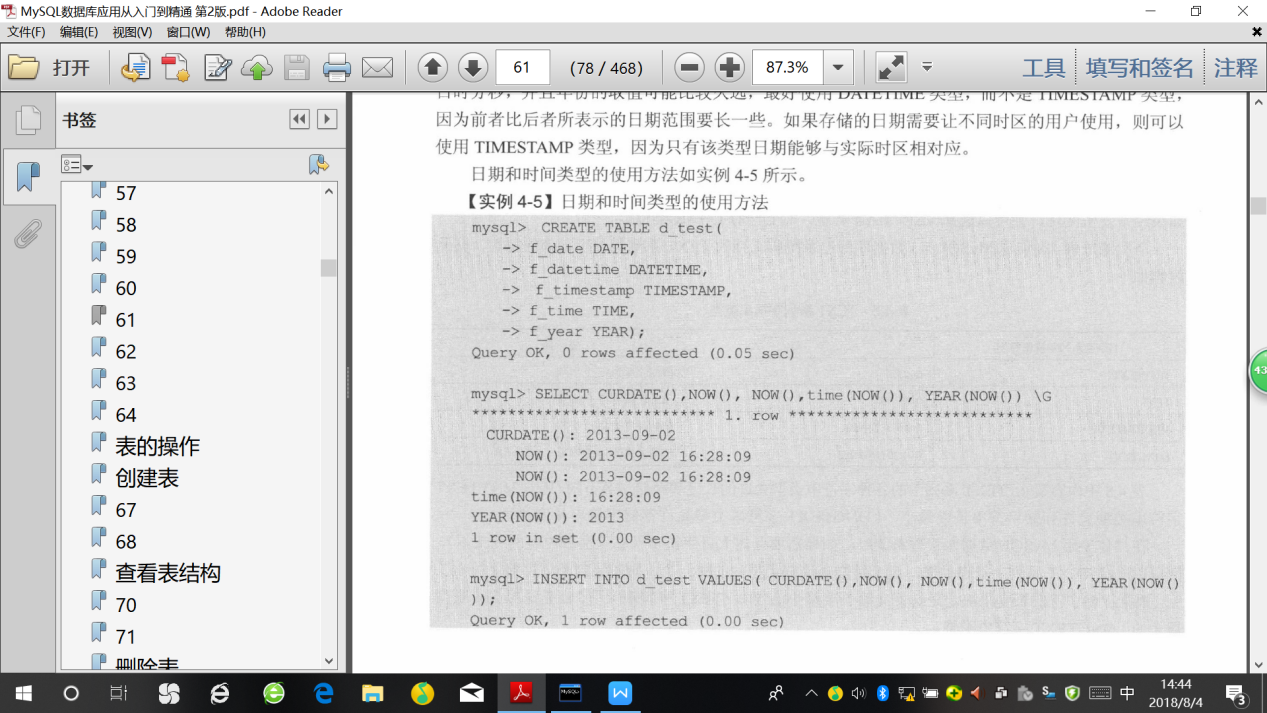
**增加外键属性**

属性 数据类型 constraint 约束名 foreign key（外键属性) references 主表（主键属性）

## 存储引擎



## 时间类型



## 数据库三范式

第一范式：每一列属性都是不可再分的属性值，确保每一列的原子性

第二范式：每一行的数据只能与其中一列相关，即一行数据只做一件事。只要数据列中出现数据重复，就要把表拆分开来。（针对联合主键）（主键约束）

第三范式：数据不能存在传递关系，即每个属性都跟主键有直接关系而不是间接关系。

## 存储过程

### 定义

存储过程是一个预编译的SQL语句，优点是允许模块化的设计，就是说只需创建一次，以后在该程序中就可以调用多次。如果某次操作需要执行多次SQL，使用存储过程比单纯SQL语句执行要快。

### 调用

1）可以用一个命令对象来调用存储过程。 2）可以供外部程序调用，比如：java程序。

### 优点

1）存储过程是预编译过的，执行效率高。 2）存储过程的代码直接存放于数据库中，通过存储过程名直接调用，减少网络通讯。 3）安全性高，执行存储过程需要有一定权限的用户。 4）存储过程可以重复使用，可减少数据库开发人员的工作量。 缺点：移植性差

## 12、从数据库中随机取50条

select \* from (select \* from t\_example order by dbms\_random.random) where rownum <= 50

## 13、truncate与 delete区别

TRUNCATE TABLE 在功能上与不带 WHERE 子句的 DELETE 语句相同：二者均删除表中的全部行。但 TRUNCATE TABLE 比 DELETE 速度快，且使用的[系统](https://www.2cto.com/os/" \t "https://www.2cto.com/database/201710/_blank)和事务日志资源少。 DELETE 语句每次删除一行，并在事务日志中为所删除的每行记录一项。

TRUNCATE TABLE 通过释放存储表数据所用的数据页来删除数据，并且只在事务日志中记录页的释放。 TRUNCATE,DELETE,DROP 放在一起比较：  
TRUNCATE TABLE ：删除内容、释放空间但不删除定义。  
DELETE TABLE: 删除内容不删除定义，不释放空间。  
DROP TABLE ：删除内容和定义，释放空间。

## 行转列、列转行

1. 使用decode函数 2）使用case when语句

## 游标

游标（用来存储多条查询数据的一种数据结构（结果集），它有一个指针，用来从上往下移动，从而达到遍历每条记录的作用，然后允许用户对指定位置的数据进行处理），逐条处理数据时，游标十分重要

## 锁

使某些数据的拥有者，在某段时间内甭使用某些数据或数据结构。

锁粒度是被封锁目标的大小,封锁粒度小则并发性高,但开销大,封锁粒度大则并发性低但开销小

SQL Server支持的锁粒度可以分为为行、页、键、键范围、索引、表或数据库获取锁

## ANY关键字查询

=ANY：功能与关键字IN一样

>ANY(>=ANY):比子查询记录中返回记录最小的还要大（大于等于）的数据记录

<ANY(<=ANY):比子查询记录中返回记录最大的还要小（小于等于）的数据记录

## 常用指令

### 删除重复数据

delete from table where data in(select data from (select data from table group by name having count(\*)>1)as table2)limit 1;

### 得到总分最高的学生名字

第一种：select name,sum(score) from student2 group by name order by sum(score) desc limit 1;

第二种：select name,sum(score)from student2 group by name having sum(score)>=all(select sum(score) from student2 group by name);

### 得到每个班总分最高的同学

（4）写sql语句，找出两门以上成绩不及格的学生的学号

select name from student2 where score<60 group by name having count(score)>=2;

（5）统计每个人的挂科数目

select name,sum(score < 60) from student2 group by name;

# 三十一、虚拟内存

## 虚拟[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)是[计算机系统](https://baike.baidu.com/item/%E8%AE%A1%E7%AE%97%E6%9C%BA%E7%B3%BB%E7%BB%9F/7210959" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)[内存管理](https://baike.baidu.com/item/%E5%86%85%E5%AD%98%E7%AE%A1%E7%90%86/5633616" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)的一种技术。它使得[应用程序](https://baike.baidu.com/item/%E5%BA%94%E7%94%A8%E7%A8%8B%E5%BA%8F/5985445" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)认为它拥有连续的可用的[内存](https://baike.baidu.com/item/%E5%86%85%E5%AD%98/103614" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)（一个连续完整的[地址空间](https://baike.baidu.com/item/%E5%9C%B0%E5%9D%80%E7%A9%BA%E9%97%B4/1423980" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)），而实际上，它通常是被分隔成多个[物理内存](https://baike.baidu.com/item/%E7%89%A9%E7%90%86%E5%86%85%E5%AD%98/2502263" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)碎片，还有部分暂时存储在外部[磁盘存储器](https://baike.baidu.com/item/%E7%A3%81%E7%9B%98%E5%AD%98%E5%82%A8%E5%99%A8/2386684" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)上，在需要时进行[数据交换](https://baike.baidu.com/item/%E6%95%B0%E6%8D%AE%E4%BA%A4%E6%8D%A2/1586256" \t "https://baike.baidu.com/item/%E8%99%9A%E6%8B%9F%E5%86%85%E5%AD%98/_blank)。

虚拟内存是Windows 为作为内存使用的一部分硬盘空间

## 虚拟内存技术的核心就是利用了局部性原理，把所要运行的进程中的数据不全部加载到内存中执行，而是加载一部分，当CPU在请求页表时，发现页表中的页表条目中的有效位为0但是被虚拟存储系统分配了的虚拟页时，就会把这个虚拟页从磁盘中调度到内存中（往往磁盘中的数据不常用，而在内存中的物理页的数据是频繁使用的数据），这样一来，我们就实现了多个进程同时加载到内存中并且还占用不是很多的内存的效果了

# 三十二、const的作用

让实参变量拥有只读属性，防止无意的代码修改，能提高程序的健壮性

## 用const修饰函数的参数：

### const只能修饰输入参数

如果输入参数采用“指针传递”，那么加const修饰可以防止意外地改动该指针，起到保护作用；

如果输入参数采用“值传递”，由于函数将自动产生临时变量用于复制该参数，该输入参数本来就无需保护，所以不要加const修饰。

对于非内部数据类型的输入参数，应该将“值传递”的方式改为“const引用传递”，目的是提高效率。例如将voidFunc(A a) 改为voidFunc(const A &a)。

对于内部数据类型的输入参数，不要将“值传递”的方式改为“const引用传递”。否则既达不到提高效率的目的，又降低了函数的可理解性。例如voidFunc(int x) 不应该改为voidFunc(const int &x)。

## 用const修饰函数参数的返回值

如果给以“指针传递”方式的函数返回值加const修饰，那么函数返回值（即指针）的内容不能被修改，该返回值只能被赋给加const修饰的同类型指针

如果函数返回值是值传递，如 int fun()，加const就没有意义。

## const成员函数

const关键字只能放在函数声明的尾部

任何不会修改数据成员的函数都应该声明为const类型，函数中数据成员都不能修改（想要修改就给数据成员加上mutable关键字），也不能调用非const函数

# 三十三、静态成员函数 -静态成员函数是类中的特殊的成员函数 -静态成员函数没有隐藏的this指针 -静态成员函数可以通过类名直接访问 -静态成员函数可以通过对象访问 -静态成员函数只能直接访问静态成员变量（函数），而**不能直接访问普通成员变量**（函数），**可以通过对象访问**

# 三十四、宏定义

## 1、内联函数和宏定义

1. 内联函数在运行时可调试，而宏定义不可以;
2. 编译器会对内联函数的参数类型做安全检查或自动类型转换（同普通函数），而宏定义则不会；
3. 内联函数可以访问类的成员变量，宏定义则不能；
4. 在类中声明同时定义的成员函数，自动转化为内联函数。

## const 和宏定义

（1）define宏是在预处理阶段展开。

　　const常量是编译运行阶段使用。

（2）类型和安全检查不同

　　define宏没有类型，不做任何类型检查，仅仅是展开。

　　const常量有具体的类型，在编译阶段会执行类型检查。

（3）const可以调试，宏定义不能调试

# 三十五、条件编译作用

#ifndef #define 宏就是用来避免头文件重复包含的，条件编译是为了代码能在不同的环境中都编译通过，通用性更好

# 三十六、ASSERT()

1、预处理宏， assert(expr), 先计算表达式expr, 如果为假，那么它会输出信息并终止程序执行。如果用if else实现同样功能的话，就会从函数开始括到函数尾。  
2、assert是宏不是函数，定义在cassert头文件中。  
3、使用assert的缺点是，频繁的调用会极大的影响程序的性能，增加额外的开销；  
4、完成调试后，不必从源代码中删除assert()语句，因为宏NDEBUG有定义时，宏assert()的定义为空，即可以通过在包含#include assert.h>或#include< csaaert >的语句之前插入 #define NDEBUG 来禁用assert调用

5、assert只有在Debug版本中才有效，如果编译为Release版本则被忽略（程序一般分为Debug 版本和Release 版本）；

6、assert用法注意：  
A：不要一起判断多个条件，否则不知道是哪个有问题。  
B：不能改变变量的值、  
C： assert语句后面空一行

# 三十七、include“”和include<>

#include<>直接从编译器自带的函数库中寻找文件  
#include""是先从自定义的文件中找 ，如果找不到在从函数库中寻找文件

# 三十八、编译

## 源程序是如何变成可执行文件？

### 预编译：处理源代码中的伪指令和一些特殊字符，并对一些相关的代码进行替换。比如宏定义，条件编译，头文件，特殊字符。

### 编译：检查语法并对代码进行优化，将文本文件 .i 翻译成 .s 文件，得到汇编语言程序。

### 汇编：将 .s 文件转换成机器语言指令也就是二进制代码，并将结果保存在目标文件 .o 中。

### 链接：将所有的目标文件链接到一起形成可执行文件，分为动态链接和静态链接。

# 三十九：链接具体过程

## 链接程序的主要工作就是将有关的目标文件彼此相连接，也即将在一个文件中引用的符号同该符号在另外一个文件中的定义连接起来，使得所有的这些目标文件成为一个能够按操作系统装入执行的统一整体。

## 1）静态链接 在这种链接方式下，函数的代码将从其所在地**静态链接库中被拷贝到最终的可执行程序中**。这样该程序在被执行时这些代码将被装入到该进程的虚拟地址空间中。静态链接库实际上是一个目标文件的集合，其中的每个文件含有库中的一个或者一组相关函数的代码。 2） 动态链接 在此种方式下，**函数的代码被放到称作是动态链接库或共享对象的某个目标文件中**。链接程序此时所作的只是在最终的可执行程序中记录下共享对象的名字以及其它少量的登记信息。在此可执行文件被执行时，动态链接库的全部内容将被映射到运行时相应进程的虚地址空间。动态链接程序将根据可执行程序中记录的信息找到相应的函数代码。

# 四十、release 和debug的区别

Debug通常称为调试版本，它包含调试信息，并且不做任何优化，便于程序员调试程序。

Release称为发布版本，它往往是进行了各种优化，使得程序在代码大小和运行速度上都是最优的，以便用户很好的使用。不同在于一组编译选项

Debug和release在初始化变量时所做的操作是不同的，debug是将每个字节维都赋成0xcc，而release赋值近似于随机。

# 四十一、fopen文本模式和二进制模式的区别

1. 使用二进制方式进行读文件时，会原封不动的读出全部的内容，写文件的时候，会把内存缓冲区的内容原封不动的写到文件中。
2. 使用文本方式进行读文件时，会将回车换行符号CRLF(0x0D OxOA)全部转换成单个的换行符号LF(OxOA)，写文件的时候，会将换行符号LF( OxOA)全部转换成回车换行符号CRLF(0x0D OxOA)。

# 四十二、指针优势和缺点

## 优点：

（1）可以提高程序的编译效率和执行速度，使程序更加简洁。  
（2）通过指针被调用函数可以向调用函数处返回除正常的返回值之外的其他数据，从而实现两者间的双向通信。  
（3）利用指针可以实现动态内存分配。  
（4）指针还用于表示和实现各种复杂的数据结构，从而为编写出更加高质量的程序奠定基础。

（5）利用指针可以直接操纵内存地址，从而可以完成和汇编语言类似的工作。

（6）更容易实现函数的编写和调用。

## 缺点：

指针编写的程序也更容易隐含各式各样的错误，同时程序的可读性也会大打折扣。

# 四十三、队列和栈

## 相同点

1.都是线性结构。

2.插入操作都是限定在表尾进行。

3.都可以通过顺序结构和链式结构实现。

4.插入与删除的时间复杂度都是O（1），在空间复杂度上两者也一样。

5.多链栈和多链队列的管理模式可以相同。

## 不同点

1.删除数据元素的位置不同，栈的删除操作在表尾进行，队列的删除操作在表头进行。

2.应用场景不同；常见栈的应用场景包括括号问题的求解，表达式的转换和求值，函数调用和递归实现，深度优先搜索遍历等；常见的队列的应用场景包括计算机系统中各种资源的管理，消息缓冲器的管理和广度优先搜索遍历等。

3.顺序栈能够实现多栈空间共享，而顺序队列不能。

# 四十四、字节对齐以及为什么要字节对齐

各个硬件平台对存储空间的处理上有很大的不同。**一些平台对某些特定类型的数据只能从某些特定地址开始存取**。比如有些架构的CPU在访问一个没有进行对齐的变量的时候会发生错误,那么在这种架构下编程必须保证字节对齐.其他平台可能没有这种情况，但是最常见的是**如果不按照适合其平台要求对数据存放进行对齐，会在存取效率上带来损失**。比如有些平台每次读都是从偶地址开始，如果一个int型（假设为32位系统）如果存放在偶地址开始的地方，那么一个读周期就可以读出这32bit，而如果存放在奇地址开始的地方，就需要2个读周期，并对两次读出的结果的高低字节进行拼凑才能得到该32bit数据。显然在读取效率上下降很多。

1. 便于cpu快速访问
2. 合理的利用字节对齐可以有效地节省存储空间

# 四十五、C++类初始化

1. 类里面的任何成员变量在定义时是不能初始化的。  
   2、一般的数据成员可以在构造函数中初始化。（构造初始化列表初始化和构造函数体内赋值初始化）  
   3、const数据成员必须在构造函数的初始化列表中初始化。（道理很简单，const成员第一次数据初始化就是发生在类构造数据初始化时，一旦过了此时就会出现问题）。  
   4、static在类的定义外面初始化。（int Test::cCount = 0;）   
   5、数组成员是不能在初始化列表里初始化的。  
   6、不能给数组指定显式的初始化。

# 四十六、C和C++的区别

1、C语句是面向结构的语言，[C++](https://www.baidu.com/s?wd=C++&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)是[面向对象](https://www.baidu.com/s?wd=%E9%9D%A2%E5%90%91%E5%AF%B9%E8%B1%A1&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)的语言

2、C和C++动态管理内存的方法不一样，C是使用malloc/free函数，而C++除此之外还有new/delete关键字

3、C++支持函数重载，而C不支持函数重载

4、C++中有引用，而C没有

5、C++特有输入输出流

# 四十七、编译语言和解释语言

编译型语言在**程序执行之前**，有一个**单独的编译**过程，将程序翻译成机器语言就不用再进行翻译了。

解释型语言，是在**运行的时候将程序翻译成机器语言**，所以运行速度相对于编C/C++ 等都是编译型语言，而Java，C#等都是解释型语言。

虽然Java程序在运行之前也有一个编译过程，但是并不是将程序编译成机器语言，而是将它编译成字节码（可以理解为一个中间语言）。  
在运行的时候，由JVM将字节码再翻译成机器语言。  
注：脚本语言一般都有相应的脚本引擎来解释执行。 他们一般需要解释器才能运行。JAVASCRIPT,ASP,PHP,PERL,Nuva都是脚本语言。C/C++编译、链接后，可形成独立执行的exe文件。

### 编译型语言：

1. **编译型语言最大的优势之一就是其执行速度**。用C/C++编写的程序运行速度要比用Java编写的相同程序快30%-70%。
2. 编译型程序比解释型程序消耗的内存更少。
3. 不利的一面——编译器比解释器要难写得多。
4. 编译器在调试程序时提供不了多少帮助——有多少次在你的C语言代码中遇到一个“空指针异常”时，需要花费好几个小时来明确错误到底在代码中的什么位置。
5. 可执行的编译型代码要比相同的解释型代码大许多。例如，C/C++的.exe文件要比同样功能的Java的.class文件大很多。
6. 编译型程序是面向特定平台的因而是**平台依赖**的。
7. 编译型程序不支持代码中实现安全性——例如，一个编译型的程序可以访问内存的任何区域，并且可以对你的PC做它想做的任何事情（大部分病毒是使用编译型语言编写的）
8. 由于松散的安全性和平台依赖性，编译型语言不太适合开发因特网或者基于Web的应用。

### 解释型语言：

1. **解释型语言提供了极佳的调试支持**。一名Java程序员只需要几分钟就可以定位并修复一个“空指针异常”，因为Java运行环境不仅指明了异常的性质，而且给出了异常发生位置具体的行号和函数调用顺序（著名的堆栈跟踪信息）。这样的便利是编译型语言所无法提供的。
2. 另一个优势是解释器比编译器容易实现
3. 解释型语言最大的优势之一是其**平台独立性**
4. 解释型语言也可以保证**高度的安全性**——这是互联网应用迫切需要的
5. 中间语言代码的大小比编译型可执行代码小很多
6. 平台独立性，以及严密的安全性是使解释型语言成为适合互联网和Web应用的理想语言的2个最重要的因素。
7. 解释型语言存在一些严重的缺点。解释型应用占**用更多的内存和CPU资源**。这是由于，为了运行解释型语言编写的程序，相关的解释器必须首先运行。解释器是复杂的，智能的，大量消耗资源的程序并且它们会占用很多CPU周期和内存。
8. 由于解释型应用的decode-fetch-execute（解码-抓取-执行）的周期，它们比编译型程序慢很多。
9. 解释器也会做很多代码优化，运行时安全性检查；这些额外的步骤占用了更多的资源并进一步降低了应用的运行速度。

# 四十八、结构体（struct）和类（class）

1. 默认的继承访问权限不同：struct是public的，class是private的

2、成员的默认访问权限。class的成员默认是private权限，struct默认是public权限。

### 3、结构体是一种值类型，而类是引用类型。值类型用于存储数据的值，引用类型用于存储对实际数据的引用。那么结构体就是当成值来使用的，类则通过引用来对实际数据操作。

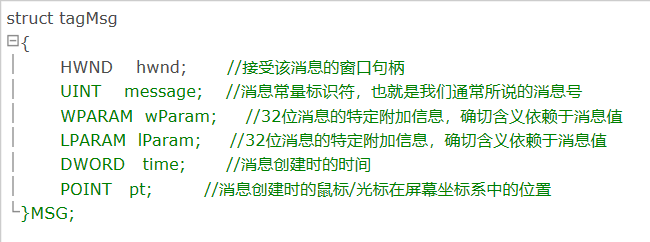
# 四十九、左值和右值

左值指的是既能够出现在等号左边也能出现在等号右边的变量(或表达式)，右值指的则是只能出现在等号右边的变量(或表达式).

# 五十、window消息机制

## 定义：

消息就是一个结构体



wParam

通常是一个与消息有关的常量值，也可能是窗口或控件的句柄。

lParam

通常是一个指向内存中数据的[指针](https://baike.baidu.com/item/%E6%8C%87%E9%92%88" \t "https://baike.baidu.com/item/windows%E6%B6%88%E6%81%AF%E5%A4%84%E7%90%86%E6%9C%BA%E5%88%B6/_blank)。

## MFC消息分类：

窗口消息、命令消息和控件通知消息。（ON\_MESSAGE ，ON\_COMMAND, , ON\_NOTIFY）

(1)窗口消息:WM\_XXX，除WM\_COMMAND之外，所有以WM\_开头的消息

      窗口消息(Window Message)一般与窗口的内部运作有关，如：创建窗口、绘制窗口和销毁窗口等。通常，消息是从系统发送到窗口，或从窗口发送到窗口。

(2)命令消息:WM\_COMMAND

      命令消息一般与处理用户请求相关，当用户单击一个菜单项或工具栏时，命令消息产生，并被发送到能处理该请求的类对象(如：装载文件、编辑文本和保存选项等)。

(3)控件通知:有多种格式  
      通常，控件通知在某些重要事件发生时，由控件窗口发送到父窗口，如打开一个组合框。控件通知为父窗口进一步控制子窗口提供了机会。例如，打开一个组合框时，父窗口可以用组合框初建时得不到的消息填充它。

把一个消息发送到窗口有3种方式：发送（send）、寄送（post）和广播（board）。

## Windows消息运行原理：

**运行程序->事件操作引发消息->消息先存在系统消息队列->再存入到应用程序消息队列->用消息循环提取消息->处理消息->再返回消息队列**

**或：**

## 一个消息从产生到被一个窗口响应，其中有5个步骤：

1) 系统中发生了某个事件。

2) Windows把这个事件翻译为消息，然后把它放到[消息队列](https://baike.baidu.com/item/%E6%B6%88%E6%81%AF%E9%98%9F%E5%88%97" \t "https://baike.baidu.com/item/windows%E6%B6%88%E6%81%AF%E5%A4%84%E7%90%86%E6%9C%BA%E5%88%B6/_blank)中。

3)应用程序从消息队列中接收到这个消息，把它存放在TMsg记录中。

4)应用程序把消息传递给一个适当的窗口的窗口过程。

5) 窗口过程响应这个消息并进行处理。

项目：首先自定义一个消息：WM\_USER+X(数字)，发送消息用的是postmessage,找到那个的窗口句柄所在的线程，把消息放到该线程的消息队列里。定义一个宏，作为消息号，用on\_message把宏和要调用的函数绑定在一起，设函数的时候有两个参数，wparam和lparam,根据两个参数的不同调用不同分支。

例：::PostMessage(pScopeView->GetSafeHwnd(), MSG\_SINGLEN\_BTNRUN, 0, 0);

# 五十一、vc6.0线程之间的通信

## 临界区变量

## 事件变量

初始化：createevent

控件事件 触发事件：setevent 每次触发后，必有一个或多个处于等待状态下的线程变成可调度状态

线程一开始 handle x=WaitForSingleObject

## 消息传递的机制

# 五十二、PostMessage和SendMessage的区别

## 1、     PostMessage 是异步的，SendMessage 是同步的。

          PostMessage 只把消息放到队列，不管消息是不是被处理就返回，消息可能不被处理；

          SendMessage等待消息被处理完了才返回，如果消息不被处理，发送消息的线程将一直处于阻塞状态，等待消息的返回。

## 2、同一个线程内：

             SendMessage 发送消息时，由USER32.DLL模块调用目标窗口的消息处理程序，并将结果返回，SendMessage 在同一个线程里面发送消息不进入线程消息队列；PostMessage 发送的消息要先放到消息队列，然后通过消息循环分派到目标窗口（DispatchMessage）。

## 3 、不同线程：

             SendMessage 发送消息到目标窗口的消息队列，然后发送消息的线程在USER32。DLL模块内监视和等待消息的处理结果，直到目标窗口的才处理返回，SendMessage在返回之前还需要做许多工作，如响应别的线程向它发送的SendMessage().PostMessge() 到别的线程的时候最好使用PostThreadMessage   代替。PostMessage()的HWND 参数可以为NULL，相当于PostThreadMessage() + GetCrrentThreadId.

## 4 、系统处理消息

             系统只处理（marshal)系统消息（0--WM\_USER)，发送用户消息（用户自己定义）时需要用户自己处理。

             使用PostMessage,SendNotifyMessage,SendMessageCallback等异步函数发送系统消息时，参数不可以使用指针，因为发送者不等待消息的处理就返回，接收者还没有处理，指针就有可能被释放了，或则内容变化了。

在控制别的应用程序的时候，经常需要等待直到某个功能结束，例如:

 打开一个窗口-->等待直到窗口结束 ，这个时候就可以用到SendMessage

 如果在打开这个窗口后仍然需要对该窗口的界面进行设置，比如Edit的value等等,比如：

  打开一个窗口-->控制窗口的control的属性

 这个时候就需要PostMessage

# 五十三、 MFC的消息机制 MFC使用一种消息映射机制来处理消息，一个消息与消息处理函数一一对应的消息映射表（BEGIN\_MESSAGE\_MAP() ），以及消息处理函数的声明和实现代码。当窗口接收到消息的时候，会到消息映射表中查找消息的处理函数，然后消息处理函数进行处理

# 五十四、Windows消息映射

Windows程序都维护有自己的消息队列，保持队列消息(当然也有非队列消息，他们直接发给窗口)，并用消息循环  
对消息进行处理。

消息循环首先通过GetMessage取得消息并从队列中移除，对于加速键，会调用TranslateAccelerator函数，对其进行翻译和处理，如果处理成功就不在调用translateMessge. 否则进行消息的转换和派发。让目的窗口的窗口过程来处理消息。  
while (GetMessage(&msg, NULL, 0, 0))   
{   
if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))   
{   
TranslateMessage(&msg);   
DispatchMessage(&msg);   
}   
真正处理消息的是所谓的窗口过程（LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)），这个函数的参数记录了过程对应的窗口、消息的ID以及参数，在其内部开发者可以实现。

# 五十五、AfxBeginThread创建线程

m\_AquisitionThread = AfxBeginThread(AquisitionThread（函数）, this, 0, 0, NULL);

1、AfxBeginThread函数调用的函数要么是全局函数要么是类内的静态函数；

2、可以将类指针作为参数传入调用函数，通过此指针就可以访问类中的成员变量和成员函数，否则被访问的成员变量和成员函数也必须是静态的。

# 五十六、Windows消息范围及意义

#define WM\_USER 0x0400   
#define WM\_APP 0x8000

0到WM\_USER-1   
Messages reserved for use by the system   
系统预留使用的消息

WM\_USER到0x7FFF   
Integer messages for use by private window classes   
被私有窗口类使用的消息

WM\_APP到0xBFFF   
Messages available for use by applications   
被应用程序使用的消息

0xC000到0xFFFF   
String messages for use by applications   
被应用程序使用的字符串消息

大于0xFFFF   
Reserved by the system   
系统预留

# 五十七、系统调用和库函数调用

## 系统调用

实际上是指底层的一个调用，就是内核提供的、功能十分强大的一系列的函数。这些系统调用是在内核中实现的。是操作系统为用户态运行的进程和硬件设备（如CPU、磁盘、打印机等）进行交互提供的一组接口，即就是设置在应用程序和硬件设备之间的一个接口层。可以说是操作系统留给用户程序的一个接口。

## 库函数

顾名思义是把函数放到库里，是把一些常用到的函数编完放到一个文件里，供别人用。别人用的时候把所在的文件名用#include<>加到里面就可以了，一般放到lib文件里。库函数一般分为两类：一种是C语言标准规定的库函数，一类是编译器特定的库函数。libc就是一个C标准库，里面放着一些基本的函数，这些函数都被标准化了。

## 联系与区别：

### 1、事实上，系统调用所提供给用户的是直接而纯碎的高级服务，如果想要更加人性化，具有更符合特定情况的功能，那么就要我们用户自己定义，因此衍生了库函数，它把部分系统调用包装起来。比如当我们要用C语言打印一句话的时候，如果没有用到库函数printf，那么我们就需要自己实现就需要调用putc()和write()等这样一些系统函数。显得比较麻烦，所以系统调用是为了方便使用操作系统的接口，而库函数则是为了人们编程的方便。

### 2、库函数的调用是语言或者应用程序的一部分，而系统调用则是操作系统的一部分。

### 3、系统调用是**应用程序与内核交互的接口**。人们在长期的编程中发现使用系统函数有个重大的缺点，那就是程序的移植性。但是大部分的系统调用函数的速度比库函数调用的速度要快。

****系统调用****   
1.使用INT和IRET指令，内核和应用程序使用的是不同的堆栈，因此存在堆栈的切换，从用户态切换到内核态，从而可以使用特权指令操控设备   
2.依赖于内核，不保证移植性   
3.在用户空间和内核上下文环境间切换，开销较大   
4. 是操作系统的一个入口点   
****函数调用****   
1.使用CALL和RET指令，调用时没有堆栈切换   
2.平台移植性好   
3.属于过程调用，调用开销较小   
4.一个普通功能函数的调用

## API：

### 1、用户级API

替用户写好函数，方便调用，使程序员可以将焦点放在设计程序逻辑上面，而不必再编写繁琐、重复的程序，不必关注技术的细节。例如VC++，MFC，VB等都是类库和各种控件，它代替了API的神秘功能。

### 2、系统级API

如果**应用程序想要和硬件交互**，必须调用此类函数。程序员调用的是API（api函数），然后通过与系统调用共同完成函数的功能。因此，API是一个提供给应用程序的接口，一组函数，是与程序猿直接进行交互的。系统调用则不与程序员进行交互，它是根据API函数，通过一个软件中断机制向内核提交请求（trap指令类似于一个系统中断，系统也会有一个特殊的中断处理函数（interrupt handler）来处理用户的请求），以获得内核服务的接口。并不是所有的API函数都一一对应一个系统的调用，有时，一个API函数会需要几个系统调用共同来完成函数的功能，甚至一些API函数不需要调用相应的系统调用，不需要内核提供的服务。

## 回调函数

回调函数就是一个通过函数指针调用的函数，使用回调函数实际上就是在调用某个函数（通常是API函数）时，将自己的一个函数（这个函数为回调函数）的地址作为参数传递给那个函数。也就是把需要执行的代码地址给系统,系统在恰当的时候执行。这时你可以利用这个机会在回调函数中处理消息或完成一定的操作。类似于中断处理机制，中断在某方面使计算机变得智能！

系统回调函数必须是由系统调用，但是这个函数可以是用户编写的，完成相应的功能服务

# 五十八、内核对象

内核对象就是类似符号对象、事件对象、文件对象、文件映射对象、   
I / O完成端口对象、作业对象、信箱对象、互斥对象、管道对象、进程对象，信标对象、线程对象和等待计时器对象等

1、内核对象的数据结构只能被内核访问，因此应用程序无法在内存中找到这些数据结构并直接改变它们的内容

2、当调用一个用于创建内核对象的函数时，该函数就返回一个用于标识该对象的**[句柄](https://baike.baidu.com/item/%E5%8F%A5%E6%9F%84" \t "https://baike.baidu.com/item/%E5%86%85%E6%A0%B8%E5%AF%B9%E8%B1%A1/_blank)**，如果将该句柄值传递给另一个进程中的一个线程，那么这另一个进程使用你的进程的句柄值所作的调用就会失败。如果想在多个进程中共享内核对象，要通过一定的机制。

# 五十九、用户态和内核态

## ****内核态:****

CPU可以访问内存所有数据, 包括外围设备, 例如硬盘, 网卡. CPU也可以将自己从一个程序切换到另一个程序

## ****用户态:****

只能受限的访问内存, 且不允许访问外围设备. 占用CPU的能力被剥夺, CPU资源可以被其他程序获取

## 为什么有用户态和内核态：

需要限制不同的程序之间的访问能力, 防止他们获取别的程序的内存数据, 或者获取外围设备的数据, 并发送到网络

## 用户态与内核态的切换

所有用户程序都是运行在用户态的, 但是有时候程序确实需要做一些内核态的事情, 例如从硬盘读取数据, 或者从键盘获取输入等. 而唯一可以做这些事情的就是操作系统, 所以此时程序就需要先操作系统请求以程序的名义来执行这些操作.这时需要一个这样的机制: **用户态程序切换到内核态**, 但是不能控制在内核态中执行的指令。这种机制叫**系统调用**, 在CPU中的实现称之为陷阱指令(Trap Instruction)

他们的工作流程如下:

1、用户态程序将一些数据值放在寄存器中, 或者使用参数创建一个堆栈(stack frame), 以此表明需要操作系统提供的服务.

2、用户态程序执行陷阱指令

3、CPU切换到内核态, 并跳到位于内存指定位置的指令, 这些指令是操作系统的一部分, 他们具有内存保护, 不可被用户态程序访问

4、这些指令称之为陷阱(trap)或者系统调用处理器(system call handler). 他们会读取程序放入内存的数据参数, 并执行程序请求的服务

5、系统调用完成后, 操作系统会重置CPU为用户态并返回系统调用的结果

## EnterCriticalSection会不会进入内核态

windows实际上先让线程进入循环锁，避免转内核态，一定时间后若得不到资源则进入内核态

# 六十、测试部分

## 1、为什么选择测试岗位

答：很多人认为开发比测试的技术要求要更高，但是我觉得其实测试也需要有牢固的技术基础，而且这两个岗位本身的关注点就不同，前者在于实现功能，后者是一个质量保证，本来就没有可比性。然后呢，我觉得测试的魅力在于它是发现和分析的一个过程，发散性的思考问题。首先我需要发现问题，然后通过各种测试方法的组合去得到问题的根源，我认为这个过程很有趣也很锻炼人。而且在研究生期间也写了一些代码，写的代码越多越感觉到测试的重要性。因此我想投测试岗。

## 2、项目中用到的测试方法

答、我们项目的测试主要有四个部分，对话框，按键，界面效果还有功能指标。

首先我们是基于MFC做的开发，需要开发大量对话框，对话框的显示隐藏，各个对话框间的切换相当多，同时还有按键面板的集成，这个部分我们还是通过大量的傻瓜式的点击保证对话框显示的正确性和按键的效果。

其次，我们的项目主要是做能够实现生产的工业仪器， 主要做的就是界面和功能指标的测试，我们有产品规范要求达到某些数据的精度，用到比较多的就是黑盒测试。比如采取边界值方法，比如说我要测频率在某个范围的功率精度，那么我就可以输入一个最小值，一个略大于最小值，一个正常值，一个略小于最大值，一个最大值，测试每个输入条件下的功率精度是否符合要求。还有就是用等价类法，例如测触发功能，触发是要把电平范围设为90%的幅度范围内才能正常触发，我就会选择信号幅度最大值作为无效等价类，一些合理的值作为有效等价类，观察是否实现触发功能。

## 3、测试岗位职业规划

[1～2年]，测试技能：熟悉整个测试过程及产品业务领域，学习和掌握自动测试工具，学习测试自动化编程技术；开发和执行测试脚本，承担系统测试实施任务；学习编程语言、操作系统、网络与数据库方面的技能。

[3～4年]，测试过程：深入了解测试过程，掌握测试过程设计及改进，参与软件工作产品的同行评审；进一步了解产品业务领域，改进测试自动化编程技术，能指导初级测试工程师；加强编程语言、操作系统、网络与数据库方面的技能。

[4～5年]，测试组织工作：管理1～3名测试工程师，担任任务估算、管理及进度控制；进一步培养在软件项目管理及支持工具方面的技能。

## 4、黑盒测试

黑盒测试也称[功能测试](https://baike.so.com/doc/6972420-7195106.html" \t "_blank)，它是通过测试来检测每个功能是否都能正常使用。在测试中，把程序看作一个不能打开的黑盒子，在完全不考虑程序内部结构和内部特性的情况下，在[程序接口](https://baike.so.com/doc/7844608-8118703.html" \t "_blank)进行测试，它只检查程序功能是否按照需求规格说明书的规定正常使用，程序是否能适当地接收输入数据而产生正确的输出信息

主要试图发现：功能不正确或遗漏，界面错误，输入和输出错误，数据库访问错误，性能错误;[初始化](https://baike.so.com/doc/6380882-6594531.html" \t "_blank)和[终止](https://baike.so.com/doc/6019883-6232879.html" \t "_blank)错误等。

方法：

1. 边界值测试
2. 等价类测试：将程序输入域分为若干部分，然后从每个部分中选取少数代表性数据作为测试用例。

有效等价类：是相对于规格说明合理的，正确的有意义的输入数据的集合

无效等价类：是不合理的，错误的，无意义的…

1. 基于决策表的测试：将问题的各种情况全部列举出来，简明并避免遗漏。

条件桩：列出问题的所有条件

动作桩：列出问题的规定可能采取的操作

条件项：列出条件桩各种取值下应该采取的动作

动作项：列出了条件项的各种取值情况下应该采取的动作

1. 因果图：图解表示各种输入的组合

1、输入状态之间可能有某些依赖关系，称为约束

E约束：a和b中最多有一个可能为1，即a和b不能同时为1。  
2、I约束：a、b和c中至少有一个必须是1，即abc不能同时为0。  
3、O约束：a和b必须有一个且仅有一个为1。  
4、R约束;a是1时，b必须是1，即当a是1时，b不能是0。

2、输出条件的约束只有M约束：若结果a是1,则结果b强制为0。

## 5、白盒测试

盒子指的是被测试的软件，白盒指的是盒子是可视的，你清楚盒子内部的东西以及里面是如何运作的。白盒要考虑测试用例对程序内部的覆盖程度，最好的白盒测试是能覆盖到每一条路径包括语句覆盖。判定覆盖。条件覆盖。判定/条件覆盖。条件组合覆盖。

语句覆盖：每一条可执行语句都要执行到

判定覆盖：每个判断的取真和取假分支都要执行到

条件覆盖：判断语句中每个条件的可能都要取值一次，每个条件都要有一次真值和一次假值，不管结果走哪个分支

判定/条件覆盖：条件的可能都要取到，同时取真和取假分支也要都执行到

条件组合覆盖：符合判定条件的所有组合情况都要取到一次

## 测试的五个阶段

单元测试，集成测试，确认测试，系统测试，验收测试

单元测试：模块内部的程序错误

集成测试：模块间的组装和调用关系

确认测试：验证性能是否与用户需求一致

系统测试：模拟环境，对计算机系统进行测试（压力测试（也称为强度测试），容量测试， 负载测试，性能测试，安全测试，容错测试）

验收测试：以用户为主的测试，软件开发人员和质量保证人员参加，由用户设计测试用例

## 软件性能

      定义：软件的性能是软件的一种非功能特性，它关注的不是软件是否能够完成特定的功能，而是在完成该功能时展示出来的及时性。

　　由定义可知性能关注的是软件的非功能特性，所以一般来说性能测试介入的时机是在功能测试完成之后。另外，由定义中的及时性可知性能也是一种指标，可以用时间或其它指标来衡量，通常我们会使用某些工具或手段来检测软件的某些指标是否达到了要求，这就是性能测试。

　　性能测试定义：指通过自动化的测试工具模拟多种正常、峰值以及异常负载条件来对系统的各项性能指标进行测试。

## 对测试的理解

首先测试的目的就是用各种技术手段找到软件中潜在的各种错误和缺陷。这种缺陷可能是功能的错误，也可能是性能低下，易用性差。那我觉得测试是站在一个更高的维度去看软件，从整体上去把握一个软件，需要发散性的思维和过硬的测试技术，然后就是如何去完成一个测试过程，首先是要分析需求，可以看一些产品说明书或者文档，然后就是安排测试计划，用多少人力或者是资源之类，然后就是写测试用例，肯定是越详细，覆盖面越大越好，然后就是执行测试计划，后面就是测试记录啊测试报告啊，最后就是用户测试。

## 一个测试人员的素质

1. 逻辑思维能力，要有发散性的思维，想到一些别人考虑不到的点
2. 具备一定的编程基础
3. 知道软件工程中，测试所需要做的事情，了解性能测试、[安全测试](https://www.baidu.com/s?wd=%E5%AE%89%E5%85%A8%E6%B5%8B%E8%AF%95&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)等

3、软件测试的基本方法

4、会写的基础的[测试文档](https://www.baidu.com/s?wd=%E6%B5%8B%E8%AF%95%E6%96%87%E6%A1%A3&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)（[测试用例](https://www.baidu.com/s?wd=%E6%B5%8B%E8%AF%95%E7%94%A8%E4%BE%8B&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)、缺陷报告等）

5、会看开发文档（需求说明书、操作手册）

6、做事要细心认真，要踏实，有兴趣有求知欲，要花心思琢磨。

## 9、测试案例题

### （1）网页搜索：

首先是边界值测试：不输文字，直接点搜索；输入\*\*字符，能成功跳转；输入\*\*字符，会截断；\*\*字符无法识别；输入英文；输入空格；其他符号；

区间测试：有意义的关键字；乱码；

界面测试：图片，字体，颜色，按钮等；

与鼠标的交互；

安全性测试： 被删除、加密、授权的数据，不允许被查出来的，是否有安全控制设计

兼容性测试：不同数据库存储下，不同操作系统下，不同语种，不同浏览器下的兼容性

### （2）购物网站从登陆到下单的整个测试设计

### 购物车测试案例

界面测试：

　　·打开页面后，页面的布局是否合理，显示是否完整；

　　·鼠标浮动在购物车按钮，迷你购物车界面显示是否正常；

　　·不同卖家的商品在不同的table区域显示，区分明显；

　　·页面的tooltips能正常显示；

[功能测试](http://www.51testing.com/html/86/javascript:;" \t "http://www.51testing.com/html/86/_self)：

　　·所有页面链接功能正常，可以点击到正确页面；

　　·页面关联本地软件阿里旺旺的icon点击后，能打开软件；

　　·从商品信息页面添加的商品能显示在购物车中；

　　·购物车页面打开的同时，在其他页面添加了商品，购物车页面刷新后，新的商品能显示；

　　·若未登录，点击购物车，则提示用户输入用户名和密码，或者提示其他的非注册用户购物方式；

　　·商品未勾选的状态下，结算按钮是灰色无法点击的；

　　·勾选商品后，已选商品的总价会显示，结算按钮变高亮可点击工作；

　　·勾选商品，点击结算按钮后，进入确认订单信息页面；

　　·购物车页面中，可以对添加的商品信息做信息的修改，并自动保存成功；

　　·卖家在线的时候，旺旺icon高亮，反之，灰色；

　　·购物车有商品降价或者库存告急的，那么点击对应的tab，降价或者告急商品会归类后显示；

　　·购物车能添加的商品种类是有数量上限的；

　　·不要的商品，可以删除；

　　（其他特有的功能不做赘述，只讨论常见通用功能）

[性能测试](http://www.51testing.com/html/86/javascript:;" \t "http://www.51testing.com/html/86/_self)：

　　·打开购物车页面要多久；

　　可用性测试：

　　·快捷键功能知否支持

　　兼容测试：

　　·不同浏览器上的测试功能是否正常；

·app上测试

### 订单管理

### 流量充值

1）用户发过去的信息有携带key值   
2）商户系统本地数据会留存一份用户的订单信息，并且会根据每笔订单信息生成一笔支付信息（同时留存到本地）  
3）第三方支付成功，第三方有存支付订单信息   
4）充值成功，用户的流量余额有对应增加

异常用例  
1、修改用户发过去的数据：   
1）产品ID 与价值不对等---->检查点：篡改数据和key，检查商户系统报错：key值不对或者是用户数据有误。   
2）取消充值流量    
3）重复发起流量充值请求

2、商户系统-第三方之间：  
1）密钥搞错-第三方报错，不接收密钥  
2）提交商户系统里面不存在的订单/支付订单->第三方这里也是不能通过请求  
3）篡改用户支付金额-->第三方也要检查

3、第三方--用户之间:  
1）支付密码错误/余额不足   
2）取消支付   
3）重复支付[对账--->处理退款]

### （6）退款测试案例

1.用户发起退款--->该用户的订单以及支付订单号都要存在。---检查点：商户系统/第三方检查数据没有问题，可以退款成功--->交易状态改成退款

异常用例：  
1：无故发起退款：提交不存在的订单号或者支付订单号 --->订单号不存在/支付订单号不存在  
2：信息不匹配发起退款：提交订单号与支付订单号不匹配的数据--->订单号/支付订单号有误  
3：退款大于实际金额：提交的退款金额大于实际支付订单的金额-->商户系统要报错  
4：商户系统这里发过去的请求：退款金额大于实际支付金额-->第三方要报错

### （7）客户端和服务端测试

1.1.  客户端关注

尝试连接时，不联网，要有超时

对于本地无连接的测试，一定要区分断网和禁用网卡，这两种情况不同。

下载过程中，网络情况不佳或者断网，要有超时，最好也有三次重连机制，重连的时间不宜过短建议20ms

文件过大，建议需要有断点续传逻辑

要验证各种网络错误环境，最起码要包括200、302、403、404、417、500、502等错误和服务器超时、本地超时

测试时需要关注，每条url请求是否支持302跳转

需要模仿502跳转，确保调整后，客户端能够正常运行

服务器返回文件类型需要关注：文件格式错误、Html格式文件、空文件、0字节的文件。

服务器返回文件时，文件的储存路径，空间，路径的读写权限、储存路径是否已经存在文件，存在文件的数据（0字节，其他类型的文件、损坏的文件、下载的文件不完整）

服务器端返回的url值的类型、长度的容错，服务器返回文件的内容：是否加密，参数值为中英文、简繁体、特殊符号、数字、空、缺省、零、小数、负值、超长、乱码等，参数缺省，规定下载的文件个数与下载的文件实际个数不匹配。

发送的url内容，参数值中的特殊符号、中文是否已经转义；

需要测试时关注，交互时配置文件的编码问题，例如要覆盖到ASCII、unicode、ANSI等编码。

还要注意测试时，交互时配置文件中的换行与回车换行的问题，要保证这两种都可以测试通过

要注意在低权限进程中，该功能能够正常进行

该交互是否会被安全软件拦截；如果要打开浏览器访问，还要检查是否会被浏览器拦截

客户端发送的请求中是否带具有特性的User-Agent（具体根据与开发的沟通结果来验证）

基本功能测试完毕后，需要进行跟服务器端的联调。

客户端与服务器联调时，要注意确认联调case，多跟开发沟通；

1.2.  服务器端关注

对于服务器来说，要进行压力测试

客户端、服务器端分别测试后，上线前要有联调，除了走主功能外、还要结合开发与运营的意见设计联调case

服务端需要考虑是否要做安全校验，以免被攻击

服务端和客户端均通过测试后，在上线前，需要客户端和服务端进行联调测试，确认服务端和客户端均ok

客户端发送请求的内容：是否加密，参数值为中英文、简繁体、特殊符号、数字、空、缺省、超长。

### （8）微信的发送消息功能的测试

功能测试（图片、文字（大段文字）、小视频、语音）是否发送顺利，数据有没有丢失、有没有延迟，一个人发了100遍同样的数据能否发送成功，群发功能；性能能测试（并发度，响应时间，系统资源占用）、压力测试、兼容性测试（硬件兼容和软件兼容）、还有就是考虑多端登录消息的同步，

### （9）聊天工具的语音功能

发送语音对方是否能收到；收到的语音是否与发送的一致；一段语音最长能发送多长；如果超时了，语音功能会做怎样的处理，是截断还是分成多段；如果对方不在线，是否能正常发送；支持的语言；支不支持双方同时发送语音；多端同时登录的情况如何处理；

语音转化成文字；能否撤回，多久以内能撤回；能否转发；音质如何，是否清晰；能否支持耳机听和说，扬声器；兼容性测试：安卓和ios，手机版和电脑版。消息能保存多久。

### （10）杯子的测试

需求测试:查看杯子使用说明书

界面测试:查看杯子外观

功能度：用水杯装水看漏不漏；水能不能被喝到

安全性：杯子有没有毒或细菌

可\*性：杯子从不同高度落下的损坏程度

可移植性：杯子再不同的地方、温度等环境下是否都可以正常使用

兼容性：杯子是否能够容纳果汁、白水、酒精、汽油等

易用性：杯子是否烫手、是否有防滑措施、是否方便饮用

用户文档：使用手册是否对杯子的用法、限制、使用条件等有详细描述

疲劳测试：将杯子盛上水（案例一）放24小时检查泄漏时间和情况；盛上汽油（案例二）放24小时检查泄漏时间和情况等

压力测试：用根针并在针上面不断加重量，看压强多大时会穿透

跌落测试: 杯子加包装(有填充物),在多高的情况摔下不破损

震动测试: 杯子加包装(有填充物),六面震动,检查产品是否能应对恶劣的铁路/公路/航空运输

### 聊天类app测试

1. 对象  
　　对象就是聊天的联系人，包括个人账号，公共号，机构号，群组等。  
　　2. 对象的属性  
　　就是这些联系人的各个特征。  
　　个人主页（头像，昵称，签名，管理的群，管理的轻应用，2维码。。。）  
　　群 （头像，创建人，群成员，名称，消息设置，群轻应用，群类型，2维码。。。）  
　　公共号（机构号）（主页，头像，名称，简介，管理的轻应用，消息设置，2维码。。。）  
　　3. 方法。  
　　方法，就是这些对这些对象的操作。  
　　消息发送（单聊，群聊，语音，文字，图片，表情，链接，字符及长度。。。）  
　　成员管理（加人，被加，退出，被动退出，编辑，删除。。。）  
　　群组管理（创建群，消息设置，申请入群，扫二维码入群，退群，通知提醒，头像编辑，名称编辑，简介编辑，权限编辑，成员编辑。。。）  
　　消息管理（发布通知，接收通知，发文件，消息提醒，通知提醒，声音，震动，好友请求，请求处理。。。）  
　　隐私管理（黑名单，允许好友查看动态，允许陌生人查看动态，允许通过手机号查找，允许真实姓名查找。。。）  
　　权限管理（开放群（任何人入群），半开放群（验证入群），验证加好友，不需验证加好友。。。）  
　　登录退出（忘记密码，更换账号。。。）  
　　好友管理（扫二维码加人，加好友，查好友，好友推荐，群组推荐，联系人导入，拉黑名单，解除好友，备注名。。。）  
　　动态管理（发动态，发投票，点赞，表情，评论，增加，删除，分享，隐藏，编辑。。。）  
　　消息推送（在线，离线，收发，时序。。。）  
　　文件管理（接收，离线接收，预览，删除，分享，转存，文件格式，大小。。。）  
　　4. 缓存  
　　缓存这个东西需要检查的，关系到用户体验，如果页面没做缓存，打开很慢。可以清除缓存测测。  
　　5. 升级  
　　安装，卸载，升级一定要测。有的时候需要直接升级测试，因为开发可能改了表结构而没告诉你，导致升级后部分数据丢失，后果很严重。还要看升级后消息记录是否有丢失（如果可能，需要去查看数据库。）  
　　6. 网络  
　　要覆盖(2G,3G,4G,wifi),如果有条件，可以搭建一个慢网的环境，测试慢网或者断线的表现，消息是否成功发送，或者retry几次，没发出去是否有提示等。（各种响应速度是否可以接受）  
　　7. 中断  
　　要测试启动，静置，活动中等状态下的中断，看错误的操作是否有保护现场机制。。。  
　　8. 兼容性  
　　布局检查，分辨率检查，各种代表性机型检查  
　　9. 闪退

# 六十一、项目方面

## 1、为什么要用MFC，vc6.0，不用c#，qt？

答：

1）由于我们项目的核心算法是用c、c++写的，代码可能有转换成fpga代码的需求，就可以通过vivado转换成fpga代码，同时可能有支持代码移植到Linux平台的需求，而C#不支持，因此无法用c#，但是现在界面部分正在转c#，核心代码还是c/c++。（不用c#）

2）嵌入式人力投入大，无法短时间内搭系统，研发周期较短，不现实，因此采用工控机的方案，运行速度快，配置低，代码可重用。（不用linux）

3）我们的重要功能模块都是用MFC写的，因此折中考虑，我们采用平台较老的MFC。同时方案制定初期，QT还不流行，由于我们的研发周期短，只能在原有的项目上增加新功能，因此不能做较大的改动。（不用qt）

虽然这个平台有点老，但是并不影响我学习c++的知识，我一样可以学到很多东西。

## 2、用了哪几个线程？

答：采集线程，UI线程，按键线程，输入/输出线程

## 3、这个项目软件部分最大的难点？

答：板卡多，所有板卡的状态例如档位、耦合关系、滤波等控制复杂，

同时多物理量数据都需要单独处理。

## 4、如何保证数据实时性？

答：FPGA对所有板卡并行采集数据后，组成多个TLP包，因此我们取到的数据能保证实时。

## 5、有没有使用第三方库、第三方控件库？

答：第三方库：msxml3.dll（用于存配置文件）、(msimg32.lib)（用于绘图）、mysql（数据库）

第三方控件库：换肤控件、Xtremer

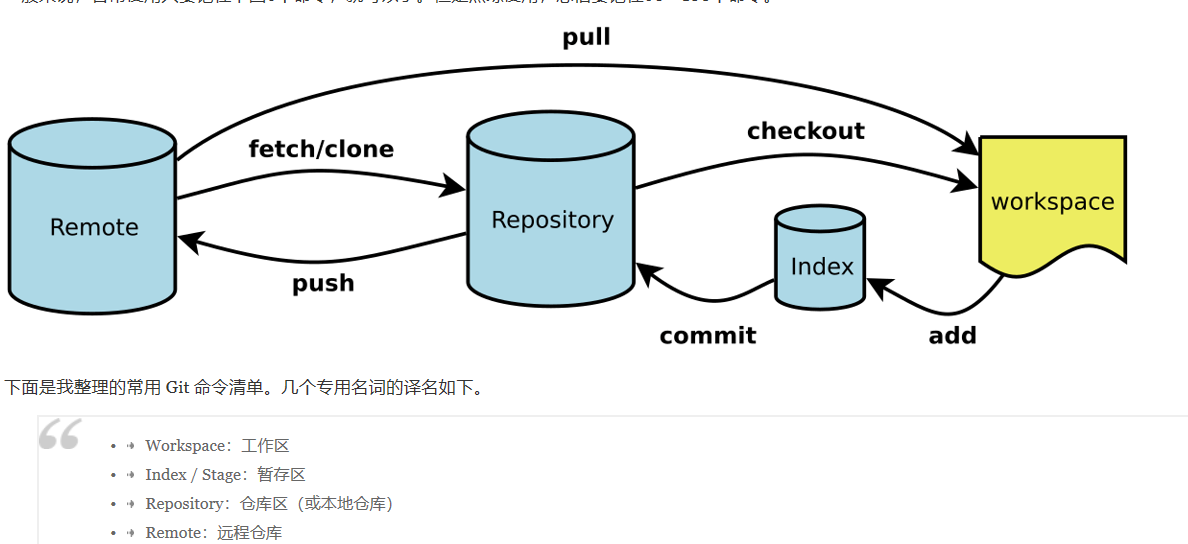
## 6、开发中你觉得最难的部分?

答：我认为我在项目开发过程中遇到的最难的部分就是我负责的测频优化的部分，它是难点在于我需要考虑的情况很多。首先，在原本的项目里频率的测量是采取测周法，但是它在高频部分的测量精度不高，针对这个问题，我们引入了测频法，我是采取了一个分段函数，先用测周法测量出一个大概的频率，再根据频率范围反过来选择用哪种方法。其次，我们的仪器有刷新速度过慢的问题，我分析了可能的几种原因，1、我们原来所用的测量周期数是固定的，设置的是20个，这样假设测1hz频率就测频完成就需要20s，时间太长了，因此我们就将周期数设为可变的，不同的频段设置合适的周期数，当然要保证这个频率是一个正常的频率，不是突变值，因此我还添加了一个锁频的功能。2、刷新过慢的还有一个原因是因为在fpga测频完成后，我们需要等待一段时间，保证数据的完整传输，这个时间是一个ns级，但是我们用的等待函数用的sleep是毫秒级，这样等待的时间就太长了，这个是我用了一个Truetime的工具，分析了工程中每一部分代码运行的时间。然后我就找了一个利用系统API的函数，实现us级的延迟。3、还有一点是我们每次测频完之后都发送了复位信号，其实发送复位的过程是很耗时的，因此我就记录了每个通道上次发送的周期数，只有在当前通道周期数变化的时候才重新发送复位信号。通过这些方法，界面刷新速度了有了明显的提高。再一块就是遇到异常值的处理，称为锁频功能。我是这样处理的，如果遇到一个值与前面两个值的差值超过我设定的范围，比如百分之一，我就认为是一个突变值，我就会把它记录下来，然后再与之后的两个测量值比较，如果是连续三次在某个范围内，我就认为是输入信号频率值的改变，就更新显示和运算的频率，否则就是个异常值，剔除了就可以了。在整个实现的过程中，我画了多次流程图来整理我的逻辑，同时我经常跟fpga的同学沟通，了解他们的流程和**需求，**清楚我需要对接的部分，我也经常跟老师讨论，最后顺利完成了这个功能。

## 7、项目中用到的设计模式

1. 对话框的实例只生成一次，并供所有类使用
2. 观察者模式：所有通道都有比如滤波、耦合等参数，所有通道都处于一个监听的状态，只要底层的某个参数更改，相应通道就要更改相应的参数。
3. 适配器模式：上层和底层不能互相访问，是通过一个接口类实现的。

# 六十二、git指令



# 六十三、软件岗面试准备

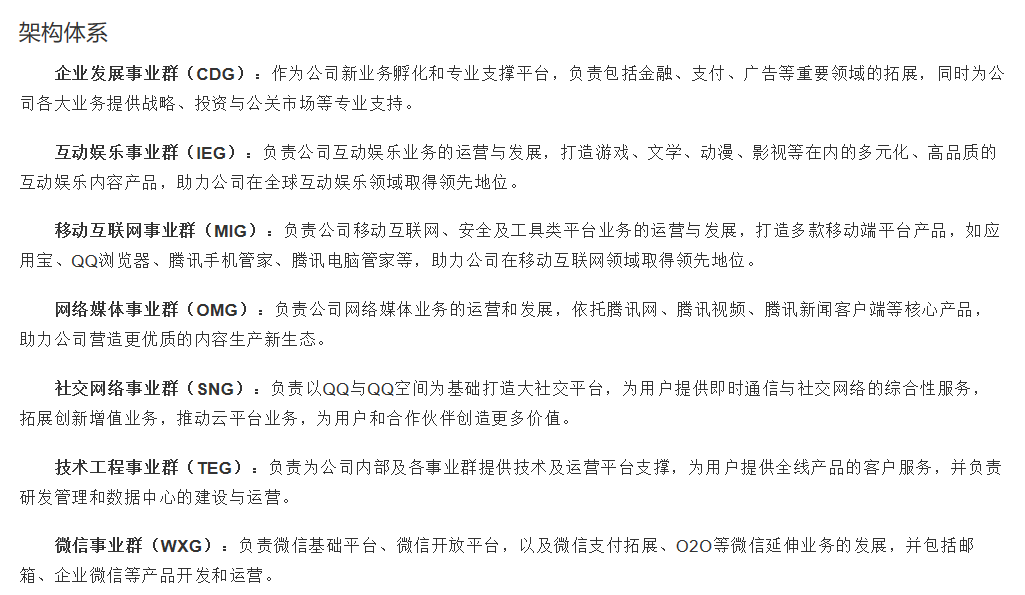
## 1、职业规划

（1）2-3年技术人员：目前我对于计算机的基础知识还不是很扎实、并且项目经验也不是很丰富、对于技术掌握还不是全面，所以你现在很有必要系统学习一下计算机的基础课程，比如数据结构、[操作系统](https://www.baidu.com/s?wd=%E6%93%8D%E4%BD%9C%E7%B3%BB%E7%BB%9F&tn=SE_PcZhidaonwhc_ngpagmjz&rsv_dl=gh_pc_zhidao" \t "https://zhidao.baidu.com/question/_blank)和计算机算法，完善知识体系。

（2）项目组长-项目经理：工作重点在同客户沟通需求、项目进度的把控、团队的沟通方面，有些公司也会需要项目经理承担团队建设的工作，不过貌似很多国内公司都忽略了团队建设这个工作了。对于项目经理来说，重点会要求沟通能力、协调能力、危机把控能力、执行力、团队管理能力，着重于沟通、管理与计划。当然也有些公司还要求项目经历要参与招标谈判，这就要求项目经理有一定的商务谈判能力。

## 2、为什么要投腾讯？

首先当然是因为腾讯是个大平台，发展的前景好，我觉得在这里我有更多发展的可能性；然后我学的是C++，我知道腾讯的大多数项目都是用C++开发的，所以说其实学C++的人应该对于腾讯都是向往的，能跟前辈学到更多的知识。然后我投的这个岗位的原因是因为要求里写用MFC进行开发，这个其实和我研究生的学习内容非常合适了，我们研究生做的两个项目都是使用的MFC开发的。



## 3、我的优势

首先我研究生的两个项目都是用VC MFC开发的，有项目使用经验，所以说在这部分我是跟腾讯的岗位需求非常符合。然后我觉得我是个很踏实认真的人，理解能力也不错，很细心也很有耐心。然后我们教研室也是类似公司的结构，有各个负责不同部分的小组，有硬件，fpga，软件组，我们每周组内和组间都会有周会，交流项目进度和各自的需求，所以说我觉得我的沟通交流能力和团队协作能力也是得到一定的培养，进公司之后也能很快适应工作节奏。还有就是我也有随团队出差的经验，所以说我的心理素质和应变能力也比较不错。我觉得我非常适合这个岗位也能很好的胜任它。

## 4、C++优点

c++既保留c语言的灵活性，简洁高效和接近汇编底层语言等的特点，又增加面向对象等特点。用c++写的代码可读性好，运行效率高。

首先C++效率高，常用语言里面仅次于C和汇编，当然，这个跟个人的写法也有很大的关系，如果封装太多，虚类太多，效率也会低  
然后，C++灵活，因为相比JAVA之类的程序更加接近底层，所以它能够更加本质的了解到程序具体的使用，你可以看到指针，可以看到具体的内存，至少指针是其他语言都不玩的吧？C++能够在更底层的地方帮你实现更加多的东西  
接着，就是C++难用，一个好的C++程序员往往比其他更高级语言的程序员要积累更多的经验，一个小小的指针就可能导致程序的整个崩溃，这是很让人抓狂的事情。调试的时候你需要关注各种可能的问题，出现问题的时候必须想尽办法去调试。

## 5、MFC优缺点

可用资源多，各种控件和已有的代码，各种参考资料。只要会C++就能上手。如果对MFC的基础类不满意，还可以自己继承添加新的成员。使用起来非常的灵活。执行效率很高。ClassWizard能够帮你快速找到需要添加的消息响应和重写虚函数。兼容性非常好，vc6写出来的程序在win7 64位下跑的都很流畅。比较适合做大型工程。

# 六十四、银行部分

## 为什么要投银行？

**首先我觉得企业的竞争压力太大，工作的不确定因素太多了，我还是想找一份工作内容和人际关系都相对稳定的工作，然后我觉得我性格比较开朗，相对于企业的紧张压抑的气氛，我认为银行也要更适合我。然后我做事非常细心，与人沟通的能力也比较强，在研究生期间我也学到了一定的专业技能，所以我认为我也能很好的胜任银行的工作。**

## 为什么要投银联而不是其他银行体系？

**首先银联是区别于其他银行体系的，它是一种跨银行的交易清算系统，要处理的数据量和管理系统的安全性都相对要高，所以对于技术方面要求要更高一点，而且我觉得银联有广阔的市场前景，所以我倾向于选择银联。**

# 六十五、string的基本操作

## string转化为字符数组(3种)

string s="abcd";

char \*a=new char;

1. strcpy(a,s.c\_str());
2. strncpy(a, str.c\_str(), 10); //复制s的前10个字符给字符数组a
3. int length=s.copy(a,3); a[length]=’\0’; //复制s的前3个字符给字符数组a
4. int length=s.copy(a,i,j); a[length]=’\0’; //复制s的下标从j开始的i个字符给数组a

## 字符数组转化为string

char \*a=”abcd”;

string s=a;

## 截取字符串

string sub1=s.substr(i,j); //截取s的下标从i开始的j个字符

string sub2=s.substr(m); //截取s的下标从m开始到末尾的字符

## 字符串初始化

1. string s(3,’a’) //将s初始化为3个’a’,即”aaa”.
2. string s(”abcd”,2) // 将s初始化为ab
3. string temp=”abcd”;string s(s,2); //将s初始化为cd
4. string str=”abcd”;String s(str);
5. string s(”abcd”);

## 赋值/增加值

string s;string s2=”ab”;

s=”abcd”;或s.assign(“abcd”);

s+=”ab”;或s.append(“ab”); s+=s2;或s.append(s2);

s.assign(str,i,j)/s.append(str,i,j) 将str的下标为i开始的j个字符赋值或增加给s

## 交换两个字符串的值

swap(s,s2);

## 插入字符

s.insert(s.begin()+i,ch); //在s的下标为i的字符**前**插入**字符**ch

## 8、删除字符

s.erase(i);//删除下标为i的元素

## 9、替换字符

s.replace(i,len,str); //替换下标从i开始的len个字符为str

## 10、比较函数（>、==、<、>=、<=）

string s,str;

int a=s.compare(str); // s>str,a>0;

s=str,a=0;

s<str,a<0;

按字母顺序从左到右依次比较，如果字母都相等，长度较长的字符串较大。

## 11、查找函数

int pos=s.find(str,index);//从s的index（默认为0）位置开始查找字符串str，并返回位置。查找失败返回-1.

# 六十六、[cin、cin.get()、cin.getline()、getline()、gets()](https://www.cnblogs.com/wft1990/p/5861529.html)

## 1、cin使用方法

## 输入一个字符，>>遇到滤掉不可见的字符（如 空格 回车，TAB 等）会滤掉；要想不滤掉，使用noskipws

e.g char ch[5];

1. cin>>ch; //输入a b c d e 只能读入a
2. for(int i=0;i<5;i++) cin>>ch[i];//输入a b c d e 读入abcde
3. for(int i=0;i<5;i++) cin>>noskipws>>ch[i];//输入a b c d e 读入a b c

## cin.get()

### cin.get(字符变量名)可以用来接收字符

e.g.

char ch;

cin.get(ch); //输入abcd 输出a

cout<<ch<<endl;

### cin.get(字符数组名，接收字符数目)用来接收一行字符串，可以接收空格

e.g.

char \*ch=new char;

cin.get(ch,5); //输入ab cdefgds 输出ab c加’\0’

cout<<ch<<endl;

## cin.getline()

### 用来接收一行字符串，可以接收空格

e.g.

char \*ch=new char;

cin.getline(ch,5); //输入ab cdefgds 输出ab c加’\0’

cout<<ch<<endl;

## getline()

接收一个字符串，可以接收空格并输出

e.g.

string s;

getline(cin,s);

cout<<s<<endl;

和cin.getline()类似，但是cin.getline()属于istream流，而getline()属于string流，是不一样的两个函数

## 5、getline()和cin输入字符串区别

cin输入字符串时，遇到空格就结束；getline可以读空格

e.g.

（1）cin>>s; //输入为 aaa bbb

cout<<s<<endl; //输出为aaa

（2）While(getline(cin,s)) //输入为aaa bbb

cout<<s<<endl; //输出为aaa bbb

## 6、gets()

接受一个字符串，可以接收空格并輸出

e.g.

char \*ch=new char;

gets(ch);

cout<<ch<<endl;

## 7、getchar()

接收一个字符

e.g

char ch;

ch=getchar();

cout<<ch<<endl;

# 六十六、函数参数列表的进栈顺序

# 六十七、结构体中柔性数组的作用

# 六十八、数据结构

## 堆

## 栈和队列

# 六十九、嵌入式操作系统

## 时间片

## 实时操作系统

## 分时操作系统

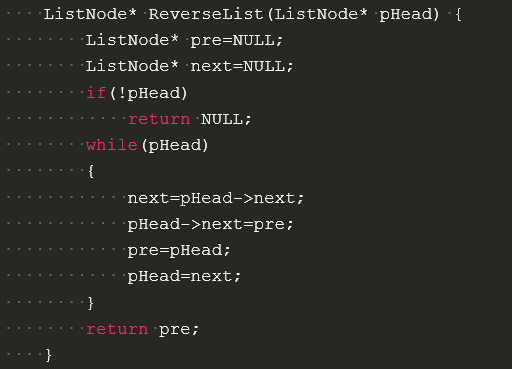
# 六十七、常考编程题

## 1、最大不重复字符串长度

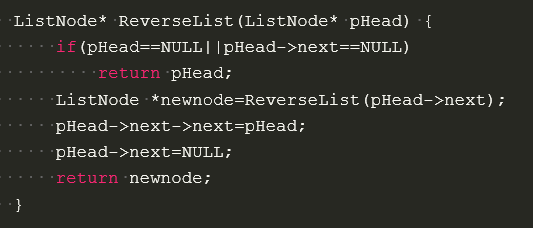


## 2、迭代法反转链表

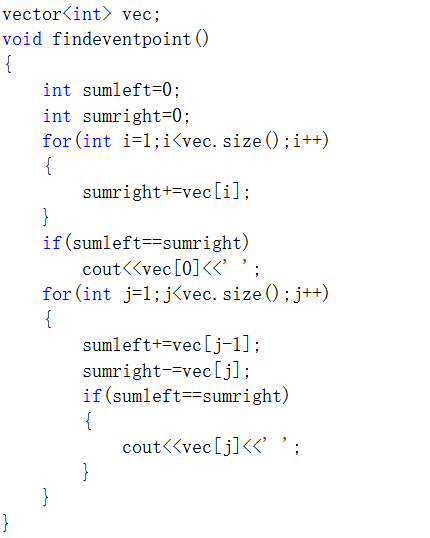
迭代：



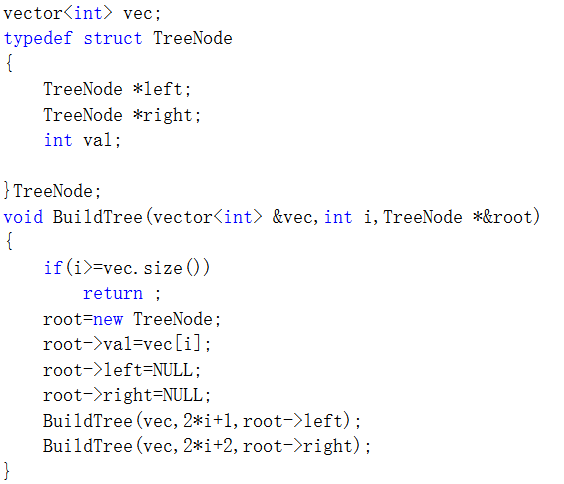
递归：

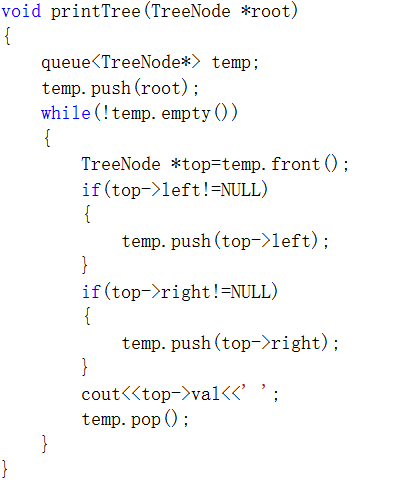


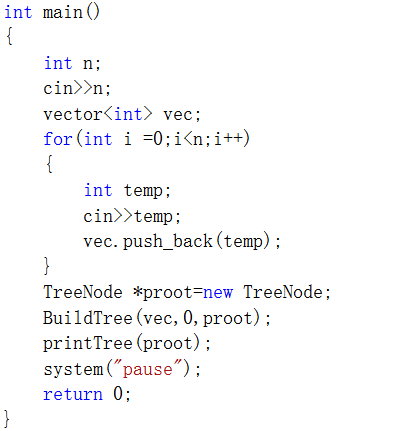
## 求一个长度为400的数组中的所有平衡点



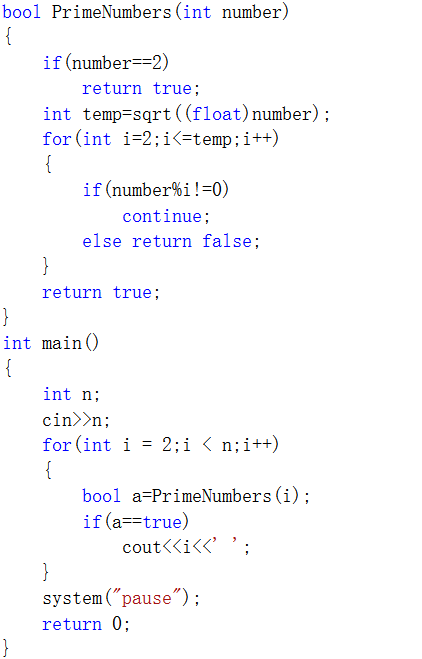
## 如何用数组定义一颗二叉树，并按行打印







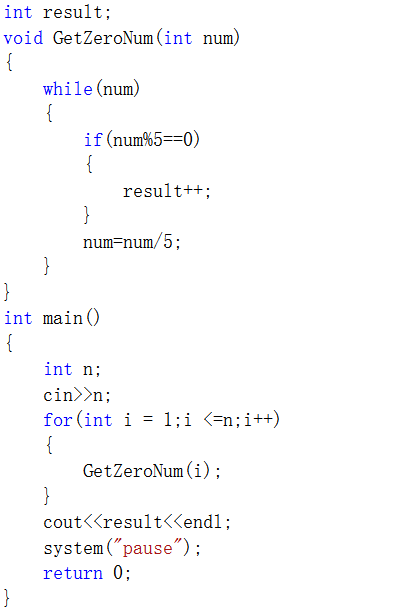
## 求比n小的质数（给定一个大于2的数，对这个数求平方根，如果这个数能被从0到平方根的数中任意一个数整除，那么这个数就是不是质数，否则这个数就是质数。）



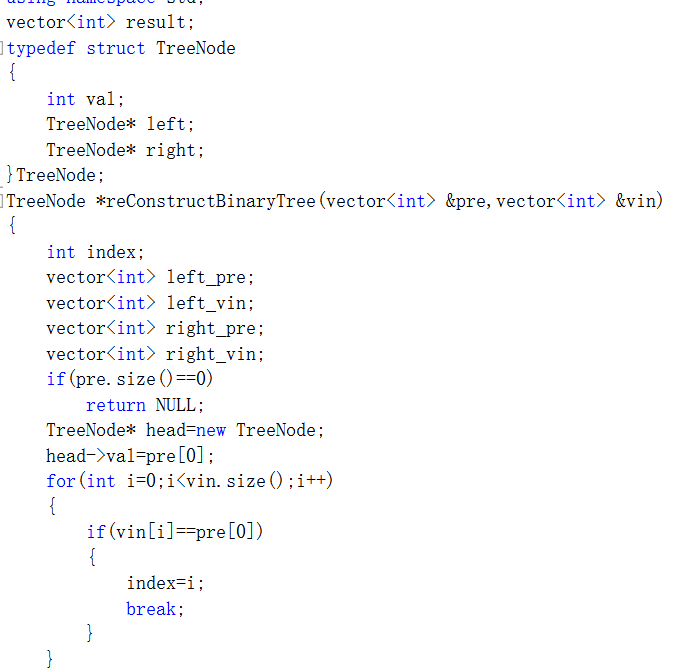
## n的阶乘，从个位开始共有连续的多少个零

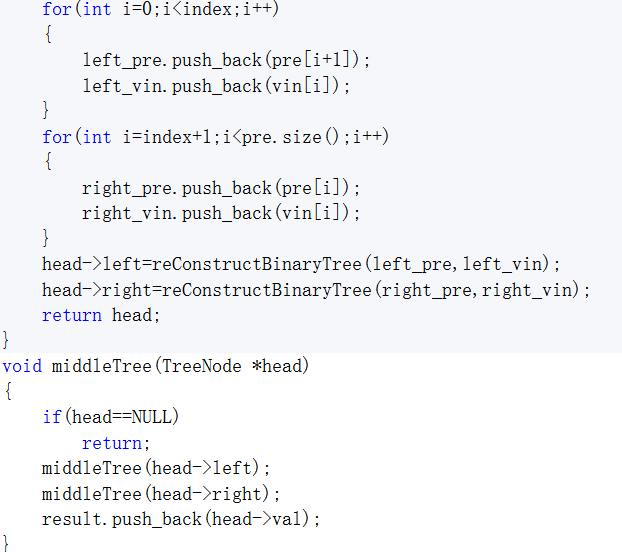
**（计算n的阶乘时，实际上就是把所有小于等于n的正整数分解成质因数，然后再将其乘到一起，那么末尾0的个数实际上就是2\*5的个数，而2的个数明显是很多很多的，所以问题就转化成了5的个数。**

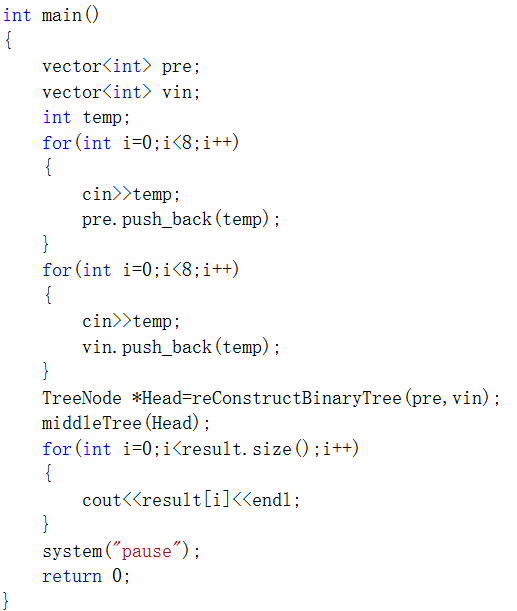
**而只有5的倍数才有5这个因数，所以，问题就进一步简化为小于等于n的数中有多少个数是5的倍数）**



## 根据前序和中序序列求后序序列

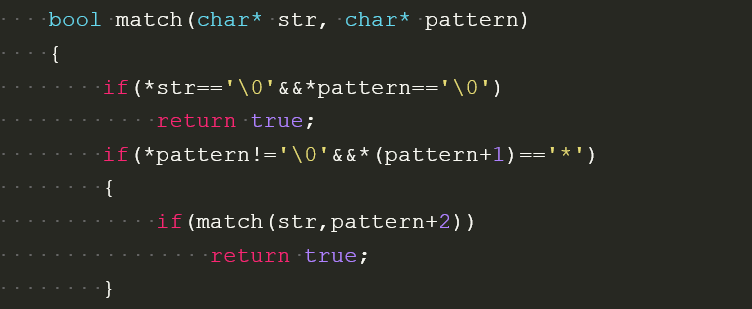


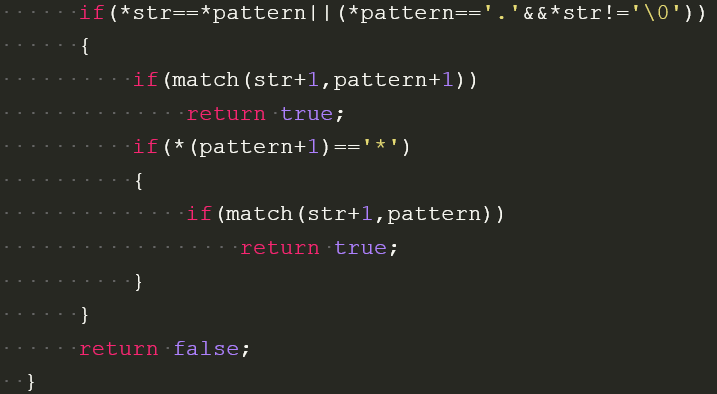




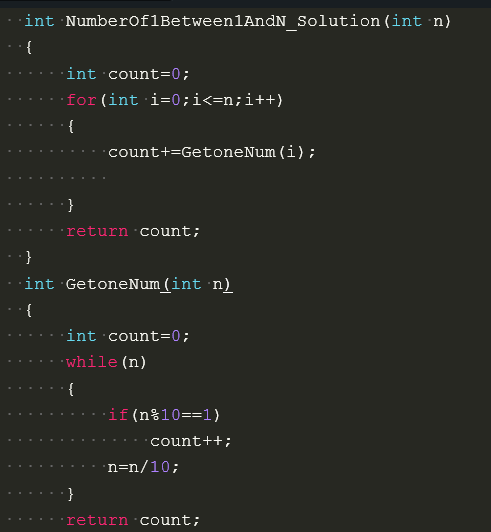
## 正则表达式匹配

（‘\*’表示’\*’前面的字符可以出现任意次数，’.’表示任意字符）

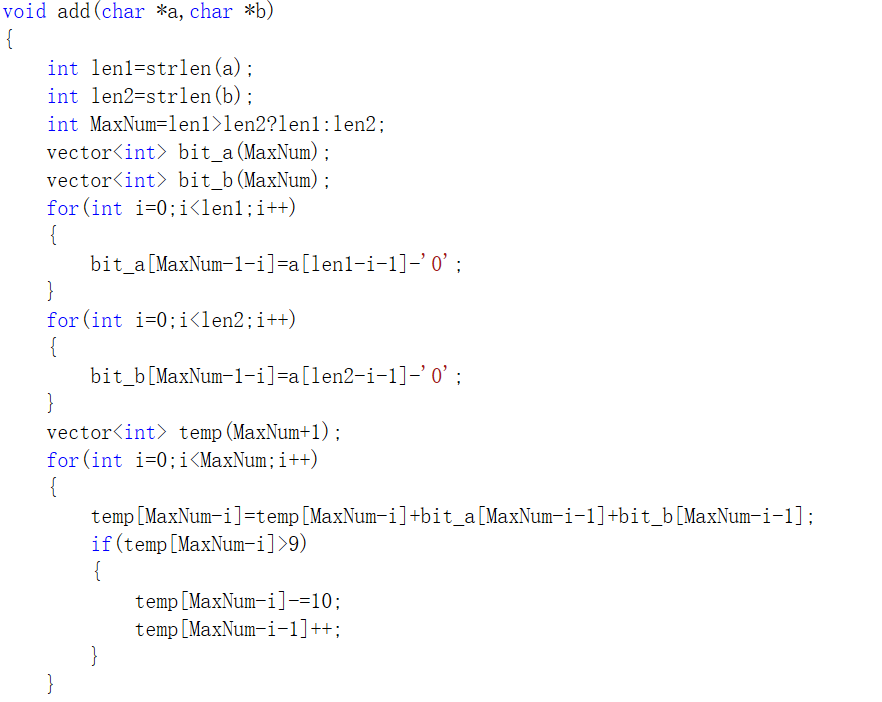


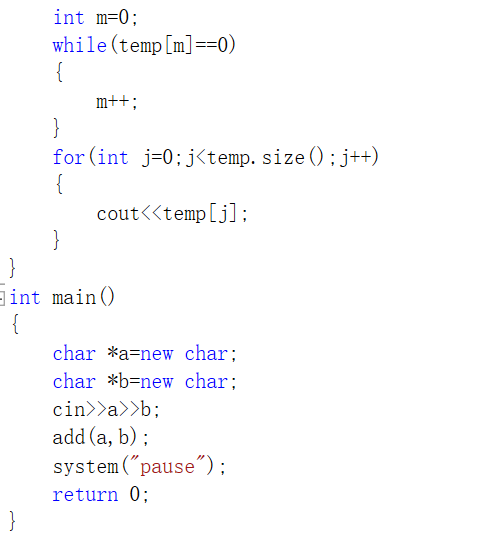


## 9、0-n中有几个数字1

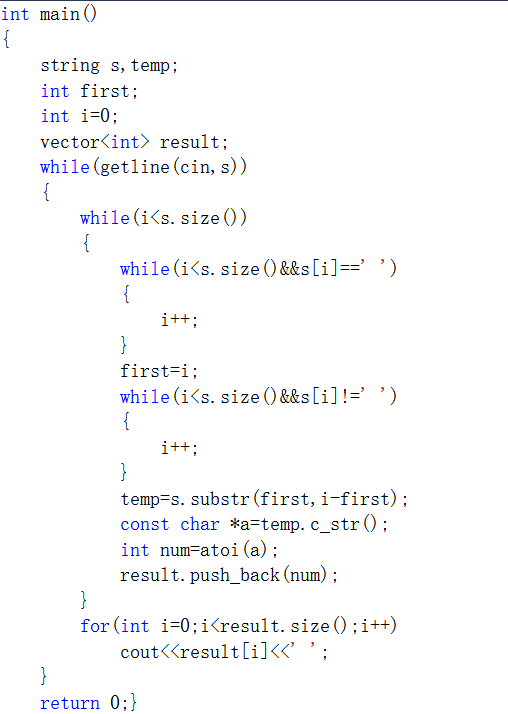


## 10、实现两个字符串的整型相加，用最基础的加法思想

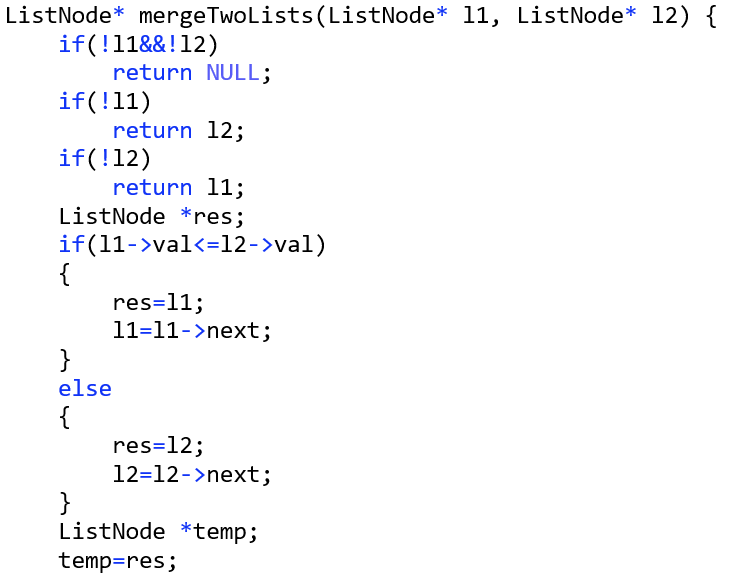


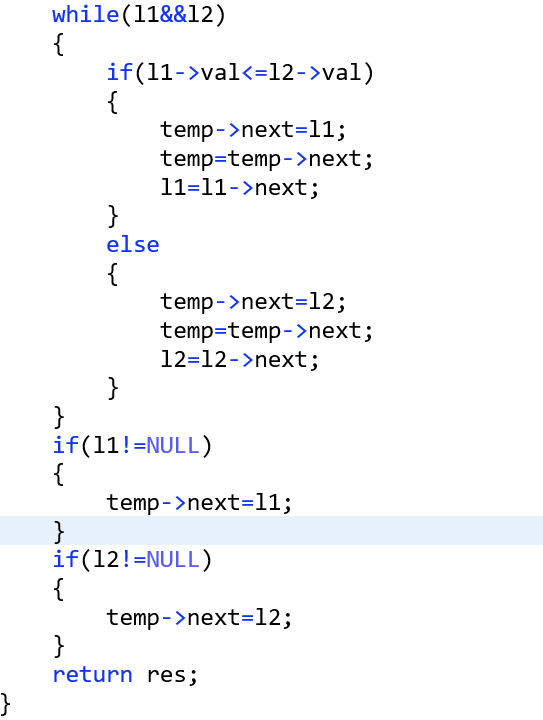


## 11、不定长输入一组数字

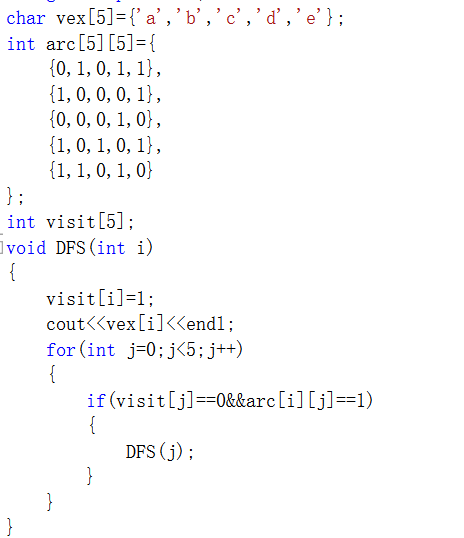


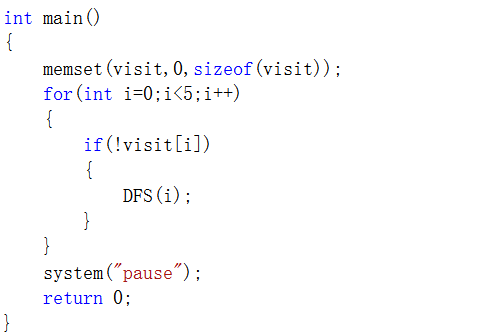
## 两个有序链表排成一个有序链表



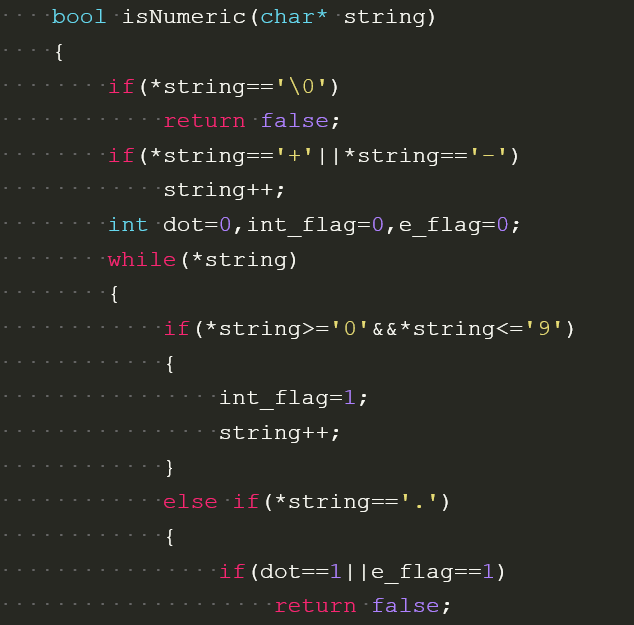


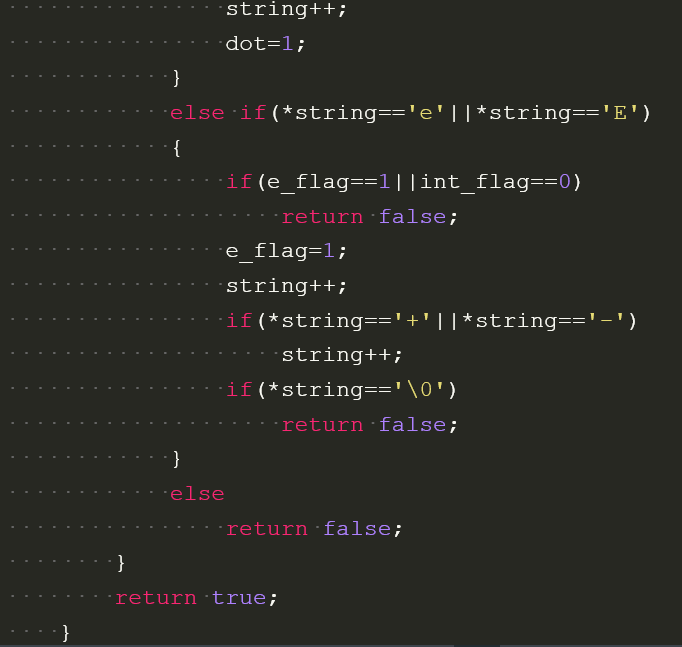
## 图的深度遍历



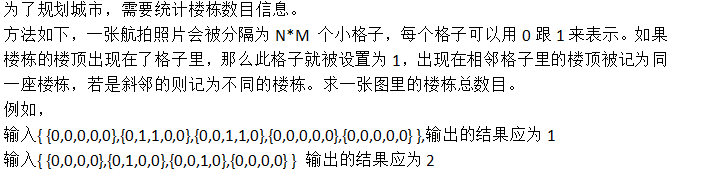


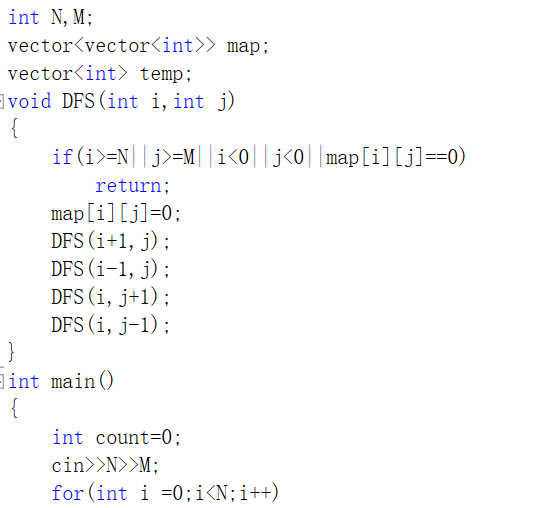
## 表示数值的字符串

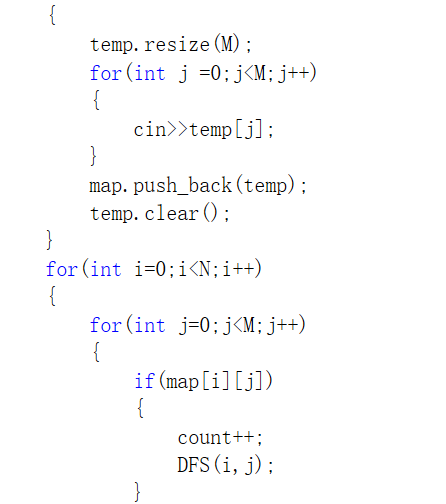


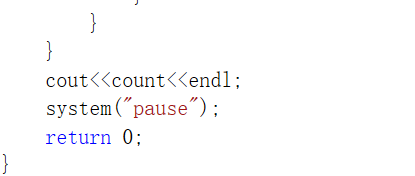


## 15、楼栋数目题

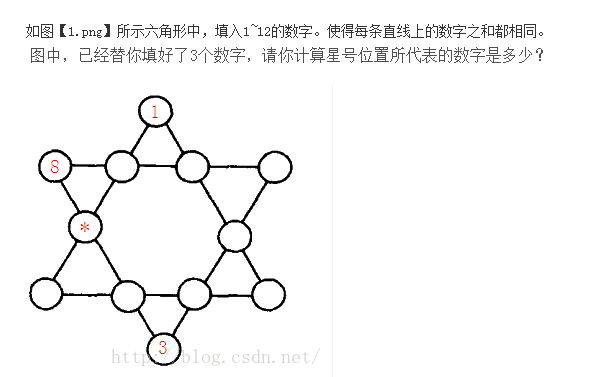


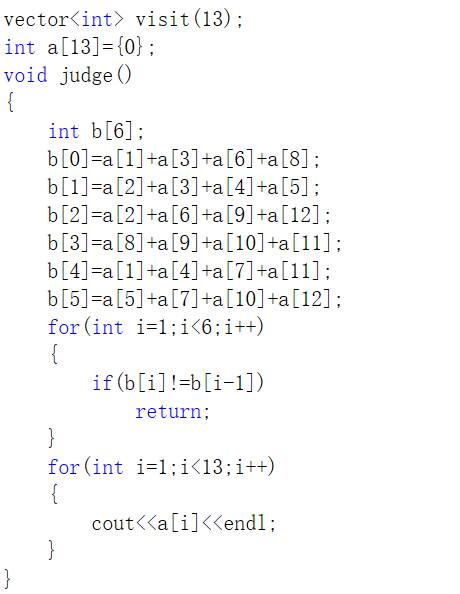


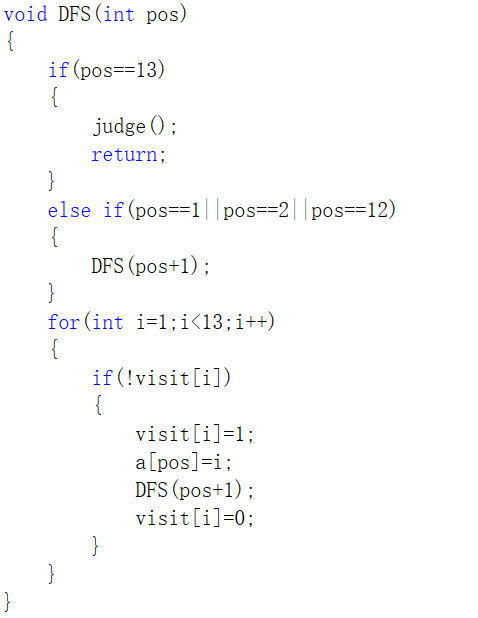


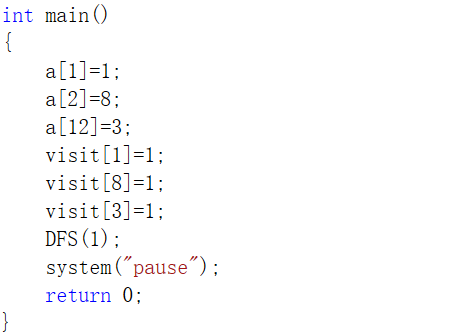


## 16、六角形填数字

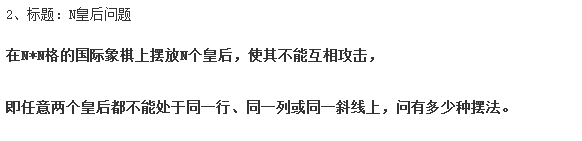


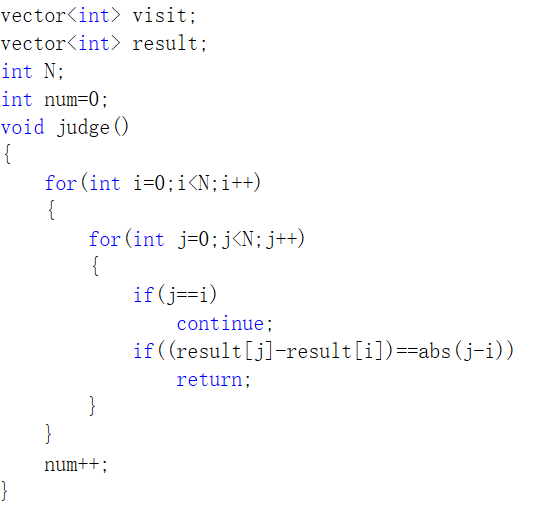


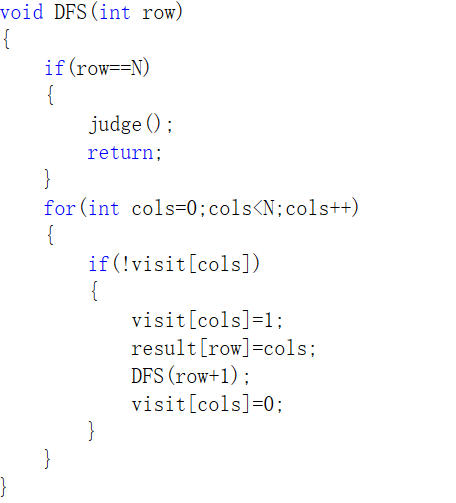


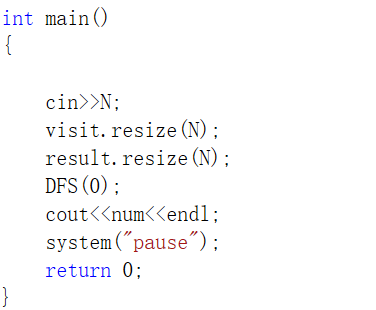


## 17、N皇后问题

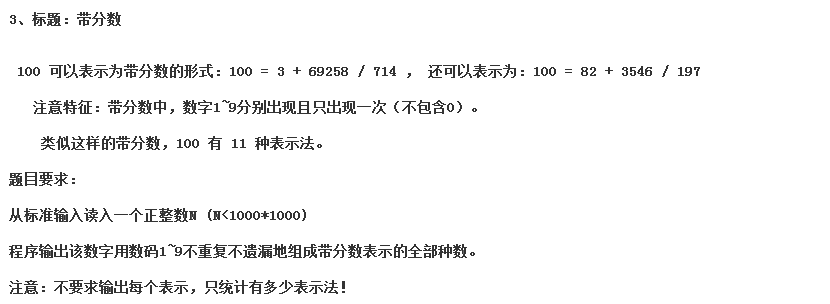


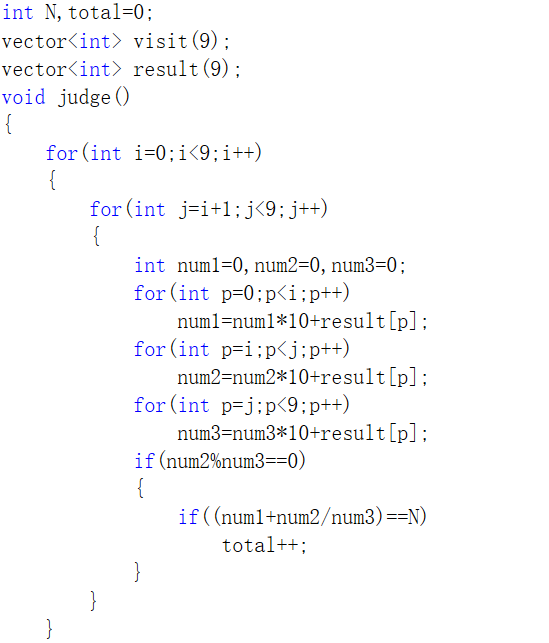


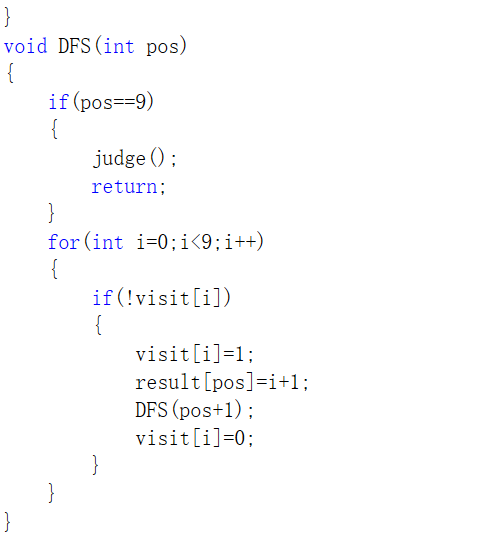


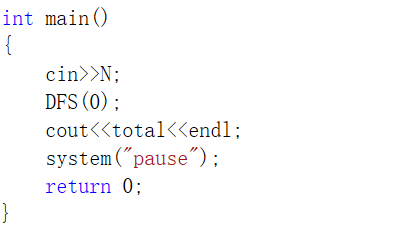


## 18、带分数





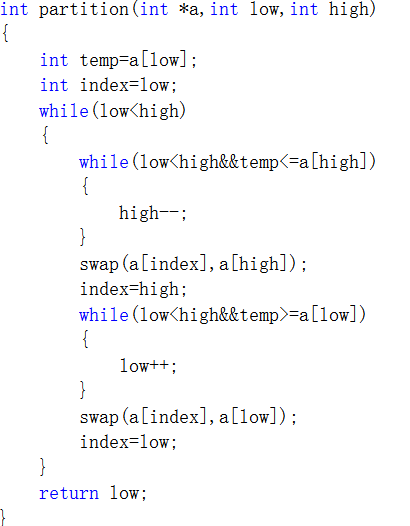


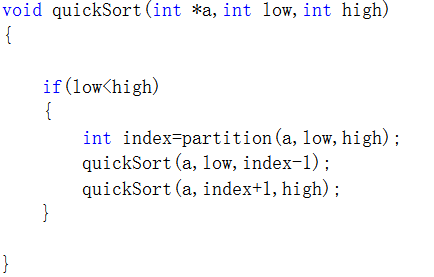


## 19、**排序**

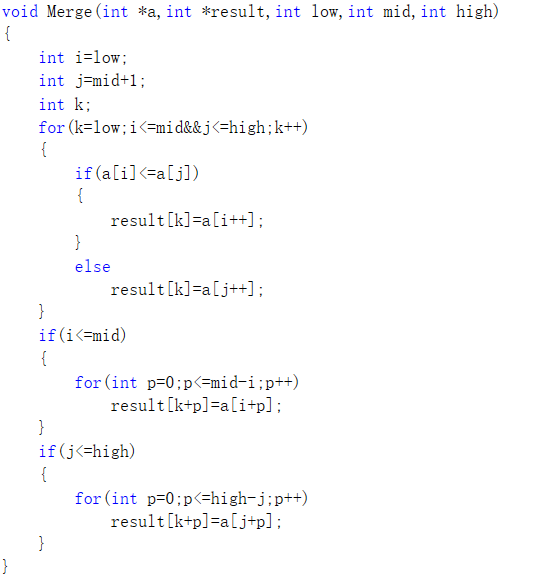


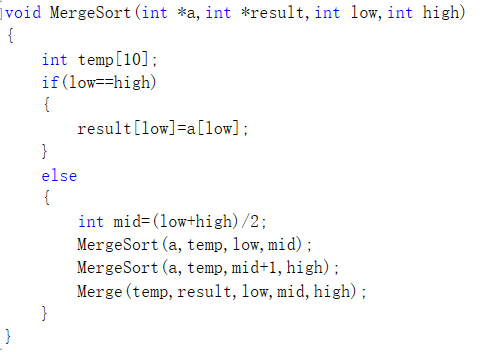
### （1）快速排序



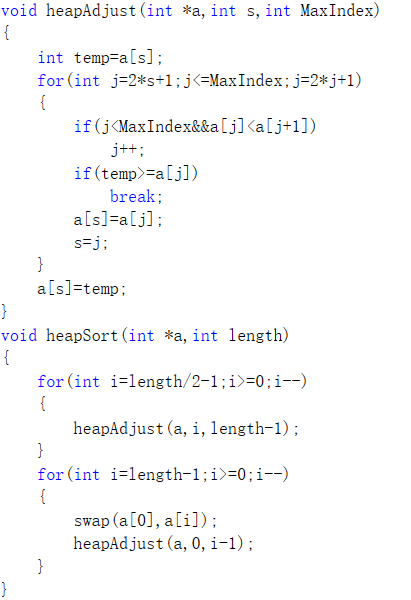


### （2）归并排序

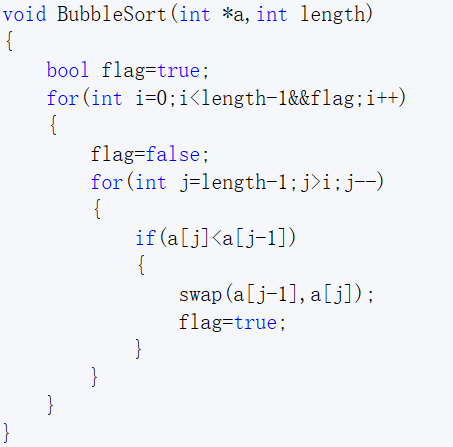




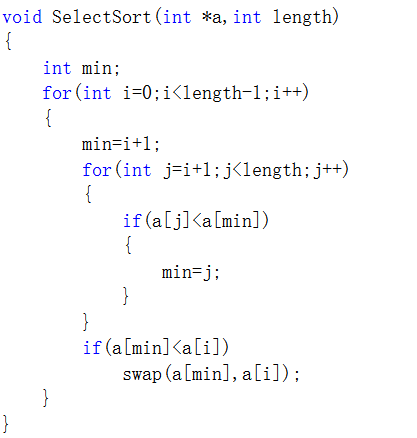
### （3）堆排序



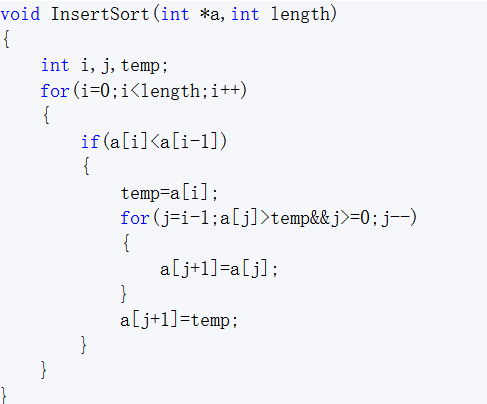
### （4）冒泡排序



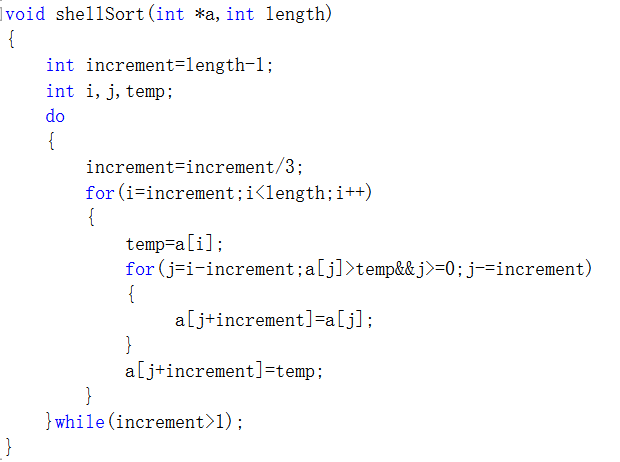
### （5）简单选择排序



### （6）直接插入排序

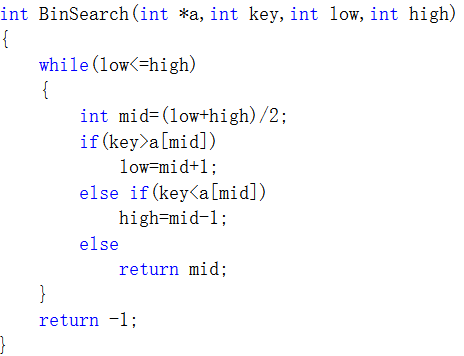


### （7）希尔排序

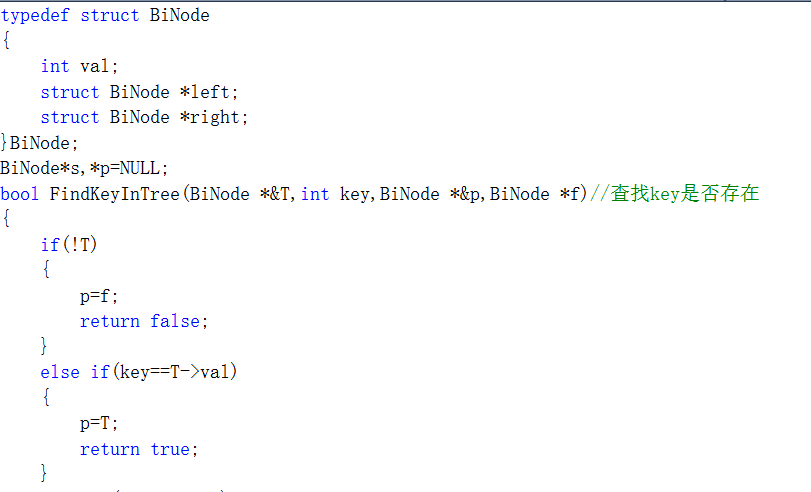


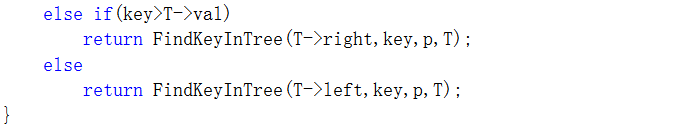
## 查找

### （1）二分查找

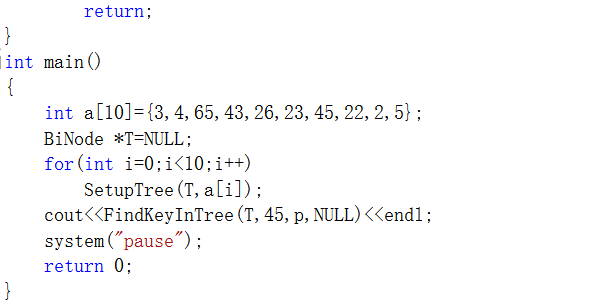


### （2）二叉排序树



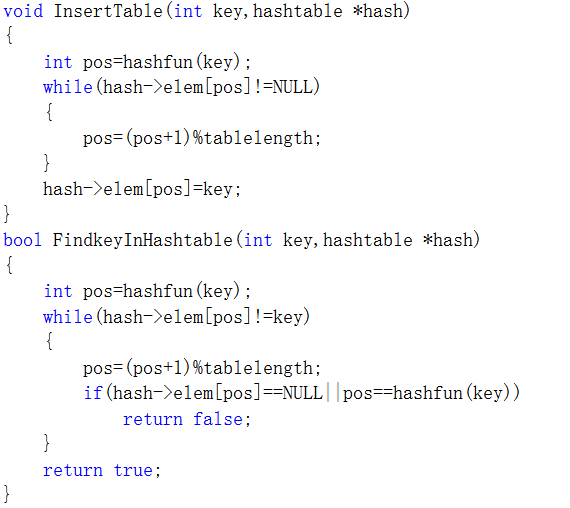


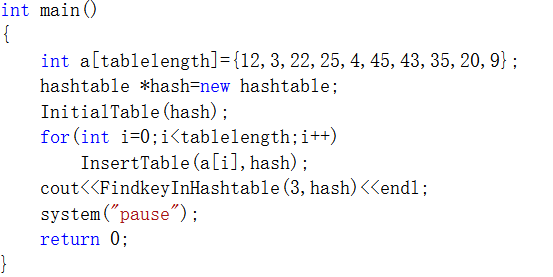




### （3）哈希查找



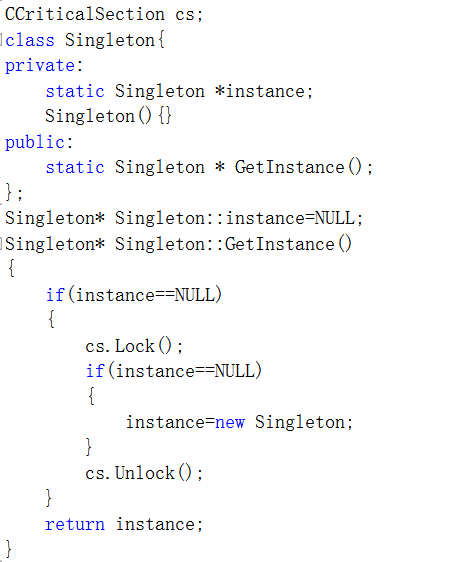


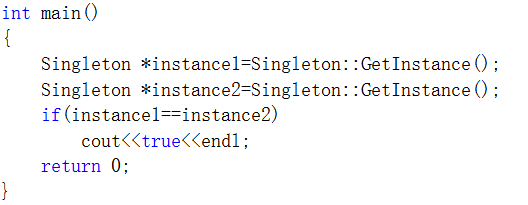


## **在一个字符串中找到第一个只出现三次的字符**

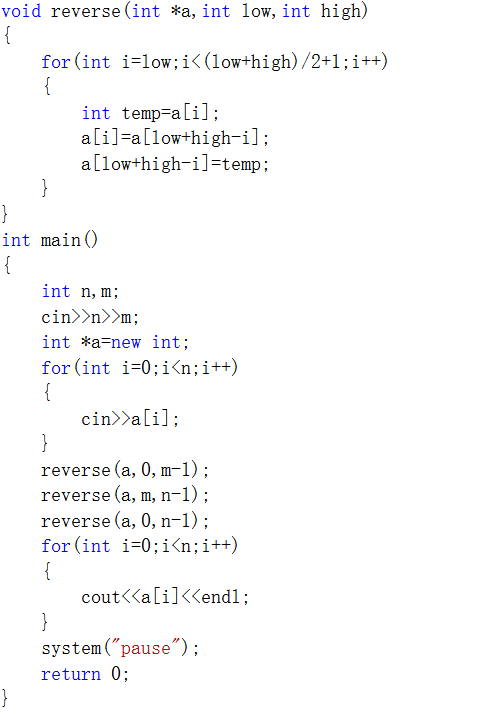


## 双重锁单例模式





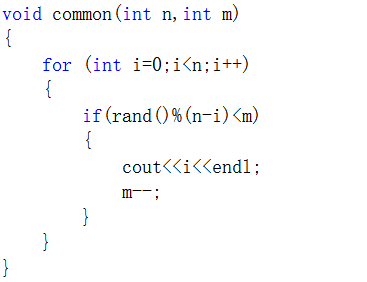
## 长度为n的整数数组循环左移m位



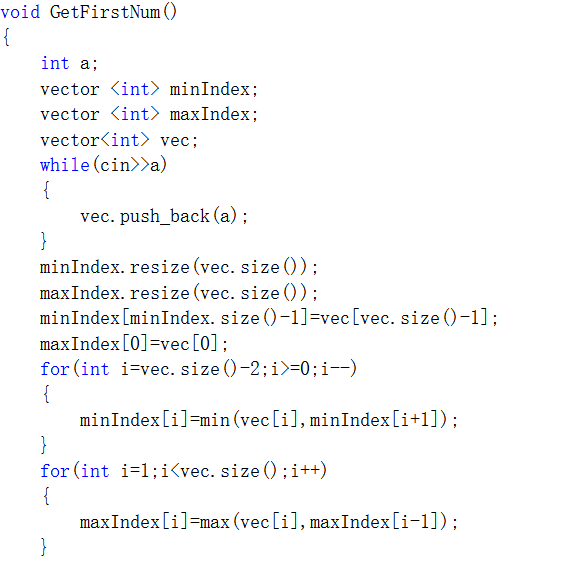
## 24、随机在[0-n]中生成不重复的m个数

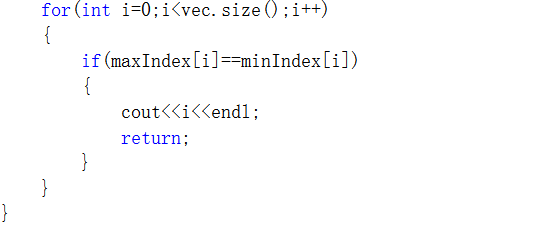


改进：



## 25、输入数组，判断数组中第一个满足其左侧的数字均小于该数，右侧均大于该数的下标



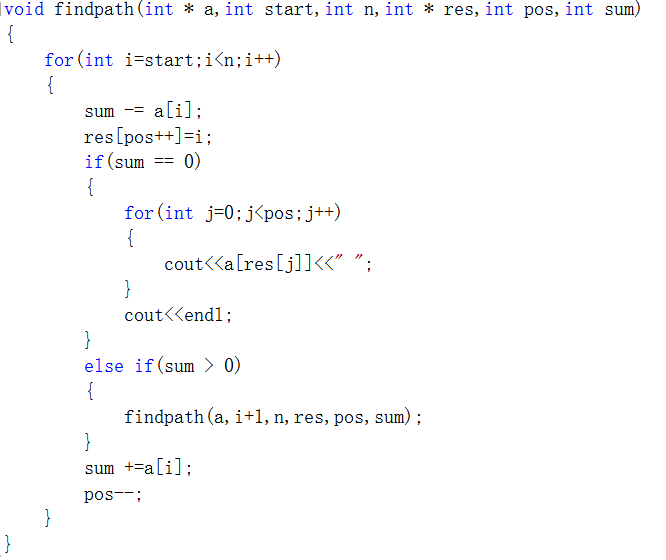


## 进制转换

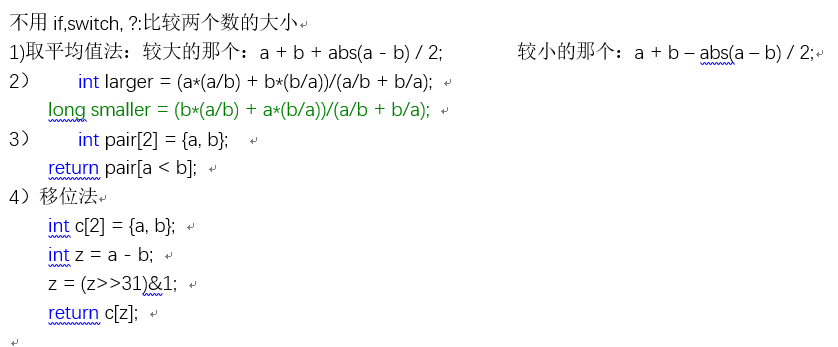
cin>>hex(oct)>>a;cout<<a<<endl; //十六(八)进制转十进制

Cin>>a;cout<<hex(oct)<<endl;//十进制转十六（八）进制

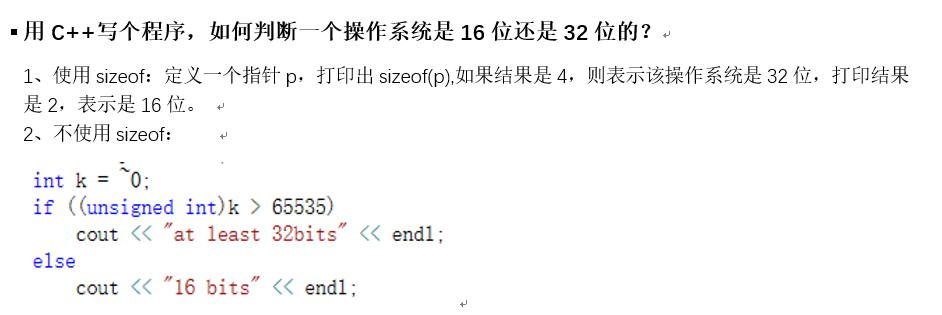
## 数组中和为100的子集



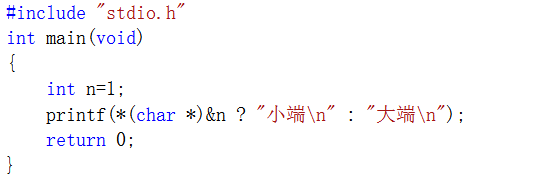
## 返回a,b中较大的数（不用if、switch，三目运算符）



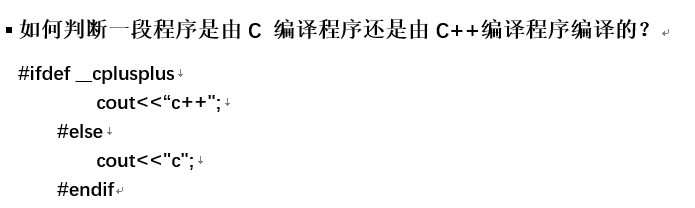
## 如何判断一个系统是16位还是32位



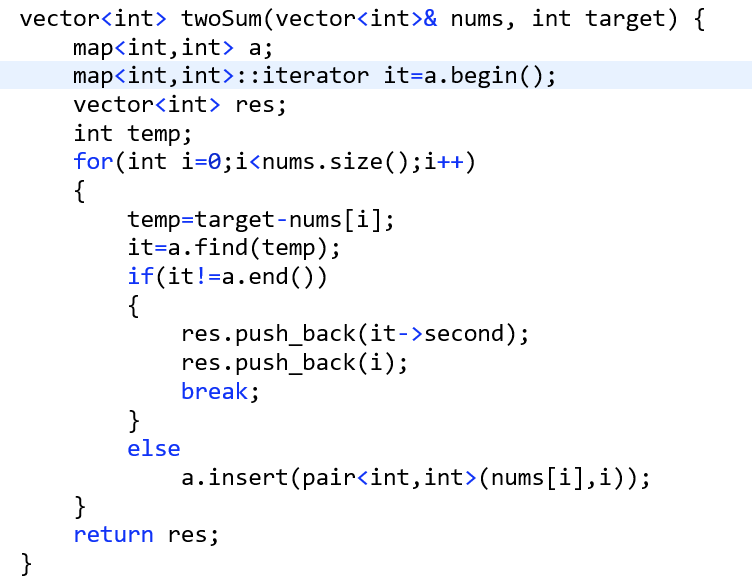
1. 如何判断是大端模式还是小端模式



## 30、如何判断是c++编译程序还是c编译程序

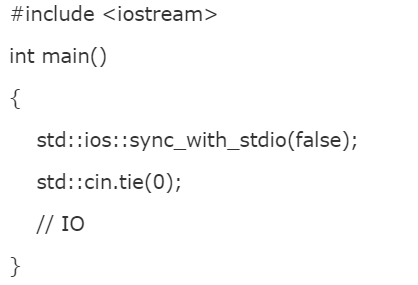


## 31、给定一个整数数组和一个目标值，找出数组中和为目标值的两个数。

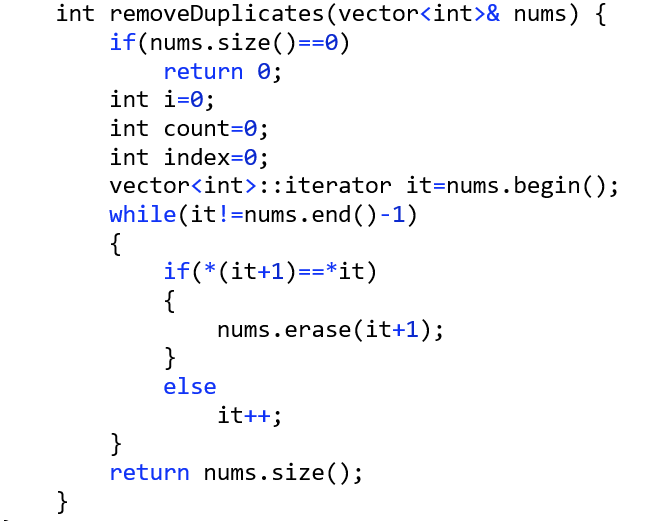


## 32、给定一个字符串 s，找到 s 中最长的回文子串

## 33、C++加速代码



## [原地](http://baike.baidu.com/item/%E5%8E%9F%E5%9C%B0%E7%AE%97%E6%B3%95" \t "https://leetcode-cn.com/problems/remove-duplicates-from-sorted-array/description/_blank)删除数组重复出现的元素，使得每个元素只出现一次



## 删除链表结点，不能从头开始访问，复杂度要求o（1）

## 写一个赋值构造函数

## 37、将链表中的每m个结点反转，不足m个就不反转