http://cv.qiaobutang.com/post/55b6ff400cf220407d66d332

网易有道2017内推选择题

1. 数据解析：ABCD

**A:XML数据结构根节点可以嵌套**

**B:** **JSONObjetWithData:options:error:** **使用缓冲区数据来解析**

**C:** **writeJSONObject:toSteam:options:error:** **使用流来解析**

**D：XML解析分为三种:SAX解析、PULL解析和DOM解析**

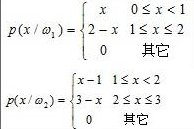
1. 大小为MAX的循环队列中，f为当前对头元素位置，r为当前队尾元素位置(最后一个元素的位置)，则任意时刻，队列中的元素个数为：**(r-f+MAX+1)%MAX**

解析：教材上说的 （也是大家记住的）求队列公式为： （rear-front+Max）%Max

该公式中的***rear指向队尾元素的下一个位置***，而本题中***r 指 向队尾元素位置***

可以带入一个小例子

1. 已知两个一维模式类别的类概率密度函数为:



先验概率P(ω1)=0.6;P(ω2)=0.4,则样本{x1=1.35,x2=1.45,x3=1.55,x4=1.65}各属于哪一类别?

解析：

概率问题基本上都是贝叶斯和全概率互相扯蛋,，他们之间往往可以通过条件概率建立联系。

本题中，要判断 xi 属于w1，还是w2，就是判断 p(w1 | xi)  和 p(w2 | xi)的大小关系。即在xi已经发生的情况下，xi 属于哪个类别（w1 ，w2）的可能性更大。比较后验概率 p(ω|x) , 哪个类的后验概率大 , 就属于哪个类。

后验概率 : p(ω|x)=p(x| ω )p( ω )/p(x), 分母 p(x ) 总是常数可以忽略 , 先验概率 p( ω ) 已知 , 计算类条件概率 p( x| ω ) , 即可得到后验概率

p(w1 | xi) = p(xiw1) / p(xi) = p(xi| w1) \* p(w1) / p(xi) = 0.6\*(2 - xi) / p(xi)   // 因为xi都在 （1，2)范围

p(w2 | xi) = p(xiw2) / p(xi) = p(xi| w2) \* p(w2) / p(xi) = 0.4\*(xi- 1) / p(xi)   // 因为xi都在 （1，2)范围

上面两等式相减，得：

delta = p(w1 | xi) - p(w2 | xi) = (1.6 - xi) / p(xi)

所以，在上诉样本中，大于1.6的，属于w2，小于1.6的，属于w1。

看了很多公司的概率题基本上都是在贝叶斯和全概率上面扯，掌握这个套路就行。

1. 程序员小李通过管道统计prog.c函数中for语句通过的次数，需要使用的指令分别是：grep、wc

解析：grep, Global Regular Expression Print, 使用正则表达式搜索文本 , 并把匹配的行打印出来 .

wc, word count, 统计指定文件中的字节数 , 字数 , 行数 , 并将统计结果显示输出 .

1. 请将下列构造函数补充完整，使得程序的运行结果是5

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include<iostream>  using namespace std;  class Sample{      public:          Sample(int x){              \_\_\_\_\_\_\_\_          }          ~Sample(){              if(p) delete p;          }          int show(){              return \*p;          }      private:          int\*p;  };  int main(){      Sample S(5);      cout<<S.show()<<endl;      return 0;  } |

A \*p=x;

B p=new int(x);

C \*p=new int(x);

D p=&x;

解析：

D ： p = &x;  表示的是，对x取地址，赋值给指针p，那么p 将指向 x 的那块内存空间，但是 x  是形式参数(也有人说是方法参数，都可以)，函数调用完了之后，内存就释放了（x的值和它所在的内存位置，都会在方法结束后失效），所以再返回 \*p（即取出那块内存空间的值），已经找不到了。所以错误。

A ：\*p = x；  表示的是 将 x 的值赋值给 P 所指向的空间，而p之前并没有指向任何地方，这个操作将是非法的。

C：\*p = new int(x)；   这个操作同 A 的结果一样。

B  ： p = new int(x);  new int(x) 新申请空间，调用完后不释放空间，所以将地址赋值给p 则p 指向了这段新申请内存空间，所以当做  \*p 返回时，即取出p所执行空间的值，所以会输出5

1. 变量z执行结果是（3）

|  |  |
| --- | --- |
| 1  2 | int a=5, b=6, c=12;  int z= (a \* b -c ) / a; |

解析：18/5 由于左右操作数均为int，输出结果为int，3.6向上取整为3.

1. 如下Java语句

|  |  |
| --- | --- |
| 1  2  3 | double x= 3.0;  int y=5;  x/=--y; |

执行后， x的值是（0.75）

解析：x/=--y相当于x=x/4，x为double类型，4转换为double类型，得到小数0.75

1. 设某棵二叉树的中序遍历序列为BADC，前序遍历序列为ABCD，则后序遍历该二叉树得到序列为（BDCA）.

解析：前序遍历：根节点-->左子树-->右子树

中序遍历：左子树 --> 根节点 ->右子树

后序遍历： 左子树 ->右子树 --> 根节点

1. 关于解释系统的叙述中，正确的是（AD）

A解释程序不是直接执行，而是转换成机器可识别码之后才能执行

B使用解释系统时会区分编译阶段和运行阶段

C目标程序可以脱离其语言环境独立执行，使用比较方便、效率较高

D一般来说，建立在编译基础上的系统在执行速度要优于建立在解释执行基础上的系统

解析：

A： 计算机不能直接理解任何除及其语言以外的语言，所以必须要把程序员所写的程序语言翻译成机器语言，计算机才能执行程序。将其他语言翻译成机器语言的工具，被称为编译器。 编译器翻译的方式有两种：一个是编译，一个是解释，相对应的是编译型语言和解释型语言。  
B：编译语言程序执行分为编译、执行两个阶段  
C：编译语言如JAVA是跨平台语言，编译一次可以在别的系统中不需再次编译直接执行

D选项：编译语言首先将源码编译成字节码或者编译成二进制码 ，C就是把源码编译成二进制码，Java编译成字节码。编译语言执行的效率往往比较高。因为它会对编译好的代码直接执行，而编译好的代码已经接近机器语言，无需再去做解释的工作。 解释性语言不同，在执行的时候才把源代码调到执行它的地方，比如是引擎或者解释器，调用一句执行一句。

目标程序是 编译系统生成的，解释系统不生成目标程序。

编译： 源代码->目标代码

解释：源代码->中间代码->目标代码

目标代码是机器可直接执行的代码

不管编译还是解释,都需要转为机器识别的才能执行, 只不过解释是靠虚拟机或者其他机制

脚本的特点是，脚本本身不编译为机器码。而是依托于寄主（虚拟机，脚本解释器等）。

其实真正起执行作用的是寄主。脚本命令寄主按照脚本的需求来执行操作。

而常规的编译型的代码，通过编译器编译成独立的可执行文件。可执行文件本身就包含了执行语句。它自己来执行自己所需的操作。

你可以简单这么理解：常规编译型的代码，是奔跑的人。而脚本是骑马的人，脚本解释器就是他的坐骑。真正跑起来的是马而不是人。

一般脚本执行效率会低一些，但开发起来容易一些。（只是一般情况）

1. 在一个10阶的B-树上，每个树根结点中所含的关键字数目最多允许为(9)个，最少允许为(4)个。

解析：关键字的个数n必须满足： [ceil(m / 2)-1]<= n <= m-1  （ceil为取上整）

<http://www.cnblogs.com/oldhorse/archive/2009/11/16/1604009.html>

**B树**

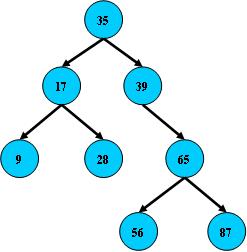
       即二叉搜索树：

       1.所有非叶子结点至多拥有两个儿子（Left和Right）；

       2.所有结点存储一个关键字；

       3.非叶子结点的左指针指向小于其关键字的子树，右指针指向大于其关键字的子树；

       如：



       B树的搜索，从根结点开始，如果查询的关键字与结点的关键字相等，那么就命中；

否则，如果查询关键字比结点关键字小，就进入左儿子；如果比结点关键字大，就进入

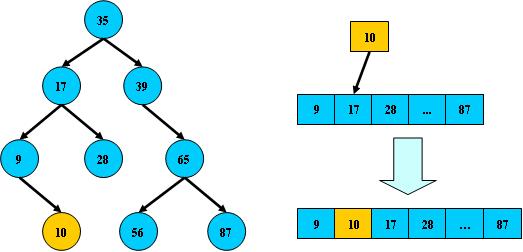
右儿子；如果左儿子或右儿子的指针为空，则报告找不到相应的关键字；

       如果B树的所有非叶子结点的左右子树的结点数目均保持差不多（平衡），那么B树

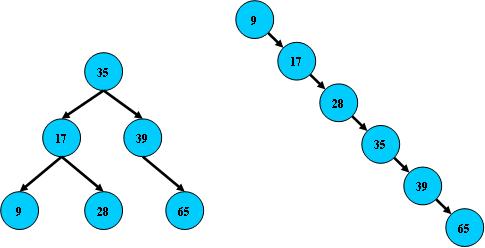
的搜索性能逼近二分查找；但它比连续内存空间的二分查找的优点是，改变B树结构

（插入与删除结点）不需要移动大段的内存数据，甚至通常是常数开销；

       如：



   但B树在经过多次插入与删除后，有可能导致不同的结构：



   右边也是一个B树，但它的搜索性能已经是线性的了；同样的关键字集合有可能导致不同的树结构索引；所以，使用B树还要考虑尽可能让B树保持左图的结构，和避免右图的结构，也就是所谓的“平衡”问题；

       实际使用的B树都是在原B树的基础上加上平衡算法，即“平衡二叉树”；如何保持B树结点分布均匀的平衡算法是平衡二叉树的关键；平衡算法是一种在B树中插入和删除结点的策略；

**B-树**

       是一种多路搜索树（并不是二叉的）：

       1.定义任意非叶子结点最多只有M个儿子；且M>2；

       2.根结点的儿子数为[2, M]；

       3.除根结点以外的非叶子结点的儿子数为[M/2, M]；

       4.每个结点存放至少M/2-1（取上整）和至多M-1个关键字；（至少2个关键字）

       5.非叶子结点的关键字个数=指向儿子的指针个数-1；

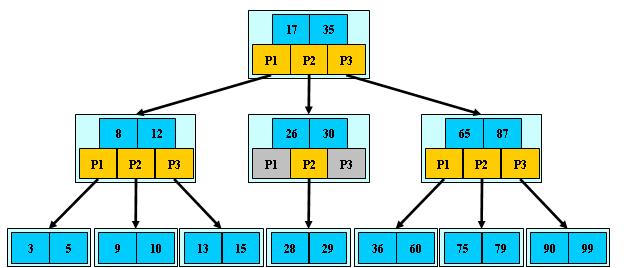
       6.非叶子结点的关键字：K[1], K[2], …, K[M-1]；且K[i] < K[i+1]；

       7.非叶子结点的指针：P[1], P[2], …, P[M]；其中P[1]指向关键字小于K[1]的

子树，P[M]指向关键字大于K[M-1]的子树，其它P[i]指向关键字属于(K[i-1], K[i])的子树；

       8.所有叶子结点位于同一层；

       如：（M=3）



       B-树的搜索，从根结点开始，对结点内的关键字（有序）序列进行二分查找，如果

命中则结束，否则进入查询关键字所属范围的儿子结点；重复，直到所对应的儿子指针为

空，或已经是叶子结点；

B-树的特性：

       1.关键字集合分布在整颗树中；

       2.任何一个关键字出现且只出现在一个结点中；

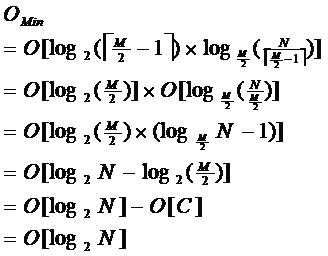
       3.搜索有可能在非叶子结点结束；

       4.其搜索性能等价于在关键字全集内做一次二分查找；

       5.自动层次控制；

       由于限制了除根结点以外的非叶子结点，至少含有M/2个儿子，确保了结点的至少

利用率，其最底搜索性能为：



       其中，M为设定的非叶子结点最多子树个数，N为关键字总数；

       所以**B-树的性能总是等价于二分查找（与M值无关）**，**也就没有B树平衡的问题**；

       由于M/2的限制，在插入结点时，如果结点已满，需要将结点分裂为两个各占

M/2的结点；删除结点时，需将两个不足M/2的兄弟结点合并；

**B+树**

       B+树是B-树的变体，也是一种多路搜索树：

       1.其定义基本与B-树同，除了：

       2.非叶子结点的子树指针与关键字个数相同；

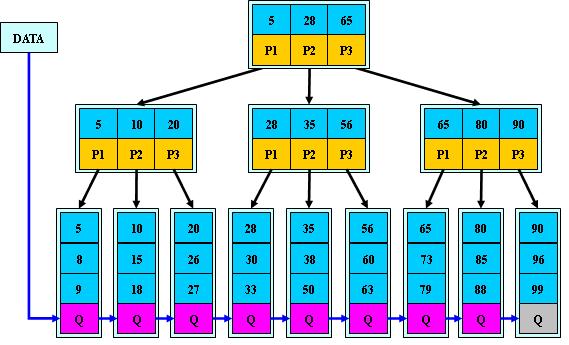
       3.非叶子结点的子树指针P[i]，指向关键字值属于[K[i], K[i+1])的子树

（B-树是开区间）；

       5.为所有叶子结点增加一个链指针；

       6.所有关键字都在叶子结点出现；

       如：（M=3）



   B+的搜索与B-树也基本相同，区别是**B+树只有达到叶子结点才命中（B-树可以在**

**非叶子结点命中）**，其**性能也等价于在关键字全集做一次二分查找**；

       B+的特性：

       1.所有关键字都出现在叶子结点的链表中（稠密索引），且链表中的关键字恰好

是有序的；

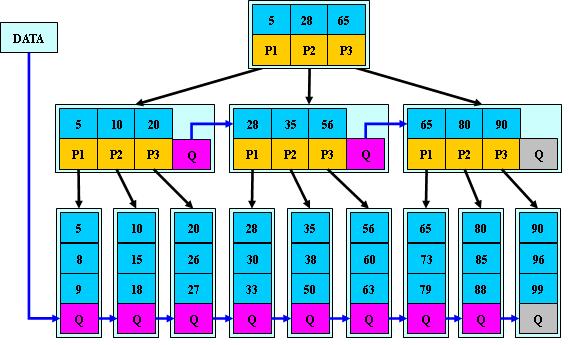
       2.不可能在非叶子结点命中；

3.非叶子结点相当于是叶子结点的索引（稀疏索引），叶子结点相当于是存储（关键字）数据的数据层；

       4.更适合文件索引系统；

**B\*树**

       是B+树的变体，在B+树的非根和非叶子结点再增加指向兄弟的指针；



**B\*树定义了非叶子结点关键字个数至少为(2/3)\*M，即块的最低使用率为2/3**

（代替B+树的1/2）；

       B+树的分裂：当一个结点满时，分配一个新的结点，并将原结点中1/2的数据

复制到新结点，最后在父结点中增加新结点的指针；B+树的分裂只影响原结点和父

结点，而不会影响兄弟结点，所以它不需要指向兄弟的指针；

       B\*树的分裂：当一个结点满时，如果它的下一个兄弟结点未满，那么将一部分

数据移到兄弟结点中，再在原结点插入关键字，最后修改父结点中兄弟结点的关键字

（因为兄弟结点的关键字范围改变了）；如果兄弟也满了，则在原结点与兄弟结点之

间增加新结点，并各复制1/3的数据到新结点，最后在父结点增加新结点的指针；

       所以，B\*树分配新结点的概率比B+树要低，空间使用率更高；

**小结**

       B树：二叉树，每个结点只存储一个关键字，等于则命中，小于走左结点，大于

走右结点；

       B-树：多路搜索树，每个结点存储M/2到M个关键字，非叶子结点存储指向关键

字范围的子结点；

       所有关键字在整颗树中出现，且只出现一次，非叶子结点可以命中；

       B+树：在B-树基础上，为叶子结点增加链表指针，所有关键字都在叶子结点

中出现，非叶子结点作为叶子结点的索引；B+树总是到叶子结点才命中；

       B\*树：在B+树基础上，为非叶子结点也增加链表指针，将结点的最低利用率

从1/2提高到2/3；

1. 下列程序编译时会出现错误，请根据行号选择错误位置( )

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | #include <iostream>  using namespace std;  class A{    int a1;  protected:    int a2;  public:    int a3;  };  class B: public A{    int b1;  protected:    int b2;  public:    int b3;  };  class C:private B{    int c1;  protected:    int c2;  public:    int c3;  };  int main(){    B obb;    C obc;    cout<<obb.a1;//1    cout<<obb.a2;//2    cout<<obb.a3;//3    cout<<obc.b1;//4    cout<<obc.b2;//5    cout<<obc.b3;//6    cout<<obc.c3;//7    return 0;  } |

正确答案: A D

A 1,2

B 2,5,7

C 3,4,7

D 4,5,6

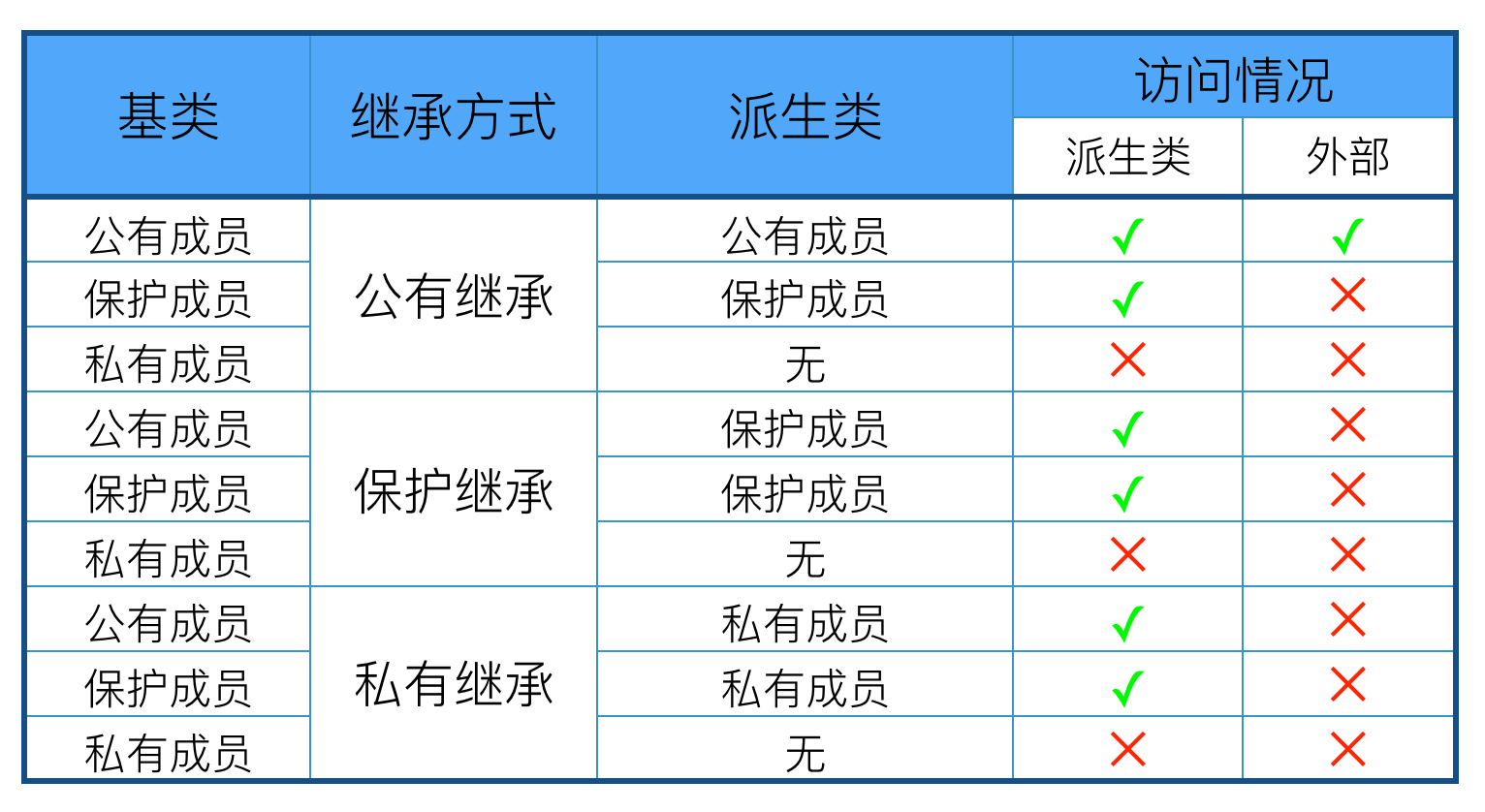
**解析：派生类可以继承定义在基类中的成员，但是派生类的成员函数不一定有权访问从基类继承而来的成员**，访问权限受下面因素影响。

* 继承方式；
* 基类成员的访问权限(即public/private/protected)。

继承有三种方式，即公有(Public)继承、私有(Private)继承、保护(Protected)继承。（私有成员不能被继承）

* 公有继承就是将基类的公有成员变为自己的公有成员，基类的保护成员变为自己的保护成员。
* 保护继承是将基类的公有成员和保护成员变成自己的保护成员。
* 私有继承是将基类的公有成员和保护成员变成自己的私有成员。

三种继承方式的比较:



**类的继承后方法属性变化：**

private 属性不能够被继承。

使用private继承，父类的protected和public属性在子类中变为private；

使用protected继承，父类的protected和public属性在子类中变为protected；

使用public继承，父类中的protected和public属性不发生改变;

**private, public, protected 访问标号的访问范围：**

private：只能由1.该类中的函数、2.其友元函数访问。

不能被任何其他访问，该类的对象也不能访问。

protected：可以被1.该类中的函数、2.子类的函数、以及3.其友元函数访问。

但不能被该类的对象访问。

public：可以被1.该类中的函数、2.子类的函数、3.其友元函数访问，也可以由4.该类的对象访问。

注：友元函数包括3种：设为友元的普通的非成员函数；设为友元的其他类的成员函数；设为友元类中的所有成员函数。

1. 一磁带机有9道磁道，带长700m，带速2m/s，每个数据块1k字节，块间间隔14mm。如果数据传输率为128000字节/秒,求记录位密度为（64000）字节/m.

解析：因为**数据传输率**（C）=**记录位密度**（D） x   **线速度**( V )

所以 D=C/V= 128000/2=64000

1. 以下几条路由，10.1.193.0/24,10.1.194.0/24,10.1.196.0/24,10.1.198.0/24，如果进行路由汇聚，则能覆盖这几条路由地址的是（C）

A 10.1.192.0/22

B 10.1.200.0/22

C 10.1.192.0/21

D 10.1.224.0/20

解析：

10.1.193.0的二进制：00001010.00000001.11000 001.00000000

10.1.194.0的二进制：00001010.00000001.11000 010.00000000

10.1.196.0的二进制：00001010.00000001.11000 100.00000000

10.1.198.0的二进制：00001010.00000001.11000 110.00000000

子网掩码为：11111111.11111111.11111 000.00000000 （十进制为255.255.248.0）即可覆盖。

子网掩码是用来划分主机号和网络号，上面的二进制中只有001， 010， 100， 110这三位以后不同。

取前面21位相同为网络号，后面11位为主机号。

故为：        00001010.00000001.11000 000.00000000 (10.1.192.0)

子网掩码为 11111111.11111111.11111 000.00000000 (255.255.248.0, 缩写为21)

网易2016研发工程师笔试题（二）

1. 设有两个事务T1,T2,其并发操作如下所示,下面评价正确的是()

步骤          T1         T2

1          读A=100

2                      读A=100

3.         A=A+10写回

4.                     A=A-10写回

正确答案: D

A 该操作不能重复读

B 该操作不存在问题

C 该操作读"脏"数据

D 该操作丢失修改

解析：**丢失修改**

下面我们先来看一个例子，说明并发操作带来的数据的不一致性问题。

考虑飞机订票系统中的一个活动序列:  
甲售票点（甲事务）读出某航班的机票余额A,设A=16.  
乙售票点（乙事务）读出同一航班的机票余额A,也为16.  
甲售票点卖出一张机票,修改余额A←A-1.所以A为15,把A写回数据库.  
乙售票点也卖出一张机票,修改余额A←A-1.所以A为15,把A写回数据库.

结果明明卖出两张机票，数据库中机票余额只减少1。

归纳起来就是：两个事务T1和T2读入同一数据并修改，T2提交的结果破坏了T1提交的结果，导致T1的修改被丢失。前文（2.1.4数据删除与更新）中提到的问题及解决办法往往是针对此类并发问题的。但仍然有几类问题通过上面的方法解决不了，那就是：

**不可重复读**

不可重复读是指事务T1读取数据后，事务T2执行更新操作，使T1无法再现前一次读取结果。具体地讲，不可重复读包括三种情况：  
事务T1读取某一数据后，事务T2对其做了修改，当事务1再次读该数据时，得到与前一次不同的值。例如，T1读取B=100进行运算，T2读取同一数据B，对其进行修改后将B=200写回数据库。T1为了对读取值校对重读B，B已为200，与第一次读取值不一致。  
事务T1按一定条件从数据库中读取了某些数据记录后，事务T2删除了其中部分记录，当T1再次按相同条件读取数据时，发现某些记录神密地消失了。  
事务T1按一定条件从数据库中读取某些数据记录后，事务T2插入了一些记录，当T1再次按相同条件读取数据时，发现多了一些记录。（这也叫做幻影读）   
**读"脏"数据**

读"脏"数据是指事务T1修改某一数据，并将其写回磁盘，事务T2读取同一数据后，T1由于某种原因被撤消，这时T1已修改过的数据恢复原值，T2读到的数据就与数据库中的数据不一致，则T2读到的数据就为"脏"数据，即**不正确的数据**。

产生**上述三类数据不一致性的主要原因是并发操作破坏了事务的隔离性**。并发控制就是要用正确的方式调度并发操作，使一个用户事务的执行不受其它事务的干扰，从而避免造成数据的不一致性。

1. 以下哪个不是与Mysql服务器相互作用的通讯协议(B)

A TCP/IP

B UDP

C 共享内存

D Socket

解析：MySQL实现了四种通信协议

1、**TCP/IP协议**，通常我们通过来连接MySQL，各种主要编程语言都是根据这个协议实现了连接模块

2、**Unix Socket协议**，这个通常我们登入MySQL服务器中使用这个协议，因为要使用这个协议连接MySQL需要一个物理文件，文件的存放位置在配置文件中有定义，值得一提的是，这是所有协议中最高效的一个。

3、**Share Memory协议**，这个协议一般人不知道，肯定也没用过，因为这个只有windows可以使用，使用这个协议需要在配置文件中在启动的时候使用–shared-memory参数，注意的是，使用此协议，一个host上只能有一个server，所以这个东西一般没啥用的，除非你怀疑其他协议不能正常工作，实际上微软的SQL Sever也支持这个协议

4、**Named Pipes协议**，这个协议也是只有windows才可以用，同shared memory一样，使用此协议，一个host上依然只能有一个server，即使是使用不同的端口也不行，Named Pipes 是为局域网而开发的协议。内存的一部分被某个进程用来向另一个进程传递信息，因此一个进程的输出就是另一个进程的输入。第二个进程可以是本地的（与第一个进程位于同一台计算机上），也可以是远程的（位于联网的计算机上）。正因为如此，假如你的环境中没有或者禁用TCP/IP环境，而且是windows服务器，那么好歹你的数据库还能工作。使用这个协议需要在启动的时候添加–enable-named-pipe选项。

1. CPU和两台输入/输出设备(I1,I2)多道程序设计环境下,同时有三个作业J1,J2,J3进行,这三个作业

使用CPU和输入/输出设备的顺序和时间如下所示:

J1:I2(35ms);CPU(15ms);I1(35ms);CPU(15ms);I2(25ms)

J2:I1(25ms);CPU(30ms);I2(35ms)

J3:CPU(30ms);I1(25ms);CPU(15ms);I1(15ms);

假定CPU,I1,I2都能并行工作,J1的优先级最高,J2次之,J3优先级最低,优先级高的作业可以抢占优先级低的作业的CPU,但不能抢占I1,I2,作业从J3开始到完成需要多少时间?

解析：图示



1. 文件exer1的访问权限为rw-r--r--,现要增加所有用户的执行权限和同组用户的写权限,下列哪个命令是对的？

正确答案: A   你的答案: A (正确)

chmod a+x,g+w exer1

chmod g+w exer1

chmod 765 exer1

chmod o+x exer1

解析：  
u-g-o分别表示用户－用户组－其他用户，用a表示所有的用户；即可以表示成为chmod a+x g+w exer1；  
用数字4-2-1分别表示rwx 则这里u:4+2+1=7，g:4+2+1=7，o:4+1=5。即同样可以表示成为chmod 775 exer1；这道题可能出现的歧义的地方在于，很多人认为增加同组用户的写权限应该为rw，也就是6，但是，前面还有一句是“增加所有用户的执行权限”，所以，同组用户应该为r+w+x=7

1. 新建一个管理员用户admin,需要使用的参数(useradd -u 0 -o admin)

解析：u表示创建用户.g表示创建用户组 -e指定账号的有效期限，缺省表示永久有效。-f指定在密码过期后多少天即关闭该账号。-p表示创建密码

-u 用户号 指定用户的用户号；因为系统用户的用户号为0，故指定用户号为0 如果同时有-o选项，则可以重复使用其他用户的标识号；因为系统本身存在用户号为0的系统用户，故应该使用该参数。

u表示创建用户.

g表示创建用户组

-e指定账号的有效期限，缺省表示永久有效。

-f指定在密码过期后多少天即关闭该账号。

-p表示创建密码

1. 设fp已定义,执行语句fp=fopen("file","w");后,以下针对文本文件file操作叙述的选项错误的是:

正确答案: A C D

可以随意读和写

只能写不能读

可以在原有内容后追加写

写操作结束后可以从头开始读

解析：



1. 下面()组命令,将变量count值赋值为1

正确答案: B

DIM@count=1 count=1

DECLARE @count SELECT @count=1

DECLARE count SELECT count=1

DIM @count SELECT @count=1

解析：  
DECLARE @count表示选定变量，SELECT @count＝1表示为count赋值；这里的赋值语句还可以使用SET @count＝1；但是select和set在赋值语句中有些区别：SELECT可以在一条语句里对多个变量同时赋值,而SET只能一次对一个变量赋值，即如果数据表中存在多个count变量，则不能使用SET赋值；

1. 下面重载乘法运算符的函数原型声明中正确的是:

正确答案: A B D

MyClass operator \*(double ,MyClass);

MyClass operator \*(MyClass ,MyClass);

MyClass operator \*(double ,double);

MyClass operator \*(MyClass ,double);

解析：对于一个运算符函数来说，它或者是类的成员，或者至少包含一个类类型的参数； 操作符重载允许将标准C++操作符用于类对象；

1. 依次把结点的关键字的值为50,30,20,150,130,40,80,70,85,15的记录插入到初始化为空的平衡二叉排序树中,在插入过程中平衡树条件如被破坏,则进行必要的调整,得到的平衡二叉排序树的深度为(4)(深度从1开始)

解析：   
可以用公式，n＋1对2取对数，结果向上取整；但是这里可以自己构建一颗平衡二叉树，然后计算出树的深度；

1. 截止JDK1.8版本,java并发框架支持锁包括?

正确答案: A B D   你的答案: A B C D E (错误)

读写锁

自旋锁

X锁

乐观锁

排他锁

解析：

1、自旋锁 ,自旋，jvm默认是10次吧，有jvm自己控制。for去争取锁

2、阻塞锁 被阻塞的线程，不会争夺锁。

3、可重入锁 多次进入改锁的域

4、读写锁

5、互斥锁 锁本身就是互斥的

6、悲观锁 不相信，这里会是安全的，必须全部上

7、乐观锁 相信，这里是安全的。

8、公平锁 有优先级的锁

9、非公平锁 无优先级的锁

10、偏向锁 无竞争不锁，有竞争挂起，转为轻量锁

11、对象锁 锁住对象

12、线程锁

13、锁粗化 多锁变成一个，自己处理

14、轻量级锁 CAS 实现

15、锁消除 偏向锁就是锁消除的一种

16、锁膨胀 jvm实现，锁粗化

17、信号量 使用阻塞锁 实现的一种策略

18、排它锁：X锁，若事务T对数据对象A加上X锁，则只允许T读取和修改A，其他任何事务都不能再对A加任何类型的锁，直到T释放A上的锁。这就保证了其他事务在T释放A上的锁之前不能再读取和修改A。

1. 下列哪个IP地址可以分配给一台计算机?

正确答案: D

256.1.3.4

197.3.11.0

199.5.89

11.15.33.235

解析：

互联网的网络地址分为A~E五类，其中

A类地址：0.0.0.0 ~ 127.255.255.255

主机号是后24位

B类地址：128.0.0.0 ~ 191.255.255.255

主机号是后16位

C类地址：192.0.0.0 ~ 223.255.255.255

主机号是后8位

D类地址：224.0.0.0 ~ 239.255.255.255

后28位为多播组号

E类地址：240.0.0.0 ~ 255.255.255.255

后27位待用

所以，A错。

主机号全为0的时候，表示一个网段

主机号全为1的时候，是一个指向网络的广播。关于广播的具体分类请看TCP/IP卷一的第12章。

所以B代表一个网段。

C的话，参照TCP/IP详解卷一第三章 IP:网际协议的图3-3，代表了一个直接相连的网络。

D作为一个A类地址，是可以分配给一台计算机的。

网易2016实习研发工程师选择题

1、以下关于mysql复制关系，描述错误的一项是？

正确答案: C   你的答案: C (正确)

A mysql支持master-slave复制，也支持master-master复制

B一般情况下，异步复制的性能比半同步复制好，但后者相对更为安全

C基于row格式的复制由于传输的数据量比较大，一般不建议使用

D在有多个slave参与的半同步复制中，master并不一定需要等待全部slave返回

解析：Mysql复制分成三步：   
(1) master将改变记录到二进制日志(binary log)中（这些记录叫做二进制日志事件，binary log events）；   
(2) slave将master的binary log events拷贝到它的中继日志(relay log)；

(3) slave重做中继日志中的事件，将改变反映它自己的数据。