STL 中 vector 的实现原理 (衍生：Map, Set 等实现原理)

参考答案

vector 的数据安排以及操作方式，与 **array** 非常相似。两者的唯一区别在于空间的运用的灵活性。**array** 是静态空间，一旦配置了就不能改变；要换个大（或小）一点的房子，可以，一切琐细都得由客户端自己来：首先配置一块新空间，然后将元素从旧址一一搬往新址，再把原来的空间释还给系统。**vector** 是动态空间，随着元素的加入，它的内部机制会自行扩充空间以容纳新元素。因此，**vector** 的运用对于内存的合理利用与运用的灵活性有很大的帮助，我们再也不必因为害怕空间不足而一开始要求一个大块头的 **array** 了，我们可以安心使用 **array**，吃多少用多少。

vector 的实现技术，关键在于其对大小的控制以及重新配置时的数据移动效率。一旦 **vector** 的旧有空间满载，如果客户端每新增一个元素，**vector** 的内部只是扩充一个元素的空间，实为不智。因为所谓扩充空间（不论多大），一如稍早所说，是”**配置新空间/数据移动/释还旧空间**“的大工程，时间成本很高，应该加入某种未雨绸缪的考虑。稍后我们便可看到 **SGI vector** 的空间配置策略了。

另外，由于 **vector 维护的是一个连续线性空间，所以 vector 支持随机存取** 。

注意：**vector** 动态增加大小时，并不是在原空间之后持续新空间（因为无法保证原空间之后尚有可供配置的空间），而是以原大小的两倍另外配置一块较大的空间，然后将原内容拷贝过来，然后才开始在原内容之后构造新元素，并释放原空间。因此， **对 vector 的任何操作，一旦引起空间重新配置，指向原 vector 的所有迭代器就都失效了** 。这是程序员易犯的一个错误，务需小心。

给定 N 张扑克牌和一个随机函数，设计一个洗牌算法

参考答案

```
1   void shuffle(int cards[],int n)
2   {
3           if(cards==NULL)
4                   return ;
5
6           srand(time(0));
7
8           for(int i=0;i<n-1;++i)
9           {
10                  //保证每次第 i 位的值不会涉及到第 i 位以前
11                  int index=i+rand()%(n-i);
12                  int temp=cards[i];
13                  cards[i]=cards[index];
14                  cards[index]=temp;
15          }
16  }
```

25 匹马，5 个跑道，每个跑道最多能有 5 匹马进行比赛，最少比多少次能比出前 3 名？前 5 名？

参考答案

找出前 3 名最少需要 7 场就可以确定。首先我们将 25 匹 马分成 5 组，分别为 ABCDE，5 组分别进行比赛决出各小组名次；接着让各小组第一进行比赛决出冠军，我们假设各小组第一分别是 A1, B1, C1, D1, E1，并且速度 A1 > B1 > C1 > D1 > E1；接着 2，3 名可以在一场比赛内决出，分别由 A2, A3, B1, B2, C1 参赛；这样总共进行了 5 + 1 + 1 = 7 场；找出前 5 名的思路和这个类似。

100 亿个整数，内存足够，如何找到中位数？内存不足，如何找到中位数？

参考答案

内存足够的情况： 可以使 用类似 quick sort 的思想进行，均摊复杂度为 O(n)，算法思想如下：

- 随机选取一个元素，将比它小的元素放在它左边，比它大的元素放在右边
- 如果它恰好在中位数的位置，那么它就是中位数，可以直接返回
- 如果小于它的数超过一半，那么中位数一定在左半边，递归到左边处理
- 否则，中位数一定在右半边，根据左半边的元素个数计算出中位数是右半边的第几大，然后递归 到右半边处理

内存不 足的情况：

方法 ： 分法

思路：一个重要的线索是，这些数都是整数。整数就有范围了，32 位系统中就是[--2^32, 2^32--1]，有了范围我们就可以对这个范围进行二分，然后找有多少个数 于 Mid,多少数大于 mid，然后递归， 和基于 quicksort 思想的第 k 大 方法类似

方法二：分桶法 思路：化大为小，把所有数划分到各个小区间，把每个数映射到对应的区间 里，对每个区间中数的 个数进行计数，数一遍各个区间，看看中位数落在哪个区间，若够小，使 用基于内存的算法，否则 继续划分

请简述智能指针原理，并实现一个简单的智能指针。

参考答案

智能指针是一种资源管理类，通过对原始指针进行封装，在资源管理对象进行析构时对指针指向的内存进行释放；通常使用引用计数方式进行管理，一个基本实现如下：

```
1  class Object;
2  class SmartPointer;
3
4  class Counter
5  {
6    friend  class SmartPointer;
7  public:
```

```
 8    Counter()
 9    {
10      ptr = NULL;
11      cnt =  0;
12    }
13    Counter(Object* p)
14    {
15      ptr = p;
16      cnt =  1;
17    }
18    ~Counter()
19    {
20      delete ptr;
21    }
22  private:
23    Object* ptr;
24    int cnt;
25  };
26
27  class SmartPointer
28  {
29  public:
30    SmartPointer(Object* p)
31    {
32      ptr_counter =  new Counter(p);
33    }
34    SmartPointer(const SmartPointer &sp)
35    {
36      ptr_counter = sp.ptr_counter;
37      ++ptr_count->cnt;
38    }
39    SmartPointer& operator=(const SmartPointer &sp)
40    {
41      ++sp.ptr_counter->cnt;
42      --ptr_counter->cnt;
43      if (ptr_counter->cnt ==  0)
44      {
45        delete ptr_counter;
46      }
47      ptr_counter = sp.ptr_counter;
48    }
49    ~SmartPointer()
50    {
51      - -ptr_counter->cnt;
```

```
52        if (ptr_counter->cnt ==  0)
53        {
54          delete ptr_counter;
55        }
56      }
57    private:
58      Counter* ptr_counter;
59
60    };
```

如何处理循环引用问题？

参考答案

刚刚研究过这个问题。

何为循环引用

如果有两个或者以上的对象，它们彼此引用，就会造成循环引用。如下面的例子

```
class Node {
   Node next ;;
}
Node a = new Node ();;
Node b = new Node ();;
a . next = b ;;
b . next = a ;;
```

代码中，a 对象引用了 b 对象，b 对象也引用了 a 对象，这种情况下 a 对象和 b 对象就形成了循环引用。

引用计数 GC 处理

什么是引用计数

引用计数是一种垃圾回收的形式，每一个对象都会有一个计数来记录有多少指向它的引用。其引用计数会变换如下面的场景

- 当对象增加一个引用，比如赋值给变量，属性或者传入一个方法，引用计数执行加 1 运算。
- 当对象减少一个引用，比如变量离开作用域，属性被赋值为另一个对象引用，属性所在的对象被回收或者之前传入参数的方法返回，引用计数执行减 1 操作。
- 当引用计数变为 0，代表该对象不被引用，可以标记成垃圾进行回收。

如何处理

实际上单纯的基于引用计数实现的计数器无法处理循环引用带来的问题。

CPython 的垃圾回收就是采用引用计数,采用引用计数的主垃圾回收器会清理垃圾，对于那些因为循环引用无法清理的对象，CPython 会不时启动一个辅助的基于引用遍历的垃圾回收器来清理它们。

引用遍历 GC 处理

什么是引用对象遍历

垃圾回收器从被称为 GC Roots 的点开始遍历遍历对象，凡是可以达到的点都会标记为存活，堆中不可到达的对象都会标记成垃圾，然后被清理掉。 GC Roots 有哪些

- 类，由系统类加载器加载的类。这些类从不会被卸载，它们可以通过静态属性的方式持有对象的引用。注意，一般情况下由自定义的类加载器加载的类不能成为 GC Roots
- 线程，存活的线程
- Java 方法栈中的局部变量或者参数
- JNI 方法栈中的局部变量或者参数
- JNI 全局引用
- 用做同步监控的对象
- 被 JVM 持有的对象，这些对象由于特殊的目的不被 GC 回收。这些对象可能是系统的类加载器，一些重要的异常处理类，一些为处理异常预留的对象，以及一些正在执行类加载的自定义的类加载器。但是具体有哪些前面提到的对象依赖于具体的 JVM 实现。

如何处理

基于引用对象遍历的垃圾回收器可以处理循环引用，只要是涉及到的对象不能从 GC Roots 强引用可到达，垃圾回收器都会进行清理来释放内存。

总结

基于引用计数的垃圾回收器无法处理循环引用导致的内存泄露问题，但是其在主流的 JVM 中很少，几乎所有的 JVM 都是采用引用对象遍历的方法，垃圾回收器都会处理循环引用潜在的问题。

请实现一个单例模式的类，要求线程安全

参考答案

在 C++之后，下面的实现是个线程安全的单例模式实现：

```
1  class Singleton
2  {
3  private:
4    Singleton();
5    Singleton(const Singleton &s);
6    Singleton& operator=(const Singleton &s);
7  public:
8    static Singleton* GetInstance()
9    {
10     static Singleton instance;
11     return &instance;
```

```
12      }
13  };
```

下面的结构体大小分别是多大（假设 32 位机器）？

```
 1  struct A {
 2  char a;
 3  char b;
 4  char c;
 5  };
 6
 7  struct B {
 8  int a;
 9  char b;
10  short c;
11  };
12
13  struct C {
14  char b;
15  int a;
16  short c;
17  };
18
19  #pragma pack(2)
20  struct D {
21  char b;
22  int a;
23  short c;
24  };
```

参考答案

3，8，12，8

引用和指针有什么区别？

参考答案

本质：引用是别名，指针是地址，具体的：

- 指针可以在运行时改变其所指向的值，引用一旦和某个对象绑定就不再改变
- 从内存上看，指针会分配内存区域，而引用不会，它仅仅是一个别名
- 在参数传递时，引 用会做类型检查，而指针不会
- 引用不能为空，指针可以为空

const 和 define 有什么区别？

参考答案

本质：define 只是字符串替换，const 参与编译运行，具体的：
• define 不会做类型检查，const 拥有类型，会执行相应的类型检查

• define 仅仅是宏替换，不占 用内存， 而 const 会占用内存

• const 内存效率更高，编译器通常将 const 变量保存在符号表中，而不会分配存储空间，这使得它成 为一个编译期间的常量，没有存储和读取的操作

define 和 inline 有什么区别？

参考答案

本质：define 只是字符串替换，inline 由编译器控制，具体的：
• define 只是简单的宏替换，通常会产生二义性；而 inline 会真正地编译到代码中
• inline 函数是否展开由编译器决定，有时候当函数太大时，编译器可能选择不展开相应的函数

malloc 和 new 有什么区别？

参考答案

1,malloc 与 free 是 C++/C 语言的标准库函数，new/delete 是 C++的运算符。它们都可用于申请动态内存和释放内存。

2,对于非内部数据类型的对象而言，光用 maloc/free 无法满足动态对象的要求。对象在创建的同时要自动执行构造函数，对象在消亡之前要自动执行析构函数。由于 malloc/free 是库函数而不是运算符，不在编译器控制权限之内，不能够把执行构造函数和析构函数的任务强加于 malloc/free。

3,因此 C++语言需要一个能完成动态内存分配和初始化工作的运算符 new，以一个能完成清理与释放内存工作的运算符 delete。注意 new/delete 不是库函数。

4,C++程序经常要调用 C 函数，而 C 程序只能用 malloc/free 管理动态内存。

5、new 可以认为是 malloc 加构造函数的执行。new 出来的指针是直接带类型信息的。而 malloc 返回的都是 void 指针。

C++中 static 关键字作用有哪些？

参考答案

1、隐藏：当同时编译多个文件时，所有未加 static 前缀的全局变量和函数都具有全局可见性。
static 可以用作函数和变量的前缀，对于函数来讲，static 的作用仅限于隐藏.
2、static 的第二个作用是保持变量内容的持久：存储在静态数据区的变量会在程序刚开始运行时就完成初始化，也是唯一的一次初始化。
共有两种变量存储在静态存储区：全局变量和 static 变量，只不过和全局变量比起来，static 可以控制变量的可见范围，
说到底 static 还是用来隐藏的。虽然这种用法不常见
3、static 的第三个作用是默认初始化为 0（static 变量）

4、C++中的作用

1）不能将静态成员函数定义为虚函数。

2）静态数据成员是静态存储的，所以必须对它进行初始化。 （程序员手动初始化，否则编译时一般不会报错，但是在 Link 时会报错误）

3）静态数据成员在<定义或说明>时前面加关键字 static。

C++中 const 关键字作用有哪些？？

参考答案

- 修饰变量
- 修饰成员函数，表示该成员函数不会修改成员变量

C++中成员函数能够同时用 static 和 const 进行修饰？

参考答案

否，因为 static 表 示该函数为静态成员函数，为类所有；而 const 是用于修饰成员函数的，两者相矛盾

下面三个变量分别代表什么含义？

const int* ptr;;

int const* ptr;;

int* const ptr;;

参考答案

前两个代表指向 const 变量的指针，即指针所指向的对象是 const 的，不能使用指针修改；最后一个代表 const 指针，即指针本身是 const 的，不能指向其他地址

C++中包含哪几种强制类型转换？他们有什么区别和联系？

参考答案

• reinterpret_cast: 转换一个指针为其它类型的指针。它也允许从一个指针转换为整数类型, 反之亦 然. 这个操作符能够在非相关的类型之间转换. 操作结果只是简单的从一个指针到别的指针的值的 二进制拷贝. 在类型之间指向的内容不做任何类型的检查和转换? class A{};;

class B{};;

A* a = new A;;

B* b = reinterpret_cast**(a);;**

• static_cast: 允许执行任意的隐式转换和相反转换动作（即使它是不允许隐式的）,例如：应用到类 的指针上, 意思是说它允许子类类型的指针转换为父类类型的指针(这是一个有效的隐式转换), 同 时, 也能够执行相反动作: 转换父类为它的子类

class Base {};;

class Derive:public Base{};;

Base* a = new Base;;

Derive *b = static_cast(a);;

• **dynamic_cast:** 只用于对象的指针和引用. 当用于多态类型时，它允许任意的隐式类型转换以及相 反过程. 不过，与 **static_cast** 不同，在后一种情况里（注：即隐式转换的相反过程）,**dynamic_cast** 会检查操作是否有效. 也就是说, 它会检查转换是否会返回一个被请求的有效的完整对象。检测在 运行时进行. 如果被转换的指针不是一个被请求的有效完整的对象指针，返回值为 **NULL**. 对于引用 类型，会抛出 **bad_cast** 异常

• **const_cast:** 这个转换类型操纵传递对象的 **const** 属性，或者是设置或者是移除,例如：

```
class C{};;
const C* a = new C;;
C *b = const_cast(a);;
```

下面两段代码的输出分别是什么？

```
1   class Base{
2           public:
3                 virtual  void Print()  const{
4                       cout <<  "Print in Base" << endl;
5                 }
6   };
7   class Derive::public base
8   {
9           public:
10                void Print()  const{
11                      cout <<  "Print in Derive" << endl;
12                }
13  };
14  void Print(const Base* base){
15          base->Print();
16  }
17  int main(){
18          Base b;
19          Derive d;
20          print(&b);
21          print(&d);
22          return 0;
23  }
```

```
1   class Base{
2           public:
3                 void Print()  const{
4                       cout <<  "Print in Base" << endl;
5                 }
6   };
7   class Derive::public base
8   {
9           public:
10                void Print()  const{
```

```
11                      cout << "Print in Derive" << endl;
12                  }
13  };
14  void Print(const Base* base){
15          base->Print();
16  }
17  int main(){
18          Base b;
19          Derive d;
20          print(&b);
21          print(&d);
22          return 0;
23  }
```

参考答案

考察对虚函数的基本理解

第一个：Print in Base, Print in Derive

第二哥：Print in Base, Print in Base

简述 C++虚函数作用及底层实现原理

参考答案

要点是要答出虚函数表和虚函数表指针的作用。C++中虚函数使用虚函数表和 虚函数表指针实现，虚函数表是一个类的虚函数的地址表，用于索引类本身以及父类的虚函数的地址，假如子类的虚函数重写了父类的虚函数，则对应在虚函数表中会把对应的虚函数替换为子类的 虚函数的地址；虚函数表指针存在于每个对象中（通常出于效率考虑，会放在对象的开始地址处）， 它指向对象所在类的虚函数表的地址；在多继承环境下，会存在多个虚函数表指针，分别指向对应 不同基类的虚函数表。

一个对象访问普通成员函数和虚函数哪个更快？

参考答案

访问普通成员函数更快，因为普通成员函数的地址在编译阶段就已确定，因此在访问时直接调 用对应地址的函数，而虚函数在调用时，需要首先在虚函数表中寻找虚函数所在地址，因此相比普 通成员函数速度要慢一些

在什么情况下，析构函数需要是虚函数？

参考答案

若存在类继承关系并且析构函数中需要析构某些资源时，析构函数需要是虚函数，否则当使用父类指针指向子类对象，在 delete 时只会调用父类的析构函数，而不能调用子类的析构函数，造成内存泄露等问题

内联函数、构造函数、静态成员函数可以是虚函数吗？

参考答案

都不可以。内联函数需要在编译阶段展开，而虚函数是运行时动态绑定的，编译时无法展开；构造函数在进行调用时还不存在父类和子类的概念，父类只会调用父类的构造函数，子类调用子类 的，因此不存在动态绑定的概念；静态成员函数是以类为单位的函数，与具体对象无关，虚函数是 与对象动态绑定的，因此是两个不冲突的概念；

构造函数中可以调用虚函数吗？

参考答案

可以，但是没有动态绑定的效果，父类构造函数中调用的仍然是父类版本的函数，子类中调用的仍然是子类版本的函数

简述 C++中虚继承的作用及底层实现原理?

参考答案

虚继承用于解决多继承条件下的菱形继承问题，底层实现原理与编译器相关，一般通过虚基类 指针实现，即各对象中只保存一份父类的对象，多继承时通过虚基类指针引用该公共对象，从而避 免菱形继承中的二义性问题。

1000 个灯围成一个环，初始状态是熄灭的，按一个灯，它以及它的左右两盏灯的状态会改变，问 如何让所有灯都亮？

参考答案

挨个按一遍。思路是每个灯只会被 3 个位置改变状态，挨个按一遍恰好每个位置被改变了奇数次 状态

n 条直线最多能将一个平面分成多少部分？

参考答案

num[n] = num[n -- 1] + n 思路：第 n 条直线总能和前 面 n--1 条直线形成 n--1 个交点，将第

n 条直线 分成 n 份，每一份会多分出一个平 面；num[n] = 1 + n*(n+1)/2;;

n 个平面最多能将一个空间切成多少部分？

参考答案

显然，当这 n 个平面满足以下条件时，所分割的部分数是最多的。 1、 这 n 个平面两两相交； 2、 没有三个以上的平面交于一点； 3、 这 n 个平面的交线任两条都不平行。 对于一般情况一下子不易考虑，我们不妨试着从简单的，特殊的情况入手来寻找规律。设 n 个 平面分空间的部分数为 an，易知 当 n=1 时，an=2 ； 当 n=2 时，an=4 当 n=3 时，an=8 当 n=4 时，情况有些复杂，我们以一个四面体为模型来观察，可知 an=15 ； 从以上几种情况，很难找出一个一般性的规律，而且当 n 的值继续增大时，情况更复杂，看来这样不行。那么，我们把问题在进一步简单化，将空间问题退化到平面问题：n 条直线最多可将平面分割成多少个部分？（这 n 条直线中，任两条不平行，任三条不交于同一点），设 n 条直线最多可将平面分割成 bn 个部分，那么 当 n=1,2,3 时，易知平面最多被分为 2，4，7 个部分。 当 n=k 时，设 k 条直线将平面分成了 bk 个部分，接着当添加上第 k+1 条直线时，这条直线与前 k 条直线相交有 k 个交点，这 k 个交点将第 k+1 条直线分割成 k 段，而每

一段将它所在的区域一分为二，从而增加了 K+1 个区域，故得递推关系式 b(k+1)=b(k)+(k+1)，即 b(k+1)--b(k)=k+1 显然当 k=1 时, b(1) =2,当 k=1,2,3.....n--1 时，我们得到 个式子： b(2)--b(1)=2;; b(3)--b(2)=3;; b(4)--b(3)=4;; b(5)--b(4)=5;; …… b(n)--b(n--1)=n;; 将这 n--1 个式子相加，得 b(n)=1/2*(n^2+n+2)，即 n 条直线最多可将平面分割成 1/2*(n^2+n+2) 个部分。 我们来归纳一下解决这个问题的思路：从简单情形入手，确定 b(k)与 b(k+1)的递推关系，最后得出结论。 现在，我们回到原问题，用刚才的思路来解决空间的问题，设 k 个平面将空间分割成 a(k)个部分，再添加上第 k+1 个平面，这个平面与前 k 个平面相交有 k 条交线，这 k 条交线，任意三条不共点，任意两条不平行，因此这第 k+1 个平面就被这 k 条直线分割成 b(k)个部分。 而这 b(k)个部分平面中的每一个，都把它所通过的那一部分空间分割成两个较小的空间。所以，添加上这第 k+1 个平面后就把原有的空间数增加了 b(k)个部分。由此的递推关系式 a(k+1)=a(k)+b(k)， 即 a(k--1)--a(k)=b(k) 当 k=1,2,3.....n--1 时，我们得到如下 n--1 个关系式 a(2)--a(1)=b(1);; a(3)--a(2)=b(2);; …… a(n)--a(n--1)=b(n--1);; 将这 n--1 个式子相加，得 a(n)=a(1)+(b(1)+b(2)+b(3)+.....+b(n--1)) 因为 b(n)= 1/2*(n^2+n+2),a(1)=2 所以 a(n)=2+{1/2*(1^2+1+2)+(2^2+2+2)+(3^2+3+2)+.....+((n--1^2)+(n--1)+2)} =(n^3+5*n+6)/6 问题的解：由上述分析和推导可知，n 个平面最多可将平面分割成 =(n^3+5*n+6)/6

两个机器人,初始时位于数轴上的不同位置。给这两个机器人输入一段相同 的程序,使得这两个机 器人保证可以相遇。程序只能包含“左移 n 个单位”、“右移 n 个单位”,条件判断语句 If,循环语句 while, 以及两个返回 Boolean 值的 函数 “在自己的起点处”和“在对方的起点处”。你不能使用其它的变量和 计数器，请写出该程序

参考答案

答案合理即可，下面是一个参考:

```
1   while (!at_other_robot_start())
2     move_right(1);
3   while (true)
4     move_right(2)
```

有 n 个人互相比赛（n 已知), 一个人输掉 4 次就出局(不能继续比赛),赢 7 次通过(可以继续比赛), 问最多通过人数?

参考答案

方程法，无人出局条件下，每个人最多送 3 次助攻，设 a 个胜者，b 个败者 7a <= 3n + b;; a + b = n;;

两个软硬程度一样的鸡蛋，它们在某一层摔下会碎，有个 100 层的建筑，要求最多用两个鸡蛋确 定鸡蛋安全下落的临界位置，给出临界位置？如果是 n 层楼，m 个鸡蛋，请给出确定临界位置的算法

参考答案

动态规划方法

100 层楼 2 个鸡蛋：f[n] = min(1 + max(i -- 1, f[n -- i])) i = 1……n

n 层楼 m 个鸡蛋：f[i, 0] = 0;; f[n,m] = min(1 + max(f[i -- 1, m -- 1], f[n -- i, m]) i = 1…n

n 个人，只有 1 个人是明星，明星所有人都认识，但明星不认识其他任何人，如何找到该明星？如果 n 很大很大，如果改进你的算法？

参考答案

线性扫描一遍，两两比较，每次比较都会排出一个人：若 a 认识 b，则 a 一定不是明星；若 a 不认 识 b，则 b 一定不是明星；n 很大的情况下可以采用分布式方法，每个机器处理一部分数据，最后每个 机器选出一个候选，归并

给 50 个硬币,面值可以不同,排成一排,两个人轮流取,只能从两端取,先取的人如何保证取到的币值大于等于另一个人

参考答案

看奇数位的数的和以及偶数位的数的和哪个大，若奇数位的大选第一个，否则选最后面的（这里假设奇数位和偶数位的和不同）若出现相同，可以采用动态规划的思路来做

一个绳子从一头开始烧是 1 小时，要求想办法测出 45 分钟。

参考答案

用甲乙两根绳.先同时点燃甲的两端以及乙的一端，当甲烧完后过去了 30 分钟,因为是两头烧,所以就是 30 分钟.而乙已经烧了 30 分钟,还有 30 分钟的量,此时再点燃乙的另一端.双管齐下,剩下的 30 分钟只烧了 15 分钟,加上一开始已经烧了 30 分钟,共 45 分钟

100 个囚犯从前往后坐成一列。坐在最后面的那个囚犯能够看到其余 99 个囚犯，坐在最前面的 那个囚犯啥也看不见。看守给每个囚犯戴上一顶黑色的或者白色的帽子。然后，看守会从后往前依 次叫这些囚犯猜测自己头顶上的帽子的颜色。如果哪个囚犯猜对了，他就自由了。坐在前面的每一 个囚犯都可以听到后面的囚犯的猜测。如果这 100 个囚犯事先可以商量好一种策略，那么最理想的策 略是什么？

参考答案

100 个囚犯从前往后坐成一列。坐在最后面的那个囚犯能够看到其余 99 个囚犯，坐在最前面的那个囚犯啥也看不见。看守给每个囚犯戴上一顶黑色的或者白色的帽子。然后，看守会从后往前依次叫这些囚犯猜测自己头顶上的帽子的颜色。如果哪个囚犯猜对了，他就自由了。坐在前面的每一个囚犯都可以听到后面的囚犯的猜测。如果这 100 个囚犯事先可以商量好一种策略，那么最理想的策略是什么？

囚犯们可以乱猜一通，最坏情况下所有人都猜错，平均下来则会有 50 个人猜对。这个题有趣的地方就在于，100 个囚犯事先可以商量一种策略，也就是说坐在后面的囚犯可以用他的猜测给坐在前面的囚犯透露一些信息。很显然，坐在最后面的囚犯是不可能保证自己猜对的，他猜黑猜白都只有一半的几率猜对，似乎没什么区别；但囚犯可以事先约定好一种暗号，即最后一个囚犯猜黑表示什么意思，猜白表示什么意思。比如，最后一个囚犯可以猜测和他前面的囚犯的帽子一样的颜色，这就相当于用他的猜测告诉了他前面那个囚犯该猜什么，于是坐倒数第二的囚犯可以保证被释放；此时，坐在倒数第三个位置上的囚犯面对与刚才坐最后的囚犯相同的处境，他同样可以用他的猜测提示出他前面那个人的帽子颜色。这样下去，可以保证至少 50 个人猜对，平均情况则有 75 个人猜对。这不是最佳的策略。

不可思议的是，最佳策略可以保证，除了坐在最后面的囚犯以外，其余 99 个囚犯都能猜对。

你能想出这样的策略是什么吗？继续看下去前不妨先想一下。

前面那种策略的问题在于，坐在最后面的那个人透露出的信息不多。他完全可以透露出与全局相关的一些信息，因此以后所有的人都可以用这条信息。比如，他可以数一数他前面 99 个人一共有多少顶白帽子，并约定他猜“黑”表示他前面共有偶数顶白帽，他猜“白”表示他前面共有奇数顶白帽。坐倒数第二的那个人也数一数他前面 98 个人的白帽子个数：如果他数出来的个数与先前透露出的个数一奇一偶，则他自己肯定戴的是白帽子；如果他数出来的和先前透露的结果奇偶性相同，则他自己戴的肯定是黑帽子。这样，坐倒数第二的保证可以猜对了。那接下来咋办呢？不要忘了，其他囚犯能听到刚才那人猜的是什么，并且知道他的猜测保证是对的。这相当于每个人不仅能看到坐他前面的所有人的帽子颜色，还知道他背后那些人的帽子颜色，结合最初的那个奇偶性信息，接下来的每一个人都可以猜出自己脑袋上的帽子颜色。这样下去，至少 99 个囚犯可以保证被释放。这种策略显然是最佳的，不可能再有什么策略能保证所有人都被释放，因为至少坐最后的那个人不可能保证自己猜对。

如何等概率地从 n 个数中随机抽出 m 个数？

上题中如果 n 的大小不确定（可以认为是 个数据流），如何做？

参考答案

每个位置 i 以(m -- k)/(n -- i + 1)的概率决定当前数是否选，k 为前面已经抽出的数的个数蓄水池采样法

给定一个能够生成 0,1 两个数的等概率随机数生成器”，如何生成 个产生 0,1,2,3 的等概率随机数生成器?

和上题类似，如何用 rand7 生成 rand9?

参考答案

将两个 0,1 随机生成器级联，每次产生两个数，则可能的结果有(0,0), (0,1), (1,0), (1,1)，分别映 射到 0, 1, 2, 3 即可

两个 rand7 可以产生 49 种可能，扔掉后面的 4 种，保留前 45 个，并平均分成 9 份，每次产生一个结果时，假如没落在对应区间中就扔掉，否则根据落在哪个区间判断是 0----8 中哪个

有一枚硬币,以 p 的概率产生正面，以 1--p 的概率产生背面，如何利用它产生个 0.5 概率的生成器？

参考答案

将两枚硬币级联，只保留“正反”，“反正”两种结果，其他两种结果扔掉

A,B,C 三人轮流扔硬币，第一个扔到正面的人算赢，问三个人赢的概率分别为多大？

参考答案

P(B) = 1/2*P(A);; P(C) = 1/4 * P(A);; P(A) + P(B) + P(C) = 1;;

得 P(A) = 4/7, P(B) = 2/7, P(C) = 1/7

A 有 n 个硬币,B 有 n+1 个硬币,谁丢的正面多谁赢,问 A 不输的概率？

参考答案

可前 n 轮，有 3 种情况，设 P(A>B) = x, P(A == B) = y，由对称性 P(A<B) = x，则有 2x + y = 1
现在来看 B 扔最后一个硬币的情况：

- **假如之前 A>B，则无论怎么扔，A 都不会输，最多平**
- **如果 A==B,则 B 扔了正面，A 才会输，这是 0.5y**
- **如果 A<B，则无论 B 怎么扔，A 都输,所以是 x**

所以 A 输的概率是：x + 0.5y = 0.5 * (2x + y) = 0.5，A 不输的概率是 1 -- 0.5 = 0.5

一个机器人在原点，右边有一个距离为 k 的点，机器人以 p 的概率右移一步， 1--p 概率左移一步， 问经过 M 步机器人处于 k 点的概率?

参考答案

k 步右移，剩下的 M -- k 步一半左移一半右移，所以 M<k 和(M -- k)%2 == 1 的情况概率为 0，其他情 况就是 M 中选 k + (M--k)/2 步的概率

扔硬币直到连续两次出现正面，求扔的期望次数

参考答案

假设期望次数是 E，我们开始扔，有如下几种情况：

- 扔到的是反面，那么就要重新仍，所以是 0.5*(1 + E)
- 扔到的是正面，再扔一次又反面了，则是 0.25*(2 + E)
- 扔到两次，都是正面，结束，则是 0.25*2

所以递归来看 E = 0.5*(1 + E) + 0.25*(2 + E) + 0.25*2，解得 E = 6

同样可以实现互斥，互斥锁和信号量有什么区别？

参考答案

信号量是一种同步机制，可以当作锁来用，但也可以当做进程/线程之间通信使用，作为通信使用时不一定有锁的概念；互斥锁是为了锁住一些资源，是为了对临界区做保护

请用普通的互斥锁编程实现一个读写锁

参考答案

下面是可参考的伪代码:

```
count_mutex = mutex_init();
write_mutex = mutex_init();
read_count =  0;

void read_lock {
  lock(count_mutex);
  read_count++;
  if (read_count ==  1) {
    lock(write_mutex);
  }
  unlock(count_mutex);
```

```
12  }
13
14  void read_unlock {
15    lock(count_mutex);
16    read_count--;
17    if (read_count ==  0) {
18      unlock(write_mutex);
19    }
20    unlock(count_mutex);
21  }
22
23  void write_lock {
24    lock(write_mutex);
25  }
26
27  void write_unlock {
28    unlock(write_mutex);
29  }
```

编程实现三个线程 ABC，并让它们顺次打印 ABC

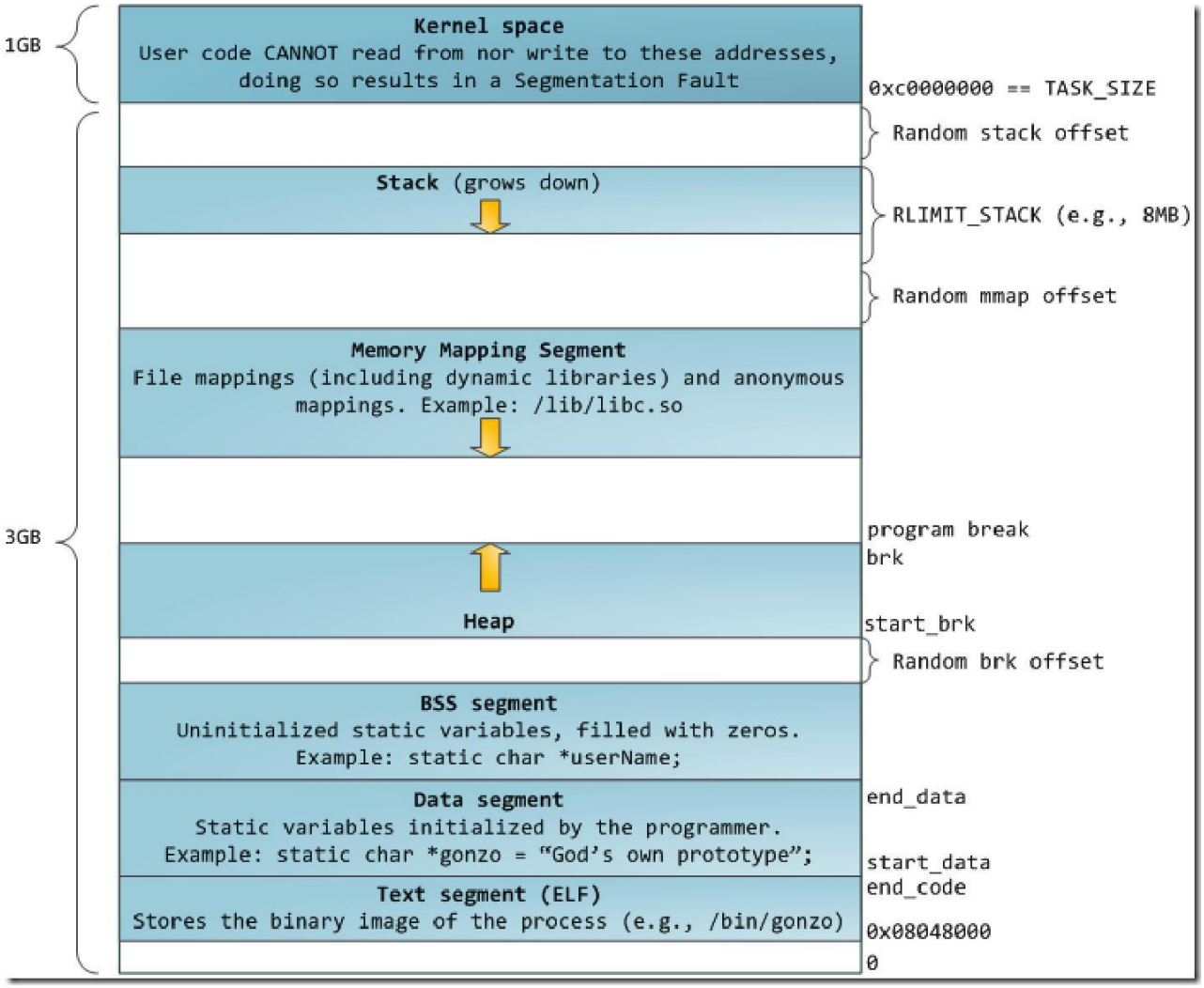
参考答案

思路：设置三个信号量：S1, S2, S3，S2 由 S1 post，S3 由 S2post, S1 由 S3 post，由 A 线程先开 始打印，其他线程必然在等待信号量，所以三个线程 一定会按照信号量的顺序来打印

简述 Linux 进程内存空间分为哪几个段？作用分别是什么？

参考答案

参考这个图



简述 Linux 内存分配----伙伴系统 原理

参考答案

伙伴系统，其思想是：把内存块分成不同的组(1,2,4,8,16,32....)；分配内存时找到能够满足条件 的最小的块；如果找不到，就找大的块，然后一分为 2，分配一块，留一块；回收时：如果有相邻的同样大小的块，则合并

简述 Malloc 实现原理

参考答案

可以基于伙伴系统实现，也可以使用基于链表的实现

- 将所有空闲内存块连成链表，每个节点记录空闲内存块的地址、大小等信息
- 分配内存时，找到大小合适的块，切成两份，一分给用户，一份放回空闲链表
- free 时，直接把内存块返回链表
- 解决外部碎片：将能够合并的内存块进行合并

使用 mmap 读写文件为什么比普通读写函数要快？

参考答案

mmap 函数：可以将文件映射到内存中的一段区域，普通函数读写文件：用户空间 buffer 内核空间 buffer 磁盘；mmap 映射之后：用户空间 buffer 进程内存空间，省掉了拷贝到内核空间的时间?

Linux 中如何实现 Signal?

参考答案

基于软中断，不同 Signal 对应不同中断处理函数

设计一个抽象类，使得它可以完成有序数组归并的任务

参考答案

设计合理即可，下 面是一个参考代码

```
1  class Sequence {
2  public:
3    virtual Object* next() =   0;
4    virtual bool hasNext() =   0;
5    virtual Object* top() =   0;
6    virtual add(Object* obj) =  0;
7  };
8
9  class Object {
10 public:
11   virtual bool lessThan(Object* another);
12 };
```

设计一个多终端日志打印的接口，使得它可以动态支持不同终端的日志打印

参考答案

设计合理即可，关键点是利用动态绑定实现接口与实现的分离，下面是一个参考答案:

```
1   class BaseTerminal {
2   public:
3     virtual  void PrintLog(const string& message)  const =  0;
4   };
5
6   class Screen:  public BaseTerminal {
7   public:
8     void PrintLog(const string& message)  const =  0;
9   };
10
11  class Network:  public BaseTerminal {
```

```
12  public:
13    void PrintLog(const string& message)  const =  0;
14  };
15
16  void LogPrint(const BaseTerminal &terminal);
```

设计并实现一个 LRU Cache

参考答案

下面是一个参考思路

- 重要数据结构：key--value 存储、LRU 存储；
- key--value 存储：hash_table/map，LRU：链表，因为可以快速实现增加、删除
- 如何更新 Cache: 找到 key 在链表中的位置，删除并将它插到表头，同时更新 key 到链表位置的映射
- 快速找到最不常访问的元素：链表尾

设计一个数据结构，能够支持插入、删除、返回最大值、最小值、随机返回一个数的操作

参考答案

设计合理即可，下面是一个参考思路：

- 插入、删除、最大、最小：使用 set 实现，复杂度 O(logn)
- 如何实现 random 使用数组，将所有数据放入数组中，random 时随机返回数组元素
- 记录每个元素在数组中的下标 • 删除时首先将对应元素和最后一个元素交换，删除最后一个元素 复杂度 O(1)

设计一个 Query suggestion 的服务

参考答案

设计合理即可，下面是一个参考思路：

- 使用 Trie 树记录每个 Query 出现的频次（或其他权值）
- 在每个 Trie 树的节点设置一个最小堆，记录以当前节点的字符串为前缀的 query 中出现 topK 次数的 query
- 更新 query 频次时，沿着 trie 树中的路径更新路径上节点中的最小堆
- 用户输入 query 时，直接返回对应 trie 树节点的最小堆中 topK 的 query 即可

什么是双数组 Trie 树？它的实现原理是什么？？

参考答案

可参考这篇介绍文章：http://linux.thai.net/~thep/datrie/datrie.html

设计 qps (query per sec)函数，用它控制 api 调用，使得 api n 毫秒内只能被调用 m 次?

参考答案

设计合理即可，下面是一个参考思路：

- 维护一个窗口，窗口有左右两个边界；窗口内为从最后一次访问开始向前 n 毫秒所有的访

问?

- 当新来一个访问，更新窗口右边界，打新的时间戳；向右移动窗口左边界，将距当前 n 毫秒外的访 问删除
- 统计次数看是否满足<= m 次?

如何设计一个短网址服务系统？

参考答案

设计合理即可，实现思路：将 url 哈希到一个唯一的数值，将这个数值转化为一个字符串；另外还需要考虑系统负载等因素

如何设计一个网页爬虫系统？

参考答案

设计合理即可，实现思路：使用 bfs 算法进行网站爬取；使用 master 节点作为控制节点控制 work 节点进行网站爬取；使用分布式队列做任务调度；使用 key--value 存储（如 redis) 做网页判重

给一个超过 100G 大小的 log file, log 中存着 IP 地址, 设计算法找到出现次数最多的 IP 地址？

与上题条件相同，如何找到 top K 的 IP？如何直接用 Linux 系统命令实现？

参考答案

Hash 分桶法：

- 将 100G 文件分成 1000 份，将每个 IP 地址映射到相应文件中：file_id = hash(ip) % 1000
- 在每个文件中分别求出最高频的 IP，再合并 Hash 分桶法：
- 使用 Hash 分桶法把数据分发到不同文件
- 各个文件分别统计 top K
- 最后 Top K 汇总

Linux 命令，假设 top 10：sort log_file | uniq --c | sort --nr k1,1 | head --10

给定 100 亿个整数，设计算法找到只出现一次的整数

参考答案

Hash 分桶法，将 100 亿个整数映射到不同的区间，在每个区间中分别找只出现一次的整数

给两个文件，分别有 100 亿个整数，我们只有 1G 内存，如何找到两个文件交集

参考答案

关键点：扫描每个整数是否出现过，如何节省内存？使用 bitmap

1 个文件有 100 亿个 int，1G 内存，设计算法找到出现次数不超过 2 次的所有整数?

参考答案

Bitmap 扩展： 用 2 个 bit 表 示状态，0 未出现过，1 出现过 1 次，2 出现了 2 次或以上

给两个文件，分别有 100 亿个 query，我们只有 1G 内存，如何找到两个文件交集？分别给出精确 算法和近似算法?

参考答案

精确算法：Hash 分桶法

- 将两个文件中的 query hash 到 N 个小文件中，并标明 query 的来源
- 在各个小文件中找到重合的 query
- 将找到的重合 query 汇总 近似算法：BloomFilter

如何扩展 BloomFilter 使得它支持删除元素的操作？

参考答案

将 Bloomfilter 中的每一位扩展为一个计数器，记录有多少个 hash 函数映射到这一位；删除的时候，只有当“引用计数”变为 0 时，才真正将该位置 0

如何扩展 BloomFilter 使得它支持计数操作？

参考答案

将 Bloomfilter 中的每一位扩展为一个计数器，每个输入元素都要把对应位置加 1，从而支持计数 操作。计数个数为，所有映射到的位置中计数的最小值

给上千个文件，每个文件大小为 1K—100M。给 n 个词，设计算法对每个词找到所有包含它的文件，你只有 100K 内存

参考答案

使 用 trie 树即可

有一个词典，包含 N 个英文单词，现在任意给一个字符串，设计算法找出包含这个字符串的所有英文单词

参考答案

一个可行的思路：给输入字符串，利 用字母建立倒排索引，索引中存储该字母 出现在哪

个单词以及在单词中位置；查询时，利用倒排找到所有的单词，并求交集并且位置要连续