第⼀讲：浅谈国内笔试⾯试风格及准备⽅法

1.实现一个Memcpy函数

#include <iostream>

using namespace std;

void \* My\_memcpy(void \* dst, const void \* src, size\_t count)

{

void \* ret = dst;

if(dst == NULL || src == NULL || count <= 0)

{

return NULL;

}

while(count--)

{

\*(char\*)dst = \*(char\*)src;

dst = (char\*)dst + 1;

src = (char\*)src + 1;

}

return dst;

}

int main()

{

char \* a = "Hello World!";

char b[20];

My\_memcpy(b, a+5,5);

b[5] = '\0';

cout << b << endl;

return 0;

}

strcpy和memcpy主要有以下3方面的区别。

1）、复制的内容不同。strcpy只能复制字符串，而memcpy可以复制任意内容，例如字符数组、整型、结构体、类等。

2）、复制的方法不同。strcpy不需要指定长度，它遇到被复制字符的串结束符"\0"才结束，所以容易溢出。memcpy则是根据其第3个参数决定复制的长度。

3）、用途不同。通常在复制字符串时用strcpy，而需要复制其他类型数据时则一般用memcpy

2.STL中vector的实现原理 (衍⽣： Map, Set等实现原理)

1）、Vector是顺序容器，是一个动态数组，支持随机存取、插入、删除、查找等操作，在内存中是一块连续的空间。在原有空间不够情况下自动分配空间。vector随机存取效率高，但是在vector插入元素，需要移动的数目多，效率低下。

2）、Map是关联容器，以键值对的形式进行存储，方便进行查找。关键词起到索引的作用，值则表示与索引相关联的数据。以红黑树的结构实现，插入删除等操作都在O(logn)时间内完成

3）、Set是关联容器，set中每个元素只包含一个关键字。set支持高效的关键字查询操作——检查一个给定的关键字是否在set中。set也是以红黑树的结构实现，支持高效插入、删除等操作

3.给定N张扑克牌和一个随机函数，设计一个洗牌算法

       假定Ｎ=54，首先，我们有一个随机函数发生器，能够产生1-54之间的随机数，如何保证抽第一张牌是54中可能，抽第二张牌是53中可能，……

       可以这样做，假设扑克牌是一个54维的数组card， 我们要做的就是从这个数组中随机取一个元素，然后在剩下的元素里再随机取一个元素… 这里涉及到一个问题，就是每次取完元素后，我们就不会让这个元素参与下一次的选取。

         int index=i+rand()%(n-i);//保证前面已经确定的元素不会参加下面的选取

       我们要实现的目的是以等概率的方式将这54个数随机打乱排列，因此，可以这样处理：

       第一次抽牌在初始54张牌中，将 随机产生的牌x，与第一个元素互换，

       第二次抽牌在剩下的53张牌中，将 随机产生的牌y，与第二个元素互换，

       void shuffle(int cards[],int n)

       {

          if(cards==NULL)

         {

                     return ;

                 }

         srand(time(0));

         for(int i=0;i<n-1;++i)

        {

     //保证每次第i位的值不会涉及到第i位以前

     int index=i+rand()%(n-i);//保证前面已经确定的元素不会参加下面的选取

     int temp=cards[i];

     cards[i]=cards[index];

     cards[index]=temp;

        }

        }

4 .25匹马， 5个跑道，最少比多少次能比出前3名？前5名？

前3名最少需要比7次，理由如下：

       由于共有25匹马和5个赛道，故可以分成五组进行比赛，假设第一小组是1~5，第二组是6~10，第三组是11~15，第四组是16~20，第五组是21~25。其优胜马的编号分别是1,6,11,16,21，小组第二编号分别是2,7,12,17,22。可知只需要求前三名，只需将优胜的马再进行一次比赛，假设得出第一为1号，而之前可知16,21分别为其小组第一，小组内其他马跑的慢于它们，故在取前三时不考虑。而后只需将2,3,6,11,16进行比赛即可决出前三名。

前5名最少需要比8次，理由如下：

      由于共有25匹马和5个赛道，故可以分成五组进行比赛，假设第一小组是1~5，第二组是6~10，第三组是11~15，第四组是16~20，第五组是21~25。其优胜马的编号分别是1,6,11,16,21，小组第二编号分别是2,7,12,17,22。可知只需要求前五名，需将优胜的马再进行一次比赛，假设得出第一为1号。将2号和6,11,16,21号进行比赛跑的最快的是第二快的马，假设2号是第二快的马21号是跑的最慢的马，则21号马所在组不再考虑。将3,4号马和6,11,16号马进行比赛（3,4号第一名所在组的下一匹马和第二名所在组的下一匹马），由于只需要三个名额，故跑的最慢的两匹马和其组全部淘汰。即决出了前五名。

5.进程和线程有什么区别？

      进程是一个具有独立功能的程序关于某个数据集合的一次运行活动。它可以申请和拥有系统资源，是一个动态的概念，是一个活动的实体。它不只是程序的代码，还包括当前的活动，通过程序计数器的值和处理寄存器的内容来表示。

      进程是一个“执行中的程序”。程序是一个没有生命的实体，只有处理器赋予程序生命时，它才能成为一个活动的实体，我们称其为进程。

      通常在一个进程中可以包含若干个线程，它们可以利用进程所拥有的资源。在引入线程的操作系统中，通常都是把进程作为分配资源的基本单位，而把线程作为独立 运行和独立调度的基本单位。由于线程比进程更小，基本上不拥有系统资源，故对它的调度所付出的开销就会小得多，能更高效的提高系统内多个程序间并发执行的 程度。

      线程和进程的区别在于，子进程和父进程有不同的代码和数据空间，而多个线程则共享数据空间，每个线程有自己的执行堆栈和程序计数器为其执行上下文。多线程主要是为了节约CPU时间，发挥利用，根据具体情况而定。线程的运行中需要使用计算机的内存资源和CPU。

      线程与进程的区别归纳：

 a.地址空间和其它资源：进程间相互独立，同一进程的各线程间共享。某进程内的线程在其它进程不可见。

   b.通信：进程间通信IPC，线程间可以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。

    c.调度和切换：线程上下文切换比进程上下文切换要快得多。

      d.在多线程OS中，进程不是一个可执行的实体。

进程是具有一定独立功能的程序关于某个数据集合上的一次运行活动,进程是系统进行资源分配和调度的一个独立单位。线程是进程的一个实体,是CPU调度和分 派的基本单位,它是比进程更小的能独立运行的基本单位。线程自己基本上不拥有系统资源,只拥有一点在运行中必不可少的资源(如程序计数器,一组寄存器和 栈),但是它可与同属一个进程的其他的线程共享进程所拥有的全部资源。

6 .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？

       内存足够的情况： 可以使⽤用类似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：

无重复数字：

　　bitmap方法

有重复数字：

　   既然要找中位数，很简单就是排序的想法。那么基于字节的桶排序是一个可行的方法 ：

　　思想：将整形的每1byte作为一个关键字，也就是说一个整形可以拆成4个keys，而且最高位的keys越大，整数越大。如果高位keys相同，则比较次高位的keys。整个比较过程类似于字符串的字典序。

　    第一步:把10G整数每2G读入一次内存，然后一次遍历这536,870,912个数据。每个数据用位运算">>"取出最高8位 (31-24)。这8bits(0-255)最多表示255个桶，那么可以根据8bit的值来确定丢入第几个桶。最后把每个桶写入一个磁盘文件中，同时在 内存中统计每个桶内数据的数量，自然这个数量只需要255个整形空间即可。

　　代价：(1) 10G数据依次读入内存的IO代价(这个是无法避免的，CPU不能直接在磁盘上运算)。(2)在内存中遍历536,870,912个数据，这是一个 O(n)的线性时间复杂度。(3)把255个桶写会到255个磁盘文件空间中，这个代价是额外的，也就是多付出一倍的10G数据转移的时间。

　 　第二步：根据内存中255个桶内的数量，计算中位数在第几个桶中。很显然，2,684,354,560个数中位数是第1,342,177,280个。假 设前127个桶的数据量相加，发现少于1,342,177,280，把第128个桶数据量加上，大于1,342,177,280。说明，中位数必在磁盘的 第128个桶中。而且在这个桶的第1,342,177,280-N(0-127)个数位上。N(0-127)表示前127个桶的数据量之和。然后把第 128个文件中的整数读入内存。(平均而言，每个文件的大小估计在10G/128=80M左右，当然也不一定，但是超过2G的可能性很小)。

　　代价：(1)循环计算255个桶中的数据量累加，需要O(M)的代价，其中m<255。(2)读入一个大概80M左右文件大小的IO代价。

       注意，变态的情况下，这个需要读入的第128号文件仍然大于2G，那么整个读入仍然可以按照第一步分批来进行读取。

　　第三步：继续以内存中的整数的次高8bit进行桶排序(23-16)。过程和第一步相同，也是255个桶。

　　第四步：一直下去，直到最低字节(7-0bit)的桶排序结束。我相信这个时候完全可以在内存中使用一次快排就可以了。

 　　整个过程的时间复杂度在O(n)的线性级别上(没有任何循环嵌套)。但主要时间消耗在第一步的第二次内存-磁盘数据交换上，即10G数据分255个文件写回磁盘上。一般而言，如果第二步过后，内存可以容纳下存在中位数的某一个文件的话，直接快排就可以了。

7 .Java垃圾回收机制

       Java的垃圾回收机制是Java虚拟机提供的能力，用于在空闲时间以不定时的方式动态回收无任何引用的对象占据的内存空间。

       需要注意的是：垃圾回收回收的是无任何引用的对象占据的内存空间而不是对象本身，很多人来我公司面试时，我都会问这个问题的，70%以上的人回答的含义是回收对象，实际上这是不正确的。

       System.gc()

       Runtime.getRuntime().gc()

      上面的方法调用时用于显式通知[JVM](https://www.baidu.com/s?wd=JVM&tn=44039180_cpr&fenlei=mv6quAkxTZn0IZRqIHckPjm4nH00T1Y3nyndmW-BPjTznAf4nynd0ZwV5Hcvrjm3rH6sPfKWUMw85HfYnjn4nH6sgvPsT6KdThsqpZwYTjCEQLGCpyw9Uz4Bmy-bIi4WUvYETgN-TLwGUv3EnHT4nj61nWb3)可以进行一次垃圾回收，但真正垃圾回收机制具体在什么时间点开始发生动作这同样是不可预料的，这和抢占式的线程在发生作用时的原理一样。