**剑指Offer——大疆笔试题+知识点总结**

**情景回顾**

* 时间:2016.9.23 12：00-14：00   19：00-21：00
* 地点:山东省网络环境智能计算技术重点实验室
* 事件:大疆笔试
* 注意事项：要有大局观，该舍弃的还是要舍弃，不要在一道编程题上占用超过30分钟的时间。当你思考了15分钟，还没有好的解决方式的时候，毅然舍弃！

总体来说，大疆的题目不算太难。尤其是编程题，思路很清晰，不涉及到太复杂的算法。只有第三题可能涉及到递归操作(软肋)。其中第二题有关完全二叉树的操作，只需要知道完全二叉树的四个重要性质就对了,借助数学运算就可将问题解决。

1. 性质1 在二叉树的第i层上至多有2^(i-1)个节点（i>=1）.
2. 性质2 深度为k的二叉树至多有(2^k)-1个节点(k>=1).
3. 性质3 对任何一棵二叉树T，若叶子节点数为m,度为2的节点数为n,则m=n+1.
4. 性质4 具有n个节点的完全二叉树的深度为log2n(向下取整)+1。

这几次考试均涉及到了C++中计算结构体大小sizeof()问题。看完下面的内容，你就会明白了。

影响结构体的sizeof的因素：

    1）不同的系统（如32位或16位系统）：不同的系统下int等类型的长度是变化的，如对于16位系统，int的长度（字节为2，而在32位系统下，int的长度为4；因此如果结构体中有int等类型的成员，在不同的系统中得到的sizeof值是不相同的。

    2）编译器设置中的对齐方式：对齐方式的作用常常会让我们对结构体的sizeof值感到惊讶。

    对齐

    为了能使CPU对变量进行高效快速的访问，变量的起始地址应该具有某些特性，即所谓的“对齐”。例如对于4字节的int类型变量，其起始地址应位于4字节边界上，即起始地址能够被4整除。变量的对齐规则如下（32位系统）：

请看下面的结构：

[复制代码](javascript:void(0);)

1 struct MyStruct

2

3 {

4

5 double dda1;

6

7 char dda;

8

9 int type;

10

11 };

[复制代码](javascript:void(0);)

    对结构MyStruct采用sizeof会出现什么结果呢？sizeof(MyStruct)为多少呢？也许你会这样求：

    sizeof(MyStruct)=sizeof(double)+sizeof(char)+sizeof(int)=13

    但是当在VC中测试上面结构的大小时，你会发现sizeof(MyStruct)为16。你知道为什么在VC中会得出这样一个结果吗？

    其实，这是VC对变量存储的一个特殊处理。为了提高CPU的存储速度，VC对一些变量的起始地址做了”对齐”处理。在默认情况下，VC规定各成员变量存放的起始地址相对于结构的起始地址的偏移量必须为该变量的类型所占用的字节数的倍数。下面列出常用类型的对齐方式(vc6.0,32位系统)。

类型    对齐方式（变量存放的起始地址相对于结构的起始地址的偏移量）

char    偏移量必须为sizeof(char)即1的倍数

int     偏移量必须为sizeof(int)即4的倍数

float   偏移量必须为sizeof(float)即4的倍数

double  偏移量必须为sizeof(double)即8的倍数

short   偏移量必须为sizeof(short)即2的倍数

    各成员变量在存放的时候根据在结构中出现的顺序依次申请空间，同时按照上面的对齐方式调整位置，空缺的字节VC会自动填充。同时VC为了确保结构的大小为结 构的字节边界数（即该结构中占用最大空间的类型所占用的字节数）的倍数，所以在为最后一个成员变量申请空间后，还会根据需要自动填充空缺的字节。

    下面用前面的例子来说明VC到底怎么样来存放结构的。

[复制代码](javascript:void(0);)

1 struct MyStruct

2

3 {

4

5 double dda1;

6

7 char dda;

8

9 int type;

10

11 };

[复制代码](javascript:void(0);)

    为上面的结构分配空间的时候，VC根据成员变量出现的顺序和对齐方式，先为第一个成员dda1分配空间，其起始地址跟结构的起始地址相同（刚好偏移量0刚好为sizeof(double)的倍数），该成员变量占用sizeof(double)=8个字节；接下来为第二个成员dda分配空间，这时下一个可以分配的地址对于结构的起始地址的偏移量为8，是sizeof(char)的倍数，所以把dda存放在偏移量为8的地方满足对齐方式，该成员变量占用 sizeof(char)=1个字节；接下来为第三个成员type分配空间，这时下一个可以分配的地址对于结构的起始地址的偏移量为9（8+1），不是sizeof(int)=4的倍数，为了满足对齐方式对偏移量的约束问题，VC自动填充3个字节（这三个字节没有放什么东西），这时下一个可以分配的地址对于结构的起 始地址的偏移量为12（8+1+3），刚好是sizeof(int)=4的倍数，所以把type存放在偏移量为12的地方，该成员变量占sizeof(int)=4个 字节；这时整个结构的成员变量已经都分配了空间，总的占用的空间大小为：8 1 3 4=16，刚好为结构的字节边界数（即结构中占用最大空间的类型所占用的字节sizeof(double)=8）的倍数，所以没有空缺的字节需要填充。所以整个结构的大小为：sizeof(MyStruct)=8+1+3+4=16，其中有3个字节是VC自动填充的，没有放任何有意义的东西。

    下面再举个例子，交换一下上面的MyStruct的成员变量的位置，使它变成下面的情况：

[复制代码](javascript:void(0);)

1 struct MyStruct

2

3 {

4

5 char dda;

6

7 double dda1;

8

9 int type;

10

11 }；

[复制代码](javascript:void(0);)

    这个结构占用的空间为多大呢？在VC6.0环境下，可以得到sizeof(MyStruc)为24。结合上面提到的分配空间的一些原则，分析下VC怎么样为上面的结构分配空间的。（简单说明）

struct MyStruct

{

  char dda; //偏移量为0，满足对齐方式，dda占用1个字节；

  double dda1; //下一个可用的地址的偏移量为1，不是sizeof(double)=8的倍数，需要补足7个字节才能使偏移量变为8（满足对齐方式），因此VC自动填充7个字节，dda1存放在偏移量为8的地址上，它占用8个字节。

  int type；//下一个可用的地址的偏移量为16，是sizeof(int)=4的倍数，满足int的对齐方式，所以不需要VC自动填充，type存放在偏移量为16的地址上，它占用4个字节。

}； //所有成员变量都分配了空间，空间总的大小为1+7+8+4=20，不是结构的节边界数（即结构中占用最大空间的类型所占用的字节数sizeof (double)=8）的倍数，所以需要填充4个字节，以满足结构的大小为sizeof(double)=8的倍数。

    所以该结构总的大小为：sizeof(MyStruc)为1+7+8+4+4=24。其中总的有7+4=11个字节是VC自动填充的，没有放任何有意义的东西。

**new与malloc的区别**

参考网址：

<http://blog.csdn.net/miss_acha/article/details/7279915>

<http://www.cnblogs.com/huhuuu/archive/2013/11/19/3432371.html>

**计算十六进制表示形式中1的个数**

PS:唉，做错了,多加了1。对于较大的数，自己应该首先用简单的数验证一下的，而不至于算错！

**package cn.edu.ujn.offersword;**

**import java.util.Scanner;**

**public class C2\_10\_NumberOf1InBinary {**

**/\*\***

**\***

**\* 二进制中1的个数**

**\***

**\* 题目描述**

**输入一个整数，输出该数二进制表示中1的个数。其中负数用补码表示。**

**\***

**\* @author SHQ**

**\*/**

**public static void main(String[] args) {**

**Scanner in = new Scanner(System.in);**

**while(in.hasNext()){**

**int n = in.nextInt();**

**System.out.println(NumberOf1(n));**

**}**

**}**

**// n 中有多少个1就循环多少次**

**private static int NumberOf1(int n){**

**int cnt = 0;**

**while(n != 0){**

**++cnt;**

**n = (n - 1) & n;**

**}**

**return cnt;**

**}**

**// n 中有多少位二进制就循环多少次**

**private static int NumberOf1\_(int n){**

**int cnt = 0;**

**int flag = 1;**

**while(flag != 0){**

**if((n & (flag)) != 0)**

**cnt++;**

**flag = flag << 1;**

**}**

**return cnt;**

**}**

**}**

**解决hash冲突的方法**

与DES、AES加密方式相比，RSA、ECC加密方式属于非对称加密(答对)？

参考文献:<http://www.willrey.com/support/ssl_DES.html>

BigEndian形式为 43981，16进制无符号数占2个字节，存储在LittleEndian中的形式为\_\_\_\_\_？

参考

Big Endian      10110100

Little Endian   00101101

    实际上，由于CPU存储数据操作的最小单位是一个字节，其内部的比特序是什么样对我们的程序来说是一个黑盒子。也就是说，你给我一个指向0xB4这个数的指针，对于big endian方式的CPU来说，它是从左往右依次读取这个数的8个比特；而对于little endian方式的CPU来说，则正好相反，是从右往左依次读取这个数的8个比特。而我们的程序通过这个指针访问后得到的数就是0xB4，字节内部的比特序对于程序来说是不可见的，其实这点对于单机上的字节序来说也是一样的。

    那可能有人又会问，如果是网络传输呢？会不会出问题？是不是也要通过什么函数转换一下比特序？嗯，这个问题提得很好。假设little endian方式的CPU要传给big endian方式CPU一个字节的话，其本身在传输之前会在本地就读出这个8比特的数，然后再按照网络字节序的顺序来传输这8个比特，这样的话到了接收端不会出现任何问题。而假如要传输一个32比特的数的话，由于这个数在littel endian方存储时占了4个字节，而网络传输是以字节为单位进行的，little endian方的CPU读出第一个字节后发送，实际上这个字节是原数的LSB，到了接收方反倒成了MSB从而发生混乱。

**大疆编程题**

1.有关矩阵的操作，子矩阵增值与查询其和。

2.考察完全二叉树，给出完全二叉树的顶点数，求出其深度与叶子节点的数目。

3.长度为n的方格，刷3种颜色的颜料，相邻的方格颜料颜色不能相同，且首尾方格颜色不能相同。每个方格必须涂色。计算一共有多少种涂色方式。

4.问答题 集群网关分发请求至处理服务器的方法及其优缺点。

     实际上考察集群负载均衡策略。

https://www.cnblogs.com/csdn-sunhuaqiang/articles/5903710.html