比特就业课假期作业-C和数据结构作业答案

出题老师：

> C选择题：黄坤（day01-day08）qq：3587670086
>
> C编程题：张文超（day01-day08）qq：3627274478
>
> 数据结构选择题：吴都（day09-day16）qq：1226631755
>
> 数据结构编程题：鲍松山（day09-day16）qq：365690203

作业说明：

> 1、本次作业涵盖内容为C语言和数据结构相关知识点
>
> 2、如果对试卷上的题目，或者答案有问题，可以联系对应老师哦~~
>
> 3、同学们添加老师时备注：姓名+比特班级哦~

day01

一、选择题

1、

答案解析：
正确答案：D
char为有符号类型，占1个字节，也就是8位，其中最高位是符号位，取值范围为-128~127；a=101+27=128，128表示为1000 0000，作为补码放在内存中，符号位为1，在与int类型的sum进行加计算时会整型提升，高位补1，再转成原码为-128，sum=200+(-128)=72

2、

答案解析：
正确答案：D
1024的二进制是：0000 0000 0000 0000 0000 0100 0000 0000；分析得*((char *)(&value))的作用是获取value变量的低地址8位数据，若数据是采用大端存储方式，则低地址对应的是数据的高位，即最左边的8位0，则condition=0，不执行两个if语句，则value不变，还是1024;若数据是采用小端存储方式，则低地址对应的是数据的低位，即最右边的8位0，则condition=0，不执行两个if语句，value值还是1024

3、

答案解析：
正确答案：A
para和p都是指针，32位机器上指针都是4个字节

4、

答案解析：
正确答案：C
调用函数func时传的是s的值，形参p的改变，并不会改变s本身，*s拿到的还是首地址的字符'1'

5、

答案解析：
正确答案：CD
数组D是一个二维数组，函数传参时数组名退化为首元素地址，就是第一行的地址，是一维数组的地址，为int(*)[8]类型，C正确，B选项是指针数组的，这里不行；若想写成数组的形式，则列不能省行可以省，D选型格式是对的，A选项不对

二、编程题

1、【答案解析】：

暴力破解：对 `[0, n]` 区间内的每个数字求平方，然后对n的几位数字进行是否相等判断，若相等则 `count++` 但是这样做效率较低，每个数字都要计算每一位数是否相等， `n` 是 `m` 位数，则需要进行循环 `n*m` 次才能得到结果.

更优思想： 两位数的时候只需要对平方取模100进行比较，三位数平方取模1000， 四位数取模10000.....

```
#include <stdio.h>
#include <math.h>
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        long count = 0, base = 10;
        for (int i = 0; i <= n; i++){
            long pow_n = pow(i, 2);
            if (i == base) base *= 10;//两位数的时候成为100，三位数的时候成为1000....
            if (pow_n % base == i) count++;
            /*
            int tmp = i;
            while(tmp){
                if (tmp%10 != pow_n%10) break;
                tmp/=10;
                pow_n/=10;
            }
            if (tmp == 0)count++;
            */
        }
        printf("%d\n", count);
    }
    return 0;
}
```

2、【答案解析】

编写一个质数判断函数，然后对 `[2, n]` 区间的数字进行逐个判断，若为质数则计数 `count++` 即可。

质数判断： `[2, sqrt(num)]` 区间内的数字能被 `num` 整除，则表示不是质数

```
#include <stdio.h>
#include <math.h>
int is_prime(int num) {
    for (int i = 2; i <= sqrt(num); i++){
        if (num % i == 0) {
            return 0;
        }
    }
    return 1;
}
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        int count = 0;
        for (int i = 2; i <= n; i++) {//0和1不算质数所以从2开始
            if (is_prime(i))
                count++;
        }
        printf("%d\n", count);
    }
    return 0;
}
```


day02

一、选择题

1、

答案解析：
正确答案：D
char *s[6]={ "ABCD", "EFGH", "IJKL", "MNOP", "QRST", "UVWX" };
以上语句定义了一个字符指针数组s。首先这是一个数组，这个数组里存储的是字符指针，也就是说s[1], s[2] ...等存储的都是字符指针，类型是 char*。而数组名是指向第一个元素的常量指针，因此s是指向指针的指针，所以函数fun的形参定义是char **。fun(s)将指针s的值传递给形参p，所以p = s，因此for(i=0;i<4;i + +)printf("%s",p[i]);中printf("%s",p[i])等价于printf("% s",s[i])。注意，虽然s[i]中存储的不是字符串，而是char*类型的指针，printf在按照%s形式打印字符串的时候，看到s[i]中存放的地址，输出s[i]存储的指针指向的字符串。所以最后输出为D

2、

答案解析：
正确答案：C
在二维数组中a[1]是第一行的数组名，数组名表示首元素的地址，即a[1][0]的地址，所以a[1]+1表示的是a[1][1]的地址，所以D可以取得正确的值；指针操作*(a+1)与a[1]等价，所以B也可以取得正确的值；二维数组在内存中连续存储，所以A中a[0][0]的地址加5可以取得正确值；C选项错误，应改为*(&a[1][0]+1)，因为a[1]就表示a[1][0]的地址

3、

答案解析：
正确答案：B
这是一个标准的函数指针数组，s先与[5]结合，说明s是一个数组，数组的元素是void (*)(int)类型的函数指针，该指针指向的函数参数为int，返回值为void

4、

答案解析：
正确答案：D
int *k[10][30];表示的是指针数组，一共有10*30=300个元素，在64位系统下，每个指针的长度是8字节，因此总长度为2400字节

5、

答案解析：
正确答案：C
A选项，f(a)传参时，a会退化成指向其首元素的地址，类型是 int*，不符。B选项，b是二维数组，传参时会退化成指向其首元素的指针，也就是b[0]的地址,b[0]的类型是int [4],故&b[0]类型是int(*)[4]，不符。D选项，&a是数组a的地址，其类型是int(*)[4]，不符。C选项，q是一个指针数组，在初始化时用b[0]、b[1]、b[2], 此时b[0]、b[1]、b[2]会退化成指向各首元素的指针（int* 类型，因此类型符合，可以用它们初始化）。q传参时，退化成指向其首元素的指针，即 int**，符合

二、编程题

1、【答案解析】：

暴力破解：将字符串每个字符，逐个与剩下的字符进行比较如果相等则换下一个，最终如果都没有相等重复则跳出循环，得到第一个非重复字符下标。

更优思想：小写字母 [a, z] 的 ascii 值为 [97, 122]，大写字母 [A, Z] 的 ascii 值为 [65, 90]

这时候只需要定义一个统计各个字符出现次数的表（数组），然后遍历字符串，以字符的 ascii 值为下标对表中对应位置的计数 +1，完成后，重新遍历字符串，从前往后判断哪个字符在表中统计的次数是 1 即可。

```
int FirstNotRepeatingChar(char* str ) {
    // write code here
    //小写字母[a, z]-[97,122], 大写字母[A, Z]-[65,90]
    char table[128] = {0};//以字符ascii值为下标，存放各个字符出现的次数
    int len = strlen(str);
    for (int i = 0; i < len; i++) {
        if (table[str[i]] < 256)//防止字符出现次数过多，导致溢出截断
            table[str[i]] += 1; //以字符ascii值为下标，对应位置次数+1
    }
    //从第0个字符开始在表中查看对应字符的出现次数，第一个次数为1则找到
    for (int j = 0; j < len; j++) {
        if (table[str[j]] == 1){
            return j;
        }

    }
```

```
    return -1;
}
```

2、【答案解析】：

暴力破解：遍历字符串，将每个字符与剩余的字符进行逐个比较，判断是否相同，若有相同则返回`false`即可，好在测试用例的字符串都不长，否则估计要超时...

建表标记思想：`ascii`字符的`ascii`值都不会超过`255`，定义一张表（数组）用于标记出现的字符，如果哪个字符在进行标记的时候发现已经标记过，那就意味着有重复字符，不是全都不同。

```
bool isUnique(char* astr){
    //ascii字符的ascii值都不会超过255， 定义一张表（数组）用于标记出现的字符，
    //如果哪个字符在进行标记的时候发现已经标记过，那就意味着有重复字符，不是全都不同。
    for (char *s = astr; *s != '\0'; s++) {
        for (char *e = s+1; *e != '\0'; e++) {
            if (*s == *e) return false;
        }
    }
    return true;
    /*
    char table[256] = {0};
    while(*astr) {
        if (table[*astr] == 1) {
            return false;
        }
        table[*astr] = 1;
        astr++;
    }
    */
    return true;
}
```


day03

一、选择题

1、

答案解析：
正确答案：ABD
A选项，首先类型就不匹配，getchar()函数返回值是int，只能赋值给整型，此时p为char*类型。B选项，p指针在定义的时候没有分配内存，这行代码在运行的时候会报野指针错误。C选项，指针p指向数组s。D选项，*p代表p指向内存的内容，这里要使用p = "china"才正确

2、

答案解析：
正确答案：A
char **ptr[] 是一个二级指针数组，存的是s这个指针数组中每个元素的地址，p是一个三级指针，ptr这个二级指针数组的数组名会退化成一个三级指针赋值给p，++p相当于是在ptr数组上往后跳了一个元素的位置,*p就是ptr数组第二个元素的内容,也就是指向s数组中s+2这个元素的地址,**p就得到了s数组中s+2这个元素的内容, s数组中保存的是字符串的首地址,那就得到了"pink"这个元素的首地址,在**p+1就是这个地址向后偏移一个字节,也就指到了'i'上,所以printf("%s",**p+1);输出的就是"ink"

3、

答案解析：
正确答案：A
题目主要就考指针类型是否一样，char s[3][10]中s运算时会退化为数组指针，类型为char (*)[10]，所指向的每个数组长度为10；char (*k)[3]很明显k就是一个数组指针，类型也为 char (*)[3]，所指向的每个数组的长度为3；char *p类型为char *指针，s[0]代表二维数组第一行，运算时会退化为第一行首元素地址，类型为char *。在没有强制类型转换的情况下，只有类型完全相同的指针才能相互赋值

4、

答案解析：
正确答案：B
因为sizeof是计算数据结构的大小，数组就是一种数据结构，sizeof(str)此时str代表整个数组。对于"ab"，编译器都会自动在后面添一个0，占据一个字节，共3个字节，故数组str的长度为3，元素是char类型，故最终结果为3

5、

答案解析：
正确答案：CD
我们知道使用(*f1)(s)与f1(s)效果是相同的。*f1(s)中()的优先级比*高，先结合为f1(s)，返回结果是char，*f1(s)是对返回结果char类型数据解引用，是错的，改成(*f1)(s)是对的，故D正确；对ch取地址得到char *，作为参数没问题，C正确；A错在对char解引用，B错在参数类型不匹配

二、编程题

1、【答案解析】：

这道题我的第一想法是将两个字符串逐个字符串异或，如果为`0`则就完全一致。但是不行，因为有可能另一个字符串中多出了偶数个相同字符的情况。

而其他的解法，说白了给两个字符串各自统计一个字符出现次数表，最终只需要判断两张表中相同字符的出现次数是否一致就可以了。

```
bool CheckPermutation(char* s1, char* s2){
    int table1[256] = {0}, table2[256] = {0};
    int len1 = strlen(s1);
    int len2 = strlen(s2);
    if (len1 != len2) return false;//长度都不同，直接不可能了
    while(*s1 != '\0' && *s2 != '\0') {
        table1[*s1] += 1;//以字符ascii值为下标对s1字符串的字符进行计数
```

```
        table2[*s2] += 1;//以字符ascii值为下标对s2字符串的字符进行计数
        s1++;
        s2++;
    }
    for (int i = 0; i < 256; i++) {
        if (table1[i] != table2[i]) { //判断两个字符串相同位置是否计数相同即可
            return false;
        }
    }
    return true;
}
```

2、【答案解析】：

回文串不管怎么排列，肯定是最多只有一个字符是出现奇数次的，其他的字符出现肯定都是偶数次。因此统计字符出现次数，如果有`1`个以上的字符出现次数是奇数，则表示肯定不是回文串的排列。

```
bool canPermutePalindrome(char* s){
    int table[256] = {0};
    for (char *p = s; *p != '\0'; p++) {
        table[*p] += 1;//以字符ascii值为下标对字符出现次数进行统计
    }
    int count = 0;
    for (int i = 0; i < 256; i++) {
        if (table[i] % 2 == 1) count++; //出现奇数次的字符如果超过1个就表示肯定不是回文串
        if (count > 1) {
            return false;
        }
    }
    return true;
}
```


day04

一、选择题

1、

```
答案解析：
正确答案：C
p是char*类型，每次++，后移一个地址，char *p = str[0]相当于char *p = "stra"，p先指向其中的字符's'，printf输出遇到0停止，第一次输出"stra"，p++后，指向字符't'，第二次输出"tra"，第三次输出"ra"
```

2、

答案解析：
正确答案：A
p是一个数组指针，指向的数组元素是int，长度是4，p被赋值为m的首地址，此时p+1在数组m里会跳过4个元素，相当于是将m中的元素四个一组进行管理，p[1]拿到第二组{5，6，7，8}，而p[1]是5的地址，p[1][2]就是这一组里第三个元素7

3、

答案解析：
正确答案：D
计算时p1、str退化为首元素地址，则p1+2相当于是字符串"cd"的首地址，p2+1相当于是字符串"BCD"的首地址，strcat函数将两个字符串拼接，并把新字符串"cdBCD"的首地址返回，再执行strcpy(str+2,"cdBCD");str+2是数组第3个元素'z'的地址所以，strcpy(str+2,"cdBCD")也就是将'z'及其后的字符用"cdBCD"替换，而前面的"xy"不受影响 所以最后输出"xycdBCD"

4、

答案解析：
正确答案：C
A选项错误，通常C语言中两个字符串比较大小，是借助strcmp()函数进行的。B选项中strlen返回的是字符串除结束符'\0'的部分，实际占用内存大小加上结尾符。D选项两个字符串拼接后，原本每个字符串后都会有一个字符串结束符'\0'，而拼接后，字符串结束符只剩下一个，所以占用空间会减一

5、

答案解析：
正确答案：D
内层while循环的作用让指针it2跳过字符'c'，第一次越过后停在原字符串第一个'd'上，而在之前it1和it2是同步的，赋值不会改变字符串内容，此时it1停在第一个'c'上，*it1++ = *it2++;语句将'c'替换为'd'，字符串变更为"abddcccd"，同时指针it1加加停在第四个字符'd'上，it2再次跳过字符'c'，停在最后一个'd'上，赋值后，字符串不变，再后移外层循环遇到0结束

二、编程题

1、【答案解析】：

需要注意的是简写规则：

1. `strlen(str) >= 10` 的单词需要简写
2. 简写首先是首字符，其次是简写的长度`strlen(str) - 2`，也就是省略的数据长度，最后是末尾字符。

```
#include <stdio.h>
int main()
{
   int n;
   while(~scanf("%d", &n)) {
      char str[101] = {0};
      scanf("%s", str);
      int len = strlen(str);
      if (len < 10) {
         printf("%s\n", str); //短单词不用缩写

         continue;
```

```
        }
        printf("%c%d%c\n", str[0], len-2, str[len-1]);//首字符，省略长度，末尾字符
    }
    return 0;
}
```

2、【答案解析】：

以前常规的字符串替换，如果所需空间更多，则需要将后边的字符向后移动，给前边保留足够的替换空间，以这种方式遍历字符串逐个字符替换，比如：

```
char str[] = "a b c";  替换第一个空格的时候需要将 "b c" 向后移动2个字符才能放下 '%20'
```

但是这样的话效率太低，因为本身遍历字符串需要循环 strlen(str) 次， 如果最坏情况下全部都是空格，则每个都要替换，意味着每个字符进行替换的时候都要将后边字符向后移动一次，太恐怖了。

更优思想：先遍历字符串统计空格有多少个，然后计算替换后所需空间大小，然后从最后一个字符开始，从后往前逐个填充替换即可。

```
char* replaceSpaces(char* S, int length){
    int count = 0;
    for (int i = length - 1; i >= 0; i--) {
        if (S[i] == ' ') count++;//统计空格的个数
    }
    int r_end = length + count * 2; //实际所需数组空间的长度
    int c_end = length - 1;//字符串的末尾
    S[r_end--] = '\0';//先设置字符串结尾标志
    while(c_end >= 0){ //从后往前开始遍历替换
        if (S[c_end] == ' ') {//当前字符是空格则向末尾添加 %20
            S[r_end--] = '0';
            S[r_end--] = '2';
            S[r_end--] = '%';
            c_end--;
            continue;
        }
        S[r_end--] = S[c_end--];//当前是非空格则直接赋值即可
    }
    return S;
}
```

day05

一、选择题

1、

答案解析：
正确答案：A
因为arr是一个2行4列的二维数组，每一行可以存放最多三个字符的字符串。执行strcpy(art[0],"you");后则将字符串"you"放置在数组arr的第一行上，即a[0][0]='y',a[0][1]='o',a[0][2]='u',a[0][3]='\0'(字符串结尾符)，执行strcpy(arr[1],"me")；后则将字符串"me"放置在数组arr的第二行上，即a[1][0]='m'，a[1][1]='e',a[1][2]='\0'(字符串结尾符)，由于二维数组在内存中是以行序存放的，执行arr[0][3]='&';语句后，第一行与第二行将被视为一个字符串"you&me"，而数组名则是该字符串的首地址。所以正确答案是A

2、

答案解析：
正确答案：B
要注意的是64位的情况下指针是占8个字节的,剩余详细了解结构体内存对齐的规则

3、

答案解析：
正确答案：C
因为mon给了初始值3，之后的成员会在此基础上递增

4、

答案解析：
正确答案：AB
虽然d1，d2地址相同，但存储的是二进制，浮点型的10和整型的10，二进制是不同的，所以读取出来的结果也是不同的，故CD错误

5、

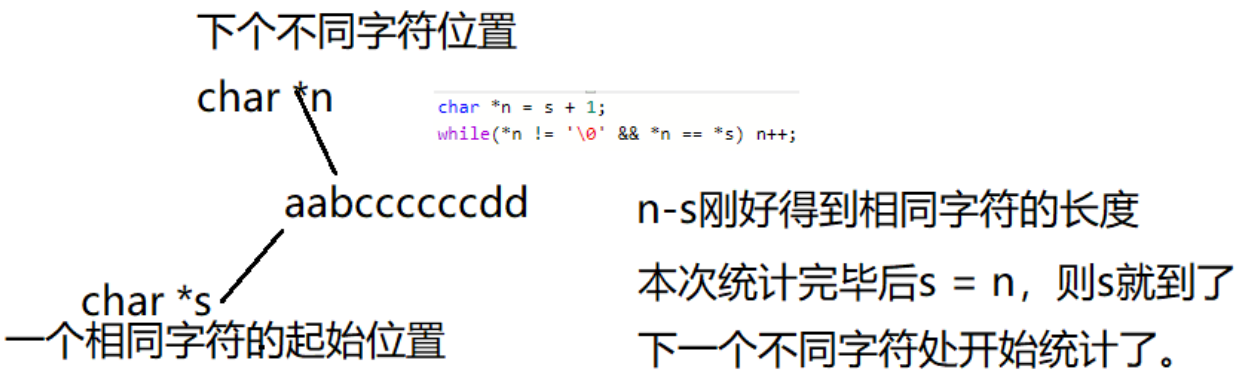
答案解析：
正确答案：D
不管用户程序怎么用malloc，在进程结束的时候，用户程序开辟的内存空间都将会被回收。

二、编程题

1、【答案解析】：

这道题的思路是遍历字符串，统计每个字符的出现次数就行，麻烦的是数字转字符串会稍微麻烦一些，其实遍历数字的每一位数，然后加上 '0' 字符，就成为对应的数字字符了。 通过这种方式实现数字转字符串操作。

可以调研一下sprintf这个函数，要是会用这个函数就能简化很多了~~




```
int itoa(char *str, int num){//简单的一个数字转字符串函数，将转换后的数字字符串放到str空间中
    char tmp[16] = {0};
    int count = 0;
    while(num) {
        tmp[count++] = (num % 10) + '0';//将个位数字转换为对应数字字符放到tmp空间中（逆序的）
        num /= 10;
    }
    for(int i = 0; i < count; i++) {
        str[i] = tmp[count - i - 1]; //逆序将数字字符放入传入的str空间中
    }
    return count;//返回转换的数字字符串长度
}
char* compressString(char* S){
    int len = strlen(S);
    //申请新的空间，注意空间有可能所需会变大，比如abc -> a1b1c1
    char *newstr = (char *)calloc(len, 3);
    int pos = 0;
    char *s = S;
    while(*s != '\0') {
        char *n = s + 1;
        while(*n != '\0' && *n == *s) n++;//走到下一个不同字符处停下
        int l = n - s;//相同字符的长度
        newstr[pos++] = *s;// 先赋值对应字符
        pos += itoa(newstr + pos, l);//编写了一个数字转字符函数，将数字放到字符串空间指定位置
        s = n;//下一个不同的字符处
    }
    return strlen(newstr) >= len ? S : newstr;
}
```


2、【答案解析】：

注意：这道题是二进制的奇数位和偶数位的交换，只需要把偶数位左移一位，把奇数位右移一位即可

```
int exchangeBits(int num){
    int odd = 0b10101010101010101010101010101010;//保存偶数位
    int even = 0b01010101010101010101010101010101;//保留奇数位
    odd &= num;
    even &= num;
    //偶数位右移一位变成奇数位，奇数位左移一位变成偶数位，然后相加即可
    return (even<<1) + (odd>>1);
}
```

day06

一、选择题

1、

答案解析：
正确答案：D
选项A：malloc函数用来动态地分配内存空间，其原型void* malloc (size_t size); size 为需要分配的内存空间的大小，以字节（Byte）计。所以A正确。选项B：realloc函数，其原型extern void *realloc(void *mem_address, unsigned int newsize);如果有足够空间用于扩大mem_address指向的内存块，则分配额外内存，并返回mem_address。realloc是从堆上分配内存的，当扩大一块内存空间时，realloc()试图直接从堆上现存的数据后面的那些字节中获得附加的字节，即如果原先的内存大小后面还有足够的空闲空间用来分配，加上原来的空间大小= newsize，得到的是一块连续的内存。所以B正确。选项C：realloc函数，如果原先的内存大小后面没有足够的空闲空间用来分配，那么从堆中另外找一块newsize大小的内存，并把原来大小内存空间中的内容复制到newsize中，返回新的mem_address指针，而后释放原来mem_address所指内存区域，同时返回新分配的内存区域的首地址。所以C正确。选项D：free函数，作用是释放内存,内存释放是标记删除, 只会修改当前空间的所属状态,并不会清除空间内容。所以D错误

2、

答案解析：
正确答案：B
题目中的char data[0]或写成char data[]，即为柔性数组成员；在计算机结构体大小的时候data不占用struct的空间，只是作为一个符号地址存在。因此sizeof的值是两个指针所占字节，即4+4=8字节

3、

答案解析：
正确答案：B
malloc给开辟空间把地址强转为int *给了p，fut(&p,a);函数调用时把指针变量的地址传给形参s，**s解引用拿到的是开辟的那块内存，并把二维数组元素赋值进去，数组第二行第二列是9，故在print时*p得到的是9

4、

答案解析：
正确答案：D
free释放的内存不一定直接还给操作系统，可能要到进程结束才释放。malloc不能直接申请物理内存，它申请的是虚拟内存

5、

答案解析：
正确答案：B
w+以纯文本方式读写，而wb+是以二进制方式进行读写。 mode说明：w打开只写文件，若文件存在则文件长度清为0，即该文件内容会消失。若文件不存在则建立该文件。w+打开可读写文件，若文件存在则文件长度清为零，即该文件内容会消失。若文件不存在则建立该文件。wb只写方式打开或新建一个二进制文件，只允许写数据。wb+读写方式打开或建立一个二进制文件，允许读和写。r打开只读文件，该文件必须存在，否则报错。r+打开可读写的文件，该文件必须存在，否则报错。 rb+读写方式打开一个二进制文件，只允许读写数据。a以附加的方式打开只写文件。a+以附加方式打开可读写的文件。ab+读写打开一个二进制文件，允许读或在文件末追加数据。加入b字符用来告诉函数库打开的文件为二进制文件，而非纯文字文件

二、编程题

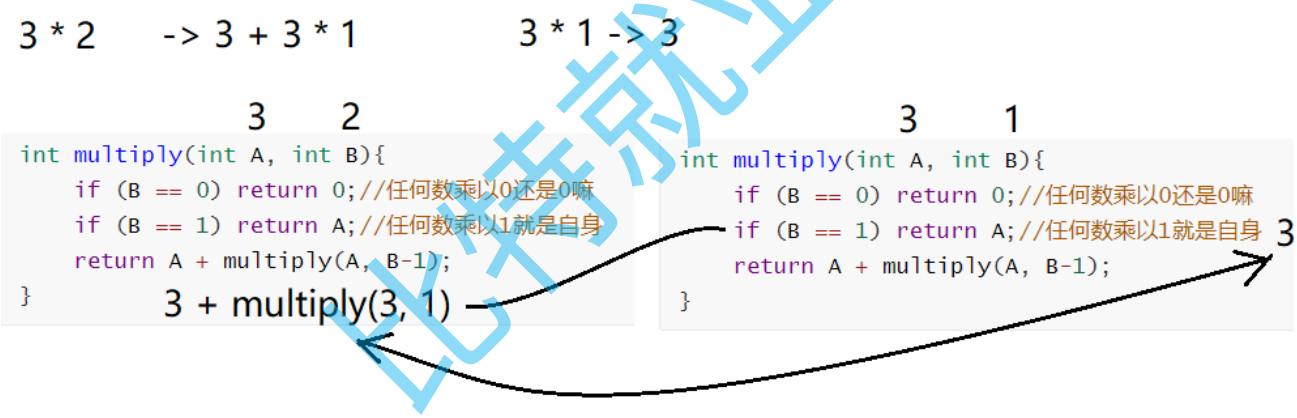
1、

【答案解析：】

这道题的递归思路其实很简单，A，B两数相乘，就是把第A数字相加B次

递归思想：`3 * 2`

```
3 * 2 == 3 + 3 * 1
3 * 1 == 3
```




```
// multiply(3, 3) -> 3+multiply(3, 2) -> 3+multiply(3, 1) -> 3,逐层返回替换相加的过程
int multiply(int A, int B){
    if (B == 0) return 0;//任何数乘以0还是0嘛
    if (B == 1) return A;//任何数乘以1就是自身
    return A + multiply(A, B-1);
}
```

2、

【答案解析】：

这道题暴力破解即可，首先是7的倍数的需要统计，其次是数字中包含7的数字也需要统计，循环获取数字的每一位进行判断就行，但是要注意不要重复统计了，比如`70`，既是`7`的倍数，也包含`7`，然而只需要统计一次即可。

如果不想暴力破解，可以考虑n的范围是在30000以内，因此一个数字只要符合以下任一条件即可：

```
i % 7 == 0                  整除
i % 10 == 7                 个位是7
(i / 10) % 10 == 7          十位是7
(i / 100) % 10 == 7         百位是7
(i / 1000) % 10 == 7        千位是7
```

```
#include <stdio.h>
int main()
{
    int n;//n的范围在30000以内
    while(~scanf("%d", &n)) {
        int count = 0;
        for (int i = 7; i <= n; i++) {
            if ((i % 7) == 0 || (i % 10) == 7 || ((i / 10) % 10) == 7 ||
                ((i / 100) % 10) == 7 ||  ((i / 1000) % 10)  == 7)
                count++;
            /*
            if (i % 7 == 0) {
                count++; //7的倍数
                continue;
            }
            //能走下来就是非7的倍数，检测是否包含7即可
            int num = i;
            while(num) {//获取数字每一位进行判断
                if ((num % 10) == 7){//数字中包含7
                    count++;
                    break;
                }
                num /= 10;
            }
            */
        }
        printf("%d\n", count);
    }
    return 0;
}
```


day07

一、选择题

1、

```
答案解析：
正确答案：A
函数rewind()功能：文件内部的位置指针重新指向一个流（数据流/文件）的开头。注意：不是文件指针而是文件内部的位置指针，随着对文件的读写文件的位置指针（指向当前读写字节）向后移动。而文件指针是指向整个文件，如果不重新赋值文件指针不会改变。rewind函数作用等同于(void)fseek(stream, 0L, SEEK_SET); 原型：void rewind(FILE *stream);
```

2、

答案解析：
正确答案：D
第一个参数是指要打开的文件名的字符串，所以答案A、B是错误的；文件路径分隔符\必须使用转义字符，即\\，所以答案C也是错误的。故选择答案是D

3、

答案解析：
正确答案：A
打开文件采用的是w方式，该方法表示会将原文件清除，然后再重新写入

4、

答案解析：
正确答案：A
B选项，fread函数是二进制输入；C选项，fopen函数是打开文件；D选项，fgets函数是文本行输入

5、

答案解析：
正确答案：D
fputc()写入成功时返回写入的字符，失败时返回EOF即-1，返回值类型为int也是为了容纳这个负数

二、编程题

1、【答案解析】：

求一个数的阶乘中有多少个`尾数0`， 这里就需要灵活一点了`尾数0`是怎么来的？是乘以`10`来的，而`10=2*5`，阶乘中`2`这个因子很多，但是`5`并不多，因此我们只需要统计`5`出现了多少次即可。

暴力方法：遍历`5`的倍数，判断每个`5`的倍数中有多少个`5`，比如`25=5*5`，因此25是需要被统计两次的，这种方法直接，但是效率不高，因为当数字比较大的时候循环次数就比较多了....

更优思想：以`10`的阶乘为例`10!=1*2*3*4*5*6*7*8*9*10`，每`5`个数字就会出现一次`5`，所以`10 / 5`就是`5`出现的次数。

但是特殊的是`25，125....`这种其中包含多个`5`的，`25=5*5，125=5*5*5...`，他们本身只被统计了一次，`25`还剩`1个乘数5`，`125`还剩`1个乘数25`，因此这时候就需要再次除以`5`，得到剩余`5`的个数，直到最后结果小于`5`为止。

```
int trailingZeroes(int n){
    int count = 0;
    while(n >= 5) {
        count += n / 5;
        n /= 5;
    }
    /*
    for (int i = 0; i <= n; i += 5) {
        //判断5的倍数中包含几个5，比如25实际上是5*5有两个
        int num = i;
        while(num > 0 && num % 5 == 0) {
            num /= 5;
```

```
            count++;
        }
    }
    */
    return count;
}
```

2、

【答案解析】：

这道题我在做的时候第一想法是，直接`printf("%.0f", num)`让它自动四舍五入不就行了，但是这个不行，因为浮点数精度的问题，当数据超过浮点类型自身精度的时候，就具有了不确定性。

因此考虑给浮点数加上`0.5`，然后赋值给整形，而浮点数给整形赋值这里存在一个特殊的地方在于只赋值整数部分，小数部分全部忽略，因此只要浮点大于等于`0.5`加上`0.5`后就会进位了，而如果小于`0.5`，则最终小数部分被忽略。

```
#include <stdio.h>
int main()
{
    float num1;
    while(~scanf("%f", &num1)) {
        int num2 = num1 + 0.5;//浮点数给整形赋值时只获取整形部分，小数被丢弃
        printf("%d\n", num2);
    }
    return 0;
}
```

day08

一、选择题

1、

```
答案解析：
正确答案：D
A正确，#define定义的宏是在预处理阶段进行替换的，是仅仅的文本替换，并不会检查其合法性。const常量是在编译、运行阶段进行使用的。B正确，const定义的常量依然是内置类型等，所以会对其进行类型安全检查。C正确，宏定义在程序中使用了几次在预处理阶段就会被展开几次，并不会增加内存占用，但是宏定义每展开一次，代码的长度就要发生变化，而const常量也会为其分配内存（如果是动态申请空间肯定就是堆中了）。D错误，const定义的常量只有一次拷贝没问题，而define定义的变量在内存中并没有拷贝，因为所有的预处理指令都在预处理时进行了替换
```

2、

```
答案解析：
正确答案：D
替换完的样子是++a+b*++b+c，对于b*++b，操作符的优先级只能决定自增++运算在*运算的前面，但是我们并没有办法得知，*操作符的左操作数的获取在右操作数之前还是之后求值，所以结果是不可预测的，是有歧义的
```

3、

答案解析：
正确答案：C
预处理过程读入源代码之后，会检查代码里包含的预处理指令，完成诸如包含其他源文件、定义宏的处理。C选项中，诸如__DATA__、__FILE__、__LINE__、__STDC_VERSION__、__TIME__等，属于预定义宏，因此是在预处理阶段的

4、

答案解析：
正确答案：B
A选项，预处理命令行不能以分号结尾。C选项，预处理命令行可以出现在程序的最后一行。D选项，预处理命令行作用域是整个文件

5、

答案解析：
正确答案：A
define在预处理阶段就把main中的a全部替换为10了。另外，不管是在某个函数内，还是在函数外，define都是从定义开始直到文件结尾，所以如果把foo函数的定义放到main上面的话，则结果会是50..50

二、编程题

1、【答案解析】：

这道题其实本身并不难，有了具体的公式直接循环往后推导就可以得到结果。但是当测试用例有多个的情况下，每次重新计算第k个数据，效率就要差很多了，毕竟每个数都要从头重新计算一遍。

因此程序进入后首先可以先生成一个较大的数列，后边测试用例中需要第几个直接取出即可。

```
#include <stdio.h>
int main()
{
    long n;
    long arr[100000] = {1, 2};
    //先生成数列，注意这里存入的是取模后的结果，否则数据会越界
    for (int i = 2; i < 100000; i++) {
        arr[i] = (arr[i-1] * 2 + arr[i-2]) % 32767;
    }
    scanf("%d", &n);//获取测试用例个数
    for (int i = 0; i < n; i++) {
        int num;
        scanf("%d", &num);
        printf("%ld\n", arr[num-1]);//循环打印每个测试用例的对应结果就行
    }
    return 0;
}
```

2、【答案解析】：

字符个数统计，只需要遍历字符串，根据不同的字符进行不同的计数即可

```
['a', 'z'] || ['A', 'Z'] 这是字母计数
['0', '9']  这是数字字符，一定要注意不是 [0,9]， 数字和数字字符是不一样的
[ ] 空格字符，
上边三种以外就是其他字符了。
```

```c
#include <stdio.h>
int main()
{
    char str[1001];
    while(gets(str) > 0) {
        char *ptr = str;
        int character = 0, space = 0, digit = 0, other = 0;
        while(*ptr != '\0') {  //统计各种字符个数
            if ((*ptr >= 'a' && *ptr <= 'z')||(*ptr >= 'A' && *ptr <= 'Z')) character++;
            else if (*ptr == ' ') space++;//空格计数
            else if (*ptr >= '0' && *ptr <= '9') digit++;//数字字符计数
            else other++;//其他字符计数
            ptr++;
        }
        printf("%d\n%d\n%d\n%d\n", character, space, digit, other);
    }
    return 0;
}
```

day09

一、选择题

1、

```
答案解析：C
1.Func函数的执行次数：F(N) = 2N^2+M
2.根据大O的渐进表示法， 只保留高阶项， 如果最高阶项存在且不是1，则去除与这个项目相乘的常数
所以，时间复杂度为O(N^2)
综上：选择C选项
```

2、

```
答案解析：C
递归函数求时间复杂度的方式：单次递归的时间复杂度*总的递归次数
func函数会被递归调用n次，单次递归的时间是常数，所以时间复杂度为O(n)
综上：选择C
```

3、

答案解析：D
1.该题当中定义变量i, i的初始值为1， 循环判断为真的判断条件为i <= n, 循环体执行语句为 i = i * 2, 在循环体当中， 每次执行循环体， i的数值都会乘以2
2.所以什么时候该函数执行完毕， 取决于什么时候跳出循环体，也就是什么时候,i<=n为假，也就是说， 当i>n的时候， 就不会在进入while循环， 该函数也就执行完毕
3.该函数的基本运算是i=i*2，设其执行时间为T(n)，则2T(n)<=n，即T(n)<=log2n=O(log2n)。
综上：选择D

4、

答案解析：B
1.顺序表插入操作需要考虑空间是否足够，如果不够需要先增容，再进行插入。
综上：选择B选项

5、

答案解析：B
1.顺序表插入元素，需要移动元素，这里需要把[i, X - 1]区间的元素全部向后移动一次，故移动的次数为X - 1 - i + 1
2.举一个例子：顺序表中元素为{ 1,2,3,4,5,6,7}总共7个元素，i=3（4的位置）在第i个位置前插入元素data需要搬移元素个数， 需要移动， 7， 6， 5， 4，才能在在原本4之前空出位置插入data, 也符合X-i, 即 7（顺序表长度） - 3（i的位置） = 4（移动4次）
综上：选择B选项

二、编程题

1、**【解题思路】：**

此题主要考察数组的操作，动态数组求和其实就是i位置的值等于下标0到i的求和，只要控制好循环条件即可。

【代码实现】：

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* runningSum(int *nums, int numsSize, int *returnSize)
{
    int *ans = (int *)malloc(sizeof(int) * numsSize);
    for(int i=0; i<numsSize; ++i)
    {
        int sum = 0;
        //对i之前的数据累加求和
        for(int j=0; j<=i; ++j)
        {
            sum += nums[j];
        }
        ans[i] = sum;
    }
    *returnSize = numsSize;
    return ans;
}
```

2、**【解题思路】：**

此题主要考察数组的二分查找，要使用二分查找，前提是数据已经排序，题目强调为有序数组，故满足条件，首先将目标值跟首位元素相比，如果小于最小值或者大于最大值即可得到插入位置，如果数据在数组中，则采用二分查找即可。

【代码实现】：

```
int searchInsert(int *nums, int numsSize, int target)
{
    //判断边界值
    if(target < nums[0])
        return 0;
    if(target > nums[numsSize-1])
        return numsSize;

    //二分查找
    int left = 0, right = numsSize-1;
    while(left <= right)
    {
        int mid = (left + right) / 2;
        if(target == nums[mid])
            return mid;

        if(target < nums[mid])
            right = mid - 1;
        else
            left = mid + 1;
    }
    return left;
}
```


day10

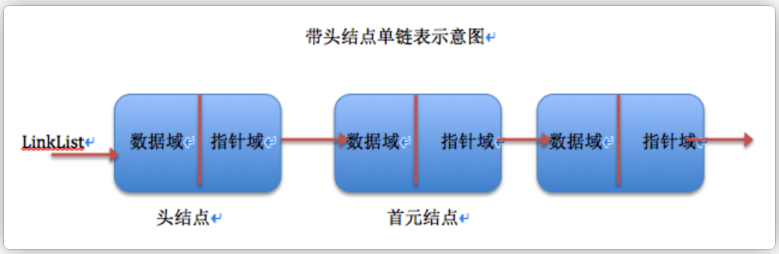
一、选择题

1、

```
答案解析：C
1.二叉树属于树形结构，不是线性的，队列，链表，顺序表都属于线性表
综上：选择C选项
```

2、

```
答案解析：B
带头单链表如下图所示：
```



所以， 当链表为空，则头节点的指针域指向NULL， 如下图：

综上， 选择B选项

3、

答案解析：A
1.顺序表是在计算机内存中以数组的形式保存的线性表。所以直接用下标获取元素的速度非常快。
2.链表都是使用指针串联起来， 获取一个元素的时候， 需要进行遍历。
综上：选择A选项

4、

答案解析：B
双向循环链表的尾节点即是头节点的前驱结点, 所以选择 “p->next == L&&L->prior==p”
综上：选择B选项

5、

答案解析：C
考察单链表的插入操作，核心就是新旧地址的互换。 分两步：
1、将旧结点的指针域(即pPre->Link,它存放着接下来的那个结点的地址)赋值给新结点的指针域（pNew->Link），这一步是因为：为了完成插入，新结点应该指向旧结点原来指向的元素。
2、将指向新结点的指针（pNew，即新结点的地址）赋值给旧结点的指针域（pPre->Link）,以让旧结点指向新结点。
只要将pPre->Link，pNew，pNew->Link这些地址（指针）的关系搞清就行。
综上： 选择C选项

二、编程题

1、**【解题思路】：**

此题主要是一个二分查找的变种考察，如果不考虑效率，可以直接使用顺序查找，但肯定不是面试官想要的结果，所以还得另辟蹊径。 如果中间的数小于最右边的数，则右半段是有序的，若中间数大于最右边数，则左半段是有序的，我们只要在有序的半段里用首尾两个数组来判断目标值是否在这一区域内，这样就可以确定保留哪一半了。

【代码实现】：

```
int search(int* nums, int numsSize, int target)
{
    int left = 0, right = numsSize-1;
    while(left <= right)
    {
        int mid = (left + right) / 2;
        if(nums[mid] == target)
            return mid;
        else if(nums[mid] < nums[right])
        {
            if(nums[mid]<target && target<=nums[right])
                left = mid + 1;
            else
                right = mid - 1;
        }
        else
        {
            if(nums[left]<=target && target<nums[mid])
                right = mid - 1;
            else
                left = mid + 1;
        }
    }
    return -1;
}
```

2、

【解题思路】：

此题主要考察单链表操作，对链表进行遍历提取节点的值，在把该值化为十进制数即可

【代码实现】：

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
int getDecimalValue(struct ListNode* head)
{
    int ans = 0;

    while(head != NULL)
```

```
    {
        ans = (ans<<1) + head->val;  //左移1位，相当于乘以2
        head = head->next;
    }
    return ans;
}
```

day11

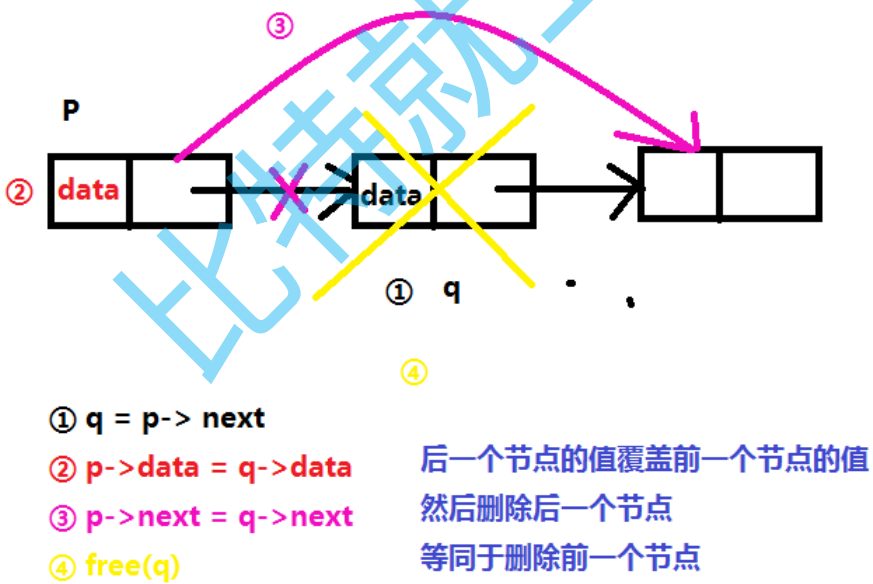
一、选择题

1、

答案解析：D
链栈由于采用了链表的方式作为存储方式，入栈时，使用malloc申请空间后，用指针相连接，所以节点个数没有限制，但是出栈时，如果栈中的元素个数为0，则不能继续出栈，所以需要判断当前栈是否为空。
综上：选择D选项

2、

答案解析：A
参考如下的图解



3、

答案解析：C
1.顺序表便于随机存取，无须为表中元素之间的逻辑关系增加额外的存储空间，花费的存储空间少。
2.链表对频繁插入和删除数据操作，优势比较明显。
综上：选择C选项

4、

答案解析：A
队列是一种操作受限的线性表，其限制为允许在队头进行删除操作，在队尾进行插入操作，它的特点就是先进先出。
综上：选择A选项

5、

答案解析：A
1.顺序栈空间大小是固定的，当元素存满了需要扩容，而链表不需要
2.链表的节点一般都是动态分配内存，如果在没有其他限制情况下，只有动态申请内存失败的时候才算是栈满.
综上：选择A

二、编程题

1、

【解题思路】：

此题主要考察单链表的反转操作，核心思路主要是借助临时头结点，找到需要翻转链表的前驱位置，然后将要翻转链表的节点摘下进行子链表的头部插入即可，题目不难，主要就是要注意一些边界条件的处理，以及借助临时头结点后，能够使代码处理更简单。

【代码实现】：

```
/**
 * struct ListNode {
 *  int val;
 *  struct ListNode *next;
 * };
 */
struct ListNode* reverseBetween(struct ListNode *head, int m, int n )
{
    if(head==NULL || head->next==NULL || m==n)
        return head;

    //借助临时头结点，可以统一所有的情况进行处理，尤其是翻转的链表从第一个节点开始
    struct ListNode *new_head = (struct ListNode*)malloc(sizeof(struct ListNode));
    new_head->next = head;

    //pre指向翻转子链表的前驱节点
    struct ListNode *pre = new_head;
    for(int i=1; i<m; ++i)
        pre = pre->next;

    //head指向翻转子链表的首部
    head = pre->next;
    for(int i=m; i<n; ++i)
    {
        //将p节点摘下进行子链表的头部插入
        struct ListNode *p = head->next;
        head->next = p->next;
        p->next = pre->next;
        pre->next = p;
```

```
    }
    head = new_head->next;
    free(new_head);
    return head;
}
```

2、**【解题思路】：**

此题主要借助链表考察前缀和的求解，要删去总和值为零的连续链表节点，只需维护每一个节点之前的所有和，此时如果该节点与前面节点的和(即前缀和)相加结果为0，则该节点就可以抵消掉前面的节点，消除的时候就是把next指针指向和的下一个节点，然后在重复该过程，直到整个链表求解结束。

【代码实现】：

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* removeZeroSumSublists(struct ListNode *head)
{
    struct ListNode* pre = (struct ListNode*)malloc(sizeof(struct ListNode));
    pre->next = head;
    struct ListNode* p = pre;
    while (p)
    {
        int sum = 0;
        struct ListNode* q = p->next;
        while (q)
        {
            sum += q->val;
            if (sum == 0)
            {
                p->next = q->next;

                //释放被删除的节点
                struct ListNode *del = p->next;
                while(del != q->next)
                {
                    struct ListNode *tmp = del;
                    del = del->next;
                    free(tmp);
                }
            }
            q = q->next;
        }
        p = p->next;
    }

    head = pre->next;
```

```
    free(pre);
    return head;
}
```

day12

一、选择题

1、

答案解析：C
队列: 先进先出 从头插入, 尾部数据操作， 队列是先进先出
栈: 先进后出 从头插入, 从头取出数据， 栈是后进先出
所以：栈当中， ABC依次进栈 则依次出栈的顺序为CBA；队列当中，ABC依次入队 则依次出队的顺序为ABC

2、

答案解析：D
A选项：A进A出，B进B出，C进C出，D进D出。
B选项：ABCD依次进入，DCBA依次输出。
C选项：A进A出，BC进CB出，D进D出。
D选项：没有办法满足条件
综上：选择D

3、

答案解析：B
1.全部入栈之后出栈：CBA
2.部分入栈出栈：BAC，ACB，BCA
3.入栈一个立即出栈： ABC
4.CAB不存在，因为C出栈之后，AB已经出栈，且A不会在B之上
综上：共有5种组合， 选择B选项

4、

答案解析：
正确答案：C
1.根据队列的性质，出队队列的顺序等于入队队列，也就递推等于出栈顺序。所以给题目直接可以转为求出栈顺序不可能的选项。
2.具体分析：
题目给出的入栈：e1,e2,e3, e4,e5,e6，以及栈的一端入栈，两端出栈。
A选项：e2,e4,e3,e5,e1,e6 操作步骤：e1、e2入栈，e2出栈，e3、e4入栈，e4出栈，e3出栈，e5入栈，e5出栈，e1出栈，e6入栈，e6出栈。
B选项：e2,e5,e1,e3,e4,e6 操作步骤：e1、e2入栈，e2出栈，e3、e4、e5入栈，e5出栈，e1出栈，e3出栈，e4出栈，e6入栈，e6出栈
C选项：e5,e1,e6,e3,e2 ,e4 操作步骤：e1、e2、e3、e4、e5入栈，e5出栈，e1出栈，e6入栈，e6出栈，e3无法出栈（因为下面出口还有e2，上面出口还有e4）------> 所以C错误。
D选项：e4,e1,e3,e5,e2,e6 操作步骤：e1、e2、e3、e4入栈，e4出栈，e1出栈，e3出栈，e5入栈，e5出栈，e2出栈，e6入栈，e6出栈
综上：选择C选项

5、

答案解析：B
这道题是考虑循环队列，对于循环队列，空间长度为N是固定的
1.如果front == rear，则（rear - front）== 0， 实际空间长度就是0。
1.如果front < rear，则（rear - front）> 0， 实际空间长度就是（rear-front）。
举例 front = 0，rear = 4, 实际元素就是数组下标为0，1，2，3的元素， 共4个
2.如果front > rear，则（rear - front）< 0，实际长度就是（rear + N - front）% N。
举例front = 1， rear = 0 , 数组长度为4， 则实际元素就是数组下标为1，2，3的元素， 共3个
结论：为了统一两种情况 所以给出的结果为（rear - front + N）% N
综上：选择B选项

二、编程题

1、

【解题思路】：

此题主要考察对链表的操作，借助链表的形式完成两数求和。求解过程为，每次取两个链表的对应节点，求和之后将其放入新节点，并尾部插入到新链表，因为链表是反向存储，直到将两个链表求解结束，值得注意的是，两个数的长度不一定相同，所以在一个链表结束之后，还需判断另一个链表是否是有剩余节点。

【代码实现】：

```
/*
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
//定义全局变量p指向每次插入节点的前驱节点，方便在addItem函数中进行节点链接
struct ListNode *p = NULL;


//完成每一项相加，并进行节点链接，进位sign采用地址传递，因为函数内部会修改进位并在下一次计算时进行使用
```

```
void addItem(int a, int b, int *sign)
{
    struct ListNode *s = (struct ListNode*)malloc(sizeof(struct ListNode));
    int tmp = a + b + *sign;
    if(tmp >= 10)
    {
        tmp -= 10;
        *sign = 1;
    }
    else
        *sign = 0;
    s->val = tmp;
    s->next = NULL;

    //链接节点
    p->next = s;
    p = s;
}

struct ListNode* addTwoNumbers(struct ListNode* l1, struct ListNode* l2)
{
    struct ListNode *head = (struct ListNode *)malloc(sizeof(struct ListNode));
    head->next = NULL;

    //全局变量初始化
    p = head;

    int sign = 0;
    while(l1!=NULL && l2!=NULL)
    {
        addItem(l1->val, l2->val, &sign);
        l1 = l1->next;
        l2 = l2->next;
    }

    while(l1 != NULL)
    {
        addItem(l1->val, 0, &sign);
        l1 = l1->next;
    }

    while(l2 != NULL)
    {
        addItem(0, l2->val, &sign);
        l2 = l2->next;
    }

    if(sign != 0)
        addItem(0, 0, &sign);

    struct ListNode *tmp = head;
    head = head->next;

    free(tmp);
```

```
    return head;
}
```

2、

【解题思路】：

此题主要考察栈的应用，利用栈记录栈顶历史最大值，即就是字符串的括号嵌套最大深度。

【代码实现】：

```
int maxDepth(char *s)
{
    int len = strlen(s);
    //申请栈空间，利用数组模拟栈结构
    char *stack = (char *) malloc(len + 1);
    int top = 0;

    int maxDepth = 0;
    for (int i = 0; i < len; i++)
    {
        if (s[i] == '(')
        {
            stack[top++] = s[i];
            maxDepth = maxDepth > top ? maxDepth : top;
        }
        else if (s[i] == ')')
            top--;
    }
    free(stack);
    return maxDepth;
}
```


day13

一、选择题

1、

```
答案解析：C
首先我们要了解的是栈和队列表达的算法思想，栈是先进后出，队列是先进先出，
根据题意，abcdefg依次进栈，而出队顺序是bdcfeag，由此，我们联想元素进栈时也在出栈和入队，
所以我们根据最终的出队顺序推出进栈顺序的出栈和入队。
            进栈    出栈   此时栈当中剩下   入队出队    出队顺序    队列剩下   所需栈空间
第一次：    ab      b       a          b        b         空       s(2)
    第一次需要栈当中有两个空间，因为至少要容纳ab同时入栈
第二次：    cd      dc      a          dc       dc        空       s(3)
    第二次需要栈当中有三个空间，加上原本的a, cd同时入栈还需要两个空间
第三次：    ef      fe      a          fe       fe        空       s(3)
    第三次需要栈栈当中有三个空间，加上原本的a, ef同时入栈还需要两个空间
第四次：    无      a       空         a        a         空       none

    第四次只需要把原先的a进行出栈即可
```

第五次：g　　g　　空　　g　　g　　空　　s(1)
　　第五次需要进栈g, 所以需要1个空间
通过上述 “所需栈空间”， 可以得知栈至少需要3个空间
综上：选择C选项

2、

答案解析：C
首先，栈的先进后出原则大家应该是知道的。
根据题意 p 2 = 3，可以知道 p1 的可能情况有三种：1，2 或 4 。（看到有些人只想到了 1，2）
为啥这样想呢？这里估计还有一个关键是要考虑到 n 的大小。
当 n = 3 时， p2 = 3 的话，那么 p 1 有两种情况 1 和 2 。
　如果 p 1 = 1 ， 那么 p 3 = 2 ；
　如果 p 1 = 2 ， 那么 p 3 = 1 ；
此时的话我们就可以看到 p 3 只有两种可能 1 或者 2 （n - 1）个。

当 n > 3 时： p 2 = 3 的话，那么 p 1 有三种情况 1 ， 2 和 4 。
　如果 p 1 = 1 ， 那么 p 3 = 2，4，5，... n （n - 2）个
　如果 p 1 = 2 ， 那么 p 3 = 1，4，5，... n （n - 2）个
　如果 p 1 = 4 ， 那么 p 3 = 2，5，6，... n （n - 3）个
此时的话我们就可以看到 p 3 的情况有 1，2，4，5，... n （n - 1）个。
综上所述就是 p 3 可能取值的个数是 （n - 1）个。
综上：选择C选项

3、

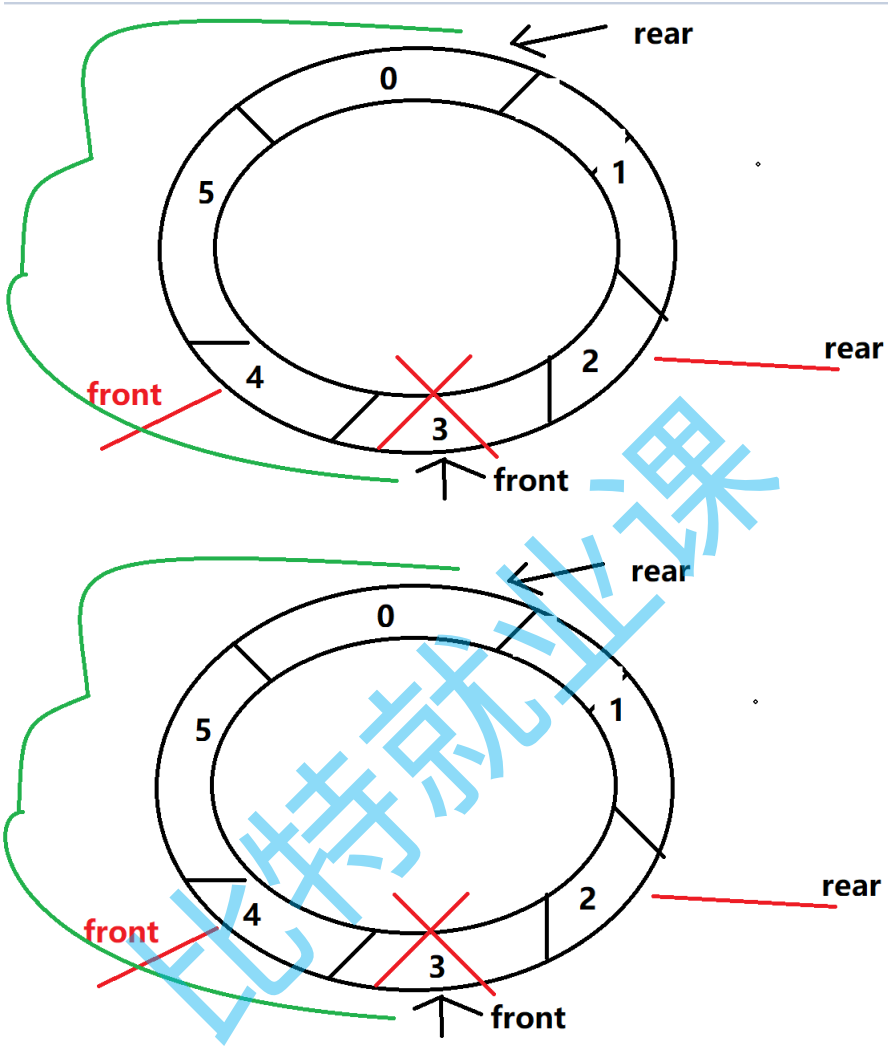
答案解析：D
本题考察知识点是:栈的性质是后进先出
所以:分析如下 栈里的元素最多为2+1=3
进栈元素　　栈里元素(栈底->栈顶)
a进栈　　a
b进栈之后出栈　　a
c进栈　　a c
d进栈之后出栈　　a c
c出栈　　a
e进栈　　a e
f进栈之后出栈　　a e
e出栈　　a
a出栈
综上：选择D选项

4、

答案解析：B
队列中元素的个数：(rear-front+QueueSize)%QueueSize
根据题意得：（rear - 47 + 60）% 60 = 50， 所以：rear = 37
综上：选择B

5、

答案解析：B
1.队列添加元素是在对尾，删除元素是在对头；
2.添加元素，尾指针rear+1;删除元素，头指针front+1;
3.本题中，删除一个元素，front+1,也就是3+1=4；添加2个元素，rear+2,也就是0+2=2；
可以参考下面的图解：



二、编程题

1、

【解题思路】：

此题主要考察栈的使用，将待整理字符串 s 的字符依次入栈，每入栈一个字符就对栈进行检查：栈顶的两个元素是否互为大小写。如果是则弹出栈顶的两个元素。最后栈内的字符即为答案。

【代码实现】：

```
char * makeGood(char *s)
{
    int len = strlen(s);
    //使用字符数组模拟栈结构

    char *st = (char *)malloc(len + 1);
```

```
    int top = 0;
    for(int i=0; i<len; ++i)
    {
        st[top++] = s[i];
        //判断栈顶的两个元素是否互为大小写
        if(top>=2 && (st[top-1]+32==st[top-2] || st[top-1]-32==st[top-2]))
            top -= 2; //删除栈顶的两个元素
    }
    st[top] = '\0';
    return st;
}
```

2、

【解题思路】：

此题变相考察二叉树的遍历，对二叉树进行先序遍历，用哈希数组判断当前值是否出现过（出现过就是1，未出现就是0），每出现一个没有在哈希数组中统计过的元素哈希数组就新增一个值，最后统计哈希数组有多少不为0的值即可。

【代码实现】：

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
int hash[1001];
void preOrder(struct TreeNode* root)
{
    if(root)
    {
        hash[root -> val]++;
        preOrder(root -> right);
        preOrder(root -> left);
    }
}

int numColor(struct TreeNode* root)
{
    memset(hash, 0, sizeof(hash));
    preOrder(root);
    int sum = 0;
    for(int i = 1; i < 1001; i++)
    {
        if(hash[i])
            sum++;
    }

    return sum;
```

```
}
```

day14

一、选择题

1、

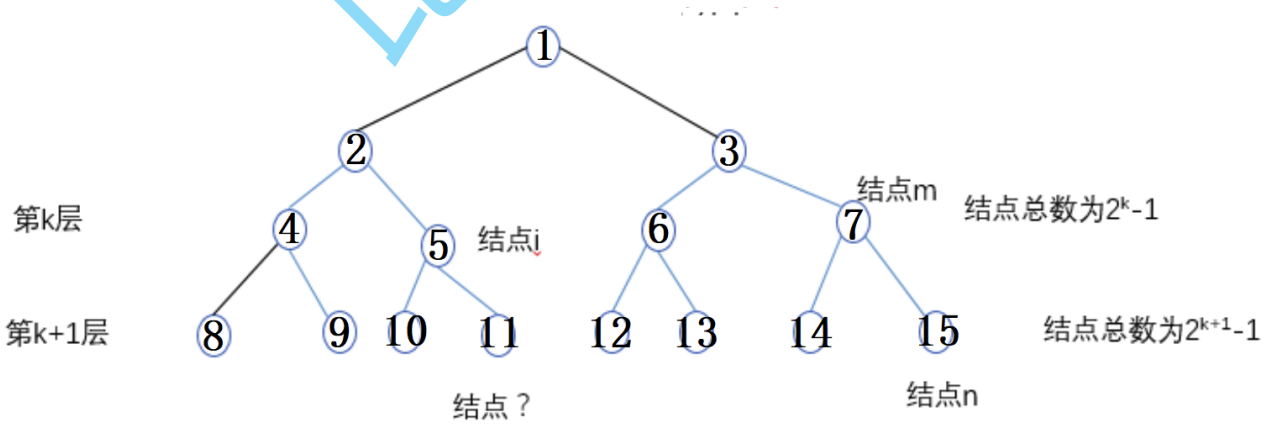
```
答案解析：A
前序遍历：先根节点， 再左子树， 再右子树
中序遍历：先左子树， 再根节点， 再右子树， 针对排序二叉树而言， 该遍历就是有序序列
后序遍历：先左子树， 再右子树， 再根节点。
综上：选择A
```

2、

```
答案解析：C
完全二叉树一共有8个叶结点，而普通二叉树最少可以有1个叶结点，相差最大 15-8=7
```

3、

```
答案解析：D
必须是完全二叉树才能确定，若是，选择B选项。如下图所示：
根据二叉树的性质5:
1.对于具有n个结点的完全二叉树，如果按照从上至下从左至右的数组顺序对所有节点从0开始编号，则对
于序号为i的结点有：
    1.1 若i>0，i位置节点的双亲序号：(i-1)/2；i=0，i为根节点编号，无双亲节点
    1.2 若2i+1<n，左孩子序号：2i+1，2i+1>=n否则无左孩子
    1.3 若2i+2<n，右孩子序号：2i+2，2i+2>=n否则无右孩子
上述的性质的节点从0开始编号， 而题目当中是从1开始编号， 所有右孩子的序号为 2i+1
```



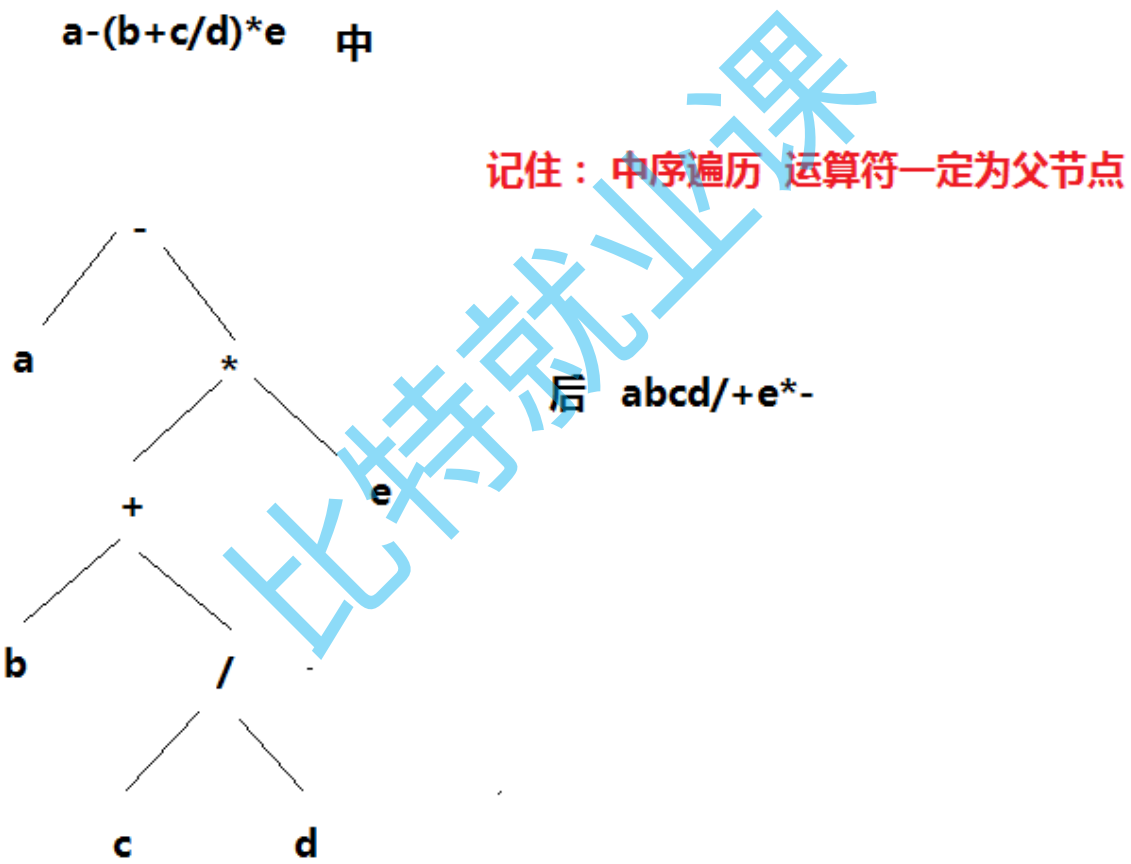
节点？（节点11）的序号 = 节点i（节点5）* 2 + 1；
即节点？的序号为 = 5 * 2 + 1 = 11

4、

答案解析：C， D
中序遍历：先遍历左子树，然后遍历根结点，最后遍历右子树。
A选项：dbac　　B选项：cbda　　C选项：abcd　　D选项：abcd
综上：选择C, D选项

5、

答案解析：D
1.先使用中缀表达式构建二叉树
1.1 第一个 “-”则将表达式分为左右子树， 左子树为“a”, 右子树为“(b+c/d)*e”
1.2 在1.1中的右子树“(b+c/d)*e”， 通过“*”将再次分为左右子树， 左子树为“(b+c/d)”， 右子树为“e”
1.3 在1.2中的左子树“(b+c/d)”， 通过“+”将再次分为左右子树， 左子树为“b”， 右子树为“c/d”
1.4 在1.3中的右子树“c/d”， 通过“/”将再次分为左右子树， 左子树为“c”， 右子树为“d”
所以构建出来的二叉树如下图所示， 后缀表达式也就不难求解， 先左， 再右， 再根， 即“abcd/+e*-”
综上：选择D选项



二、编程题

1、

【解题思路】：

此题主要考察二叉树的深度遍历， `pathVal:` 计算各路径和； `sum:` 统计各路径和；由于结点值为0,1；表示二进制数，利用二进制计算方法由最高位至最低位： `pathVal=(pathVal<<1)+root->val` ；先序遍历树结点，用 `pathVal` 计算至当前节点的值，当该节点为叶节点则达到路径末尾；此时统计 `*sum+=pathVal` ；

【代码实现】：

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
void dfs(struct TreeNode* root, int pathVal, int *sum)
{
    if(root == NULL)
        return;
    pathVal = (pathVal<<1)+root->val; //相当于pathVal*2 + root->val

    //到达叶子节点则统计该路径的和
    if(root->left==NULL && root->right==NULL)
        *sum+=pathVal;

    //递归深度遍历左子树
    dfs(root->left, pathVal, sum);

    //递归深度遍历右子树
    dfs(root->right,pathVal, sum);
}

int sumRootToLeaf(struct TreeNode* root)
{
    int sum=0;
    dfs(root,0,&sum);
    return sum;
}
```


2、

【解题思路】：

此题主要考察二叉树，是一个变相的二叉树遍历，二叉树的坡度等于左树的坡度+右树的坡度+根节点的坡度即可，题目比较简单，可以直接根据题意写出递归程序。

【代码实现】：

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
```

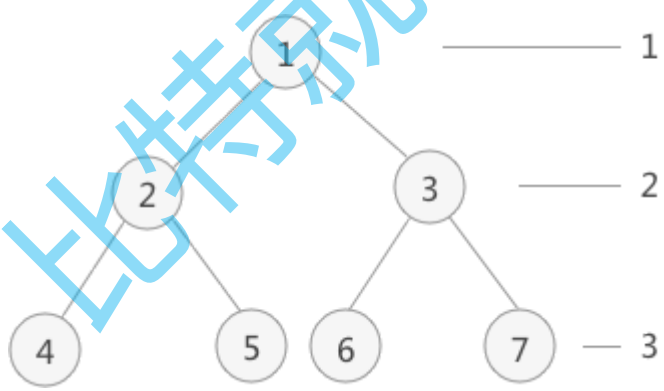
```
int getSum(struct TreeNode *root)
{
    if (root == NULL)
        return 0;
    return getSum(root->left) + getSum(root->right) + root->val;
}

int findTilt(struct TreeNode* root)
{
    if(root == NULL)
        return 0;
    return findTilt(root->left) + findTilt(root->right) + abs(getSum(root->left) - getSum(root->right));
}
```

day15

一、选择题

1、

答案解析：A
1.在按层次遍历二叉树的算法中，需要借助的辅助数据结构是队列。
2.扩展补充：二叉树的先序、中序、后序遍历的非递归实现需要借助栈这一数据结构实现。
如下题给出一个示例：



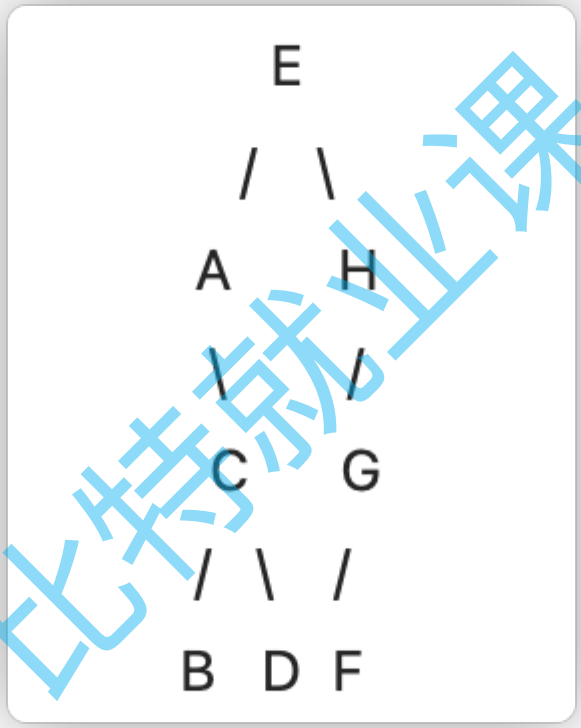
1.首先，根结点 1 入队；
2.根结点 1 出队，出队的同时，将左孩子 2 和右孩子 3 分别入队；
3.队头结点 2 出队，出队的同时，将结点 2 的左孩子 4 和右孩子 5 依次入队；
4.队头结点 3 出队，出队的同时，将结点 3 的左孩子 6 和右孩子 7 依次入队；
5.不断地循环，直至队列内为空。
综上：选择A

2、

答案解析： B
1.先序遍历的规则： 先根节点， 再左子树， 再右子树
2.中序遍历的规则： 先左子树， 在跟节点， 再右子树
3.从前序获取根，在中序中找根的位置，左侧是跟的左子树，右侧是跟的右子树
4.从前序遍历结果知道根是A 前序下一个元素是B 那说明如果A有左子树，则左子树个根一定是B，故排除A和C,D选项明显错误：节点个数都不够
综上选择B选项

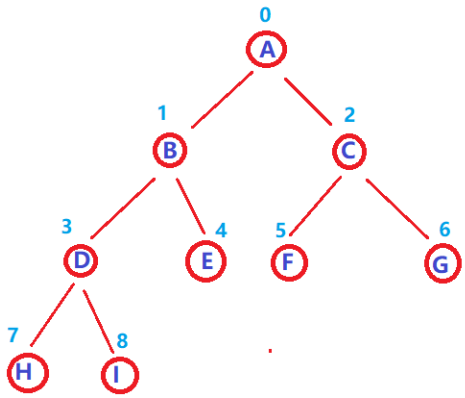
3、

答案解析：B
如下图， 根据题干还原的二叉树



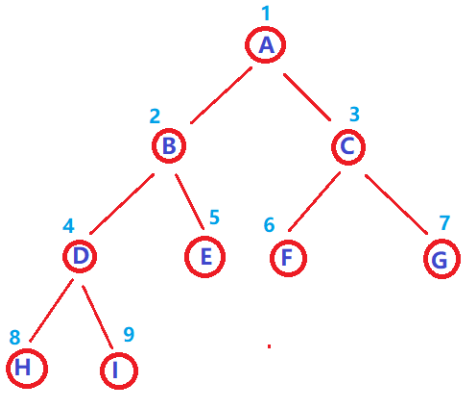
4、

答案解析： C
注意：一般完全二叉树编号方式都是从根为0开始的，见课件二叉树性质位置。
此题完全二叉树根的编号是从1开始的，通过观察可得：某节点下标为i时，其双亲节点的编号为i/2
该公式不用记，用到时忘记了画个图推导下就出来了，故选择C



图A

假设完全二叉树中有n个节点：按照层序的方式，根据点从0开始编号
当节点编号i不等于0时，该节点的双亲为(i-1)/2，如果i=0，则i为根节点
当节点编号为i时，若2*i+1 < n 则编号为i的节点的左孩子为：2*i+1
当节点编号为i时，若2*i+2 < n 则编号为i的节点的右孩子为：2*i+2



图B

假设完全二叉树中有n个节点：按照层序的方式，根节点从1开始编号
当节点编号i不等于1时，该节点的双亲给i/2，如果i=1，则i为根节点
当节点编号为i时，若2*i < n 则编号为i的节点的左孩子为：2*i
当节点编号为i时，若2*i+1 < n 则编号为i的节点的右孩子为：2*i+1

注意：上课中讲的二叉树性质5，根节点是从0开始编号的，即图A的方式，一般的编号规则都是和图A一致的
但是有些选择题可能是按照图B的方式编号的

5、

答案解析：B
先序遍历：根左右；后序遍历：左右根
要满足题意，则只有,根左<----->左根，根右<---------->右根
所以高度一定等于节点数
综上： 选择B选项

二、编程题

1、

【解题思路】：

此题主要考察二叉树层次遍历的应用，使用队列进行层次遍历，在遍历的过程中进行判断，一旦不满足奇偶树的条件，就可立即返回false,如果满足，继续层次遍历，直至整棵树层次遍历完成。

【代码实现】：

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     struct TreeNode *left;
 *     struct TreeNode *right;
 * };
 */
#define N 100002
bool isEvenOddTree(struct TreeNode* root)
{
    //申请队列空间，使用数组模拟队列
    struct TreeNode *queue[N];

    int front = 0;
```

```
    int rear = 0;
    int odd = 0;   //odd用于判断奇偶层[0:偶数层，1:奇数层]
    int pre;      //表示当前行判断节点的前驱节点

    queue[rear++] = root;
    while(front != rear)
    {
        int cnt = rear - front;
        if (odd == 1)
            pre = INT_MAX;
        if (odd == 0)
            pre = INT_MIN;

        for (int i = 0; i < cnt; i++)
        {
            root = queue[front++];
            //偶数行判断
            if ((odd == 1) && (root->val % 2 != 0 || pre <= root->val))
                return false;

            //奇数行判断
            if ((odd == 0) && (root->val % 2 != 1 || pre >= root->val))
                return false;

            pre = root->val;
            if (root->left)
                queue[rear++] = root->left;
            if (root->right)
                queue[rear++] = root->right;
        }

        //控制奇偶行标记
        odd = (odd + 1) % 2;
    }

    return true;
}
```


2、

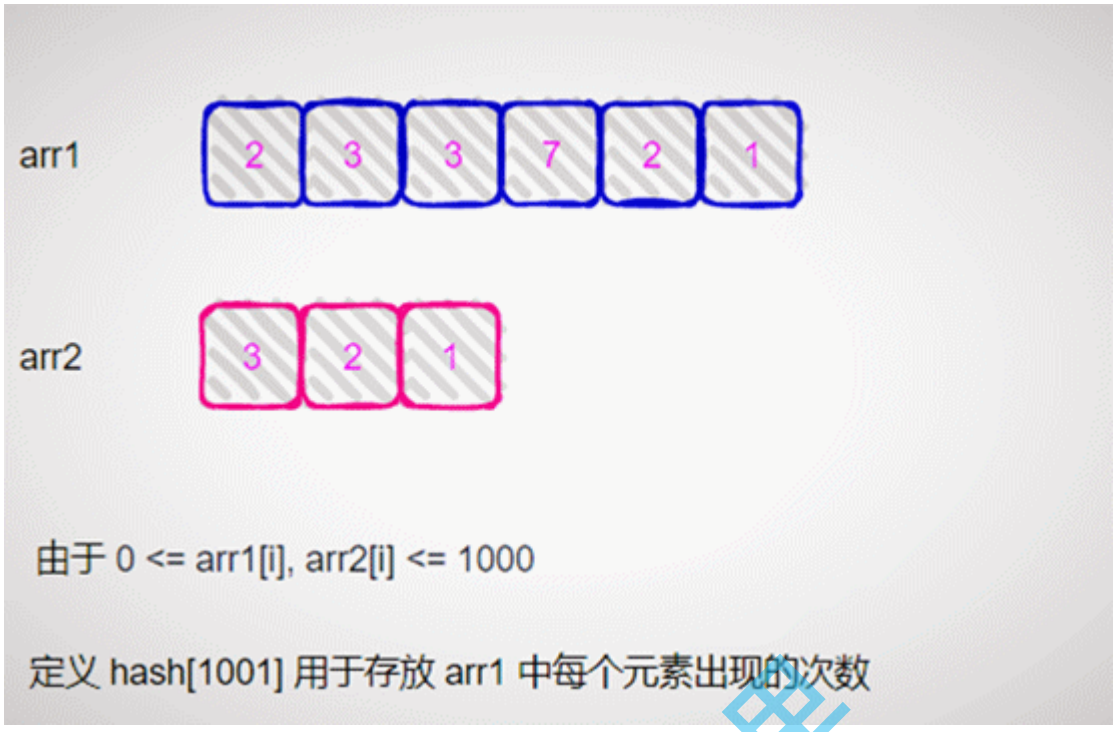
【解题思路】：

此题主要考察数组的排序操作，其思路为:

1、记录`arr1`数字出现的次数

2、找到`arr2`和`arr1`都出现的数字

3、找`arr1`有，`arr2`没有的。 以`arr1 = [2,3,3,7,2,1], arr2 = [3,2,1]`为例子，如下图示



【代码实现】：

```
/**
 * Note: The returned array must be malloced, assume caller calls free().
 */
int* relativeSortArray(int *arr1, int arr1Size, int *arr2, int arr2Size,
                       int *returnSize)
{
    int arr[1001]={0};

    //记录arr1数字出现的次数次数
    for(int i=0; i<arr1Size; i++)
    {
        arr[arr1[i]]++;
    }

    int j = 0;
    //找到在arr2和arr1都出现的数字,更新arr1
    for(int i=0; i<arr2Size; i++)
    {
        while(arr[arr2[i]] > 0)
        {
            arr1[j] = arr2[i];
            j++;
            arr[arr2[i]]--;
        }
    }

    //找arr1有，arr2没有的
    for(int i=0; i<1001; i++)
    {

        while(arr[i] > 0)
```

```
        {
            arr1[j++]=i;
            arr[i]--;
        }
    }

    *returnSize=arr1Size;
    return arr1;
}
```

day16

一、选择题

1、

```
答案解析： D
1.当二叉树为完全二叉树时， 深度最小。
2.根据二叉树的性质“若规定根节点的层数为1，具有n个结点的满二叉树的深度，h=log2(n+1), 计算log2(1001)约等于9.97
3.另一种计算方式： 当第一层深度为1时， 深度为n的满完全二叉树节点为2^n - 1。
所以： 2^n - 1 >= 1000; 则n >= 10;
综上： 选择D选项
```

2、

```
答案解析：C
n0为叶子结点,n1为度为1的结点,n2为度为2的结点
1. 定理n2 = n0 -1;
2. 因为该树的节点数为偶数，所以n1必为1.
所以由以上可得：n0+n1+n2 = 626  =>  n0 + n0-1 + 1 = 626 =>n0 = 313
综上：选择C选项
```

3、

```
答案解析：A
1.n0为叶子结点,n1为度为1的结点,n2为度为2的结点,n为总结点；则：n=n0+n1+n2.
已知n0=30,n1=20.二叉树中度为 2 的结点数目 = 叶子结点数目 - 1 ,所以n2=n0-1=29。n=30+20+29=79。所以总节点数为79
综上：选择A选项
```

4、

```
答案解析：C
1.树的度：节点的分叉数，二叉树的度只能为 0,1,2。记为n0, n1, n2
2. 完全二叉树: n0 + n1 + n2 = 1000，由于节点总数为偶数， 所以度为1的只有1个，
      即n0+1+n2 = n2+1+1+n2=1000
      2*n2=998  则最终求解得n2=499
综上：选择C选项
```

5、

答案解析： C
X=A+B*(C-D)/E
1)扫描X为数字,进行输出 X
2)扫描=为操作符,进栈,栈中元素为:=
3)扫描A为数字,进行输出A,输出的元素为X,A 栈中元素为:=
4)扫描+为操作符,由于+>=(栈顶)进栈,栈中元素为:+,=
5)扫描B为数字,进行输出B,输出的元素为X,A,B 栈中元素为:=
6) 扫描*为操作符,由于 *>+(栈顶) 进栈,栈中元素为:*,+,=
7)扫描(为操作符,进栈，栈中元素为:(, *,+,=
8)扫描C为数字,进行输出C，输出的元素为X,A,B,C 栈中元素为:(, *,+,=
9) 扫描-为操作符,进栈，栈中元素为:-,(, *,+,=
10)扫描D为数字,进行输出D，输出的元素为X,A,B,C,D 栈中元素为:-(, *,+,=
11)扫描)为操作符,进行弹出操作,直到遇见(为止,输出元素为:- ,总的 输出元素为X,A,B,C,D,- ， 栈中元素为: *,+,=
12) 扫描/为操作符,此时栈顶元素为*，由于*，/优先级相同，依次弹出直到遇见比/优先级低的为止,然后/进栈 输出元素为:*,总的输出为: 输出的元素为X,A,B,C,-,* 栈中元素为:/ ,+,=
13) 扫描E为数字,进行输出E，输出的元素E,总的输出元素为 X,A,B,C,D,-,*,E 栈中元素为: / ,+,=
14)到此扫描结束,依次弹出栈中剩余的元素,输出的的元素为 / ,+,=, 总的输出元素为 X,A,B,C,D,-,*,E, / ,+,=
综上：选择C选项

二、编程题

1、

【解题思路】：

此题主要考察字符串的操作， 题中明确指出几点：1、数字个数不超过9；2、无前导空格；3、无后缀空格；一次for循环遍历找到数字，并定义结构体存储这些数字中对应的字符串，类似hash的方式；因此需要定义一个结构体，结构体的大小可以为10（即数字个数不超过9）；结构体中存储字符串与字符串长度；for循环中找到数字n后，放到对应的结构体[n]中，保存字符串与字符串长度；最后直接遍历存储了多少个字符串即可。

【代码实现】：

```
typedef struct WordRecord
{
    char *p; //字符串指针，需要开辟数据空间
    int len; //字符串长度
}WordRecord;

char * sortSentence(char * s)
{
    //开辟10个空间，0下标不用
    WordRecord wr[10];

    int len = strlen(s);
    int i, j, maxValue, sIdx;
    i = j = maxValue = sIdx = 0;

    for (i=0; i < len; ++i)
    {
        //寻找数字

        int sentenceIdx = s[i] - '0';
```

```
        if (sentenceIdx >= 1 && sentenceIdx <= 9)
        {
            wr[sentenceIdx].len = i - sIdx + 1;
            wr[sentenceIdx].p = (char *)malloc(sizeof(char) * wr[sentenceIdx].len);
            for (j = 0; j < wr[sentenceIdx].len - 1; j++)
            {
                wr[sentenceIdx].p[j] = s[sIdx + j];
            }
            wr[sentenceIdx].p[j] = ' ';
            sIdx = i + 2; //跳过数字和空格

            //记录单词的最大个数
            maxValue = maxValue < sentenceIdx ? sentenceIdx : maxValue;
        }
    }
    for (i = 0, j = 1; j <= maxValue; j++)
    {
        for (int k = 0; k < wr[j].len; k++)
        {
            s[i++] = wr[j].p[k];
        }
        //释放字符串空间
        free(wr[j].p);
        wr[j].p = NULL;
    }
    s[i - 1] = '\0';
    return s;
}
```

2、

【解题思路】：

此题主要考察对数组的操作，需要用到排序和滑动窗口的思想。1、排序，我们只需要依次找到相邻两个连续相同元素的子序列，检查该这两个子序列的元素的之差是否为 1。2、双指针滑窗 ，begin 指向第一个连续相同元素的子序列的第一个元素，end 指向相邻的第二个连续相同元素的子序列的末尾元素，如果满足二者的元素之差为 1，则当前的和谐子序列的长度即为两个子序列的长度之和，等于end−begin+1。

【代码实现】：

```
int Cmp(const void *a, const void *b)
{
    return *(int*)a - *(int*)b;
}

// 排序 + 双指针滑动窗口
int findLHS(int* nums, int numsSize)
{
    //先对数组排序，从小到大
    qsort(nums, numsSize, sizeof(nums[0]), Cmp);
```

```
    int ret = 0;
    int begin = 0;
    for (int end = 1; end < numsSize; end++)
    {
        if (nums[end] - nums[begin] > 1)
        {
            //若差值大于1，则左指针向右移，缩小二者差值
            begin++;
        }
        if (nums[end] - nums[begin] == 1)
        {
            // 差值为1，计算长度并取最大值
            ret = ret > end-begin+1 ? ret : end-begin+1;
        }
    }
    return ret;
}
```