比特就业课假期作业-C语言作业答案

出题老师：

> C选择题：黄坤（day01-day16） qq：3587670086
>
> C编程题：张文超（day01-day16）qq：3627274478

作业说明：

> 1、本次作业涵盖内容为C语言相关知识点
>
> 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：C
swap函数调用时用的是全局变量，主函数中定义的变量只在主函数中有效，因为主函数也是一个函数，它与其他函数是平行关系；输出语句这里，考虑局部优先的原则，故选C
```

2、

```
答案解析：
正确答案：B
本题B选项考查转义字符，有如下格式，但八进制数字是0-7，没有8，故B选项中'\8'是错误的
\ddd    ddd表示1到3个八进制数    如：\130    转义为    字符X
\xhh    hh表示1到2位十六进制数    如：\x30    转义为    字符0
```

3、

```
答案解析：
正确答案：ACD
因为#define是宏定义，仅仅是直接替换,INT_PTR a, b; 进行宏替换后代码是这样的：int *a, b;这里的int *是a的类型，b的类型是int，故此次b只是int类型。而typedef是把该类型定义一个别名，别名是一个独立的类型了，使用这个类型创建的变量都是这个类型的。
所以 a，c，d才是指针类型
```

4、

答案解析：
正确答案：C
给定条件表达式(M)?(a++):(a--)。 (表达式1)? (表达式2): (表达式3)为三目运算符。
计算规则为：先判断表达式1是否为真，若为真，则计算表达式2，并将表达式2的结果作为整个表达式最终的结果，表达式3不计算；否则，表达式3的结果为最终结果，表达式2不计算。 在此表达式中，若M=0，为假，计算a--；若M≠0，为真，计算a++；若要求与M等价，则要满足M取0时为假，取非0数值时为真。 C选项中：假定M取0，则M表示假，当M是0时,表达式M!=0不成立，为假，计算a--；当M取非0数值时，M为真，表达式M!=0成立，为真，计算a++；符合题意，选C

5、

答案解析：
正确答案：AB
&c和c两个地址值是一样的，程序的效果相同，也没错，但同时也必须把变量b的地址给scanf,故CD错误，AB正确

二、编程题

1、【答案解析】：

这里首先要清楚n位数中最大的数字，实际上就是`10^n - 1`。 比如：

```
1位数:  10^1 - 1
2位数:  10^2 - 1
3位数:  10^3 - 1
...
```

这个清楚后动态申请空间，将数值填入就可以了，需要注意的是数组下标从0开始，而数值从1开始

```
int* printNumbers(int n, int* returnSize ) {
    *returnSize = pow(10, n) - 1; //确定最大的数字
    int *arr = (int *)malloc(sizeof(int)*(*returnSize));//申请足够大小的空间
    for (int i = 0; i < *returnSize; i++) {
        arr[i] = i+1;//下标从0开始，而数值从1开始
    }
    return arr;
}
```

2、【答案解析】：

这道题简单解法其实将每个月的天数枚举出来，然后根据当前月份向前累加满月的天数，然后再加上当前月所在的天数。最终考虑平闰年的`2`月份区别是否增加一天。

其中需要注意的是平年和闰年的判断，而且是闰年的月份大于`2`的时候，也就是`2`月走完，总天数才能加`1`（比如`2000年2月18日`，虽然是闰年，但是`2月`都没走完那是不能加上闰年多出的一天的）.

```
#include <stdio.h>
int is_leap_year(int year) {
    if (year % 4 == 0 && year % 100 != 0 || year % 400 == 0){
        return 1;
    }
    return 0;
}
```

```c
int main()
{
    int month_day[] = {0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};
    int year, month, day;
    while (~scanf("%d %d %d", &year, &month, &day)){
        int total_day = day;//先把当前月份天数加上
        if (is_leap_year(year) && month > 2) { //若闰年，且月份大于2月，则在平年基础上+1
            total_day += 1;
        }
        for (int i = month - 1; i > 0; i--) {
            total_day += month_day[i];//向前累加每月的天数即可
        }
        printf("%d\n", total_day);
    }
    return 0;
}
```

day02

一、选择题

1、

答案解析：
正确答案：A
这里考查转义字符，注意：\\ 表示字符'\'，\123表示字符'{'，\t表示制表符，这些都是一个字符

2、

答案解析：
正确答案：B
宏只是替换，替换后NUM的样子是(2+1+1)*2+1/2，计算得8

3、

答案解析：
正确答案：C
此题注意静态局部变量的使用，static改变了i的生命周期，第一次调用函数：i初值是1，递归第二次调用函数时，i还是第一次那个变量，值已经变成了2，再一次调用函数时i就是3，依次类推

4、

答案解析：
正确答案：B
const在*的左边，则指针指向的变量的值不可直接通过指针改变(可以通过其他途径改变)；在*的右边，则指针的指向不可变。简记为"左定值，右定向"，(1)和(2)const都在*的左边，(3)中const在*的右边，所以应该选择B。

5、

答案解析：
正确答案：D
A选项，没有考虑内存对齐。B选项，考察double类型的比较，由于浮点数存在误差，不能直接判断两个数是否相等，通常采用比较两数之差的绝对值是否小于一个很小的数字（具体的可自己设定这样一个数，作为误差）来确定是否相等。C选项，a为数组首地址是常量不能改变，所以A,B,C都是错的，选择D

二、编程题

1、【答案解析】：

这道题的关键在于知道规律后，能够找到第`n`个数据立方的起始奇数，从这个起始奇数开始，组成连续的n个奇数项之和的表达式即可。

比如：`3^3`的起始奇数是`7`， 则`{7, 9, 11}`3个奇数求和表达式`7 + 9 + 11`。

而起始奇数有个规则：`m^3`的起始奇数值等于`m * (m - 1) + 1`

奇数起始项规律：
首先所有奇数项构成一个差值为2的等差数列， 1 3 5 7 9 ....
其次，1的起始奇数是第1个等差数列项，2的起始奇数是第2个等差数列项，3的起始奇数是第4个等差数列项...
形成规律： 1 2 4 7....，而他们的差值分别是1 2 3 4 5...，所以第n项就是一个从1开始到n-1的等差数列之和+1

因此当有了需求m的立方，首先计算他的第一个奇数项是总体的第几个 。
等差数列求和公式 Sn=n(a1+an)/2 m * (m - 1) / 2
等差数列第n项公式 an=a1+(n-1)d 1 + ((m * (m - 1) / 2) + 1 - 1) * 2
最终得到m的立方的表达式起始奇数： m * (m - 1) + 1

```c
#include <stdio.h>
int main()
{
    int m;
    while(~scanf("%d", &m)){
        int start = m * (m - 1) + 1;//找到对应m^3的起始奇数
        char buf[10240] = {0};
        //sprintf(buf, format, ...) 与printf用法类似，格式化字符串但是不用于打印而是放到一个buf中
        sprintf(buf, "%d", start);//先将起始奇数转换成为字符串存入buf中
        for (int i = 1; i < m; i++) {
            //然后将紧随随后的m-1个奇数数字转换为字符串，按照指定格式放入buf中
            //%s+%d，要求先有一个字符串，然后是+符号，然后是个数字的格式，对应是buf原先的数据，和奇数
            sprintf(buf, "%s+%d", buf, start+=2);
        }
        printf("%s\n", buf);
    }
    return 0;
}
```

2、【答案解析】：

这道题了解了等差数列求和公式`Sn=n(a1+an)/2`就简单了，根据题目得知`a1=2`，而等差数列第n项也有具体公式`an=a1+(n-1)d`，而公差为3， 这时候只需要套入公式计算即可。

```c
#include <stdio.h>
int main()
{
    int n, a1 = 2;
    while(~scanf("%d", &n)){
        int an = a1 + (n - 1) * 3;//等差数列第n项计算
        printf("%d\n", n * (a1 + an) / 2); //等差数列求和打印
    }
    return 0;
}
```

day03

一、选择题

1、

答案解析：
正确答案：A
参数a是指针，要接收地址，BD错误。参数b可以接收的是char*，而&c的类型是char(*)[10]，C错误

2、

答案解析：
正确答案：ABCD
如果 const 位于 * 的左侧，则 const 就是用来修饰指针所指向的变量，即指针指向为常量；*c和*d不能变。
如果 const 位于 * 的右侧，则 const 就是修饰指针本身，即指针本身是常量；e和f不能变。

3、

答案解析：
正确答案：A
全局变量i，在main()中修改为5，第一次在prt()中执行循环输出三次'*'，i被修改为8，回到main()中第二次调用prt()时，i<8为假，循环结束没输出，执行一次print("\t")，再次回到主函数后i++变为9，i<=8为假，循环结束；

4、

答案解析：
正确答案：D
a+=a-=a*a等价于a=a+(a=a-a*a)，即先计算a=a-a*a，所以此时a的值为3-3*3=-6，再计算-6+(-6)=-12赋值给a，所以a的值为-12，也就是整个表达式的值，所以应选择D

5、

答案解析：
正确答案：D
只有条件为真时才进行循环，ABC中1为真，D中0为假

二、编程题

1、【答案解析】：

这道题其实通过`scanf`捕捉数据即可，统计负数个数，以及正数格式，并且在统计正数个数的过程中求取正数总和，最后计算得出平均数即可。需要注意的是所有数字中0是不统计在内的。

```
#include <stdio.h>
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        int count1 = 0, count2 = 0, tmp;
        float sum = 0;
        for (int i = 0; i < n; i++) {
            scanf("%d", &tmp);
            if (tmp < 0) {
                count1++;   //统计负数个数
            } else if (tmp > 0) {
                sum += tmp; //正数求和
                count2++;   //统计大于0的正数个数，这样是因为题目说明0不算在内
            }
        }
        printf("%d %.1lf\n", count1, sum / count2);
    }
    return 0;
}
```

2、【答案解析】：

暴力破解：遍历数组找出最小值即可

更优思想：采用二分查找，这个题主要分析三种旋转情况 [1, 2, 3, 4, 5]，使用中间值与右端进行比较。

1. 中间大于右边 [3, 4, 5, 1, 2]，这种情况下，最小数一定在右边；则left = middle + 1
2. 中间等于右边 [1, 0, 1, 1, 1], 这个是[0, 1, 1, 1, 1] 旋转过来的，这时候需要缩小范围 right--;，注意不能是 left++，因为是非降序数组，所以要缩小右边范围，把较小值向右推，符合我们的判断规则。
3. 中间小于右边 [5, 1, 2, 3, 4], 这种情况下，最小数字则在左半边；则right = middle

```
int minNumberInRotateArray(int* rotateArray, int rotateArrayLen ) {
    if (rotateArrayLen == 0) return 0;
    int left = 0, right = rotateArrayLen - 1, mid;
    if (rotateArray[right] > rotateArray[left]) return rotateArray[0];
    while(left < right) {
        mid = left + (right - left) / 2;
        if (rotateArray[mid] > rotateArray[right]) left=mid+1;
        else if (rotateArray[mid] == rotateArray[right]) right--;
        else right = mid;
    }
    return rotateArray[left];
}
```

day04

一、选择题

1、

答案解析：
正确答案：D
对于for循环，其中第一项初始化表达式只执行一次，因此ch只从输入流中取一个字符，之后就再不会取字符，因此会死循环

2、

答案解析：
正确答案：C
此题技巧是耐心，考查while循环和循环嵌套的理解，初值m=65,n=14；循环1判断m!=n为真，来到循环2判断m>n为真，执行m=m-n;直到m=9,n=14；循环2结束来到循环3判断n>m为真，执行n=n-m;直到m=9,n=5；循环3结束回到循环1，如此往复直到m==n时，循环结束

3、

答案解析：
正确答案：D
代码switch语句中没有break，则每次找到入口进入后，顺序执行到代码块结束为止。例如当c为'A'时，从case 'A'进入，先后执行v1+=1;v0+=1;v2+=1;，而当c为'p'时，从default进入，先后执行v0+=1;v2+=1;，容易看出最终v0和v2是相等的

4、

答案解析：
正确答案：B
这是一个升序插入排序算法，读完即懂，第i次排序时，t=a[i]作为临时变量保存这一次待插入值，j=i-1故而while循环中j是从下标i的前一项开始向下标0遍历，判断t<a[j]为真时a[j+1]=a[j]，j+1在遍历之初是等于i的，也就是将下标i位置用前边较大的值覆盖，依次把前边的元素后移，直到a[j]不大于t的时候将t插入到下标j+1位置，使得前i个元素达到有序，方便第i+1次排序操作，所以第i次排序时前边i-1个元素都是有序的

5、

答案解析：
正确答案：C
外循环有n次，当i=0,内循环为n次,当i=1,内循环为n-1次,当i=2时，内循环为n-2次，以此类推，总次数为n+(n-1)+(n-2)+.....+2+1,就是个等差数列，等于n(n+1)/2

二、编程题

1、【答案解析】：

使用标记的方式就可以找出重复的数字，数组中出现过哪个数字就把对应数字作为下标在对应位置1，表示已经标记出现过，如果哪个数据对应位已经置1，则表示就是重复的数字。有了重复的数字，拿`[1, n]`的总和减去去掉重复数据的数组总和就是丢失的数据。 其实使用标记法时出现的数字对应位每次`++`，则最后出现0次的就是丢失，出现2次的就是重复的，这样的方式也可以，不过需要多次遍历。

```
int* findErrorNums(int* nums, int numsSize, int* returnSize){
    *returnSize = 2;
    //遍历nums数组，将其中数据对应的位置1， 哪一位如果已经重置过则意味着数据重复了
    int *arr = (int *)calloc(numsSize + 1, sizeof(int));//申请numsSize个整形空间，并初始化为0
    int *ret = (int *)calloc(*returnSize, sizeof(int));//申请2个整形空间，并初始化为0
    int cur_sum = 0, old_sum = 0;
    for (int i = 0; i < numsSize; i++) {
        if (arr[nums[i]] == 1) { //这个数字在上边数组的对应位置已经置过1了，则重复
            ret[0] = nums[i];//找到重复的数字
        }
        arr[nums[i]] = 1; //将标记数组的对应数据位置1
        old_sum += i + 1; // 1~n的求和
        cur_sum += nums[i]; //当前数组中的数据求和（多了一个重复的，少了一个丢失的）
    }
    ret[1] = old_sum - (cur_sum - ret[0]);//原始总和，减去去掉重复后的当前总和就是丢失的数字
    free(arr);
    return ret;
}
```

2、【答案解析】：

这道题只需要将字符串从头到尾的每种字符（大写字符，小写字符，数字，其他字符）分别统计出来后。然后逐个判断是否符合条件即可。而条件的判断包含有：

- 长度不小于8
- 不能以数字开头
- 只能包含字母和数字
- 大小写和字符必须具备两种以上

```
#include <stdio.h>
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        for (int i = 0; i < n; i++) {
            char password[101] = {0};
            int upper = 0, lower = 0, digit = 0, other = 0;
            scanf("%s", password);//捕捉输入的密码
            if (strlen(password) < 8) {//密码长度小于8
                printf("NO\n");
                continue;
            }
            if (password[0] >= '0' && password[0] <= '9') {//密码以数字开头
                printf("NO\n");
                continue;
            }
            char *ptr = password;
            while(*ptr != '\0') {  //统计各种字符个数
                if (*ptr >= 'a' && *ptr <= 'z') lower++;
                else if (*ptr >= 'A' && *ptr <= 'Z') upper++;
                else if (*ptr >= '0' && *ptr <= '9') digit++;
                else other++;
```

```
                ptr++;
            }
            if (other > 0) { // 有其他字符（注意：密码只能由数字和字母组成）
                printf("NO\n");
                continue;
            }
            //大写，小写，数字，必须具有两种以上，而比较运算真则1，假则0
            if ((upper>0) + (lower>0) + (digit>0) < 2) { // 密码只有一种字符
                printf("NO\n");
                continue;
            }
            printf("YES\n");
        }
    }
    return 0;
}
```

day05

一、选择题

1、

```
答案解析：
正确答案：D
一个字母对应的大写和小写之间的ASCII码值相差32，而且小写的大于大写的。所以题中'e'和'E'之间的ASCII码值相差32（ch[j]+'e'-'E'相当于ch[j]+32）。一个字母从大写转化为小写就是在它自身上+32，小写转大写则是-32
```

2、

```
答案解析：
正确答案：B
因print(“*”)函数调用的返回值是字符串中字符的个数，即为1。所以while后面的条件恒为真，所以循环控制表达式与'0'是等价的（字符'0'不是0）。正确答案是B
```

3、

```
答案解析：
正确答案：C
程序首先考虑ch的ASCII码值是不是奇数，再看是不是小写字母，同时满足时被改为大写字母
```

4、

```
答案解析：
正确答案：D
D选项与众不同，其他都是a==0时输出y，a!=0时输出x
```

5、

答案解析：
正确答案：ABCD
此题旨在整理跳出多层循环的方法，每个选项都是正确的，代码为伪代码，condition代表逻辑表达式

二、编程题

1、【答案解析】：

采用遍历也能搞定，不过数组为非降序，采用二分查找的思想最优，先二分找到最左边的数字位置，再二分查找最右边的数字位置，两个位置相减+1就是长度了

中间比找的值大：则要找的数字肯定在右边， left = mid + 1;

中间比找的值小：则要找的数字肯定在左边， right = mid - 1;

中间值与找的值相同：

- 找的最左边数字：如果mid就是left，则返回mid就行，否则重置right=mid-1，把中心不断向左偏移
- 找的最右边数字：如果mid就是right，则返回mid就行，否则重置left=mid+1，把中心不断向右偏移

```
int get_last_or_first_idx(int *data, int len, int k, int flag) {//flag:0-找左边， 1-找右边
    int left = 0, right = len - 1, mid;
    while(left <= right) {
        mid = left + (right - left) / 2;
        if (data[mid] > k)
            right = mid - 1;
        else if (data[mid] < k)
            left = mid + 1;
        else {
            if(flag == 0) {//flag==0时，找最左边的数字
                if(mid == left || data[mid-1] != k) return mid;
                else right = mid - 1;//把中心向左推
            }else {//flag==1时，找最右边的数字
                if (mid == right || data[mid+1] != k) return mid;
                else left = mid + 1;//把中心向右推
            }
        }
    }
    return -1;
}
int GetNumberOfK(int* data, int dataLen, int k ) {
    if (dataLen == 0) return 0;
    int left = get_last_or_first_idx(data, dataLen, k, 0);
    int right = get_last_or_first_idx(data, dataLen, k, 1);
    if (left == -1 && right == -1) return 0; //表示没有找到k这个数据
    return right - left + 1;
}
```

2、【答案解析】：

其实问需要修改多少个比特位，问的就是有多少个比特位不同而已，因为有多少位不同就修改多少位。

```
int get_bin_count(int num) {
    int count = 0;
    for (int i = 0; i < 32; i++) {
        if ((num >> i) & 1)
            count++;
    }
    return count;
}
int convertInteger(int A, int B){
    return get_bin_count(A^B);
}
```

day06

一、选择题

1、

答案解析：
正确答案：A
break语句通常用在循环语句和switch语句中。当break用于switch语句中时，可使程序跳出switch而执行switch以后的语句；当break语句用于do-while、for、while循环语句中时，可使程序终止循环而执行循环后面的语句，即满足条件时便跳出循环。continue语句的作用是跳过循环体中剩余的语句而强行执行下一次循环。B、C和D三个选项中均有错误。因此A选项正确

2、

答案解析：
正确答案：D
逻辑或运算如果前表达式为真，后表达式不计算，第一次循环时i为0，执行i++，第二次循环时i为1，是个真值，不再执行i++，也就死循环了

3、

答案解析：
正确答案：C
do-while循环中的循环体通常都是复合语句代码块，A错误，while(表达式)后面要写分号，B错误，while不能省，D错误

4、

答案解析：
正确答案：D
A选项数组传参只需要写数组名就行，a[]时错误的，B选项第二个参数写成了大写，错了。C选项第二个参数是浮点数，但是fun函数的第二参数是数组不匹配，fun函数参数x需要传一个数组或者float *指针，只有D选项的形式是正确的。

5、

答案解析：
正确答案：A
一个函数不写返回值类型，默认的返回类型是int，但不提倡这么做

二、编程题

1、【答案解析】：

暴力破解：双重循环遍历数组，对每个元素判断是否是其他元素的两倍。或者先遍历一遍找出最大值，然后遍历一遍判断是否是其他数字二倍。

更优思想：一次遍历找出最大的数字和次大的数字，判断最大的数字是否是次大数字2倍即可。

```
int dominantIndex(int* nums, int numsSize){
    if (numsSize == 1) return 0;//特殊情况只有一个元素则特殊处理
    int max, sec, idx;
    //先对最大和次大进行选择赋值，注意max和sec不能随意赋初值，因为有可能你赋予的初值就是最大值
    //因此要使用数组中的数据进行初值赋予。
    if (nums[0] > nums[1]) {
        max = nums[0];
        idx = 0;
        sec = nums[1];
    }else {
        max = nums[1];
        idx = 1;
        sec = nums[0];
    }
    for (int i = 2; i < numsSize; i++) {
        if (nums[i] > max) {     //当前元素大于max，则意味着要替换
            sec = max;           //先将原先最大的保存到sec中，则他就是次大的
            max = nums[i];       //再将最大的放到max中
            idx = i;             //保存max值的下标
        }else if (nums[i] > sec){
            //避免刚好nums[0]就是最大的情况，因为不切换最大而导致次大无法判断情况
            sec = nums[i];
        }
    }
    if (max >= sec * 2) {
        return idx;
    }
    return -1;
}
```

2、【答案解析】：

暴力破解即可，将`nums1`数组中的每一个数字，判断是否存在于`nums2`数组中，通过这种方式找出交集数据，找出之后判断这个数组是否已经在返回数组中存在，不存在则添加到返回数组中即可。

```
int* intersection(int* nums1, int nums1Size, int* nums2, int nums2Size, int* returnSize){
    static int arr[1000];
    *returnSize = 0;
    int i, j, k;
```

```
    for (i = 0; i < nums1Size; i++) {
        for (j = 0; j < nums2Size; j++) {
            if (nums2[j] == nums1[i]) break;//判断nums1[i] 是否在nums2数组中
        }
        if (j == nums2Size) {// nums1中i位置的数据在nums2数组中不存在，则非交集数据
            continue;
        }
        //只有在另一个数组中存在的数据才能走下来，判断是否已经被添加到返回数组中
        for (j = 0; j < *returnSize; j++) {
            if (nums1[i] == arr[j]) break;//判断nums1[i] 是否在 arr 这个返回数组中
        }
        if (j == *returnSize) {//不在返回数组中，则添加到返回数组中
            arr[*returnSize] = nums1[i];
            *returnSize += 1;
        }
    }
    return arr;
}
```

day07

一、选择题

1、

答案解析：
正确答案：AB
主函数中定义的局部变量只在主函数中有效，因为主函数也是一个函数，它与其他函数是平行关系，C错误；当函数有返回值时，可以出现在表达式中，D错误

2、

答案解析：
正确答案：A
在调用函数的时候，真实传递给函数的是实参，函数定义部分函数名后的参数是形参。形参和实参的名字是可以相同的，在函数调用的时候，形参是实参的一份临时拷贝，分别占用不同的内存空间，所以A正确，B错误，及时形参和实参的名字相同，也是占用不同的内存空间，所以B错误；函数如果不被调用时，函数的形参是形式上存在的，但是函数在被调用的时候，形参是要分配内存空间的，所以D错误。

3、

答案解析：
正确答案：A
代码实现了递归倒序打印字符串的功能，但是++s使得s的值发生了变化，回不到'G'的位置上，故而没有打印'G'

4、

答案解析：
正确答案：B
函数f是没有返回值的，不能给int类型变量赋值，故A错误，同时需要一个整型参数，C中f(9)不能作为f的参数，也是错的，D选项没有传参，也不能接收返回值也是错误的。

5、

答案解析：
正确答案：C
代码是一个递归函数，计算x+(x-1)+(x-2)+...+2+1即等差数列的和

二、编程题

1、【答案解析】：

这道题考察的其实就是字符排序，每个`ascii`字符在内存都有一个对应的`ascii`值，通过内存中数据的存储进行排序就行。

冒泡排序：相邻数据之间进行比较交换，将较大或较小的数据向后推到数组末尾，然后开始下一轮次大数据的冒泡过程。

```
#include <stdio.h>
int main()
{
    char str[1024] = {0};
    while(gets(str)) {
        int len = strlen(str);
        for (int i = 0; i < len; i++) {
            for (int j = 1; j < len - i; j++) {
                if (str[j] < str[j - 1]) {
                    char ch = str[j - 1];
                    str[j - 1] = str[j];
                    str[j] = ch;
                }
            }
        }
        printf("%s\n", str);
    }
    return 0;
}
```

2、【答案解析】：

从数组的0下标处开始向后逐下标统计，计算当前下标左边之和，和右边之和，进行判断，相等则为中心下标，如果数组循环结束都没有找到中心下标，则返回-1，表示没有中心下标。

```
int pivotIndex(int* nums, int numsSize){
    int i, j;
    for (i = 0; i < numsSize; i++) {//从假设中心点为0开始进行统计判断
        int l_sum = 0, r_sum = 0;//初始化左边之和和右边之和为0
        for (j = 0; j < numsSize; j++) {
            if (j < i) l_sum += nums[j]; //小于i坐标的都是左边的数字
```

```
            else if (j > i) r_sum += nums[j];//大于i坐标的都是右边的数字
        }
        if (l_sum == r_sum) {//如果两遍相等则i就是中心坐标
            return i;
        }
    }
    return -1;
}
```

day08

一、选择题

1、

答案解析：
正确答案：D
字符串的结束标志是'\0'，而'\0'的ASCII值是0，而c[2]被初始化为0，就相当于是'\0'，故字符串打印的内容只有"ab"

2、

答案解析：
正确答案：D
本题主要考虑数组越界访问的情况，二维数组的行和列都是从0开始的，对于a数组来说，行下标最大是1，列下标最大是2，D选项中1>2表达式的值是0，是正确的，其他选项行和列都可能存在越界，A是行越界，B是行和列都越界，C是列越界。

3、

答案解析：
正确答案：D
D中的'5'是一个字符常量，不能给字符型数组a初始化

4、

答案解析：
正确答案：AC
A选项：宏替换，没问题；B选项：非法定义，一维数组必须定义数组元素个数；C选项：字符'0'，转换成十进制为48，所以该选项最终为int num[48]；D选项：错误，数组定义下角标不能为变量，注：C99标准中支持了使用变量，这里不做特殊考虑。

5、

答案解析：
正确答案：BC
本题考查的是二维数组的元素访问，A选项是 正确的，X[i]就是第i行的数组名，数组名表示首元素的地址，X[i]表示第i行的第一个元素的地址，+j后就是第i行下标为j的元素的地址，整体解引用就是X[i][j]，A正确。B选项因为[]的优先级高于*，所以代码相当于**((X+i)+j),X+i+j后就越界了，并不代表X[i][j],所以错误。C选项也明显不对，X是二维数组的数组名，数组名相当于第一行的地址，X+i+j，跳过了i+j行，就越界了，C错误。D选项是标准的指针形式访问二位数组的一个元素。

二、编程题

1、【答案解析】：

这道题思路比较简单，因为题目圈定出现的字符都是`ascii`值小于127的字符，因此只需要定义一个标记数组大小为`127`，然后将字符作为数组下标在数组中进行标记，若数组中没有标记过表示第一次出现，进行计数，否则表示重复字符。

示例：查表法，`"aca"`，首先把a字符(`ascii`值为`97`)作为下标，将标记数组的第`97`位置`1`，下次如果还有`a`字符出现，到下标`'a'`或者`97`的位置一看是1就表示a已经统计过了。

```
#include <stdio.h>
int main()
{
    char tmp[501] = {0};
    while(~scanf("%s", tmp)) {
        char table[128] = {0}, *ptr = tmp;
        int count = 0;
        while(*ptr != '\0') {
            if (table[*ptr] != 1) {//判断字符ascii值作为下标的位置是否被标记过，是否是重复字符
                count++;    //当前字符的位置没有被标记过表示没有出现过，则计数+1
            }
            table[*ptr++] = 1;//将字符ascii值作为下标的位置进行标记置1
        }
        printf("%d\n", count);
    }
    return 0;
}
```

2、【答案解析】：

一个数组中有一个数字出现次数大于`n/2`，从第`0`个字符开始，假设它就是最多的那个数字，遇到相同的数字则计数`+1`， 遇到不同的则计数`-1`，其实就是互相消耗，等到计数为`0`的时候，表示本次互拼完毕，从下一个字符重新开始互拼，但是归根结底出现次数大于`n/2`的这个数字数量更多，因此也是最后保留的字符。

示例：`"23335"`首先从字符`2`开始计数`1`，遇到`3`，不同则`-1`，互拼消耗 重新从剩下的`"335"`开始的过程，这时候保存的字符为`3`，遇到`3`则计数`+1`， 遇到5则计数`-1`，在计数不为`0`时，走到末尾保存的字符就是个数超过`n/2`的字符

```
int majorityElement(int* nums, int numsSize){
    int count = 1;
    int tmp = nums[0];
    for (int i = 1; i < numsSize; i++) {
        if (tmp == nums[i]){//与保存的字符相同则计数+1
            count++;
        } else {//与保存的字符不同则计数-1
            count--;
            //计数为0表示有可能保存的字符不是最多的字符，换下一个
            if (count == 0) tmp = nums[i + 1];
        }
    }
```

```
    return tmp;
}
```

day09

一、选择题

1、

答案解析：
正确答案：D
p是一个指针数组，p[i] = &a[i*3]相当于是把数组a每3个一组分开并把每组的首地址存在p数组，此时p类似一个4行3列的二维数组，p[3][2]就是4行第3个元素12

2、

答案解析：
正确答案：A
假设每行有n个元素：那x[9][9]元素的地址 - x[4][4]元素的地址 = 0x21c-0x140=5n+5(21c和140是地址末三位的十六进制数)，这里n是43，假设x[7][7]的地址是z,x[7][7]元素的地址 - x[4][4]元素的地址 = z-0x140 = 3n+3，z = 3n+3+140 = 3*43+3+0x140 = 0x84+0x140 = 0x1c4，看地址的尾数，选择A

3、

答案解析：
正确答案：A
sizeof是C语言中的一个操作符，不是函数调用，简单的说其作用就是返回一个对象或者类型所占的内存字节数，结果是无符号整数，因此可以把它看作是整型表达式。所以选择A

4、

答案解析：
正确答案：A
变量a里边存的是字符'a'，第一次输出先加加再输出，输出的是'b';第二次输出的时候，a先赋值再加加，赋值给b的就是a原来的值，输出b的时候的还是'b'

5、

答案解析：
正确答案：A
逗号表达式是从前到后依次计算子表达式，而其结果是最后一项的值，此题去掉括号后的表达式，和原表达式是等价的，先计算4*5并赋值给x，x变为20，中间x*5并没有改变x的值，最后一项x+5值是25，也就是整个表达式的值

二、编程题

1、【答案解析】：

自除数的判断，数字的每一位都能被源数字整除，有了这个规则，咱们只需要循环获取一个数字的每一位，然后与源数字取模判断是否为`0`，如果中间有任意一次不为`0`，则表示不是自除数。

接下来，只需要遍历数组中的每个元素，判断是否是自除数即可，如果是则加入到返回数组中。

```
int* selfDividingNumbers(int left, int right, int* returnSize){
    int *ret = (int *)calloc(1000, sizeof(int));//动态申请足够大的空间用于存放返回的自除数
    *returnSize = 0;
    for (int i = left; i <= right; i++) {
        int num = i;
        while(num) {
            int remainder = num % 10;//计算余数
            if (remainder == 0 || (i % remainder) != 0) {//判断i自身与余数取模是否为0
                break;
            }
            num /= 10;
        }
        //如果num==0表示通过了每一位数的取模判断，则i就是自除数
        if (num == 0) ret[(*returnSize)++] = i;
    }
    return ret;
}
```

2、【答案解析】：

暴力不考虑其他的因素的话，将所有数据乘积起来，然后遍历数组除以当前位置数据即可。

更优解法：将乘积分为两次进行，第一次先将每个位置左边的数据乘积计算出来放到返回数组中，后边第二次循环将对应位置右边的数据乘积计算出来与返回数组对应位置的左半边乘积相乘得到结果。

示例： 一个数组 `int nums[] = {2, 3, 4}`。

```
int left = 1, right = 1;
计算左侧乘积:
第0个元素的左边乘积，  arr[0] = left  然后计算第1位左侧乘积      left*=nums[0] -> left = 1*2
第1个元素的左边乘积，    arr[1] = left  然后计算第2位左侧乘积    left*=nums[1] -> left = 1*2*3
第2个元素的左边乘积，  arr[2] = left  然后计算第3位左侧乘积      已经没必要了，因为第2元素是末尾元素了
一次循环完毕后，返回数组中每个元素存储的都是自己左侧元素的乘积。    arr[]中的值:  [1, 2, 6]
计算右侧乘积:
第2个元素的右边乘积，  arr[2] *= right 然后计算第1位右侧乘积          right*=nums[2] -> right =1*4
第1个元素的右边乘积，  arr[1] *= right 然后计算第0位右侧乘积          right*=nums[1] -> right =1*4*3
第0个元素的右边乘积，  arr[0] *= right 然后计算第-1位右侧乘积    -1位已经不需要计算了
循环完毕后，返回数组中的每个元素都是其他元素的乘积了 arr[2]*=1;   arr[1]*=4;  arr[0]*=12
```

```
int* productExceptSelf(int* nums, int numsSize, int* returnSize){
    int *ret = (int *)malloc(numsSize * sizeof(int));
    *returnSize = numsSize;
    int left = 1, right = 1;
    //第一次循环，将当前位置左边的数字乘积填入返回数组中
    for (int i = 0; i < numsSize; i++) {
        ret[i] = left;// 1  nums[0]  nums[0]*nums[1]  num[0]*nums[1]*nums[2] ....
        left *= nums[i];
    }
    //第二次循环，对于返回数组的元素从后往前进行，每次乘以右边元素的乘积
```

```
    for (int i = numsSize - 1; i >= 0; i--) {
        ret[i] *= right; //最后一个成绩不需要乘以最后元素，乘以1就行
        right *= nums[i]; //right变化：1  nums[end]  nums[end]*nums[end-1] .....
    }
    return ret;
}
```

day10

一、选择题

1、

```
答案解析：
正确答案：C
x=x&(x-1)这个表达式执行一次就会将x的2进制中最右边的1去掉，在x变成0之前，表达式能执行几次，就去掉几个1，所以这个代码实现了求一个有符号整数二进制补码中1的个数的功能，我们知道-1的补码是全1，而int类型4个字节32位，选C
```

2、

```
答案解析：
正确答案：D
此题一个关键，有符号数右移运算高位是补符号位的，负数的符号位是1，所以x永远不会变为0，是个死循环
```

3、

```
答案解析：
正确答案：C
C选项中a/b是表达式，表达式计算的结果是一个值不能做左值
```

4、

```
答案解析：
正确答案：A
w<x?w:(y<z?y:z)加个括号应该就好理解了w<x为真，返回w，即表达式的值为1
```

5、

```
答案解析：
正确答案：A
k=(n=b<a)&&(m=a);这部分的执行顺序如下：先执行n=b<a部分，其中，关系运算符优先级高于赋值运算符，所以先算b<a，得到0，n=0赋值运算的结果将作为括号内表达式的结果，即(n=b<a)&&(m=a)转换成(0)&&(m=a)，&&运算前表达式为假，则后面的括号(m=a)不运算，m值还是0，最后，&&的结果是0，即k=0
```

二、编程题

1、【答案解析】：

十进制相加思想：`15+07`，先计算不考虑进位的相加结果`12`（因为`5+7`的不考虑进位的结果是`2`，遇`10`进位嘛），然后计算进位`5+7`进位是`10`，则`10`与`12`再次相加，得到`22`，进位为`0`，则计算到此结束。

这里使用二进制求和完成，思想类似，但是二进制计算相加和进位不需要使用 `+` 符号

二进制相加思想：与十进制相同，先计算不考虑进位的相加结果（`0+0`得`0`， `1+1`进位得`0`， `1+0`得`1`），使用异或可以取得； 然后计算相加的进位结果（同`1`的位置左移一位即可），使用相与后左移取得。

示例：

```
5 0101   +   7 0111
不考虑进位的相加结果    0101^0111 -> 0010
相加的进位              0101&0111 -> 0101   因为进位左移得到   1010
1010 + 0010
不考虑进位的相加结果   1010 ^ 0010 -> 1000
相加的进位             1010 & 0010 -> 0010   因为进位左移得到   0100
1000 + 0100
不考虑进位的相加结果   1000 ^ 0100 -> 1100
相加的进位             1000 & 0100 -> 0000   进位为0结束运算
```

```
int Add(int num1, int num2 ) {
    while(num2 != 0) {//进位不为0则持续与相加结果进行相加
        int tmp = num1 ^ num2;//得到每位相加不考虑进位的数据
        num2 = (num1 & num2) << 1;//同1的位相加则会进位
        num1 = tmp;
    }
    return num1;
}
```

2、【答案解析】：

`numsSize`大小的数组，其中每个元素的数据在`[1, numsSize]`区间之内，解法其实并不复杂，以数组元素的绝对值作为下标，将对应位置的数据置为负数，比如`0`号位置是`3`，则把`3`号位置的数据重置为负值，等到数组遍历重置完毕，只有缺失的这个数字对应的位置保留正数，其他出现过的数字位置都会是负数， 要注意不要重复设置负数，因为负负得正。

示例：

```
[2, 3, 3, 2, 4] 注意数组10个元素，值为[1-10]， 但是访问下标应该在[0-9]之内,因此修改位置下标应该是值-1
0号元素是2，则将1号位置置为对应负值 [2, -3, 3, 2, 4]
1号元素是3，则将2号位置置为对应负值 [2, -3, -3, 2, 4]
2号元素是-3，绝对值为3，将2号位置为负值，但是2号位已经重置过，不需要重置，否则会变正数[2, -3, -3, 2, 4]
3号元素是-2，绝对值为2，将1号位置为负值，但是1号位已经重置过，不需要重置，否则会变正数[2, -3, -3, 2, 4]
4号元素是4，则将3号位置置为对应负值 [2, -3, -3, -2, 4]
遍历数组得到0,4两个位置的数据是大于0的，因为人家数值从1开始，因此+1后得到1， 5两个缺失的数字
```

```
int* findDisappearedNumbers(int* nums, int numsSize, int* returnSize){
    for (int i = 0; i < numsSize; i++) {
        if (nums[abs(nums[i]) - 1] > 0)
            nums[abs(nums[i]) - 1] = -(nums[abs(nums[i]) - 1]);
    }
    int *ret = (int *)malloc(sizeof(int) * (numsSize));
    *returnSize = 0;
    for (int i = 0; i < numsSize; i++) {
```

```
        if (nums[i] > 0) {
            ret[*returnSize] = i + 1;
            *returnSize += 1;
        }
    }
    return ret;
}
```

day11

一、选择题

1、

答案解析：
正确答案：D
基本数据类型的等级从低到高如下：char int long float double运算的时候是从低转到高的，表达式的类型会自动提升或者转换为参与表达式求值的最上级类型

2、

答案解析：
正确答案：A
x是有符号数-1，内存中是全1，当有符号的x和无符号数进行比较时，x会隐式类型转换被当做无符号数，是一个很大的数，这时就选择A了

3、

答案解析：
正确答案：A
A选项，取余操作两边必须是整数

4、

答案解析：
正确答案：C
(1 << 31);左移31位，并在右侧填充0，得到0x80000000，即符号位为1，其他为0，即-2147483648
int k = 1^(1 << 31 >> 31);注意，这里在右移的时候，符号位保持为1，右移后填充1，结果为0xFFFFFFFF，即-1，0x00000001^0xFFFFFFFF，即0xFFFFFFFE(-2)

5、

答案解析：
正确答案：A
一般表达式的运算是在运行时执行的，而sizeof是一个编译阶段就执行的运算符，在其内的任何运算都不执行，只推测出其中表达式结果的类型求其大小，故前后i的值不变。

二、编程题

1、【答案解析】：

这道题思路比较简单，统计连续1的个数，遇到0时表示连续中断，判断如果当前的统计数大于之前最大的则替换，然后继续下一个位置开始的统计即可。

```
int findMaxConsecutiveOnes(int* nums, int numsSize){
    int max_count = 0, cur_size = 0;;
    for (int i = 0; i < numsSize; i++) {
        if (nums[i] == 1) {
            cur_size++;
        }else {
            max_count = max_count > cur_size ? max_count : cur_size;
            cur_size = 0;
        }
    }
    max_count = max_count > cur_size ? max_count : cur_size;
    return max_count;
}
```

2、【答案解析】：

这道题的关键在于完全数的判断：完全数指的是一个数字的所有约数的和和自身相等。我们只需要从`1`开始将这个数的约数相加求和即可。

约数就是能够被数字整除，而这里简化的一个思路是数字能够被整除，则除数和结果就都是约数，这种思路下，只需要从1计算到平方根即可

比如：数字`8`， 能够整除`2`，结果是`4`，则除数`2`和结果`4`都是约数，而这两个只需要一次计算判断即可。

需要注意的是`4，9，25...`这种，除数和结果相同的情况，则除数或者结果只相加一次就够了。

```
#include <stdio.h>
#include <math.h>
int is_perfect_num(int num)
{
    int sum = 1;
    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) {//判断是否能够整除i，能整除则i和结果都是约数
            sum += i;   //与除数相加
            if (i != sqrt(num))//防止除数和结果相同的情况下重复相加
                sum += num / i; //与相除结果相加
        }
    }
    if (sum == num) return 1;
    return 0;
}
int main()
{
    int n;
    while(~scanf("%d", &n)){
        int count = 0;
        for(int i = 2; i <= n; i++) {//对n以内的数字都进行判断是否是完全数，注意1不参与判断
            if (is_perfect_num(i)) count++;
        }
        printf("%d\n", count);
```

```
    }
    return 0;
}
```

day12

一、选择题

1、

答案解析：
正确答案：A
'0'<=c<='9'并非判断x大于等于字符0,小于等于字符9，而是先执行'0'<=c，使用这个表达式的结果再和'9'比较，'0'的ASCII码值是48，'A'的ASCII码值是'65'，故'0'<c是真值1，1无疑是小于字符'9'的，最终是真

2、

答案解析：
正确答案：B
unsigned short类型的x变量2个字节保存了65530，十六进制形式为0xFFFA，x给y赋值时会整型提升，而无符号数在提升时高位补0，其实就相当于把x的值放在了y的低2个字节的空间中，故选B

3、

答案解析：
正确答案：B
i % 3 的值按1、2、0循环，可推算出ans按1、3、3、2、0、0循环，循环进行1001次，而1001%6=5，也就是ans按规律得到的第5个数为最终结果，故ans=0

4、

答案解析：
正确答案：A
单目运算符的优先级通常都比较高，具体情况可查阅运算符优先级表格

5、

答案解析：
正确答案：C
十六进制数0xF是4位1，参与运算时整型提升，高位都是0。低四位和1异或，0^1是1，1^1是0；高位和0异或，0^0是0，1^0是1。故而可以通过异或F使得a的低四位翻转，并保持高位不变

二、编程题

1、【答案解析】：

这道题只需要循环取出一个数字的每一位进行单独打印，打印完毕后换行即可。而获取数据的每一位，可以通过取每次对数字模和除以`10`来完成

示例：`129`，`129%10`得到`9`，`129/10`得到`12`，循环进行操作直到数字除以`10`得到`0`为止

```
#include <stdio.h>
int main()
{
    int num;
    while(~scanf("%d", &num)) {
        if (num == 0) {//0的情况特殊处理，因为0不会进入while循环计算余数，因此不会被打印
            printf("%d", num % 10);
            continue;
        }
        while(num > 0) {
            printf("%d", num % 10);//打印一个数字的个位数 129 % 10 得到9
            num /= 10;//通过除以10的方式去掉个位数 例如：129/10 得到12
        }
        printf("\n");
    }
    return 0;
}
```

2、【答案解析】：

这道题的解题思路不难，定义一个字符指针数组，用于保存每个单词的起始字符地址，接下来将非字母字符全部替换成为字符串结尾标志，则单词字符字母遇到结尾就结束了，相当于把一个字符串以非字母字符进行切割成为了多个字符串，最终对字符指针数组进行逆序打印每个单词即可。

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[10001] = {0};//字符串最长10000
    int row = 0;
    while(gets(str) > 0) {
        char *ptr = str;
        char *word[10000] = {NULL};
        while(*ptr != '\0') {
            //如果是个字母字符，则是单词的起始字符
            if (('z' >= *ptr && *ptr >= 'a') || ('Z' >= *ptr && *ptr >= 'A')){
                word[row++] = ptr;//保存每个单词的起始地址
                //把本次的单词字母字符走完，直到遇到非字母字符
                while(*ptr != '\0' &&
                      (('z' >= *ptr && *ptr >= 'a') ||('Z' >= *ptr && *ptr >= 'A'))) {
                    ptr++;
                }
                continue;//不能继续向下，因为下边的ptr++会跳过当前的非字母字符
            }
            *ptr = '\0';//把非字母的数据全部替换为结尾标志
            ptr++;
        }
        for (int i = row - 1; i >= 0; i--) {
            printf("%s ", word[i]);//针对所有单词的起始地址逆序开始打印即可
```

```
        }
        printf("\n");
    }
}
```

day13

一、选择题

1、

答案解析：
正确答案：C
这个作用是对整型中0的个数进行统计，x=x|(x+1);的作用是每次循环把x的二进制中从右往左数的最后一位0变成1，直道变成全1的时候x+1就溢出为全0，循环结束。2014的二进制是0000 0000 000 0000 0000 0111 1101 1110，所以结果是23

2、

答案解析：
正确答案：B
A选项的x是指针，赋值时使用a不合适，C选项在赋值时a变量还没定义，D选项中的x不是指针。

3、

答案解析：
正确答案：D
malloc函数在内存分配失败时返回NULL，其余选项都正确

4、

答案解析：
正确答案：C
D选项a计算时是首元素地址，再加1，就是a[1]的地址，AB明显对，C选项a[0]先和++结合，形成一个表达式，不能对表达式取地址，会报错

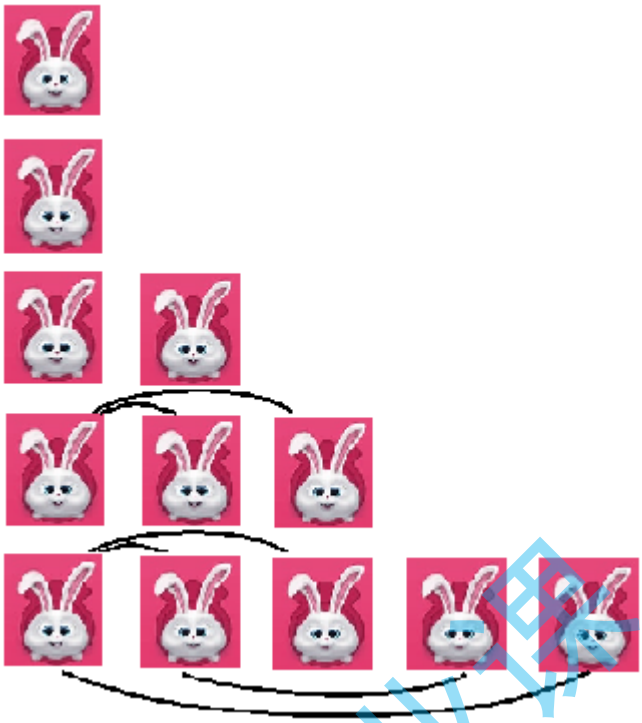
5、

答案解析：
正确答案：A
A错误，因为两个地址相加无意义也可能越界，所以规定不允许指针相加。B选项，可以求出两个数据元素储存位置之间的相隔同数据类型的元素个数，C选项，赋值，没问题，D选项，判断两指针是否相同

二、编程题

1、【答案解析】：

这道题的关键在于寻找数字之间的规律，如果细心的同学会发现这其实是一个斐波那契数列。第`n`个月的兔子数量实际上就是第`n-1`个斐波那契数。



```
#include <stdio.h>
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        int num1 = 1, num2 = 1, ret = 0;
        for (int i = 2; i < n; i++) {
            ret = num1 + num2;
            num1 = num2;
            num2 = ret;
        }
        printf("%d\n", ret);
    }
    return 0;
}
```

2、【答案解析】：

求取一个数字的平方根可以使用数学库中的`double sqrt(double num)`函数完成，接下来只需要从数字自身开始进行求和并在求和后将`n`自身计算成为自身的平方根即可。

```
#include <stdio.h>
#include <math.h>
int main()
{
    double m, n;
    while(~scanf("%lf %lf", &n, &m)) {
```

```
        double sum=0;
        while(m-- > 0) {
            sum += n;//从自身开始以及每次的平方根进行求和
            n = sqrt(n);//n成为当前自身的平方根
        }
        printf("%.2lf\n", sum);
    }
    return 0;
}
```

day14

一、选择题

1、

答案解析：
正确答案：B
循环在*t为0时停止，同时t++，t最后会停在字符串结束的'\0'之后的一个位置，t作为尾部指针减去头部指针就是整个字符串占用内存的字节数，包含\0在内；而c答案字符串长度不包括最后的\0

2、

答案解析：
正确答案：B
在*pa=a中指针pa指向a[0]；pa++返回值仍是操作之前的值；*(pa++)取pa指向的地址的值；*(pa++)*=3将该值变为原来的3倍，也就是数组a的第一个值为4.5；由于pa++之后pa指针移动了sizeof(float)个字节，所以pa指向a[1],所以值为2.5

3、

答案解析：
正确答案：A
指针q初始化为NULL，接着又解引用指针q，是错误的，对NULL指针是不能解引用的。

4、

答案解析：
正确答案：B
B选项，p是个char*类型的数组，p[1]拿到字符串"beijing"的首地址，再加3便是'j'的地址，解地址拿到'j'

5、

答案解析：
正确答案：B
A 选项描述不正确，不同类型指针一般不可以直接赋值；C选项中，p=NULL;和p=0;是等价的；D选项中，指向同一数组的两指针变量进行关系运算可表示它们所指数组元素之间的位置关系。B选项正确

二、编程题

1、【答案解析】：

遍历两个数组，统计猜中次数和伪猜中次数

猜中次数：若位置相同且颜色字符也相同在猜中次数计数器+1

伪猜中次数：颜色相同，但是在不同位置，这时候只需要除去猜中位置之外，统计两个数组中各个字符出现的数量，取较小的一方就是每种颜色伪猜中的数量了

```
int* masterMind(char* solution, char* guess, int* returnSize){
    *returnSize = 2;
    static int arr[2] = {0};
    arr[0] = 0; arr[1] = 0;//静态空间不会进行二次初始化因此每次重新初始化，可以使用memset函数
    int s_arr[26] = {0};//26个字符位 solution 四种颜色数量统计
    int g_arr[26] = {0};//26个字符位 guess 四种颜色数量统计
    for (int i = 0; i < 4; i++) {
        if (solution[i] == guess[i]) {
            arr[0] += 1;//位置和颜色完全一致则猜中数量+1
        }else {
            //统计同一位置不同颜色的两组颜色数量，伪猜中不需要对应位置相同，只需要有对应数量的颜色就行
            s_arr[solution[i]-'A'] += 1; //统计solution对应颜色字符出现次数
            g_arr[guess[i]-'A'] += 1;//统计guess对应颜色字符出现次数
        }
    }
    //在两个颜色数量统计数组中查看颜色数量，取相同位置较小的一方就是为猜中数量
    for (int i = 0; i < 26; i++) {
        arr[1] += s_arr[i] > g_arr[i] ? g_arr[i] : s_arr[i];
    }
    return arr;
}
```

2、【答案解析】：

在数组中拿到一个数字`num`后，在剩下的数字中查找是否有等于`target - num`的数字即可。

```
int* twoSum(int* numbers, int numbersLen, int target, int* returnSize ) {
    *returnSize = 2;
    static ret_arr[2] = {0};
    memset(ret_arr, 0x00, sizeof(ret_arr));//静态空间不会二次初始化，因此手动初始化
    for (int i = 0; i < numbersLen; i++) {//从第0个位置开始一个一个数字找
        for (int j = i + 1; j < numbersLen; j++) {//从第一个数字往后的数字中找出另一个数字
            //与numbers[i]相加等于target的数字找到了则i和j就是对应两个数字下标
            if (numbers[i] + numbers[j] == target) {
                ret_arr[0] = i + 1;//题目要求下标从1开始
                ret_arr[1] = j + 1;
                return ret_arr;
            }
        }
```

```
    }
    *returnSize = 0;//没有符合的下标则返回数组大小为0;
    return NULL;
}
```

day15

一、选择题

1、

```
答案解析:
正确答案: C
p是一个指针数组，p[0] = a[1];此处a[1]是二维数组的第二行的数组名，数组名表示首元素的地址，a[1]是a[1][0]的地址，所以p[0]中存储的是第2行第1个元素的地址，p[0]+1就是第二行第2个元素的地址，*(p[0]+1)就是第二行第二个元素了。所以C正确。
```

2、

```
答案解析:
正确答案: ABD
C选项，指针占几个字节要看平台，64位环境下8个字节，32位环境下4个字节
```

3、

```
答案解析:
正确答案: D
该题考察的是通过scanf函数的调用对结构体数据类型进行初始化。scanf("输入控制符", 输入参数);功能：将从键盘输入的字符转化为“输入控制符”所规定格式的数据，然后存入以输入参数的值为地址的变量中。scanf输入时要通过地址找空间，B、C用了&是正确的。name属于字符数组的数组名，相当于数组的首地址，A正确。单独的pa->age可用于输出语句获取值的形式，用在scanf中的时候需要&操作符，D错误
```

4、

```
答案解析:
正确答案: A
依题目已知情况，当n<=12时结果是正确的，说明是随着参数的变大计算过程中哪里出了问题，故而要在prod *= i;处设断点，查看原因。错误原因是数据过大时整型溢出
```

5、

```
答案解析:
正确答案: D
代码实现的思路应该是arr[i]是奇数的时候要存储起来，所以第一个空是1，最开始j是0，每次找到一个奇数就存储到arr[j]的位置，那接下里j需要+1，所以得第二个空是j++，当循环停止的时候，j其实就是奇数的个数。所以最后返回j，第三个空是j。所以选D。
```

二、编程题

1、【答案解析】：

异或：二进制比特位相同则0， 不同则1.

两个相同的数字异或得到的是0， 基于这个思路，这道题对数组中的所有数据进行逐一异或就可以解决得到奇数次的数字，因为偶数次的数字都被异或成为0了，最后单独保留了奇数次的数字。

```
#include <stdio.h>
int main()
{
    int n;
    while(~scanf("%d", &n)) {
        int num = 0, tmp = 0;
        //对每个数字进行异或，出现偶数次的就会异或为0了，而奇数次刚好剩下的就是对应数字
        for (int i = 0; i < n; i++) {
            scanf("%d", &tmp);
            num ^= tmp;
        }
        printf("%d\n", num);
    }
    return 0;
}
```

2、【答案解析】：

暴力破解： 遍历数组，从`[1, n-1]`号位置开始，哪个位置的数据大于上一个数据和下一个数据即可。

更优思想：二分思想，

中间比右边大，认为从右往左半边递增，则把`right`不断向左靠拢`right=mid`，注意不能是`mid-1`，因为这个位置有可能就是峰值点。

直到遇到中间比右边小了，意味着数据开始递降了，则`left`向右偏移，`left=mid+1`;而一旦`mid+1`位置大于了`right`，意味着刚好这个`mid+1`位置，是一个左半边-右往左递降，右半边-右往左递增的点，就是一个峰值点。

示例：

```
int arr[] = {3, 5, 4, 4, 3, 2, 1} ，这个数组中两边边界都是非峰值点
int left = 0, right = 6;
left=0,right=6,mid=3:  arr[3]=4 > arr[4]=3, 则right = mid = 3; //从右往左是递增的
left=0,right=3,mid=1:  arr[1]=5 > arr[2]=4, 则right = mid = 1; //从右往左是递增的
left=0,right=1,mid=0:  arr[0]=3 < arr[1]=5, 则left = mid + 1 = 1; //从右往左开始递降了
left > right 退出循环， 返回left，也就是1号下标位置。
```

```
int findPeakElement(int* nums, int numsLen ) {
    //边界情况处理，1个元素前后都是负无穷 以及 0号位置大于1号位置，-1位置负无穷的情况
    if (numsLen == 1 || nums[0] > nums[1]) return 0;
    //末尾位置数据大于上一个位置数据，而nums[numsLen]负无穷的情况
    if (nums[numsLen-1] > nums[numsLen-2]) return numsLen-1;
    int left = 0, right = numsLen - 1, mid;
    while(left < right) {
        mid = left + (right - left) / 2;
        if (nums[mid] < nums[mid + 1])//中间比右边小，意味着右边肯定有个峰值
```

```
            left = mid + 1;
        else //否则在左边包括当前位置肯定有个峰值
            right = mid;
    }
    return left;
}
```

day16

一、选择题

1、

答案解析：
正确答案：BC
一般float型只能精确到小数后六位（即1e-6），将float型数据的绝对值与1e-6比较，来判断是否相等（为零）。float的精度误差在1e-6；double精度误差在1e-15；所以要判断一个float型数：if(fabs(f)<1e-6)；要判断一个double型数：if(fabs(f)<1e-15)；若满足，则为零。考虑B选项是对的。若要判断float a,b是否相等，要看if(fabs(a-b)<1e-6)是否为真。C选项，考虑的是数组越界问题

2、

答案解析：
正确答案：AC
第1处两种情况之一成立都是要返回的，应该用或，此处用与错误。在语句GetMemory(&str,100);中传入str的地址，在语句char*str=NULL;中str初始化为空指针，但是str指针变量也有地址，所以参数char**p里面的p保存的是指针变量str的地址，所以调用GetMemory函数之后，动态开辟的空间的地址存放在了str中，在函数返回之后没有释放内存，但是这不会导致程序错误，只会导致内存泄漏。第3处用&str是错的，应该直接用str，是刚申请下来的空间首地址，可以用来接收字符串的copy。

3、

答案解析：
正确答案：AC
对于0x20150810
如果按照大端模式存储：从低地址到高地址：20 15 08 10　　输出从低地址到高地址：20 15 08 10
如果按照小端模式存储：从低地址到高地址：10 08 15 20　　输出从高地址到低地址：08 10 20 15
此数以int类型赋值给联合体x.a，而以结构成员b和c分开访问，分别拿到低地址的2个字节和高地址的2个字节，大端下是2015和810，小端下是810和2015

4、

答案解析：
正确答案：AB
数组下标越界：数组大小255，但是当a[255]就是256个元素，导致越界了。死循环：这个是因为无符号字符型的变量大小在0-255之间，所以说i永远不可能大于255的，是个死循环。内存泄漏：创建的临时变量，在栈中，应该会由系统自动释放，所以应该是不存在内存泄漏的问题。栈溢出：属于缓冲区溢出的一种。栈溢出是由于C语言系列没有内置检查机制来确保复制到缓冲区的数据不得大于缓冲区的大小，因此当这个数据足够大的时候，将会溢出缓冲区的范围

5、

```
答案解析:
正确答案: C
本题就是找规律，计算什么时候能遇到0
unsigned char 8位数据位，范围在0-255，所以-2（11111110）时，变成254；同理-1（11111111）时，变成255；最后减到0时，不满足循环条件，for停止。刚好173次。                                    7 4 1 ==> 共(7-1)/3+1=3次
（1-3=-2，即254，继续循环）
254 251 ... 5 2 ==> 共(254-2)/3+1=85次（2-3=-1，即255，继续循环）
255 252 ... 6 3 ==> 共(255-5)/3+1=85次（3-3=0，退出循环）                  所以总共173次
```

二、编程题

1、【答案解析】：

暴力破解：将`x`和`y`分别遍历`[1, n]`，进行判断当`x % y > k`时统计计数`count++`即可,但是这样的话当`n`的值非常大的时候循环次数将非常恐怖，需要循环`n^2`次。

更优解法： 假设输入`n=10 , k=3`;

```
当 y <=k 时，意味着任何数字取模y的结果都在 [0, k-1]之间，都是不符合条件的。
当 y = k+1=4 时，x符合条件的数字有 3,7
当 y = k+2=5 时，x符合条件的数字有 3,4,8,9
当 y = k+3=6 时，x符合条件的数字有 3,4,5,9,10
当 y = k+n时，
    x小于y当前值，且符合条件的数字数量是：y-k个，
    x大于y当前值，小于2*y的数据中，且符合条件的数字数量是：y-k个
    从上一步能看出来，在y的整数倍区间内，x符合条件的数量就是 (n / y) * (y - k)个
    n / y  表示有多少个完整的 0 ~ y区间，  y - k 表示有每个区间内有多少个符合条件的数字
    最后还要考虑的是6...往后这种超出倍数区间超过n的部分的统计
    n % y  就是多出完整区间部分的数字个数，其中k以下的不用考虑，则符合条件的是  n % y - (k-1) 个
    这里需要注意的是类似于9这种超出完整区间的数字个数 本就小于k的情况，则为0
最终公式：(n / y) * (y - k) + ((n % y < k) ? 0, (n % y - k + 1));
```

```
#include <stdio.h>
int main()
{
    long n, k;
    while(~scanf("%ld %ld", &n, &k)){
        if (k == 0) {
            printf("%ld\n", n * n);//任意数对的取模结果都是大于等于0的
            continue;
        }
        long count = 0;
        for(long y = k + 1; y <= n; y++) {
            count += ((n / y) * (y - k)) + ((n % y < k) ? 0 : (n % y - k + 1));
        }
        printf("%ld\n", count);
    }
    return 0;
}
```

2、【答案解析】：

截取字符串前`n`个字符，只需要将数组`n`下标位置的数据替换为字符串结尾标志即可

```
#include <stdio.h>
int main()
{
    char str[101];
    while(scanf("%s", str) > 0) {
        int n;
        scanf("%d", &n);
        str[n] = '\0';
        printf("%s\n", str);
    }
    return 0;
}
```