C7 题解

A 指针、指针、还是指针

第一题

正确答案是C。

A选项，字符串本身就是一个字符数组了，把字符数组写在大括号里面相当于数组套数组，肯定不能赋值给数组啦。

B选项，不能确保字符串以'\0'结尾。

C选项，定义指针后没有赋值初始值，是可以这么赋值一个字符串，且保证以'\0'结尾的，但是不推荐大家这么使用，最好还是乖乖定义数组或者使用malloc赋值给指针。

D选项，传给scanf的指针需要保证有值，因为scanf只管往地址里写，不管内存空间的申请。而刚定义的指针是未被初始化的，会RE。

第二题

正确答案是D。

[1]处相当于要“定义变量”，这里的“*”代表变量类型是指针类型，因为传入的是字符串 a ，所以 a 是一个数组，数组的本质也就是指针，所以这里不止可以写 char a[]，也可以写成B或D选项这种，字符指针的样子。

[2]处， a 是一个指针，所以 $a+i$ 也是一个指针，对应的是数组 a 从开始位置后移 i 这么多个char的位置后的地址，也就是&a[i]，也就是一个地址，所以我们要对地址取值，要加一个“*”

第三题

正确答案是A。

sub函数传入了两个int变量和一个int*，也就是指针变量，将z指针对应地址的值修改为了x-y的值，因此只要按照“每次都把第三个值赋值为第二个减去第一个”的思路脑内模拟一下就能很容易得到答案是A。

这里也要注意，因为函数定义第三个值是指针，所以要对int变量取地址（&）后才能传入。

第四题

正确答案是A。

如第三腿说明的，这里 a,b 都是整型变量，需要传入的是整型指针，所以对变量取地址（&）后传入即可。

可以思考一下为什么这个函数能实现两个变量值的交换。

第五题

正确答案是D。

HINT中给出了五个字母对应的ASCII码值的十六进制，转换成十进制就是'A', 'B', 'U', 'h', 't'分别对应65，66，85，104，116。所以容易得到，A选项括号内代表('U'-'A'+1)=21，21除以7等于3，符合要求。B选项中对应('t'-'h')=12，12除以4等于3，符合要求。C和D考察++符号在前后的行为差别，在前时返回+1后的值，在后时返回+1前的值，C对应'D'-'A'=3，正确，D对应'D'-'U'=-17，错误。

B 字符串猫猫

题目分析

可以利用`strcat()`函数将分开的字符串拼接起来，然后输出指定区间的字符。

示例代码

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[1010] = {}, temp[12];
    int l, r;
    scanf("%d%d", &l, &r);
    while (scanf("%s", temp) != EOF) //读入字符串
        strcat(str, temp); //将temp连接到str后面
    for (int i = l - 1; i < r; i++) //输出[l,r]区间的字符，注意下标
        putchar(str[i]);
    return 0;
}
```

C 大小端转化

题目分析

本题的考点主要在于指针的类型转换。

- 在用`*`提取指针处的数据时，是根据`sizeof(指针所指类型)`来提取一个特定长度的数据，因此，当一个`unsigned int`型指针和一个`char`型指针相同时，`unsigned int`提取该位置处四个字节长度的数据，而`char`只提取一个字节长度的数据。
- 在对指着做`p++`的运算时，并不是指针的数值`+1`，而是加上`sizeof(指针所指类型)`，也就是说指针向后移一个所指单位数据类型的长度。

此外，题目中涉及关于大小端存储的相关知识，较为详细的解释已在题面中给出。

示例代码

```
#include <stdio.h>

void swap_ending(unsigned int * a)
{
    char *low = (char *)a;                      //用char型指针指向uint的低字节
    char *hi = low + sizeof(unsigned int) - 1;  //指向uint的高字节
    while(low < hi)
    {
        /*
        交换两个指针位置上的数据
```

```
        */
        char tem = *low;
        *low = *hi;
        *hi = tem;
        low++;
        hi--;
    }
}

int main()
{

    unsigned int a, b;
    scanf("%u%u", &a, &b);
    swap_ending(&a);
    swap_ending(&b);
    printf("%u\n", ((a % 10000) * (b % 10000)) % 10000);
    return 0;
}
```

D 矩阵的幂次 mini

难度	考点
2	循环

问题分析

由题意，可以将矩阵乘法封装成一个函数，多次调用即可。矩阵 C 计算方式如下：

$$C_{ij} = \sum_{k=1}^{p} A_{ik} B_{kj}$$

使用三层 for 循环，第一层枚举 $C(A)$ 行下标 i，第二层枚举 $C(B)$ 列下标 j，第三层枚举 A 列下标（B 行下标）k。

注意二维数组的传参。

参考代码 #1

```
#include <stdio.h>
#include <string.h>

int m, p, n, A[10][10], B[10][10], C[10][10];

void mult(int A[][10], int B[][10], int C[][10], int r, int c)
{
    for (int i = 0; i < r; i++)
    {
        for (int j = 0; j < r; j++)
        {
            C[i][j] = 0;
            for (int k = 0; k < c; k++)
                C[i][j] += A[i][k] * B[k][j];
```

```
        }
    }
}

int main()
{
    scanf("%d%d%d", &m, &p, &n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++)
            scanf("%d", &A[i][j]);
    for (int j = 0; j < m; j++)
        for (int i = 0; i < p; i++)
            scanf("%d", &B[i][j]);
    mult(A, B, C, m, p);
    memcpy(B, C, sizeof(C));
    n--;
    while (n--)
    {
        mult(B, C, A, m, m);
        memcpy(B, A, sizeof(A));
    }
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
            printf("%d ", A[i][j]);
        printf("\n");
    }
    return 0;
}
```

由于本题对空间要求较小，亦可直接开多个数组循环。

参考代码 #2

```
#include <stdio.h>

int m, p, n, A[10][10], B[10][10], ans[10][10][10];

int main()
{
    scanf("%d%d%d", &m, &p, &n);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < p; j++)
            scanf("%d", &A[i][j]);
    for (int j = 0; j < m; j++)
        for (int i = 0; i < p; i++)
            scanf("%d", &B[i][j]);
    for (int i = 0; i < m; i++)
        for (int j = 0; j < m; j++)
            for (int k = 0; k < p; k++)
                ans[0][i][j] += A[i][k] * B[k][j];
    for (int t = 1; t < n; t++)
        for (int i = 0; i < m; i++)
            for (int j = 0; j < m; j++)
                for (int k = 0; k < m; k++)
```

```
                    ans[t][i][j] += ans[t - 1][i][k] * ans[0][k][j];
    for (int i = 0; i < m; i++)
    {
        for (int j = 0; j < m; j++)
            printf("%d ", ans[n - 1][i][j]);
        printf("\n");
    }
    return 0;
}
```

E 成绩分析!

题目分析

本题主要考察指针作为函数参数。

先将成绩排序（冒泡排序、选择排序、qsort等均可），最大值和最小值即为两端，然后求和计算平均值，最后判断奇偶求中值。

示例代码

```
#include <stdio.h>
#include <stdlib.h>

int cmp(const void *_a, const void *_b) {
    return *(int *)_a - *(int *)_b;
}

void cal(int a[], int *Max, int *Min, double *ave, double *mid, int cnt) {
    int sum = 0, sum2 = 0;
    qsort(a, cnt, sizeof(a[0]), cmp);//先对成绩进行排序（升序）
    *Max = a[cnt - 1]; //计算最大值
    *Min = a[0];//计算最小值
    for (int i = 0; i < cnt; i++) {
        sum += a[i];
    }
    *ave = 1.0 * sum / cnt;//计算平均值
    if (cnt % 2)//计算中位数
        *mid = 1.0 * a[cnt / 2];
    else
        *mid = 1.0 * (a[cnt / 2] + a[cnt / 2 - 1]) / 2;
}

int main() {
    int a[1010], Max, Min;
    double mid, ave;
    while (1) {
        Min = 100, Max = 0;
        int cnt = 0; // 记录班级人数
        if (scanf("%d", &a[cnt]) == EOF)//判断是否还有数据
            break;
        cnt++;
        while (1) {
            scanf("%d", &a[cnt]);
            if (a[cnt] == -1)//判断每组数据的结束
```

```
                break;
            cnt++;
        }
        cal(a, &Max, &Min, &ave, &mid, cnt);//调用函数计算
        printf("%d %d %.2lf %.2lf\n", Max, Min, ave, mid);
    }

    return 0;
}
```

F 横幅设计

考点	难度
指针、字符串	3

题目分析

本题改编自*例7-11*，在**Hint**的提示之下，可以大大简化问题的复杂度。仅需要使用$strstr$函数循环返回匹配Son子串的位置，自定义函数将子串逆置。

需要注意的是，每次完成子串逆置之后，应该将指针后移继续匹配.

后移的方法是：`now += strlen(Son);`

示例代码

```
#include <stdio.h>
#include <string.h>
char Mum[205], Son[250], *p, *now;
void rev(char *first, char *last);
int main()
{

    scanf("%s%s", Mum, Son);
    now = Mum;
    while ((p = strstr(now, Son)) != NULL)
    {
         rev(p, p + strlen(Son) - 1);
         now += strlen(Son);
    }
    puts(Mum);
    return 0;
}
void rev(char *first, char *last)
{
    int tmp;
    while (first < last)
    {
          tmp = *last;
          *last = *first;
          *first = tmp;
          first++;
          last--;
```

```
    }
}
```

G 二分查找 AC版

众所周知，这是一道板子题，希望大家能够认真写这道题，理解思路，保存模板。

题目解读

按照题意，我们需要对普通的二分查找条件进行修改，以便于适合题意

为了查找边界，当我们查到一个与给出的 key 数值相同的元素时，我们不能直接输出，而是应当根据左（右）边界，继续向左（右）搜索，为了进一步理解下面的代码，大家可以在草稿纸上列出一段数列并进行二分模拟。

比如你可以在下面的数列中试着用你写的条件找到 $key = 4$ 时的边界：

4 4 5 5

4 5 5

4 4 5

代码

```
#include<stdio.h>
#include<stdlib.h>

int find_lowerbound(int dst[], int key, int lef, int rit)
{//左边界查找函数
    int mid = (lef+rit)/2; //为了找左边界，一般向左端取整
    while(lef<rit)//左右位置索引相同时表示找到唯一可能边界
    {
        if(dst[mid]>=key)//中间值大于等于key
        // 向左搜索
        {
            rit = mid;
        }
        else //中间值小于key
        {
            lef = mid + 1; //由于是左端取整，需要对左位置索引+1以保证能够搜到边界
         }
        mid = (lef + rit) / 2;//保证向左端取整
    }
    //判断边界是否可行
    if(dst[mid]==key)
    return mid;
    else
    return -1;
}
int find_upperbound(int dst[], int key, int lef, int rit)
{//右边界查找函数
    int mid = (lef + rit + 1)/2; //向右端取整
    while(lef<rit)
    {
        if(dst[mid]>key)//中间值大于key
```

```
        // 向左搜索
        {
            rit = mid - 1;
        }
        else //中间值小于等于key
        {
            lef = mid;
         }
        mid = (lef + rit + 1) / 2;
    }
    if(dst[mid]==key)
    return mid;
    else
    return -1;
}

int main()
{
    int dst[500005];//数列元素
    int n, cnt;//个数、查询次数
    scanf("%d%d",&n,&cnt);
    int cntn;
    //读取
    for(cntn=0;cntn<n;cntn++)
    {
        scanf("%d",&dst[cntn]);
    }
    while(cnt--)
    {
        int key;
        //读查找的关键词
        scanf("%d",&key);
        int lowerbound=0, upperbound=n-1;//边界赋初始值
        int temp;
        temp = find_lowerbound(dst, key, lowerbound, upperbound);
        if(temp!=-1)//元素存在且搜到左边界
            {
                lowerbound = temp;
                //搜右边界
                upperbound = find_upperbound(dst, key, lowerbound, upperbound);
                //特判
                if(lowerbound != upperbound)
                    printf("%d %d\n", lowerbound, upperbound);
                else
                    printf("%d\n", lowerbound);
            }
        else//该元素不存在
            printf("-1\n");
    }
    return 0;
}
```

H 四季映姬的名单查找

考点	难度
字符串，指针	3

题目分析

由题可知，给出若干个字符串，需要找出其中长度最大的，次大的和最小的。

类似于C6A-公益组织，并不需要将所有的名字全部保存下来 *（虽然很多人那道题也是全部存下来排序）*，只需要保留四个字符串的内存：最长的，次长的，最短的，以及当前读取的字符串。

每读取一个便进行一次比较，随后根据读取的字符串的长度，对最长的，次长的，最短的字符串进行改变。

朴素的方法是将字符串复制过去，此法可以使用 `strcpy` 函数来实现

在时间上更加优化的做法是不进行内存复制，而是改变指针，从而避免过多的对内存进行操作

（详情可以参考ppt例题7-4）

示例代码1——使用strcpy函数

```
#include <stdio.h>
#include <string.h>
int main(void) {
    // max,mid,min数组分别是最长，次长，最短字符串，crt是当前输入字符串；
    char max[2001] = "", mid[2001] = "", min[2001] = "", crt[2001] = "";
    int maxLEN = 0, midLEN = 0, minLEN = 2001, crtLEN = 0;
    //计数器
    int n = 0;
    while (scanf("%s", crt) != EOF) {
        if (n == 0) {
            //第一次输入，初始化各字符串
            strcpy(max, crt);
            strcpy(mid, crt);
            strcpy(min, crt);
        }
        crtLEN = strlen(crt);
        if (crtLEN > maxLEN) {
            // crt比max长，则crt是新的max，原max是新的mid
            //将max复制给mid
            strcpy(mid, max);
            midLEN = maxLEN;
            //将crt复制给max
            strcpy(max, crt);
            maxLEN = crtLEN;
        } else if (crtLEN > midLEN) {
            // crt比mid长，则crt是新的mid
            //将crt复制给mid
            strcpy(mid, crt);
            midLEN = crtLEN;
        } else if (crtLEN < minLEN) {
            // crt比min短，则crt是新的min
            //将crt复制给min
            strcpy(min, crt);
            minLEN = crtLEN;
```

```
        }
        n++;
    }
    printf("%d\n%s\n%s\n%s", n, max, mid, min);
    return 0;
}
```

示例代码2——使用指针交换

```
#include <stdio.h>
#include <string.h>
int main(void) {
    //四个存放字符串的数组，交换的过程中储存的内存本身不变
    char s1[2001] = "", s2[2001] = "", s3[2001] = "", s4[2001] = "";
    int maxLEN = 0, midLEN = 0, minLEN = 2001, crtLEN = 0;
    // max,mid,min分别指向最长，次长，最短的字符串，crt指向当前输入内容，tmp是交换用的指针
    char *max = s1, *mid = s2, *min = s3, *crt = s4, *tmp = NULL;
    //计数器，用于储存名字个数
    int n = 0;
    while (scanf("%s", crt) != EOF) {
        if (n == 0) {
            //第一次输入，初始化各字符串
            strcpy(s1, crt);
            strcpy(s2, crt);
            strcpy(s3, crt);
        }
        crtLEN = strlen(crt);
        if (crtLEN > maxLEN) {
            // crt的长度大于max的长度，同时改变最长和次长字符串
            //交换mid和max，使原先的最长字符串变为次长字符串
            tmp = max, max = mid, mid = tmp, midLEN = maxLEN;
            //交换max和crt，使新的字符串变为最长字符串
            tmp = max, max = crt, crt = tmp, maxLEN = crtLEN;
        } else if (crtLEN > midLEN) {
            // crt的长度大于mid的长度但不大于max的长度，改变次长字符串
            //交换mid和crt，使新的字符串变为次长字符串
            tmp = mid, mid = crt, crt = tmp, midLEN = crtLEN;
        } else if (crtLEN < minLEN) {
            // crt的长度小于min的长度，改变最短字符串
            //交换min和crt，使新的字符串变为最短字符串
            tmp = min, min = crt, crt = tmp, minLEN = crtLEN;
        }
        //操作完成后，crt指向的是不需要的字符串，可以用来被下次输入覆盖
        n++;
    }
    printf("%d\n%s\n%s\n%s", n, max, mid, min);
    return 0;
}
```

H 这是第几个子串?

题目分析

显然 str 的子串 $[i,j]$ 随 j 增加，字典序严格增大。因此若 str 的子串 $[l_0,r_0]$ 字典序小于子串 $[i,j]$，则对任意 $r \le r_0$，子串 $[l_0,r]$ 字典序小于子串 $[i,j]$；若 str 的子串 $[l_0,r_0]$ 字典序大于等于子串 $[i,j]$，则对任意 $r > r_0$，子串 $[l_0,r]$ 字典序大于子串 $[i,j]$。

初始化 sum, num 为 0，分别记录小于和等于子串 $[i,j]$ 的子串数量。我们可以将 l_0 从 0 遍历到 $len-1$，每次遍历，再遍历 r，比较子串 $[l_0,r]$，找到使得子串 $[l_0,r]$ 字典序大于等于子串 $[i,j]$ 的最小 $r=r_0$，则以 l_0 为起点的子串中，字典序小于子串 $[i,j]$ 的数量应为 r_0-l_0 个，加到 sum 上。如果此时子串 $[l_0,r_0]$ 与子串 $[i,j]$ 相等，则 num 自增 1。如果找不到使得子串 $[l_0,r]$ 字典序大于等于子串 $[i,j]$ 的最小 $r=r_0$，即以 l_0 为起点的子串均小于子串 $[i,j]$，则字典序小于子串 $[i,j]$ 的数量应为 $len-l_0$ 个，加到 sum 上。最后输出应为 $sum+1, sum+num$。

善用指针，$str+k$ 是 str 第 k 个字符的地址，也表示以 str 第 k 个字符为起点的字符串，可以定义一个函数，用指针传入的方式计算。具体代码如下：

示例代码

```
#include <stdio.h>
#include <string.h>
#define min(a,b) ((a)<(b)?(a):(b))
/*
lcp函数功能：
1：如果s1前len(s2)位不为s2，则返回以s1第0个字符为起点的子串中字典序小于s2的数量
2：如果s1前len(s2)位等于s2，返回-1
*/
int lcp(const char *s1, const char *s2) {
    int len = strlen(s2);
    for(int i = 0; i < len; ++i) {
        if(*(s1 + i) < *(s2 + i)) return strlen(s1); //若s1的第i位小于s2的第i位，则以s1第0个字符为起点的所有子串均小于s2，因此返回strlen(s2)
        if(*(s1 + i) > *(s2 + i)) return i; //若s1的第i位大于s2的第i位，则以s1第0个字符为起点的子串中，终点小于i的子串小于s2，因此返回i
    }
    return -1; //s1前len(s2)位与s2相同，返回-1
}

int main() {
    char str[10005] = "", substr[10005] = ""; //分别存储字符串str和子串[i,j]
    gets(str); //也可以用scanf("%s", str);
    int i, j, k;
    int len = strlen(str);
    scanf("%d%d", &i, &j);
    //strcpy(substr, str + i); substr[j - i + 1] = '\0';
    memcpy(substr, str+i, j-i+1);
    int sum = 0, num = 0; //sum, num分别记录字典序小于和等于子串[i,j]的子串数量
    for(int k = 0; k < len; ++k) {
        int t = lcp(str + k, substr);
        if(t >= 0) sum += t; //字符串str+k前len(substr)位不为substr，返回值应加到sum上
        else { //字符串str+k的前len(substr)位等于substr，
            sum += j - i; //以str+k的第0个字符为起点的子串中字典序小于substr的数量应为j-i
            ++num; //等于substr的子串数量+1
        }
```

```
    }
    printf("%d %d", sum + 1, sum + num); //输出
}
```

补充（可忽略）

通过以上分析，可以发现本题思路在于将子串 $[i,j]$ 在 str 的每一位去匹配， $s1 > s2$ 时函数 lcp 返回值为匹配的长度， $s1 < s2$ 时返回 $len(s1)$ 。本质上是在 str 的每一位去匹配，在失配的位置比较两串字符的大小，根据情况进行计算。如果没有失配对应了 lcp 中返回 -1 的情况。因此可以借助字符串匹配的 kmp 算法在 $O(len)$ 的时间复杂度内求解，但是这个内容远远超纲了，因此仅限制了时间复杂度为 $O(len^2)$ 。

Author: 哪吒

J void学计组

题目分析

本题其实难度也没有那么高，按照题意模拟即可

我们用一个变量来表示当前执行到了哪条指令，同时统计执行指令的条数以备三号输出

代码...也就百行上下吧，因为指令的操作数格式都是已经确定的，只需要写一个从字符串中读取数字出来的函数就好

这个函数在E6E其实出现过一次，原型可能叫做快读（read函数）

代码示例

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<ctype.h>
#include<string.h>
#include<time.h>
typedef long long LL;
//指令集：add,sub,ori,li,lui,beq
//syscall,nop
int n,rs,rt,rd,Imm,top;
char ins[210][200];
int reg[32];
int end=0;//是否正常结束
int pc,cnt;//pc为当前指令位置，cnt为已经执行的指令总数
int pos;

int getnum(){
    int res=0,f=1;
    int l=strlen(ins[pc]);
    int i=pos;
    while((ins[pc][i]<'0'||ins[pc][i]>'9')&&i<l){if(ins[pc][i]=='-')f=-1;i++;}
    while(ins[pc][i]>='0'&&ins[pc][i]<='9'&&i<l){res=res*10+ins[pc][i]-'0';i++;}
    pos=i;
    return res*f;
}
```

```
int main(){
    scanf("%d ",&n);
    for(int i=1;i<=n;i++)
        gets(ins[i]);
    pc=1;
    while(pc<=n){
        if(cnt>=1000000){
            printf("program can't be finished");
            return 0;
        }
        pos=0;
        if(strcmp(ins[pc],"nop")==0){cnt++;}//nop
        if(ins[pc][0]=='s'&&ins[pc][1]=='y'){//syscall
            if(reg[2]==10)//判断结束
                {end=1;break;}
            else
                cnt++;
        }
        if(ins[pc][0]=='a'){//add
            rd=getnum();
            rs=getnum();
            rt=getnum();
            reg[rd]=reg[rs]+reg[rt];
            cnt++;
        }
        if(ins[pc][0]=='s'&&ins[pc][1]=='u'){//sub
            rd=getnum();
            rs=getnum();
            rt=getnum();
            reg[rd]=reg[rs]-reg[rt];
            cnt++;
        }
        if(ins[pc][0]=='o'){//ori
            rt=getnum();
            rs=getnum();
            Imm=getnum();
            reg[rt]=reg[rs]|Imm;
            cnt++;
        }
        if(ins[pc][0]=='l'&&ins[pc][1]=='u'){//lui
            rt=getnum();
            Imm=getnum();
            reg[rt]=Imm<<16;
            cnt++;
        }
        if(ins[pc][0]=='l'&&ins[pc][1]=='i'){//li
            rt=getnum();
            Imm=getnum();
            reg[rt]=Imm;
            cnt++;
        }
        if(ins[pc][0]=='b'){//beq
            rs=getnum();
            rt=getnum();
            Imm=getnum();
```

```
            if(reg[rs]==reg[rt])
                pc+=Imm;
            cnt++;
        }
        reg[0]=0;
        pc++;
    }
    if(end){
        puts("program is finished running");
    }
    else{
        puts("program is finished running(dropped off bottom)");
    }
    cnt=0;
    REP(i,0,31){
        printf("%08x ",reg[i]);
        cnt++;
        if((cnt%4==0)&&(cnt>0))puts("");
    }
}
```