上机8解析

ALW

难度	考点
1	文件IO

题目分析

来自LINUX和WINDOWS的文件行末尾的区别是有没有`\r`，gets函数能够将`\r`读进来，而舍去`\n`，所以只要使用gets函数读取一行并检查行末尾有没有`\r`即可。

示例代码

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

int main(){
    char s[100];
    gets(s);
    if(s[strlen(s)-1]=='\r'){
        printf("WINDOWS");
    }else{
        printf("LINUX");
    }
    return 0;
}
```

B矩形的重叠面积

难度	考点
3	结构体，判断

题目分析

找到重叠部分的左右上下边界即可，如左边界是两矩形左下角坐标的较大者。注意两个矩形不相交的情况，可额外对边界进行判断，或是采用参考代码中右边界选取“预设”右边界和左边界较大值的方法，若不相交，左边界和右边界取值相同，面积自然为0
还需注意虽然坐标值在int范围内，但面积有可能超出int

参考代码

```
#include <stdio.h>
#define max(a, b) (a > b ? a : b)
#define min(a, b) (a > b ? b : a)

struct pt_2d
{
    int x,y;
};

struct rect_2d
{
    struct pt_2d left_botoom,right_top;
};

int main()
{
  struct rect_2d rect1, rect2;
  int left,right,bottom,top;
  long long area;

  scanf("%d%d%d%d",&rect1.left_botoom.x,&rect1.left_botoom.y,&rect1.right_top.
x,&rect1.right_top.y);
  scanf("%d%d%d%d",&rect2.left_botoom.x,&rect2.left_botoom.y,&rect2.right_top.
x,&rect2.right_top.y);

  left=max(rect1.left_botoom.x,rect2.left_botoom.x);
  right=max(min(rect1.right_top.x,rect2.right_top.x),left);
  bottom=max(rect1.left_botoom.y,rect2.left_botoom.y);
  top=max(min(rect1.right_top.y,rect2.right_top.y),bottom);
  area=1LL*(right-left)*(top-bottom);

  printf("%lld",area);
```

```
    return 0;
}
```

cratings

难度	考点
2	结构体

题目解析

签到题,按要求记录最大值即可

示例代码

```
#include<stdio.h>

#define M 111111
typedef struct User User;
struct User
{
    int uid,rat;
    char na[20];
};
User u[M];
int n,mx;

int main()
{
    int i;
    scanf("%d",&n);
    for(i = 0;i < n;i++)
    {
        scanf("%s%d%d",u[i].na,&u[i].uid,&u[i].rat);
        if(u[i].rat > u[mx].rat || (u[i].rat == u[mx].rat && u[i].uid <
u[mx].uid))
            mx = i;
    }
    printf("%s %d %d\n",u[mx].na,u[mx].uid,u[mx].rat);
    return 0;
}
```

D字符串函数复习

难度	考点
2	字符串

题目分析

一道简单的字符串内容复习题。主要考点就是字符串函数大杂烩，掺杂了一点点通过指针及位移访问数组的内容。

你们要的坑点：

$\forall 1 \leq i \leq 4,$

第 $3i-2$ 组数据换行符为 `\n`；

第 $3i-1$ 组数据换行符为 `\r\n`；

第 $3i$ 组数据换行符为 `\r`和`\n`混用。

第$1-3$ 组数据为样例；

第$4-6$组数据为`str1`长度达到上界的情况；

第$7-9$组数据为`str3`长度达到上界的情况；

第$10-12$组数据为待处理字符串长度达到上界的情况；

第$10-12$组数据为待处理字符串长度达到上界的情况；

第$13-15$组数据换行符为`\r\n`，其中：

第13组数据待处理字符串可能为空；

第14组数据`str2`为空，且待处理字符串里可能含有`\t`；

第15组数据中有的待处理字符串长度为20000，且行尾还有额外的`\r\n`，模拟之前有粗心助教出题时漏考虑了换行符的情况（对，就是之前最长行那题）。

示例代码

```
#include <stdio.h>
#include <string.h>
int main()
{
    char str[2][1005],instr[20005],final[30105]={0};
    char *substr=NULL;
    int len[2]={0},i,j,k;
    for(i=0; i<2; i++)
    {
        fgets(str[i],20003,stdin);
```

```
        len[i]=strlen(str[i]);
        while(str[i][len[i]-1]=='\r'||str[i][len[i]-1]=='\n') str[i][--
len[i]]=0;
    }
    while(fgets(instr,20003,stdin)!=NULL)
    {
        len[1]=strlen(instr);
        while(instr[len[1]-1]=='\r'||instr[len[1]-1]=='\n') instr[--len[1]]=0;
        substr=strstr(instr,str[0]);
        if(substr!=NULL)
        {
            if(strcmp(substr+len[0],str[1])!=0) strcat(final,substr+len[0]);
            //注意这里的指针加法操作，想想这是为什么
        }
    }
    len[1]=strlen(final);
    printf("%d\n%s#END",len[1],final);
    return 0;
}
```

E导员的生日推送

难度	考点
3	结构体，快排

题目分析

按要求写出cmp函数，注意输出格式即可

示例代码

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

struct infmt{
    char s[51];
    int y,m,d;
} a[50003];

int cmp(const void *p1,const void *p2)
{
    if(((struct infmt *)p1)->m!=((struct infmt *)p2)->m)
        return ((struct infmt *)p1)->m-((struct infmt *)p2)->m;
    if(((struct infmt *)p1)->d!=((struct infmt *)p2)->d)
        return ((struct infmt *)p1)->d-((struct infmt *)p2)->d;
    return strcmp(((struct infmt *)p1)->s,((struct infmt *)p2)->s);
}

int main()
{
    int n;
    int i;
    scanf("%d",&n);
    for(i=1;i<=n;i++)
        scanf("%s %d:%d:%d",&a[i].s,&a[i].y,&a[i].m,&a[i].d);
    qsort(a+1,n,sizeof(a[0]),cmp);
    printf("%d:%d %s",a[1].m,a[1].d,a[1].s);
    for(i=2;i<=n;i++)
        if(a[i].d!=a[i-1].d || a[i].m!=a[i-1].m)
            printf("\n%d:%d %s",a[i].m,a[i].d,a[i].s);
        else
            printf(" %s",a[i].s);
    return 0;
}
```

F四舍六入五凑偶

难度	考点
2	基本运算

解题思路

HINT中已经给出提示可以用round()函数完成四舍五入，因此只用判断尾数为5的情况即可。

示例代码

给出下面两个代码供参考，这两个代码都能AC，但是第二个代码明显优于第一个

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

int main(void)
{
    char out[20];
    double a,b,c;
    int n,bb,i,j;
    while(~scanf("%lf%d",&a,&n)){
        b=a*pow(10,n);
        bb=(int)floor(b);
        if(b-bb==0.5){//尾数为5的情况
            if(bb&1)
                bb++;
            c=bb/pow(10,n);
        }
        else//尾数不为5的情况
            c=round(b)/pow(10,n);
        sprintf(out,"%%.%df\n",n);//构造输出函数
        printf(out,c);
        for(i=0;i<strlen(out);i++)//清零out字符串
            out[i]=0;
    }
    return 0;
}
```

上面这个代码能AC存在一定偶然因素，double的读入存在误差，**这个误差和输入浮点数是否在double精度范围内无关**。可以用double读入0.4试试，你会发现实际上读入变成了0.40000000000000002。而下面这个代码使用了字符串，可以解决读入误差。

```c
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <stdlib.h>

int main(void)
{
    char out[20],in[20];
    double a,b,c;
    int n,bb,i,dot;
    while(~scanf("%s%d",in,&n)){//用字符串存储输入，这可以保证没有误差
        sscanf(in,"%lf",&a);
        b=a*pow(10,n);
        for(dot=0;in[dot]!='.';dot++);
        bb=(int)floor(b);
        if(in[dot+n+1]=='5'){//用字符串判断是否存在读入误差
            for(i=dot+n+2;i<strlen(in);i++){
                if(in[i]!='0')
                    break;
            }
            if(i>=strlen(in)){
                if(bb&1)
                    bb++;
                c=bb/pow(10,n);
            }
            else
                c=round(b)/pow(10,n);
        }
        else
            c=round(b)/pow(10,n);
        sprintf(out,"%%.%df\n",n);
        printf(out,c);
        for(i=0;i<strlen(out);i++)
            out[i]=0;
        for(i=0;i<strlen(in);i++)
            in[i]=0;
    }
    return 0;
}
```

G艾可雪想成为舞台少女!

难度	考点
3	排序、二分查找、结构体

题目分析

题意比较清晰，就是找与待查询目标值相差不超过5的所有信息。存信息可以采用struct，又由于这里$m,n \leq 10^4$，显然如果每次都顺序查找，必然TLE，故不难想到采用快排+二分查找的方法。需要注意的是，快速排序是一种不稳定的排序，若果要保证按输入顺序输出，需要手动添加一个`pos`关键字进行排序。还有注意二分的细节，比如是否越界，还有是否出现了`x+5`这种可能导致数据爆`int`的行为。

参考代码

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
struct data
{
    char name[25];
    int score;
    int pos;
}a[10005];
int cmp(const void *p1, const void *p2)
{
    struct data *x = (struct data*)p1;
    struct data *y = (struct data*)p2;
    if(x->score == y->score)
        return x->pos - y->pos;
    else
        return x->score > y->score ? 1 : -1;
}
int main()
{
    int m, n, i, x;
    scanf("%d%d", &m, &n);
    for(i = 0; i < m; i++)
    {
        scanf("%s %d", &a[i].name, &a[i].score);
        a[i].pos = i; // 存储输入顺序
    }
    qsort(a, m, sizeof(a[0]), cmp);
    while(n--)
    {
        int l = 0, r = m - 1, mid, flag = 0;
        scanf("%d", &x);
```

```
        // 二分查找，这里直接查找下界x-5即可
        while(l <= r)
        {
            mid = (l + r) / 2;
            if(x - 5 < a[mid].score)
                r = mid - 1;
            else if(x - 5 > a[mid].score)
                l = mid + 1;
            else break;
        }
        // 可能会有多个恰在下界的情况，需要找到边界，注意不要越界
        while(mid >= 0 && a[mid].score >= x - 5)
            mid--;
        mid++;
        // 顺序输出即可，如果存在解，flag置1
        while(mid <= m - 1 && abs(a[mid].score - x) <= 5)
        {
            printf("%s %d\n", a[mid].name, a[mid].score);
            flag = 1;
            mid++;
        }
        if(!flag)
            puts("POSITION ZERO");
        puts("");// 注意题目要求
    }
}
```

H矩阵幂次

难度	考点
2	循环，数学

示例代码

```
#include <stdio.h>
int main()
{
    int n, k, i, j, l, t;
    long long int a[55][55], s[55][55], u[55][55], mo = 1000000007;
    scanf("%d%d", &n, &k);
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
        {
            scanf("%lld", &a[i][j]);
            s[i][j] = a[i][j];
        }
    for (t = 2; t <= k; t++)
    {
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
            {
                u[i][j] = 0;
                for (l = 1; l <= n; l++)
                    u[i][j] = (u[i][j] + (a[i][l] * s[l][j])) % mo;
            }
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                s[i][j] = u[i][j];
    }
    for (i = 1; i <= n; i++)
    {
        for (j = 1; j <= n; j++)
            printf("%lld ", s[i][j]);
        printf("\n");
    }
    return 0;
}
```

题目分析

理清循环变量、次序，矩阵乘法的定义式

I 这么多钱怎么花完

考点	难度
递归	3

题目分析

该题相当于求一个不定方程$\sum_{i=1}^{n} p_i x_i = T$的所有正整数解。采用递归的方式枚举每种商品的购买数量x_i。当枚举出一种可行的解决方案（n个x_i都枚举完，并且总和等于T）即输出。由于枚举每个x_i的循环是升序的，因此可以保证按照字典序升序输出所有可行解。

在递归的过程中应随时记录当前已经购买过的商品的花费总和。几处可能帮助减小搜索范围的点：

1. 如果购买前面商品时消费额已经超过总钱数T了就没必要继续枚举了。
2. 每种商品可能购买的数量（即x_i）上限为当前剩下的钱数除以该种商品的单价，向下取整。（买完前几种商品之后剩下的钱哪怕只买当前这种商品也不能无限制的多买了。这个上限还可以继续往下压，因为后续的商品也需要买至少一件）

示例代码

```
#include <stdio.h>

int n, p[15], x[15], target; // n 商品个数, p 商品单价, x 购买数量, target 总钱数
int sol = 0; // 是否找到可行解

void disp() // 输出一组可行解
{
    int i;
    for (i = 1; i <= n; i++)
        printf("%d ", x[i]);
    printf("\n");
}
// depth: 当前层循环正在枚举第depth种商品
// sum: 在购买第depth种商品之前已经花掉的钱数
void dfs(int depth, int sum)
{
    if (sum > target) return ; // 中间已经超了
    if (depth > n) // 每种商品的购买数量都确定了
    {
        if (sum == target)
        {
            sol = 1;
            disp();
        }
        return ;
    }
```

```
    int i, up = (target - sum) / p[depth]; // up: 该种商品数量上限
    for (i = 1; i <= up; i++) {
        x[depth] = i;
        dfs(depth + 1, sum + i * p[depth]);
    }
}

int main()
{
    int i;
    scanf("%d", &n);
    for (i = 1; i <= n; i++)
        scanf("%d", &p[i]);
    scanf("%d", &target);
    dfs(1, 0); // 从第1件商品开始购买，此前的商品累计消费0
    if (!sol) puts("Sorry, you failed.");
    return 0;
}
```

J画蜂巢

难度	考点
3	字符转义，循环

题目分析

将一个完整的 1×1 的蜂巢存入一个二维数组 `draw` 中。另设一个二维数组 `ans` 用来存放最终输出的内容。不妨假左上角为每个蜂巢的起始位置，这样在绘制第 i 行第 j 列的蜂巢时，我们计算出这个蜂巢的起始位置，从这个位置开始，将 `draw` 中的内容复制到 `ans` 中，便可以比较容易的解决这个问题。

示例程序

```
#include <stdio.h>
// 注意转义字符的处理
char draw[][12]={
    "  +---+  ",
    " /     \\ ",
    "+       +",
    " \\     / ",
    "  +---+  "
};
char ans[1000][1000];
// 计算第 x 行 第 y 列的蜂巢的起始坐标,用 xs,ys 表示
// 因为需要两个返回值，这里使用指针
// 也可以使用结构体或者将 xs,ys 设为全局变量
void calc_start_point(int x,int y,int *xs,int *ys){
    if(y%2==1){
        *xs=4*(x-1)+1;
        *ys=6*(y-1)+1;
    }
    else{
        *xs=4*(x-1)+3;
        *ys=6*(y-1)+1;
    }
}

int main(){
    int n,m,xs,ys,i,j,k,l;
    // 将所有位置初始化为空格
    for(i=0;i<1000;i++){
        for(j=0;j<1000;j++) ans[i][j]=' ';
    }
    scanf("%d%d",&n,&m);
    for(i=1;i<=n;i++){
        for(j=1;j<=m;j++){
```

```
            // 计算起始位置
            calc_start_point(i,j,&xs,&ys);
            // 将 draw 的内容复制到 ans 中
            for(k=0;k<5;k++){
                for(l=0;l<9;l++){
                    ans[xs+k][ys+l]=draw[k][l];
                }
            }
        }
    }
    // 简单估计一个行数和列数的上界
    for(i=1;i<=5*n;i++){
        for(j=1;j<=m*9;j++) printf("%c",ans[i][j]);
        printf("\n");
    }
    return 0;
}
```

K找人

难度	考点
5	链表、循环队列

题目分析

本题有空间限制，将所有的数据全部保存下来会MLE。一个较好的解决方法是使用循环队列。

我们维护一个大小恰好为 x 的循环队列，将人名依次入队。当队列满的时候，新的人名若要进队，就将队首出队，新的人名保存在原队首的位置处。实际上，队满时，队首的人名恰是此时的“倒数第 x 个人”。

可以用循环链表实现，比较好想。

示例代码1-数组实现

```
#include <stdio.h>
#include <string.h>
int x, front = 0, rear = 0;
char q[1007][27], s[27];

int isFull()
{
    return front == (rear + 1) % x;
}

int isEmpty()
{
    return front == rear;
}

void outQueue()
{
    front = (front + 1) % x;
}

void inQueue()
{
    if (isFull())
        outQueue();
    strcpy(q[rear], s);
    rear = (rear + 1) % x;
}

int main(int argc, const char * argv[])
```

```
{
    scanf("%d", &x);
    x++;
    while (~scanf("%s", s))
        inQueue();
    printf("%s\n", q[front]);
    return 0;
}
```

示例代码2-循环链表实现

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char s[21];
typedef struct Link
{
    char name[21];
    struct Link *next;
} l, *pl;

int main(int argc, const char * argv[])
{

    int i, x;
    scanf("%d", &x);
    pl head = NULL, rear = NULL, node = NULL;
    for (i = 0; i < x; i++)
    {
        node = (pl) malloc(sizeof(l));
        if (head == NULL)
            head = node;
        if (rear != NULL)
            rear->next = node;
        rear = node;
    }
    rear->next = head;
    while (~scanf("%s", s))
    {
        strcpy(head->name, s);
        head = head->next;
    }
    printf("%s\n", head->name);
    return 0;
}
```

L 胡闹厨房

难度	考点
5	链表

题目分析

如果为每个厨师的工作清单都用一个链表保存，则题目中的四个操作分别对应了：

1. 删除链表的第一个元素
2. 在链表末尾加入一个元素
3. 将一个链表接在另一个链表的末尾
4. 打印链表

为了能够实现在链表末尾添加元素，我们需要两个指针 `head` 和 `tail`，分别指向链表的头部与尾部，即可快速完成上述的前三个操作。由于操作 2、3 的操作对象可能是空链表，因此需要注意处理下述情况：

- 操作 2：
  - 在空链表中添加一个元素
- 操作 3：
  - 在空链表后接一个空链表
  - 在非空链表后接一个空链表
  - 在空链表后接一个非空链表

这两种操作的本质实际上是相同的。只要理解并处理好了指针增删的逻辑，本题的代码写起来是非常小清新的。

示例程序

```
#include <stdio.h>

#define N (100000 + 5)
#define LEN (15 + 2)

// 链表结构
struct _Node {
    char s[LEN];
    struct _Node *nxt;
};
typedef struct _Node Node;

// 存放链表的内存池与指针
Node buf[N], *pit;
// 指向链表头尾的指针，且空链表的头尾都是 NULL
Node *head[N], *tail[N];
```

```c
int main() {
    int n, m;
    int op, a, b, i;
    Node *o;

    for (i = 0; i < N; i++)          // 初始化链表
        head[i] = tail[i] = NULL;
    pit = buf;                       // 内存池指针初始指向 buf[0]

    scanf("%d%d", &n, &m);
    while (m--) {
        scanf("%d", &op);
        if (op == 1) {
            scanf("%d", &a);
            head[a] = head[a]->nxt;
            if (head[a] == NULL) tail[a] = NULL;    // 删除链表头后变成了空链表，更新 tail
        }
        if (op == 2) {
            scanf("%d%s", &a, pit->s);
            pit->nxt = NULL;
            if (tail[a] == NULL) {           // 在空链表 a 后添加一个新元素
                head[a] = tail[a] = pit;
            } else {                         // 在非空链表 a 后添加一个新元素
                tail[a]->nxt = pit;
                tail[a] = pit;
            }
            pit++;                           // 移动内存池指针，以供下次使用
        }
        if (op == 3) {
            scanf("%d%d", &a, &b);
            if (tail[b] == NULL) {           // 在空链表 b 后添加一个链表 a
                head[b] = head[a];
                tail[b] = tail[a];
            } else {                         // 在非空链表 b 后添加一个链表 a
                tail[b]->nxt = head[a];
                if (tail[a] != NULL)         // 在非空链表 b 后添加一个非空链表 a
                    tail[b] = tail[a];
            }
            head[a] = tail[a] = NULL;
        }
        if (op == 4) {
            scanf("%d", &a);
            if (head[a] == NULL) {
                printf("Done");
            } else {
                for (o = head[a]; o != NULL; o = o->nxt)
                    printf("%s ", o->s);
```

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

    return 0;
}
```