A - Pass the Problem!!!

考点	难度
计算	1

题目分析

根据高中知识，我们知道在 $n \geqslant 50$ 时，满足 $L \geqslant n+5$ 且 $L \geqslant \left\lfloor n+\frac{n}{10000} \right\rfloor$ 的最小 L 为：$\max(n+5, \left\lfloor n+\frac{n}{10000} \right\rfloor)$。

所以我们只需要计算出这两个值并比较，取其中较大的为最终 L 值。

本题不需要使用 `long long`，但需小心处理相关数据，以避免溢出。

示例代码

```c
#include <stdio.h>
int main()
{
    int T, n, a, b, L, Li;
    Li = 67108864 / 16 * 15 / 8;
    scanf("%d", &T);
    while (T--)
    {
        scanf("%d", &n);
        a = n + 5;
        b = n + n / 10000;
        L = a > b ? a : b;
        printf("%d\n%s\n", L, L < Li ? "Pass the Problem!!!" : "Want some MLE?");
    }
}
```

B - 大家一起刷TD 2

考点	难度
数组	2

题目分析

可以利用一个长度为 48 的数组，表示第 i 天是否打卡有效，初始全部设置为 1，有了事故再设置成 0。最后统计有多少个 1 即可。

示例程序

```c
#include <stdio.h>

int main()
{
    int t, l, r, a[50], sum, i;

    scanf("%d", &t);
    for (i = 1; i <= 48; i++)
        a[i] = 1;

    while (t--)
    {
        scanf("%d%d", &l, &r);
        for (i = l; i <= r; i++)
        {
            a[i] = 0;
        }
    }

    sum = 0;
    for (i = 1; i <= 48; i++)
    {
        sum += a[i];
    }
    printf("%d", sum);

    return 0;
}
```

C - 寻狗启事

难度	考点
3	字符数组的读入、查找与指针的使用

题目分析

本题考查字符数组及字符串函数的基本使用，以及字符指针的用法。

在处理字符串时，我们可以利用 `ctype.h` 中的一系列函数来让代码变得更加清晰，如 `isalpha()` 用来判断是否是字母，`tolower()` 可以把字符转换为小写等。

使用字符指针时，可以用 `p1 - p2` 得到两个指针位置的距离，如示例中的 `p2 - s2` 可以得到当前指针对应的数组下标。

示例使用了 `strstr()` 函数，它的作用是在一个字符串中查找另一个字符串。

示例代码

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

#define N 500
char s1[N + 5], s2[N + 5];

int main()
{
    int len1, len2;
    int i;
    char *p1, *p2;

    gets(s1);
    gets(s2);

    len1 = strlen(s1), len2 = strlen(s2);

    // 将字符串全部转换为小写，方便比较
    for (i = 0; i < len1; ++i) s1[i] = tolower(s1[i]);
    for (i = 0; i < len2; ++i) s2[i] = tolower(s2[i]);

    for (p1 = s2; (p2 = strstr(p1, s1)); p1 = p2 + len1)
    {                               // 根据题目要求查找
        printf("%d ", p2 - s2); // 输出 p2 对应的下标
    }

    return 0;
}
```

D - 指针专项练习1

考点	难度
指针相关知识	3

题目分析

本题考察了与指针相关的一些基础知识，也考察了自增运算符的一些性质。

需要同学们注意的是“野指针”问题，有些指针地址不确定可能会导致严重的问题。

各小题解析将以代码注释的形式给出。

第一题

```
// 定义变量
int abc[5] = {1, 2, 3, 4, 5};
int *p;

// 给定指针p指向的地址，p指向abc[0]的地址后的一个地址即abc[1]
p = &abc[0] + 1;

// 进行指针相关运算
(*(*(&p))++)++;
// x++的运算应在本条语句执行完毕后再执行，自增的变量在最开始的时间点就已经确定了
// 本题中*&p等价于p，内层为p++，即使指针p（地址值）加1，最终p指向abc[2]
// 外层相当于（*p）++，而p起初指向的地址为abc[1]，这里也就是abc[1]++

// 输出p指向的地址所储存的值，p指向abc[2]，为3，输出3
printf("%d", *p);

```

第二题

```
// 定义变量
int abc[5] = {1, 2, 3, 4, 5};
int *p;

// 给定指针p指向的地址
p = &abc[0] + 1;

// 进行指针相关运算
(*(&p))++;// 等价于p++，p指向abc[2]
(*p)++;// p此时指向abc[2]，这里是abc[2]++

// 输出p指向的地址所储存的值,p指向abc[2]，为4，输出4
printf("%d", *p);
```

第三题

```
int abc[5] = {1, 2, 3, 4, 5};
int *p;

p = &abc[0] + 1;

(*(*(&p))++)++;

// 以上代码同第一小题
// 输出p-1指向的地址所储存的值，p-1指向abc[1]，为3，输出3
printf("%d", *(p - 1));
```

第四题

```
int x[5] = {1, 2, 3, 4,5};// x是一个整数数组，其中x[0] = 1 …… x[4] = 5
int *y[5] = {x};// y是一个指针数组，其中y[0]指向x(&x[0])，y[1]-y[4]没有确定的指向

// 进行输出，输出 x[2]可以输出3，而y[2]指向的地址不确定，所以*(y[2])的值不确定
printf("%d %d", x[2], *(y[2]));
```

第五题

```
int *x;// 定义了一个指针x，但没有确定其指向的地址

*x = 10;
// 对x的指向的地址储存的变量赋值，但是不一定可以成功赋值
// 不成功赋值可能因为：1.地址对应的储存空间只读；2.地址对应的储存空间为程序外部的空间；等

x += 10;
*x *= 2;
x -= 10;

// 由于x指向的地址储存的值不确定，所以输出的值不确定
printf("%d", *x);
```

第六题

```
// swap为一个函数，可以传入两个储存int变量的地址
// 对于函数传入地址的情况，函数不能修改地址，但是可以修改地址对应的储存空间存储的值
int swap(int *a, int *b)
{
    int temp;
    temp = *a;
    *a = *b;
    *b = temp;
    // 执行完上述操作后，*a和*b调换，并且这个调换也会在main中生效

    /*
    int *temp
    temp = a;
    a = b;
    b =temp
    */
```

```
    // 如果执行注释中的代码，a，b两个指针调换，但这个调换只在swap函数中生效，不会在main函数中生效

    return 0;
}

int main()
{
    // 定义两个变量
    int x = 20, y = 30;

    // 使用swap函数并传入两个变量
    swap(&x, &y);

    // swap中对指针对应的储存空间内部的值进行操作，在main中生效，所以x，y的值交换
    // 输出x，y，即为30，20
    printf("%d %d", x, y);
    return 0;
}
```

第七题

```
// 定义变量，其中MAX应该作为一个常量宏在程序开始时被定义
char arr_1[MAX], arr_2[MAX] = "";
char *in_line = arr_1, *longest = arr_2, *tmp;
int max_len = 0, len;
while (gets(in_line) != NULL)// 读入一行字符串
{
    len = strlen(in_line);// 判断读入字符串的长度
    if (len > max_len)// 如果程度比已有的最长的还长就讲新的函数地址存入，并记录其长度
    {
        max_len = len;
        tmp = in_line;
        in_line = longest;
        longest = tmp;
        // 使用指针进行操作省略了strcpy或其他的一些赋值操作，可以提高程序效率
    }
}

// 输出第一个出现的最长的字符串长度和内容
// 根据输入应该为10:aaaaaaaaaa
printf("%d:%s\n", max_len, longest);
```

E - 构 造 题

考点	难度
数学思维	4

题目分析

由性质中的 $1 \leq i \leq 4n$ 可以想到，只需要输出区间 $(2n, 4n]$ 内的全部偶数即可满足两两不互素、两两不整除的条件。（此解法不唯一）

示例程序

```c
#include <stdio.h>

int main()
{
    int t, n, i;

    scanf("%d", &t);
    while (t--)
    {
        scanf("%d", &n);
        for (i = 4 * n; i > 2 * n; i -= 2)
            printf("%d ", i);
        puts("");
    }

    return 0;
}
```

F - 峡谷巡逻队

难度	考点
5	字符串、指针

考点分析

由于题目中已经给出了判断回文串的函数，因此只需要枚举出所有的长度大于 1 的子串，并判断这些子串是否是回文串即可。对于一个字符串，可以通过枚举子串起点和子串长度的方式枚举出所有子串，而需要特别注意的是，由于使用指针判断是否是回文串，搭配子串的起点和字串的长度的时候需要满足不越界的要求。

示例代码

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

int ispalindrome(char *p, int n)
{
    int i;
    for (i = 0; i < n / 2; i++)
    {
        if (*(p + i) != *(p + n - 1 - i)) return 0;
    }
    return 1;
}

int main()
{
    char a[305];
    int i, n, j, count = 0;

    gets(a);
    n = strlen(a);
    for (i = 0; i < n; i++)
    {
        for (j = 2; j <= n - i; j++)
        {
            count += ispalindrome(a + i, j);
        }
    }
    if (count == 0)
        printf("No ProbIem!\n0");
    else
        printf("DiDaRen Waring\n%d", count);

    return 0;
}
```

G - Matrix of Power

难度	考点
5	二维数组，矩阵乘法

题目分析

如果只需要做单次的矩阵乘法，则只要根据矩阵乘法的定义，做一个三重循环即可完成任务。

对于本题，实际上就是再多加一重循环，让一个答案矩阵 B 初始为 $B = E$，接着做 k 次 $B \leftarrow B \cdot A$ 的过程，就得到了题目所求的答案。这里需要注意的是：中间的计算结果需要保存在一个临时矩阵中，等所有元素都完成计算后，再将临时矩阵中的内容复制给 B，否则会因为中途修改了 B 中的元素，导致后续的运算结果有误（可以想想为什么）。

示例代码

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

#define N (50 + 5)
#define MOD (299213)

int result[N][N], a[N][N], tmp[N][N];

int main()
{
    int n, t;
    int i, j, k;
    long long sum = 0;

    scanf("%d%d", &n, &t);

    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            scanf("%d", &a[i][j]);

    for (i = 1; i <= n; i++)
        result[i][i] = 1;

    while (t--)
    {
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
            {
                sum = 0;
                for (k = 1; k <= n; k++)
                    sum = (sum + 1LL * result[i][k] * a[k][j]) % MOD;
                // 这里只能先把答案先存到 tmp 矩阵，计算完了再赋值给 result 矩阵
                // 如果修改了 result 矩阵，会导致之后的运算有误
                tmp[i][j] = sum;
            }

```

```
        // 将 tmp 矩阵的答案复制到结果矩阵
        for (i = 1; i <= n; i++)
            for (j = 1; j <= n; j++)
                result[i][j] = tmp[i][j];
    }

    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            printf("%d%c", result[i][j], " \n"[j == n]);

    return 0;
}
```

H - 高精度进制转换

难度	考点
5	高精度除法

题目分析

高精度进制转换本质是模拟除法竖式，计算出余数和商。

除法竖式大家平时可能用的比较少，因此可以先手算一遍除法竖式，回顾一下这个过程，再将这个过程写成代码，具体可以参考示例程序及注释。

示例程序

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

#define maxn 10086

int main()
{
    int t, cnt, len, hi, x, k, i, a[maxn], b[maxn];
    char s[maxn];

    scanf("%d", &t);
    while (t--)
    {
        scanf("%s", s + 1);
        len = strlen(s + 1);
        for (k = 2; k <= 9; k++)
        { // 转换为 k 进制。
            for (i = 1; i <= len; i++)
                a[i] = s[i] - '0';
            cnt = 0, hi = 1; // cnt 表示进制转换后数字的长度，hi 表示十进制数当前的最高位。
            while (hi <= len)
            {
                if (!a[hi])
                { // 当前最高位为 0，直接跳过。
                    hi++;
                    continue;
                }
                x = 0; // x 表示余数。
                for (i = hi; i <= len; i++)
                {
                    a[i] += x * 10; // 把没处理的数“拉”下来，补在 x 后面。
                    x = a[i] % k;
                    a[i] /= k; // 确定商的这一位。
                }
                b[++cnt] = x; // 最终的余数即为进制转换后数字的新一位。
            }
            for (i = cnt; i; i--)
                printf("%d", b[i]); // 倒序输出每一次产生的余数。
```

```
            puts("");
        }
    }

    return 0;
}
```

H - ⑨

难度	考点
6	数组和指针的关系

题目分析

本题是一个字符串分析题，故在开始写程序之前，要先想清楚所有可能出现的情况，理清之后再开始写。

经分析，本题流程如下：

1. 重复读取字符直至数字字符或结束(`'\0'`)，遇到数字则进入 `2.`，遇到 `'\0'` 则结束程序；
2. 读取一个纯数字，记作 $L1$，判断纯数字之后的字符是否为 `'('`，是则进入 `3.`，否则提取数字 $L1$ 并回到 `1.`；
3. 判断 `'('` 之后的字符是否为数字，是则进入 `4.`，否则提取数字 $L1$ 并回到 `1,`；
4. 读取一个纯数字，记作 $L2$，判断纯数字之后的字符是否为 `')'`，是则提取数字 $L1 \times L2$ 并回到 `1.`，否则提取数字 $L1$ 并回到 `1.`，**之后需要复位，重新分析** $L2$。

这里关于 `4.` 中加粗部分解释一下。在这种情况下，不满足积的条件，但是也不能直接提取 $L1 + L2$，原因是 $L2$ 可能和后面的字符共同组成积，比如：`10(10(10)` 这种情况，应该提取到 $10 + 10 \times 10 = 110$ 而非 30。

所以，联系课上所学关于数组和指针的关系，模仿 PPT 上的例题 `7.7`，我们可以构造函数 `getValue()`，这个函数每次会从字符串中提取一个数字（纯数字或积），并返回一个指针，指示接下来需要继续分析的位置。函数接受一个 char 指针作为参数，指示这一轮从这里开始分析。通过返回指针的方式，我们可以实现 4. 中的回溯需求，只要在 2. 之后保存一下 $L1$ 后面的字符地址即可。

此外，关于如何将提取到的数字求和，可以采取全局变量或指针传参的方式，下面程序示例用了后者。

示例代码

```c
#include <ctype.h>
#include <stdio.h>

char *getDigit(char *s, int *L) // 作用类似 getValue, 提取一个纯数字存入 L
{
    int digit = 0;
    while (isdigit(*s))
    {
        digit = digit * 10 + *s - '0';
        s++;
    }
    *L = digit;
    return s;
}

char *getValue(char *s, int *r) // 提取一个数字存入 r
{
    int L1, L2;
    while (!isdigit(*s))
    { // 步骤 1 开始
```

```
        if (*s == '\0')
        { // 结束条件
            return NULL;
        }
        s++;
    }
    s = getDigit(s, &L1); // 步骤 2 开始
    char *s1 = s;         // 存储 s 地址，便于回溯
    if (*s == '(')        // 步骤 3 开始
    {
        s++;
        if (!isdigit(*s)) // 提取到一个纯数字，返回
        {
            *r = L1;
            return s;
        }
        else
        {                           // 步骤 4 开始
            s = getDigit(s, &L2); // 获取 L2
            if (*s != ')')        // 不满足积的条件，提取到一个纯数字，返回（回溯）
            {
                *r = L1;
                return s1;
            }
            else // 提取到一个积，返回
            {
                *r = L1 * L2;
                return s;
            }
        }
    }
    else
    {
        *r = L1; // 提取到一个纯数字，返回
        return s;
    }
}

int main()
{
    char *s, str[150];
    int r, sum = 0;

    scanf("%s", str);
    s = str;
    while ((s = getValue(s, &r)) != NULL)
    {
        sum += r;
    }
    printf("%d", sum);

    return 0;
}
```

J - 扫雷游戏

难度	考点
6	深度优先搜索

题目分析

本题采用递归的方法进行深度优先搜索，如果没有接触过深度优先搜索的同学可以暂且先理解为一种有技巧的递归搜索方法。

首先对输入的数据进行处理，对以一个格子为中心的 3×3 范围内进行遍历，计算雷数并决定是否是空格。

随后就进行深度优先搜索，其核心思路是不断递归。依次遍历每一个格子，每当遇到一个空格，使 op 的值 $+1$，然后以该格为起始格进入递归搜索函数。

在递归搜索函数中，首先将该格标为非空格（避免循环枚举），然后向枚举四个方向，如果在某一方向上遇到了空格就以该格为新的起始格进行递归搜索，如果没有空格就返回上级。

这样每次在主函数中调用函数，就会遍历一整块 op 中所有空格并将它们标记为非空格，这样在 $O(mn)$ 的时间内就可以遍历所有的格子并且统计出 op 的数量。

本题的常见错误有：未判断全四个方向（或者多判断），以及在矩形边界处处理不当。

示例程序

```c
#include <stdio.h>

int t, n, m;
int a[1005][1005];
char b[1005][1005];

int dx[8] = {-1, 0, 0, 1, -1, 1, -1, 1};
int dy[8] = {0, -1, 1, 0, 1, -1, -1, 1};

int vis[1005][1005];
int ans = 0;

char check(int x, int y)
{
    int cnt = 0;
    int i;
    for (i = 0; i < 8; ++i)
        cnt += a[x + dx[i]][y + dy[i]] == 1;
    return cnt ? cnt + '0' : '_';
}

void dfs(int x, int y)
{
    int i;
    vis[x][y] = 1;
    for (i = 0; i < 4; ++i)
    {
```

```
28          if (!vis[x + dx[i]][y + dy[i]])
29              dfs(x + dx[i], y + dy[i]);
30      }
31  }
32
33  int main()
34  {
35      int i, j;
36
37      scanf("%d%d", &m, &n);
38      for (i = 1; i <= m; ++i)
39      {
40          for (j = 1; j <= n; ++j)
41          {
42              scanf("%d", &a[i][j]);
43              if (a[i][j])
44                  b[i][j] = '*';
45          }
46      }
47      for (i = 1; i <= m; ++i)
48      {
49          for (j = 1; j <= n; ++j)
50          {
51              if (!a[i][j])
52                  b[i][j] = check(i, j);
53              vis[i][j] = b[i][j] == '_' ? 0 : 1;
54          }
55      }
56      for (i = 1; i <= m; ++i)
57          vis[i][0] = vis[i][n + 1] = 1;
58      for (i = 1; i <= n; ++i)
59          vis[0][i] = vis[m + 1][i] = 1;
60      for (i = 1; i <= m; ++i)
61      {
62          for (j = 1; j <= n; ++j)
63          {
64              if (!vis[i][j])
65              {
66                  ++ans;
67                  dfs(i, j);
68              }
69          }
70      }
71      printf("%d", ans);
72
73      return 0;
74  }
```