1. **字符串匹配定位**

编写函数 `int locStr(char *str1 , char *str2)` 实现字符串匹配的定位功能；若字符串 `str1` 中含有字符串 `str2`，则返回字符串 `str2` 在字符串 `str1` 中的位置，否则返回 `-1`；

```
#include <iostream>
#include <cstring>
using namespace std;

// 函数 locStr 用于查找 str2 在 str1 中的位置
// 参数：
// str1：源字符串
// str2：要查找的目标字符串
// 返回值：
// 若找到，返回 str2 在 str1 中的起始位置（从 0 开始），若未找到，返回 -1
int locStr(char *str1, char *str2) {
    int len1 = strlen(str1);  // 获取 str1 的长度
    int len2 = strlen(str2);  // 获取 str2 的长度

    // 遍历 str1，考虑可能出现 str2 的位置
    for (int i = 0; i <= len1 - len2; i++) {
        int j;
        // 比较 str1 从 i 开始的 len2 个字符与 str2 是否相同
        for (j = 0; j < len2; j++) {
            if (str1[i + j]!= str2[j]) {
                break;  // 若有字符不匹配，退出比较
            }
        }
        // 若 j 等于 len2，说明找到了 str2
        if (j == len2) {
            return i;
        }
    }
    return -1;  // 未找到，返回 -1
}

int main() {
    // 指定字符串
    char* str1 = "how are you";
    char* str2 = "aru";
    int k = locStr(str1, str2);
    if (k!= -1) {
        cout << str2 << " 在 " << str1 << " 的位置是 " << k << endl;
    } else {
        cout << str2 << " 不在 " << str1 << " 中" << endl;
    }
    return 0;
}
```

2. **随机输入 10 个整数，用冒泡排序将其实现从小到大顺序排序**，要求：

（1）采用顺序结构，程序运行时随机输入 10 个整数

（2）输出时要将排序前和排序后的结果都输出

```
#include <iostream>
using namespace std;

int main() {
    // 动态数组
    int* a;
    int n, i, j, temp;
```

```
    // 输入要排序的数字的数量，这里固定为 10
    n = 10;
    // 分配内存
    a = new int[n];
    cout << "请输入 10 个整数：" << endl;
    // 输入 10 个整数
    for (i = 0; i < n; i++) {
        cin >> a[i];
    }
    cout << "输入的数为：" << endl;
    // 输出输入的数字
    for (i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    // 冒泡排序
    for (i = 0; i < n - 1; i++) {
        // n 个数一共排 n-1 次
        // 从后往前依次将最小的数升至最前边
        for (j = n - 1; j > i; j--) {
            if (a[j - 1] > a[j]) {
                temp = a[j];
                a[j] = a[j - 1];
                a[j - 1] = temp;
            }
        }
    }
    cout << "排序后数为：" << endl;
    // 输出排序后的数字
    for (i = 0; i < n; i++) {
        cout << a[i] << " ";
    }
    cout << endl;
    // 释放存储空间
    delete[] a;
    return 0;
}
```

使用标准库中的 `vector` 容器：

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main() {
    vector<int> a;
    int n = 10;
    int num;
    cout << "请输入 10 个整数：" << endl;
    for (int i = 0; i < n; ++i) {
        cin >> num;
        a.push_back(num);
    }
    cout << "输入的数为：" << endl;
    for (int val : a) {
        cout << val << " ";
    }
    cout << endl;
    // 使用标准库中的 sort 函数排序
    sort(a.begin(), a.end());
```

```
    cout << "排序后数为: " << endl;
    for (int val : a) {
        cout << val << " ";
    }
    cout << endl;
    return 0;
}
```

3. **数字之和 (20 分)**

请设计一个递归函数，求出某个正整数的各位数字之和。该函数的雏形为`int sumDigits(int n);`

例如，`sumDigits(123456)`的返回值为`21`

```
#include <iostream>
using namespace std;

// 显然巧妙的地方在于这个递归调用
// 函数 sumDigits 用于计算一个正整数 n 的各位数字之和
// 参数:
// n: 输入的正整数
// 返回值:
// 各位数字之和
int sumDigits(int n)
{
    // 当 n 是个位数时，直接返回该数字
    if (n / 10 == 0)
    {
        return n % 10;
    }
    else
    {
        // 取 n 的最后一位数字
        int sum = n % 10;
        // 递归调用 sumDigits 函数计算 n 去掉最后一位后的数字的各位数字之和，并累加到 sum 中
        sum += sumDigits(n / 10);
        return sum;
    }
}

int main()
{
    int n;
    cout << "输入数字:" << endl;
    cin >> n;
    // 调用 sumDigits 函数计算各位数字之和并输出结果
    cout << n << " 的各个位的数字之和为:" << sumDigits(n) << endl;
    return 0;
}
```

4. **求解一元二次方程组:aX^2+bX+c=0;**

> 首先确定好分类情况(if情况)
>
> (1)a==0
>
> b==0：无
>
> b!=0：一个
>
> (2)a!=0

> dert>0：一个
>
> dert=0：一个
>
> dert<0:虚根

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
    float a, b, c, dert, x1, x2;
    cout << "输入系数 a,b,c: " << endl;
    cin >> a >> b >> c;
    // 当 a 不等于 0 时
    if (a!= 0)
    {
        dert = b * b - 4 * a * c;
        if (dert > 0)
        {
            // 计算判别式的平方根
            dert = sqrt(dert);
            x1 = (-b + dert) / (2 * a);
            x2 = (-b - dert) / (2 * a);
            cout << "有两个解:" << endl
                 << "x1=" << x1 << endl
                 << "x2=" << x2 << endl;
        }
        else if (dert == 0)
        {
            x1 = (-b) / (2 * a);
            cout << "仅有一个解:" << endl
                 << "x=" << x1 << endl;
        }
        else
        {
            // 计算虚部和实部
            float d = sqrt(-dert) / (2 * a);
            float e = (-b) / (2 * a);
            cout << "虚根为:" << endl
                 << e << "+" << d << "i" << endl
                 << e << "-" << d << "i" << endl;
        }
    }
    else
    {
        if (b == 0)
        {
            cout << "无解" << endl;
        }
        else
        {
            cout << "有一个解:" << -c / b << endl;
        }
    }
    return 0;
}
```

5. **胖胖闯关**

设有n^2个人排成的 n*n 队列，另有队列外的一人 x 试图闯过这个队列。他闯入的方式是首先与第一行第一列的人冲撞，如果 X 的体重不小于该人，则 X 将进入第一行的第二:列，然后继续冲撞该位置上的人:否则，X 将被撞至第二行第一列，继续与该位置的人冲撞。依次类推。如果 X 最后撞倒了第 n 列的某个人，则闯关成功;如果 X 最后被第 n 行的某个人撞出，则闯关失败。
你面临的输入首先是一个正整数 n，然后跟着 n 行正整数，每行又有 n 个正整数，第 i 行第 j 列的正整数表示第 i 行第 j 列的人的体重。然后又是一个正整数，表示闯关人 x 的体重

例如:
3
56 67 65
78 45 67
56 55 57
58
你需要求出的是 X 能否闯关成功，以及在闯关过程中所撞倒的人的体重总和。
上述例子的结果是:闯关成功，闯关过程中所撞倒的人的体重总和为:
56+67+45+67+57= 292.

```cpp
#include <iostream>
using namespace std;

// 开始闯关
// 参数:
// weight:闯关人的体重
// a:存储队列中人员体重的二维数组
// n:队列的行数（或列数）
void game(int weight, int** a, int n)
{
    int sum = 0, i = 0, j = 0;
    for (i = 0; i < n; i++)
    {
        // 内层循环，用于在每一行内进行冲撞
        for (; j < n; j++)
        {
            // 无论胖胖是撞倒别人，还是被撞倒，都要记录下凡是遇到过的人体重，也就是元素值
            sum += a[i][j];
            // 但是被撞倒了，就不能再在当前行上了，掉到下一行的当前列，所以内层 for 循环的列 j 是不能初始化的
            // 所以巧妙的地方就在于 j 的值是只有一次初始化，无论行怎么变化，j 都是继续增加的
            if (weight < a[i][j])
            {
                break;
            }
        }
    }
    // 只有走到了列的尽头才说明走出去了，而至于行的尽头是无法说明通关了的
    if (j == n)
    {
        cout << "胖胖闯关成功,且撞到的人总体重为:" << sum << endl;
    }
    else
    {
        cout << "胖胖闯关失败,且撞到的人总体重为:" << sum << endl;
    }
}

int main()
{
    // 在矩阵运算中也要用到这种方式的二维数组，学会如何申请空间以及释放空间
    int** a;
    int n, i, j, weight;
    cout << "输入几行几列: " << endl;
    cin >> n;
```

```
    // 申请动态内存空间 n*n
    a = new int* [n];
    for (i = 0; i < n; i++)
    {
        a[i] = new int[n];
    }
    // 输入矩阵的值
    for (i = 0; i < n; i++)
    {
        cout << "输入第" << i + 1 << "行元素:" << endl;
        for (j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
    cout << "您输入的" << n << "*" << n << "队列为:" << endl;
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    cout << "输入胖胖的体重:" << endl;
    cin >> weight;
    // 胖胖开始闯关
    game(weight, a, n);
    // 释放内存空间
    for (i = 0; i < n; i++)
    {
        delete[] a[i];
    }
    delete[] a;
    return 0;
}
```

使用 `vector` 容器的实现方式：

```
#include <iostream>
#include <vector>
using namespace std;

// 开始闯关
// 参数:
// weight：闯关人的体重
// a：存储队列中人员体重的二维向量
void game(int weight, vector<vector<int>>& a)
{
    int sum = 0;
    int n = a.size();
    int i = 0, j = 0;
    for (i = 0; i < n; i++)
    {
        for (; j < n; j++)
        {
            sum += a[i][j];
            if (weight < a[i][j])
            {
                break;
            }
```

```
        }
    }
    if (j == n)
    {
        cout << "胖胖闯关成功,且撞到的人总体重为:" << sum << endl;
    }
    else
    {
        cout << "胖胖闯关失败,且撞到的人总体重为:" << sum << endl;
    }
}

int main()
{
    int n;
    cout << "输入几行几列: " << endl;
    cin >> n;
    vector<vector<int>> a(n, vector<int>(n));
    // 输入矩阵的值
    for (int i = 0; i < n; i++)
    {
        cout << "输入第" << i + 1 << "行元素:" << endl;
        for (int j = 0; j < n; j++)
        {
            cin >> a[i][j];
        }
    }
    cout << "您输入的" << n << "*" << n << "队列为:" << endl;
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < n; j++)
        {
            cout << a[i][j] << " ";
        }
        cout << endl;
    }
    int weight;
    cout << "输入胖胖的体重:" << endl;
    cin >> weight;
    // 胖胖开始闯关
    game(weight, a);
    return 0;
}
```