

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`；

```
1  #include <iostream>
2  #include <cstring>
3  using namespace std;
4
5  // 函数 locStr 用于查找 str2 在 str1 中的位置
6  // 参数：
7  // str1: 源字符串
8  // str2: 要查找的目标字符串
9  // 返回值：
10 // 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
11 int locStr(char *str1, char *str2) {
12     int len1 = strlen(str1); // 获取 str1 的长度
13     int len2 = strlen(str2); // 获取 str2 的长度
14
15     // 遍历 str1，考虑可能出现 str2 的位置
16     for (int i = 0; i <= len1 - len2; i++) {
17         int j;
18         // 比较 str1 从 i 开始的 len2 个字符与 str2 是否相同
19         for (j = 0; j < len2; j++) {
20             if (str1[i + j] != str2[j]) {
21                 break; // 若有字符不匹配，退出比较
22             }
23         }
24         // 若 j 等于 len2，说明找到了 str2
25         if (j == len2) {
26             return i;
27         }
28     }
29     return -1; // 未找到，返回 -1
30 }
31
32 int main() {
33     // 指定字符串
34     char* str1 = "how are you";
35     char* str2 = "aru";
36     int k = locStr(str1, str2);
37     if (k != -1) {
38         cout << str2 << " 在 " << str1 << " 的位置是 " << k << endl;
39     } else {
40         cout << str2 << " 不在 " << str1 << " 中" << endl;
41     }
42     return 0;
43 }
```

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

```
1  #include <iostream>
2  using namespace std;
3
4  int main() {
5     // 动态数组
6     int* a;
7     int n, i, j, temp;
```

```

8 // 输入要排序的数字的数量，这里固定为 10
9 n = 10;
10 // 分配内存
11 a = new int[n];
12 cout << "请输入 10 个整数: " << endl;
13 // 输入 10 个整数
14 for (i = 0; i < n; i++) {
15     cin >> a[i];
16 }
17 cout << "输入的数为: " << endl;
18 // 输出输入的数字
19 for (i = 0; i < n; i++) {
20     cout << a[i] << " ";
21 }
22 cout << endl;
23 // 冒泡排序
24 for (i = 0; i < n - 1; i++) {
25     // n 个数一共排 n-1 次
26     // 从后往前依次将最小的数升至最前边
27     for (j = n - 1; j > i; j--) {
28         if (a[j - 1] > a[j]) {
29             temp = a[j];
30             a[j] = a[j - 1];
31             a[j - 1] = temp;
32         }
33     }
34 }
35 cout << "排序后数为: " << endl;
36 // 输出排序后的数字
37 for (i = 0; i < n; i++) {
38     cout << a[i] << " ";
39 }
40 cout << endl;
41 // 释放存储空间
42 delete[] a;
43 return 0;
44 }

```

使用标准库中的 `vector` 容器：

```

1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4 using namespace std;
5
6 int main() {
7     vector<int> a;
8     int n = 10;
9     int num;
10    cout << "请输入 10 个整数: " << endl;
11    for (int i = 0; i < n; ++i) {
12        cin >> num;
13        a.push_back(num);
14    }
15    cout << "输入的数为: " << endl;
16    for (int val : a) {
17        cout << val << " ";
18    }
19    cout << endl;
20    // 使用标准库中的 sort 函数排序
21    sort(a.begin(), a.end());

```

```

22     cout << "排序后数为: " << endl;
23     for (int val : a) {
24         cout << val << " ";
25     }
26     cout << endl;
27     return 0;
28 }

```

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

```

1  #include <iostream>
2  using namespace std;
3
4  // 显然巧妙的地方在于这个递归调用
5  // 函数 sumDigits 用于计算一个正整数 n 的各位数字之和
6  // 参数:
7  // n: 输入的正整数
8  // 返回值:
9  // 各位数字之和
10 int sumDigits(int n)
11 {
12     // 当 n 是个位数时，直接返回该数字
13     if (n / 10 == 0)
14     {
15         return n % 10;
16     }
17     else
18     {
19         // 取 n 的最后一位数字
20         int sum = n % 10;
21         // 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
22         sum += sumDigits(n / 10);
23         return sum;
24     }
25 }
26
27 int main()
28 {
29     int n;
30     cout << "输入数字:" << endl;
31     cin >> n;
32     // 调用 sumDigits 函数计算各位数字之和并输出结果
33     cout << n << " 的各个位的数字之和为:" << sumDigits(n) << endl;
34     return 0;
35 }

```

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

```
1 #include <iostream>
2 #include <cmath>
3 using namespace std;
4
5 int main()
6 {
7     float a, b, c, dert, x1, x2;
8     cout << "输入系数 a,b,c: " << endl;
9     cin >> a >> b >> c;
10    // 当 a 不等于 0 时
11    if (a != 0)
12    {
13        dert = b * b - 4 * a * c;
14        if (dert > 0)
15        {
16            // 计算判别式的平方根
17            dert = sqrt(dert);
18            x1 = (-b + dert) / (2 * a);
19            x2 = (-b - dert) / (2 * a);
20            cout << "有两个解:" << endl
21                << "x1=" << x1 << endl
22                << "x2=" << x2 << endl;
23        }
24        else if (dert == 0)
25        {
26            x1 = (-b) / (2 * a);
27            cout << "仅有一个解:" << endl
28                << "x=" << x1 << endl;
29        }
30        else
31        {
32            // 计算虚部和实部
33            float d = sqrt(-dert) / (2 * a);
34            float e = (-b) / (2 * a);
35            cout << "虚根为:" << endl
36                << e << "+" << d << "i" << endl
37                << e << "-" << d << "i" << endl;
38        }
39    }
40    else
41    {
42        if (b == 0)
43        {
44            cout << "无解" << endl;
45        }
46        else
47        {
48            cout << "有一个解:" << -c / b << endl;
49        }
50    }
51    return 0;
52 }
```

5. 胖胖闯关

设有 n^2 个人排成的 $n \times n$ 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:

$56+67+45+67+57= 292$.

```
1  #include <iostream>
2  using namespace std;
3
4  // 开始闯关
5  // 参数:
6  // weight: 闯关人的体重
7  // a: 存储队列中人员体重的二维数组
8  // n: 队列的行数（或列数）
9  void game(int weight, int** a, int n)
10 {
11     int sum = 0, i = 0, j = 0;
12     for (i = 0; i < n; i++)
13     {
14         // 内层循环，用于在每一行内进行冲撞
15         for (; j < n; j++)
16         {
17             // 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
18             sum += a[i][j];
19             // 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
20             // 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
21             if (weight < a[i][j])
22             {
23                 break;
24             }
25         }
26     }
27     // 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
28     if (j == n)
29     {
30         cout << "胖胖闯关成功,且撞到的人总体重为:" << sum << endl;
31     }
32     else
33     {
34         cout << "胖胖闯关失败,且撞到的人总体重为:" << sum << endl;
35     }
36 }
37
38 int main()
39 {
40     // 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
41     int** a;
42     int n, i, j, weight;
43     cout << "输入几行几列: " << endl;
44     cin >> n;
```

```

45 // 申请动态内存空间 n*n
46 a = new int* [n];
47 for (i = 0; i < n; i++)
48 {
49     a[i] = new int[n];
50 }
51 // 输入矩阵的值
52 for (i = 0; i < n; i++)
53 {
54     cout << "输入第" << i + 1 << "行元素:" << endl;
55     for (j = 0; j < n; j++)
56     {
57         cin >> a[i][j];
58     }
59 }
60 cout << "您输入的" << n << "*" << n << "队列为:" << endl;
61 for (i = 0; i < n; i++)
62 {
63     for (j = 0; j < n; j++)
64     {
65         cout << a[i][j] << " ";
66     }
67     cout << endl;
68 }
69 cout << "输入胖胖的体重:" << endl;
70 cin >> weight;
71 // 胖胖开始闯关
72 game(weight, a, n);
73 // 释放内存空间
74 for (i = 0; i < n; i++)
75 {
76     delete[] a[i];
77 }
78 delete[] a;
79 return 0;
80 }

```

使用 `vector` 容器的实现方式：

```

1 #include <iostream>
2 #include <vector>
3 using namespace std;
4
5 // 开始闯关
6 // 参数:
7 // weight: 闯关人的体重
8 // a: 存储队列中人员体重的二维向量
9 void game(int weight, vector<vector<int>>& a)
10 {
11     int sum = 0;
12     int n = a.size();
13     int i = 0, j = 0;
14     for (i = 0; i < n; i++)
15     {
16         for (; j < n; j++)
17         {
18             sum += a[i][j];
19             if (weight < a[i][j])
20             {
21                 break;
22             }

```

```
23     }
24 }
25 if (j == n)
26 {
27     cout << "胖胖闯关成功,且撞到的人总体重为:" << sum << endl;
28 }
29 else
30 {
31     cout << "胖胖闯关失败,且撞到的人总体重为:" << sum << endl;
32 }
33 }
34
35 int main()
36 {
37     int n;
38     cout << "输入几行几列: " << endl;
39     cin >> n;
40     vector<vector<int>> a(n, vector<int>(n));
41     // 输入矩阵的值
42     for (int i = 0; i < n; i++)
43     {
44         cout << "输入第" << i + 1 << "行元素:" << endl;
45         for (int j = 0; j < n; j++)
46         {
47             cin >> a[i][j];
48         }
49     }
50     cout << "您输入的" << n << "*" << n << "队列为:" << endl;
51     for (int i = 0; i < n; i++)
52     {
53         for (int j = 0; j < n; j++)
54         {
55             cout << a[i][j] << " ";
56         }
57         cout << endl;
58     }
59     int weight;
60     cout << "输入胖胖的体重:" << endl;
61     cin >> weight;
62     // 胖胖开始闯关
63     game(weight, a);
64     return 0;
65 }
```