

1. 均值方差

2. 求数组中后一个与前一个相差为 1 的数对，打印出来

例：输入：2674831

输出：(6,7) (7,8)

```
1 #include <iostream>
2 #include <cmath> // 用于 abs 函数
3 using namespace std;
4
5 int main() {
6     int size;
7     cout << "准备输入几个数： ";
8     cin >> size;
9
10    // 输入验证
11    if (size <= 0) {
12        cout << "输入的数组合法大小必须大于 0! " << endl;
13        return 1;
14    }
15
16    int* arr = new int[size];
17    cout << "输入这几个数： ";
18    for (int i = 0; i < size; i++) {
19        cin >> arr[i];
20    }
21
22    cout << "你输入的数字是： ";
23    for (int i = 0; i < size; i++) {
24        cout << arr[i] << " ";
25    }
26    cout << endl;
27
28    cout << "其中前后相差为 1 的数对有： ";
29    for (int i = 0; i < size - 1; i++) {
30        if (abs(arr[i] - arr[i + 1]) == 1) {
31            cout << "(" << arr[i] << ", " << arr[i + 1] << ") ";
32        }
33    }
34    cout << endl;
35
36    // 释放动态数组内存
37    delete[] arr;
38    return 0;
39 }
```

3. 统计连续字符个数：同 2014 专硕第二题

4. ip 地址划分并判断类型。同 2015 专硕第二题

5. 1. 设计矩阵类:私有数据成员：row，col，int** a(双重指针)

2. 重载构造函数中初始化双重指针（申请内存空间，也就是形成了二维数组，并全部赋值为 0）

a=new int*[row];//先申请每一行

for(i=0;i<row;i++)

a[i]=new int[col];再为每一行申请 col 个元素

3. 拷贝构造函数：与构造类似，也是申请二维数组空间，但是传入的参数是(constjuzhen& j2),拷贝的格式就是这样的

4. 析构函数:释放内存空间，一定要注意步骤，先释放行:每一行的元素，再释放每一行，也就是与申请的时候相反

5. = , 赋值运算符的重载：为什么要用到这个呢？（不用的话就是：juzhen j3(j1+j2);调用的就是拷贝构造函数）就是在如 j3=j1+j2;的时候我们先做的+运算符的重载，结果返回的矩阵要赋值给j3，所以我们要首先将 j3 的内存释放掉，重新申请内存，就是为了防止 j3 的行列与 j1 , j2 不同。而函数中的 j3 就是 * this这个引用，然后就是简单的将 j1 , j2 对应的每行没列元素加给 j3 就行了，最后返回 j3 就行了，注意返回的是引用，也就是*this
6. + , -运算符的重载：首先建立一个新的矩阵对象用来储存运算出来的矩阵，然后判断相加矩阵是否符合行列相同的原则，否则是无法进行加法运算的。若符合，那么就是两个 for 循环，完成相加减
7. *运算符的重载：乘法的规则：行与列相同才可以相乘，如 2 * 3 矩阵可以与 3 * 5矩阵=2 * 5 矩阵，否则是无法相乘的。符合条件，三个 for 循环：第一层 for 是第一个矩阵的行，第二层 for 是第二个矩阵的列，第三层 for 是第一个矩阵的列，
- 注意：结果是第一个矩阵的某一行 * 第二个矩阵的某一列的和，所以在第二层矩阵里面加一个 sum，最后将 sum 依次赋值给第三个矩阵对应的位置*
8. 转置矩阵的函数：也就是例如原来的 2 * 3 变为 3 * 2，那么首先是将当前的矩阵的元素赋值给一个临时的 3 * 2 矩阵，然后将原矩阵申请的空间释放，重新申请 3 * 2 的二维数组而不是 2 * 3，最后直接 a=temp 就可以了

```
1  #include <iostream>
2  using namespace std;
3
4  class Matrix {
5  private:
6      int row;    // 矩阵的行数
7      int col;    // 矩阵的列数
8      int** data; // 二维数组，存储矩阵数据
9
10 public:
11     // 构造函数：初始化矩阵并分配内存
12     Matrix(int r, int c) : row(r), col(c) {
13         data = new int*[row];
14         for (int i = 0; i < row; i++) {
15             data[i] = new int[col];
16             for (int j = 0; j < col; j++) {
17                 data[i][j] = 0; // 初始化为 0
18             }
19         }
20     }
21
22     // 拷贝构造函数：深拷贝
23     Matrix(const Matrix& other) : row(other.row), col(other.col) {
24         data = new int*[row];
25         for (int i = 0; i < row; i++) {
26             data[i] = new int[col];
27             for (int j = 0; j < col; j++) {
28                 data[i][j] = other.data[i][j];
29             }
30         }
31     }
32
33     // 析构函数：释放内存
34     ~Matrix() {
35         for (int i = 0; i < row; i++) {
36             delete[] data[i];
37         }
38         delete[] data;
39     }
40
41     // 输入矩阵元素
42     void input() {
43         cout << "请输入矩阵元素 (" << row << "x" << col << "):" << endl;
44         for (int i = 0; i < row; i++) {
```

```
45         for (int j = 0; j < col; j++) {
46             cin >> data[i][j];
47         }
48     }
49 }
50
51 // 输出矩阵元素
52 void output() const {
53     cout << "当前矩阵 (" << row << "x" << col << "):" << endl;
54     for (int i = 0; i < row; i++) {
55         for (int j = 0; j < col; j++) {
56             cout << data[i][j] << " ";
57         }
58         cout << endl;
59     }
60 }
61
62 // 赋值运算符重载
63 Matrix& operator=(const Matrix& other) {
64     if (this == &other) {
65         return *this; // 防止自赋值
66     }
67
68     // 释放原有内存
69     for (int i = 0; i < row; i++) {
70         delete[] data[i];
71     }
72     delete[] data;
73
74     // 分配新内存
75     row = other.row;
76     col = other.col;
77     data = new int*[row];
78     for (int i = 0; i < row; i++) {
79         data[i] = new int[col];
80         for (int j = 0; j < col; j++) {
81             data[i][j] = other.data[i][j];
82         }
83     }
84
85     return *this;
86 }
87
88 // 矩阵加法
89 Matrix operator+(const Matrix& other) const {
90     if (row != other.row || col != other.col) {
91         throw invalid_argument("矩阵行列不匹配，无法相加！");
92     }
93
94     Matrix result(row, col);
95     for (int i = 0; i < row; i++) {
96         for (int j = 0; j < col; j++) {
97             result.data[i][j] = data[i][j] + other.data[i][j];
98         }
99     }
100     return result;
101 }
102
103 // 矩阵减法
104 Matrix operator-(const Matrix& other) const {
105     if (row != other.row || col != other.col) {
```

```
106         throw invalid_argument("矩阵行列不匹配，无法相减！");
107     }
108
109     Matrix result(row, col);
110     for (int i = 0; i < row; i++) {
111         for (int j = 0; j < col; j++) {
112             result.data[i][j] = data[i][j] - other.data[i][j];
113         }
114     }
115     return result;
116 }
117
118 // 矩阵乘法
119 Matrix operator*(const Matrix& other) const {
120     if (col != other.row) {
121         throw invalid_argument("矩阵行列不匹配，无法相乘！");
122     }
123
124     Matrix result(row, other.col);
125     for (int i = 0; i < row; i++) {
126         for (int j = 0; j < other.col; j++) {
127             int sum = 0;
128             for (int k = 0; k < col; k++) {
129                 sum += data[i][k] * other.data[k][j];
130             }
131             result.data[i][j] = sum;
132         }
133     }
134     return result;
135 }
136
137 // 矩阵转置
138 Matrix transpose() const {
139     Matrix result(col, row);
140     for (int i = 0; i < row; i++) {
141         for (int j = 0; j < col; j++) {
142             result.data[j][i] = data[i][j];
143         }
144     }
145     return result;
146 }
147 };
148
149 int main() {
150     try {
151         Matrix m1(2, 3);
152         Matrix m2(2, 3);
153
154         cout << "输入矩阵 m1:" << endl;
155         m1.input();
156         cout << "输入矩阵 m2:" << endl;
157         m2.input();
158
159         cout << "矩阵 m1:" << endl;
160         m1.output();
161         cout << "矩阵 m2:" << endl;
162         m2.output();
163
164         cout << "m1 + m2:" << endl;
165         (m1 + m2).output();
166     }
```

```
167         cout << "m1 - m2:" << endl;
168         (m1 - m2).output();
169
170         cout << "m1 的转置:" << endl;
171         m1.transpose().output();
172
173         Matrix m3(3, 2);
174         cout << "输入矩阵 m3 (3x2):" << endl;
175         m3.input();
176         cout << "m1 * m3:" << endl;
177         (m1 * m3).output();
178     } catch (const invalid_argument& e) {
179         cerr << "错误: " << e.what() << endl;
180     }
181
182     return 0;
183 }
```