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）

```cpp
#include <iostream>
#include <cmath>  // 用于 abs 函数
using namespace std;

int main() {
    int size;
    cout << "准备输入几个数: ";
    cin >> size;

    // 输入验证
    if (size <= 0) {
        cout << "输入的数组合法大小必须大于 0! " << endl;
        return 1;
    }

    int* arr = new int[size];
    cout << "输入这几个数: ";
    for (int i = 0; i < size; i++) {
        cin >> arr[i];
    }

    cout << "你输入的数字是: ";
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;

    cout << "其中前后相差为 1 的数对有: ";
    for (int i = 0; i < size - 1; i++) {
        if (abs(arr[i] - arr[i + 1]) == 1) {
            cout << "(" << arr[i] << "," << arr[i + 1] << ") ";
        }
    }
    cout << endl;

    // 释放动态数组内存
    delete[] arr;
    return 0;
}
```

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

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

class Matrix {
private:
    int row;     // 矩阵的行数
    int col;     // 矩阵的列数
    int** data; // 二维数组，存储矩阵数据

public:
    // 构造函数：初始化矩阵并分配内存
    Matrix(int r, int c) : row(r), col(c) {
        data = new int*[row];
        for (int i = 0; i < row; i++) {
            data[i] = new int[col];
            for (int j = 0; j < col; j++) {
                data[i][j] = 0; // 初始化为 0
            }
        }
    }

    // 拷贝构造函数：深拷贝
    Matrix(const Matrix& other) : row(other.row), col(other.col) {
        data = new int*[row];
        for (int i = 0; i < row; i++) {
            data[i] = new int[col];
            for (int j = 0; j < col; j++) {
                data[i][j] = other.data[i][j];
            }
        }
    }

    // 析构函数：释放内存
    ~Matrix() {
        for (int i = 0; i < row; i++) {
            delete[] data[i];
        }
        delete[] data;
    }

    // 输入矩阵元素
    void input() {
        cout << "请输入矩阵元素 (" << row << "x" << col << "):" << endl;
        for (int i = 0; i < row; i++) {
```

```
            for (int j = 0; j < col; j++) {
                cin >> data[i][j];
            }
        }
    }

    // 输出矩阵元素
    void output() const {
        cout << "当前矩阵 (" << row << "x" << col << "):" << endl;
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                cout << data[i][j] << " ";
            }
            cout << endl;
        }
    }

    // 赋值运算符重载
    Matrix& operator=(const Matrix& other) {
        if (this == &other) {
            return *this; // 防止自赋值
        }

        // 释放原有内存
        for (int i = 0; i < row; i++) {
            delete[] data[i];
        }
        delete[] data;

        // 分配新内存
        row = other.row;
        col = other.col;
        data = new int*[row];
        for (int i = 0; i < row; i++) {
            data[i] = new int[col];
            for (int j = 0; j < col; j++) {
                data[i][j] = other.data[i][j];
            }
        }

        return *this;
    }

    // 矩阵加法
    Matrix operator+(const Matrix& other) const {
        if (row != other.row || col != other.col) {
            throw invalid_argument("矩阵行列不匹配，无法相加！");
        }

        Matrix result(row, col);
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                result.data[i][j] = data[i][j] + other.data[i][j];
            }
        }
        return result;
    }

    // 矩阵减法
    Matrix operator-(const Matrix& other) const {
        if (row != other.row || col != other.col) {
```

```
            throw invalid_argument("矩阵行列不匹配，无法相减！");
        }

        Matrix result(row, col);
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                result.data[i][j] = data[i][j] - other.data[i][j];
            }
        }
        return result;
    }

    // 矩阵乘法
    Matrix operator*(const Matrix& other) const {
        if (col != other.row) {
            throw invalid_argument("矩阵行列不匹配，无法相乘！");
        }

        Matrix result(row, other.col);
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < other.col; j++) {
                int sum = 0;
                for (int k = 0; k < col; k++) {
                    sum += data[i][k] * other.data[k][j];
                }
                result.data[i][j] = sum;
            }
        }
        return result;
    }

    // 矩阵转置
    Matrix transpose() const {
        Matrix result(col, row);
        for (int i = 0; i < row; i++) {
            for (int j = 0; j < col; j++) {
                result.data[j][i] = data[i][j];
            }
        }
        return result;
    }
};

int main() {
    try {
        Matrix m1(2, 3);
        Matrix m2(2, 3);

        cout << "输入矩阵 m1:" << endl;
        m1.input();
        cout << "输入矩阵 m2:" << endl;
        m2.input();

        cout << "矩阵 m1:" << endl;
        m1.output();
        cout << "矩阵 m2:" << endl;
        m2.output();

        cout << "m1 + m2:" << endl;
        (m1 + m2).output();

```

```
        cout << "m1 - m2:" << endl;
        (m1 - m2).output();

        cout << "m1 的转置:" << endl;
        m1.transpose().output();

        Matrix m3(3, 2);
        cout << "输入矩阵 m3 (3x2):" << endl;
        m3.input();
        cout << "m1 * m3:" << endl;
        (m1 * m3).output();
    } catch (const invalid_argument& e) {
        cerr << "错误: " << e.what() << endl;
    }

    return 0;
}
```