实验13

1 （10分）

设计一个点类(Point)，具有数据成员x,y（点的坐标），以及设置、输出数据成员及求两点之间距离的功能。再编写主函数对该类进行测试。

```cpp
#include <iostream>
#include <cmath> // 用于计算平方根
#include <iostream>
using namespace std;

// 定义点类 Point
class Point {
private:
    double x; // x坐标
    double y; // y坐标

public:
    // 构造函数
    Point(double x_val = 0, double y_val = 0) : x(x_val), y(y_val) {}

    // 设置坐标
    void setPoint(double x_val, double y_val) {
        x = x_val;
        y = y_val;
    }

    // 输出坐标
    void display() const {
        cout << "点的坐标为: (" << x << ", " << y << ")" << endl;
    }

    // 计算两点之间的距离
    double distance(const Point& other) const {
        return sqrt((x - other.x) * (x - other.x) + (y - other.y) * (y - other.y));
    }
};

int main() {
    // 定义两个点对象
    Point p1, p2;
```

```
36
37      // 设置点的坐标
38      p1.setPoint(3.0, 4.0);
39      p2.setPoint(0.0, 0.0);
40
41      // 输出点的坐标
42      cout << "第一个";
43      p1.display();
44      cout << "第二个";
45      p2.display();
46
47      // 计算并输出两点之间的距离
48      cout << "两点之间的距离为: " << p1.distance(p2) << endl;
49
50          system("pause");
51      return 0;
52  }
```

代码解释：

1. **类定义**：
   - `x` 和 `y` 是点的坐标，定义为私有数据成员。
   - 提供了构造函数初始化坐标，并有设置和显示坐标的函数。
   - `distance` 函数接受另一个 `Point` 对象作为参数，使用欧几里得公式计算两点之间的距离。
2. **主函数**：
   - 创建两个 `Point` 对象。
   - 使用 `setPoint` 方法设置坐标。
   - 使用 `display` 方法输出坐标。
   - 使用 `distance` 方法计算两点之间的距离。

2 （10分）

设计一个字符串类(Mystring)，除具有一般的输入输出字符串的功能外，还要求具有计算字符串长度、连接两个字符串等功能，其中求字符串长度和连接字符串功能不能直接调用字符串处理函数。再编写主函数对该类进行测试。

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

// 定义字符串类 Mystring
class Mystring {
private:
    char* str;      // 字符串存储
    int length;     // 字符串长度

public:
    // 构造函数：初始化空字符串
    Mystring() {
        length = 0;
        str = new char[1];
        str[0] = '\0';
    }

    // 带参构造函数：从输入字符串初始化
    Mystring(const char* input) {
        length = 0;
        while (input[length] != '\0') { // 手动计算字符串长度
            length++;
        }

        str = new char[length + 1]; // 分配内存
        for (int i = 0; i < length; i++) {
            str[i] = input[i]; // 复制字符串
        }
        str[length] = '\0'; // 添加结束符
    }

    // 析构函数：释放内存
    ~Mystring() {
        delete[] str;
    }

    // 计算字符串长度
    int getLength() const {
        int len = 0;
        while (str[len] != '\0') {
```

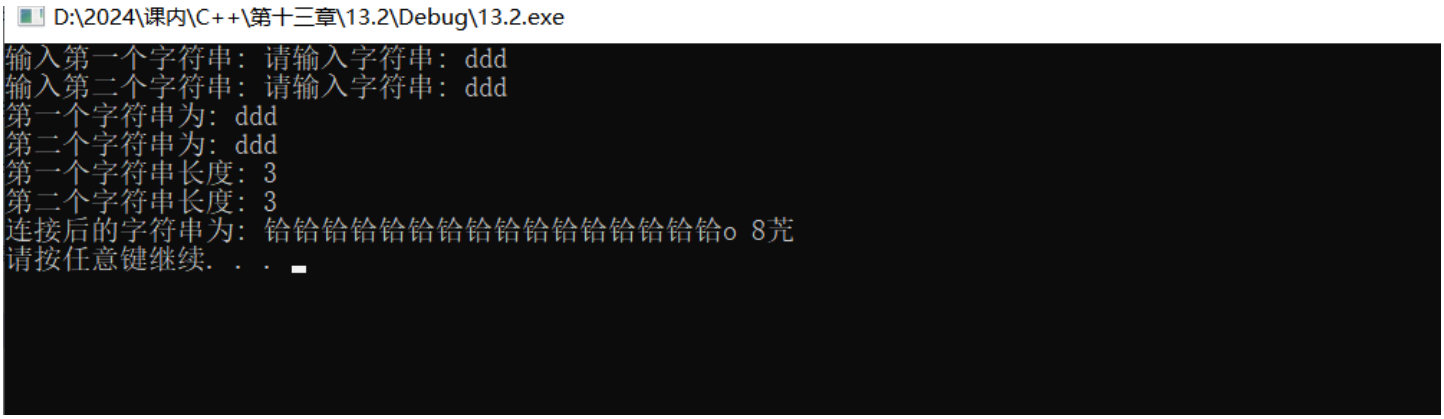
```
            len++;
        }
        return len;
    }

    // 连接两个字符串
    Mystring concat(const Mystring& other) const {
        int newLength = length + other.length; // 新字符串的长度
        char* newStr = new char[newLength + 1]; // 分配新空间

        // 复制第一个字符串
        for (int i = 0; i < length; i++) {
            newStr[i] = str[i];
        }

        // 复制第二个字符串
        for (int i = 0; i < other.length; i++) {
            newStr[length + i] = other.str[i];
        }
        newStr[newLength] = '\0'; // 添加结束符

        return Mystring(newStr);
    }

    // 输入字符串
    void input() {
        char temp[1000]; // 临时缓冲区
        cout << "请输入字符串: ";
        cin >> temp;

        // 重新分配空间并存储输入字符串
        length = 0;
        while (temp[length] != '\0') {
            length++;
        }

        delete[] str; // 释放旧空间
        str = new char[length + 1];
        for (int i = 0; i < length; i++) {
            str[i] = temp[i];
        }
        str[length] = '\0';
    }

    // 输出字符串
    void output() const {
        cout << str << endl;
```

```cpp
    }
};

int main() {
    // 创建 Mystring 对象
    Mystring str1, str2, str3;

    // 输入第一个字符串
    cout << "输入第一个字符串: ";
    str1.input();

    // 输入第二个字符串
    cout << "输入第二个字符串: ";
    str2.input();

    // 输出字符串内容
    cout << "第一个字符串为: ";
    str1.output();
    cout << "第二个字符串为: ";
    str2.output();

    // 计算并输出字符串长度
    cout << "第一个字符串长度: " << str1.getLength() << endl;
    cout << "第二个字符串长度: " << str2.getLength() << endl;

    // 连接两个字符串
    str3 = str1.concat(str2);
    cout << "连接后的字符串为: ";
    str3.output();

        system("pause");
    return 0;
}
```

代码解释：

1. **手动计算字符串长度**：
   - 使用 `while` 循环逐字符遍历字符串，直到遇到结束符 `\0`。
2. **字符串连接**：
   - 先计算新字符串的长度。
   - 分配新的内存空间，将两个字符串逐字符复制到新空间中。
3. **输入与输出**：
   - 使用临时缓冲区 `temp` 接收输入，然后手动复制内容到类的字符串数组中。

4. **内存管理**：
   - 使用动态内存分配 `new` 和 `delete`，确保不会发生内存泄漏。



3 （10分）

设计一个分数类Fraction。该类的数据成员包括分子fz和分母fm；类中还包括如下成员函数：

（1）构造函数，用于初始化分子和分母。

（2）成员函数print，将分数以 "fz/fm" 的形式输出。

（3）成员函数Reduction，用于对分数的分子和分母进行约分。

再编写主函数对该类进行测试。

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

// 定义分数类 Fraction
class Fraction {
private:
    int fz; // 分子
    int fm; // 分母

    // 求最大公约数的辅助函数（使用欧几里得算法）
    int gcd(int a, int b) {
        while (b != 0) {
            int temp = b;
            b = a % b;
            a = temp;
```

```
        }
        return a;
    }

public:
    // 构造函数：初始化分子和分母
    Fraction(int numerator = 0, int denominator = 1) {
        if (denominator == 0) { // 分母不能为0
            cout << "分母不能为0，自动设置为1" << endl;
            denominator = 1;
        }
        fz = numerator;
        fm = denominator;
        Reduction(); // 在构造时进行约分
    }

    // 成员函数：输出分数
    void print() const {
        cout << fz << "/" << fm << endl;
    }

    // 成员函数：约分分数
    void Reduction() {
        int divisor = gcd(fz, fm); // 求分子和分母的最大公约数
        fz /= divisor;
        fm /= divisor;

        // 处理分母为负数的情况（规范化分数形式）
        if (fm < 0) {
            fz = -fz;
            fm = -fm;
        }
    }
};

int main() {
    int numerator, denominator;

    // 输入分子和分母
    cout << "请输入分数的分子: ";
    cin >> numerator;
    cout << "请输入分数的分母: ";
    cin >> denominator;

    // 创建 Fraction 对象并进行测试
    Fraction frac(numerator, denominator);

```

```
63      cout << "约分后的分数为: ";
64      frac.print();
65
66      return 0;
67  }
```

代码解释：

1. **最大公约数函数 `gcd`：**
   - 使用欧几里得算法递归求解分子和分母的最大公约数。
2. **构造函数：**
   - 接收分子和分母作为参数。
   - 检查分母是否为0，若为0则默认设置为1。
   - 调用 `Reduction` 函数进行分数约分。
3. **约分函数 `Reduction`：**
   - 使用最大公约数将分子和分母同时除以公约数进行约分。
   - 如果分母为负数，将分子和分母的符号统一处理，使分母为正。
4. **输出函数 `print`：**
   - 以 `fz/fm` 的格式输出分数。
5. **主函数：**
   - 输入分数的分子和分母，创建 `Fraction` 对象，并输出约分后的结果。