Задание 1а. Написание программы в соответствии со структурным подходом на языке С++.

Условие задачи

Из входного потока вводится непрямоугольная матрица вещественных чисел двойной точности.

В каждой строке матрицы найти максимальный элемент, а затем среди найденных максимумов найти минимальный элемент. Значение найденного элемента вывести в выходной поток (на экран).

Примечание: матрица непрямоугольная, память выделяется динамически. Для задания строки матрицы используется структура.

Особенности реализации

Структура программы

Прежде всего, рассматриваемые в данном семестре задачи реализуются, а общем, достаточно объемными программными кодами. Поэтому целесообразно сразу же использовать следующие подходы:

а) текст программы следует размещать в нескольких файлах. Целесообразно выделять:

файлы-заголовки для прототипов функций и объявлений типов данных и констант;

файлы для реализации требуемых функций;

отдельный файл для тестирующей программы;

б) целесообразно программные методы, используемые для решения задачи, размещать в собственном пространстве имен;

в) в программах должна быть предусмотрена проверка ошибочных ситуаций. Там, где это необходимо, для обработки ошибок использовать исключения.

Для реализации программы в условиях данной задачи будут использованы три файла:

Prog1.h – прототипы всех функций, задание структур

Prog1.cpp – реализация всех функций

Prog1main.cpp – функция `main()`, реализующая отладку программы.

Реализация функций

Из условий задачи следует, что в ряде случаев необходимо реализовать одинаковую (по смыслу) обработку данных, но для разных типов данных (например, ввод данных типа `int` и типа `double` с защитой от ошибок) или для разных конкретных способов обработки (например, поиск максимального или минимального значений).

В первом случае необходимо иметь несколько разных функций, которые могут иметь одно и то же имя, но должны отличаться типом (и/или количеством) параметров; здесь следует воспользоваться механизмом перегрузки функций.

Соответственно, для ввода нам необходимы функции `int getNum(int &)` для ввода данных типа `int` и `int getNum(double &)` для ввода данных типа `double`.

Рассмотрим возможную реализацию этих функций.

```
// функция ввода значения типа int
int getNum(int &a)
{
    std::cin >> a;
    if (!std::cin.good())  // обнаружена ошибка ввода или конец файла
        return -1;
    return 1;
}
// функция ввода значения типа double
int getNum(double &a)
{
    std::cin >> a;
    if (!std::cin.good())  // обнаружена ошибка ввода или конец файла
        return -1;
    return 1;
}
```

Как видно из этих текстов, две функции отличаются только заданием типа параметра функции, в остальном все коды одинаковы. Поэтому данную функцию можно реализовать с помощью шаблона функций, который должен быть размещен в файле-заголовке:

```
// шаблон функций ввода одного значения произвольного типа
template <class T>
int getNum(T &a)
{
    std::cin >> a;
    if (!std::cin.good())  // обнаружена ошибка ввода или конец файла
        return -1;
    return 1;
}
```

Во втором случае можно воспользоваться одной (обобщенной) функцией, которой в качестве параметра следует передать критерий получения результата – т.е., в нашем примере, требуемую функцию сравнения. Данная функция сравнения принимает два параметра и возвращает результат истина, если необходимо выбрать первый из двух параметров. Соответственно, для поиска минимального и максимального значений из последовательности нам нужно иметь две разные функции сравнения:

– выбрать большее значение

```
int isgreater(double a, double b)
{
    return a > b;
}
```

– выбрать меньшее значение

```
int isless(double a, double b)
{
    return a < b;
}
```

Поскольку тело функций реализуется простым выражением, целесообразно данные функции объявить как `inline`- (встроенные) функции; в этом случае реализация встроенных функций также должна быть размещена в файле-заголовке.

Следовательно, программа включает в себя пространство имен `Prog1`, в котором представлены прототипы и реализация следующих функций: ввода данных типа `int` и `double` с проверкой ошибок ввода (используется ввод/вывод в системе C++); ввода непрямоугольной матрицы с необходимым выделением памяти (в стиле C++); вывода матрицы; поиска минимального (максимального) значения последовательности; функции сравнения двух элементов.

Исходный текст

Файл Prog1.h

– содержит структуры для задания непрямоугольной матрицы и прототипы всех методов

```
namespace Prog1{
    // структура для задания строки матрицы
    struct Line{
        int n; // количество элементов в строке матрицы
        double *a; // массив элементов
    };


    // шаблон функций ввода одного значения
    template <class T>
    int getNum(T &a)
    {
        std::cin >> a;
        if (!std::cin.good())    // обнаружена ошибка ввода или конец файла
            return -1;
        return 1;
    }
```

```
    // функции сравнения
    inline int isgreater(double a, double b)
    {
        return a > b;
    }

    inline int isless(double a, double b)
    {
        return a < b;
    }

    // прототипы функций
    Line* input(int &); // ввод матрицы
    void output(const char *msg, Line a[], int m); // вывод матрицы
    Line *erase(Line *&a, int m); // освобождение занятой памяти
    int minmax(Line a[], int m, double &); // вычисление главного результата
    double minmax(double a[], int m, int(*f)(double, double)); // вычисление min/max элемента
вектора
}
```

Файл Prog1.cpp

– содержит реализацию необходимых функций

```
#include <iostream>
#include "Prog1.h"

namespace Prog1{
    // функция ввода
    Line* input(int &rm)
    {
        const char *pr = ""; // будущее сообщение об ошибке
        Line *lines = nullptr;  // динамический массив строк матрицы
        int m;  // количество строк матрицы
        // сначала вводим количество строк
        do{
            std::cout << pr << std::endl;
            std::cout << "Enter number of lines: --> ";
            pr = "You are wrong; repeat please!";
            if (getNum(m) < 0) // обнаружена ошибка ввода или конец файла
                return nullptr;
        } while (m < 1);

        // выделяем память под массив структур - строк матрицы
        try{
            lines = new Line[m];
        }
        catch (std::bad_alloc &ba)
        {
            std::cout << "------ too many rows in matrix: " << ba.what() << std::endl;
            return nullptr;
        }
        for (int i = 0; i < m; i++){
            // теперь для каждой строки матрицы вводим количество столбцов
            pr = "";
            do{
                std::cout << pr << std::endl;
                std::cout << "Enter number of items in line #" << (i + 1) << " --> ";
                pr = "You are wrong; repeat please!";
                if (getNum(lines[i].n) < 0){ // обнаружена ошибка ввода или конец файла
                    erase(lines, i); // освобождение памяти, занятой ранее введенными строками
                    return nullptr;
                }
            } while (lines[i].n < 1);
```

```
        // и выделяем необходимую память под элементы строки
        try{
            lines[i].a = new double[lines[i].n];
        }
        catch (std::bad_alloc &ba)
        {
            std::cout << "------ too many items in matrix: " << ba.what() << std::endl;
            erase(lines, i);
            return nullptr;
        }

        // теперь можно вводить сами элементы данной строки матрицы
        std::cout << "Enter items for matrix line #" << (i + 1) << ":" << std::endl;
        for (int j = 0; j < lines[i].n; j++){
            if (getNum(lines[i].a[j]) < 0){ // обнаружена ошибка ввода или конец файла
                erase(lines, i + 1); // освобождение памяти, занятой ранее введенными строками
                return nullptr;
            }
        }
    }
    // формируем результат - количество строк в матрице
    rm = m;
    return lines;
}

// функция вывода
void output(const char *msg, Line *a, int m)
{
    int i, j;
    std::cout << msg << ":\n";
    for (i = 0; i < m; ++i){
        for (j = 0; j < a[i].n; ++j)
            std::cout << a[i].a[j] << "  ";
        std::cout << std::endl;
    }
}

// функция освобождения занятой памяти
Line *erase(Line *&lines, int m)
{
    int i;
    for (i = 0; i < m; i++)
        delete[] lines[i].a;
    delete[] lines;
    return nullptr;
}

// функция вычисления главного результата
int minmax(struct Line lines[], int m, double & res)
{
    double *s = nullptr;
    try{
        s = new double[m]; // вектор для получения max элементов в строке - по строкам
    }
    catch (std::bad_alloc &ba)
    {
        std::cout << ba.what() << std::endl;
        return 1;
    }
    int i;
    for (i = 0; i < m; i++)
        s[i] = minmax(lines[i].a, lines[i].n, isgreater);
    res = minmax(s, m, isless);
    delete[] s;
    return 0;
}
```

```
    // функция вычисления min/max элемента вектора
    double minmax(double a[], int m, int(*f)(double, double))
    {
        double res = a[0];
        int i;
        for (i = 0; i < m; ++i)
        if (f(a[i], res) > 0)
            res = a[i];
        return res;
    }
}
```

Файл Prog1main.cpp

– содержит отладочную функцию `main()`

```
#include <iostream>
#include "Prog1.h"

using namespace Prog1;

// основная функция
int main()
{
    Line *arr = nullptr; // исходный массив
    int m; // количество строк в матрице
    double res; // полученный результат

    arr = input(m); // ввод матрицы
    if (!arr){
        std::cout << "incorrect data" << std::endl;
        return 1;
    }
    if (minmax(arr, m, res)){  // вычисление требуемого результата
        std::cout << "Error in allocate memory" << std::endl;
        erase(arr, m);
        return 1;
    }
    output("Sourced matrix", arr, m);
    std::cout << "Result: " << res << std::endl;
    erase(arr, m); // освобождение памяти
    return 0;
}
```