УНИВЕРСИТЕТ ИТМО

Домашнее задание по C++
Операции над матрицами

Постановка задачи. Разработайте утилиту командной строки для сложения и умножения матриц.

Программа должна принимать в качестве аргументов командной строки список файлов с матрицами, разделённых командами-операциями, которые необходимо выполнить над матрицами.

Пример запуска в терминале Unix/Linux:

```
$ ./matrices mat1.txt --add mat2.txt --mult mat3.txt --mult mat4.txt
```

В терминале Windows:

```
C:\> matrices.exe mat1.txt --add mat2.txt --mult mat3.txt --mult mat4.txt
```

Необходимо поддержать две операции:

1) `--add` (начинается с двух знаков минус) — сложение матриц,

2) `--mult` — умножение матриц.

Операции необходимо выполнить в том порядке, в котором они указаны. Например, в указанном выше примере необходимо сначала сложить матрицу из `mat1.txt` с матрицей из `mat2.txt`, затем умножить результат на матрицу из `mat3.txt`, затем умножить результат на матрицу из `mat4.txt`:

$$\Big(((\mathrm{mat}_1 + \mathrm{mat}_2) \cdot \mathrm{mat}_3) \cdot \mathrm{mat}_4\Big).$$

Каждая матрица задаётся в отдельном текстовом файле. На первой строке файла указаны два натуральных числа N и M — соответственно количество строк и столбцов в матрице. Далее следует N строк по M чисел с плавающей точкой. Например:

```
2 5
7.11 3.12 8.13 2.14 0.15
3.21 7.22 3.23 2.24 9.25
```

Результирующую матрицу необходимо вывести на экран (в терминал) в том же формате, в котором заданы входные матрицы. Не допускается выводить какую-либо дополнительную или отладочную информацию.

Обработка ошибок. В случае, если аргументы командной строки не соответствуют указанному выше формату, или, если невозможно произвести операцию над матрицами (например, сложение матриц разного размера), программа должна вывести осмысленное сообщение об ошибке и завершиться с ненулевым кодом возврата (возвращаемым значением из функции `main`).

Необходимо проверять корректность открытия файла (например, приведя файловый поток к логическому типу), и, если файл открыть не удалось, выводить сообщение об ошибке и завершать программу с ненулевым кодом возврата. Если файл с матрицей успешно открылся, можно допустить, что он всегда будет успешно прочитан, и что в нём всегда корректно сохранённая матрица в описанном выше формате.

Дополнительные требования. Данная задача *не требует* использования или знания классов C++, структур или исключений. Достаточно уметь пользоваться двухмерными массивами (выделять, освобождать, и итерироваться по ним), потоками ввода/вывода, строками и функциями.

Выделение и освобождение памяти для хранения матрицы необходимо делать вручную, используя `new[]` и `delete[]` (постарайтесь не допустить дублирования кода для выделения и освобождения памяти для матриц, разумно вынести их отдельные функции). Использование классов STL для хранения или управления памятью для хранения матриц запрещено (в частности запрещено использование `std::vector` и классов умных указателей). Разрешается и рекомендуется использование классов STL для хранения остальных сущностей, помимо матриц (например, для удобной работы со строками можно использовать `std::string`). Разрешается, но не требуется, написание собственного класса матрицы.

Параметры командной строки. Для того, чтобы обработать параметры командной строки, нужно использовать следующий синтаксис функции `main`:

```
int main(int argc, char ** argv)
```

`argc` — количество параметров командной строки, а `argv` — массив строк в стиле C. Строка с номером `0` в этом массиве — это имя самой команды, строки с номерами от `1` до `(argc - 1)` — это параметры командной строки.

Например, чтобы вывести все параметры командной строки можно написать следующий код.

```
#include <iostream>

int main(int argc, char ** argv)
{
    for (int i = 1; i < argc; ++i)
    {
        std::cout << i << ": " << argv[i] << std::endl;
    }
    return 0;
}
```

Формат сдачи. В репозитории должны быть следующие файлы: `main.cpp`, `matrices.hpp`, `matrices.cpp` и файлы системы сборки — `Makefile` или `CMakeLists.txt`, если вы используете CMake.

Часто возникающие ошибки.

1. Используйте прилагающийся к заданию smoke test: в нём приведены различные варианты запуска решения с ожидаемыми результатами.

2. Для ввода/вывода необходимо использовать потоковые интерфейсы C++ (`std::cin`, `std::cout`, `std::ifstream`). Запрещено использование средств C (`fprintf()`, `fopen()`).

3. Ошибки необходимо выводить в `std::cerr`.

4. Используйте разные экземпляры класса `std::ifstream` для чтения разных файлов. Используйте то, что `std::ifstream` автоматически закроет файл, когда экземпляр класса будет уничтожен, не вызывайте `std::ifstream::close()` когда это не требуется.

5. Нет необходимости заранее проверять то, что файлы существуют: после проверки, но перед повторным открытием файл могут удалить из файловой системы. Проверяйте удалось ли открыть файл после попытки его открыть, используя приведение к `bool`.

6. Убедитесь, что директория с решением и лежащие в файлы называются в точности так, как требуется в условии.

7. В заголовочных файлах необходимо включать только минимально необходимые заголовочные файлы. Например, `<iostream>` скорее всего не нужен в `matrices.hpp`, его стоит включить в cpp файле.

8. Передавайте неизменяемые экземпляры классов по константной ссылке. Для передачи неизменяемых строк рекомендуется использовать `std::string_view`.

9. Если метод класса не изменяет состояние класса, сделайте его константным.

10. Если вы передаёте объект по указателю или ссылке и не будете его модифицировать, передавайте его явно константным.

11. Используйте передачу по ссылке вместо передачи по указателю. Указатель может быть нулевым; указатель заставляет задуматься о том, должна ли функция освобождать память, на которую указывает указатель.

12. Убедитесь, что при возникновении ошибок вы корректно освобождаете всю выделенную память.

13. Избегайте дублирования кода: вынесите код выделения и освобождения памяти в отдельные функции.

14. Не стоит использовать `using namespace std;` в заголовочных файлах, т.к. не все пользователи заголовочного файла могут хотеть вносить содержимое `namespace std` в корневую область видимости.

15. Приватные члены класса следует именовать с подчеркиванием на конце.

16. Не забывайте про необходимость вывести размер матрицы при печати матрицы в консоль.

17. Используйте корректную терминологию, её можно посмотреть, например, в Википедии[1]. В английском языке «матрица» — «matrix», «матрицы» — «matrices». Индексация в матрицах традиционно сначала по строкам («row»), затем по столбцам («column»).

18. Если вы используете исключения:

    (a) В качестве типов исключений либо используйте исключения из стандартной библиотеки (`std::runtime_error, std::invalid_argument`), либо создайте свой класс исключения, унаследовав его от `std::exception` или `std::runtime_error`.

    (b) Передавайте текст ошибки в исключении (см. `std::exception::what()`) и выводите его там, где ловите исключение.

    (c) Ловите исключения по константной ссылке, если не собираетесь их модифицировать.

19. Имена, начинающиеся с подчеркивания и большой буквы, и имена, содержащие два последовательных подчеркивания, зарезервированы для реализации компилятора и стандартной библиотеки. Такие имена нельзя использовать в стражах включения.

[1] `https://en.wikipedia.org/wiki/Matrix_(mathematics)`

20. Не стоит передавать скалярные типы (`size_t, int, double` и т.п.) по ссылке.

21. Нельзя использовать `exit()` для обработки ошибок. Функция `exit()` никогда не возвращается и выделенные ранее ресурсы никогда корректно не освобождаются (например, явно не освобождается память выделенная внутри `std::string`).

22. Старайтесь выделять замкнутую функциональность в отдельные функции. Например, печать матрицы на экран стоит оформить в виде отдельной функции, а не вложенных циклов в середине `main()`.

23. Используйте средства C++ для работы со строками. Не используйте функции C для работы со строками. Вместо `strcmp` храните строки в `std::string` (или оборачивайте в `std::string_view`) и сравнивайте с помощью `==`.

24. Используйте value initialization[2] для выделения массива, заполненного нулями, вместо явного заполнения нулями в цикле.

25. При объявлении функции или метода указывайте имена аргументов, это делает интерфейс существенно более понятным.

26. Используйте тип `size_t` для размеров и индексов, диапазона значений типа `int` может не хватить.

27. Используйте `nullptr` вместо 0 или `NULL` для нулевых указателей.

28. Проверяйте четность числа, смотря на остаток от его деления на два (`n % 2 == 0`), а не на младший бит.

[2] `http://en.cppreference.com/w/cpp/language/value_initialization`