Пояснительная записка к 1 домашнему заданию
по предмету
«Архитектура вычислительных систем»
студента Дерели Серкана.
Вариант 313.

Описание задания и выбранные параметры.

Описание задания.

По полученному варианту по формуле из описания задания был вычислен вариант задачи 1 = **5**, а также вариант задачи 2 — **23.**

1 часть задачи заключалась в написании обобщённой квадратной матрицы и 3 её конкретных реализаций: обычной квадратной матрицы с записью элементов в двумерный массив, диагональной матрицы с элементами в одномерном массиве, а также нижнетреугольной матрицы с элементами в одномерном массиве. У каждой конкретной реализации матрицы должен быть метод расчёта среднего арифметического элементов.

2 часть задачи заключалась в перемещении в начало контейнера тех матриц, чьё среднее арифметическое >= среднего арифметического средних арифметических значение элементов всех обобщённых квадратных матрицы в программе. Далее приведён кусок из условия задачи:

1 часть задания:

Обобщённый артефакт, используемый в задании.	Базовые альтернативы (уникальные параметры, задающие отличительные признаки альтернатив).	Общие для всех альтернатив переменные.	Общие для всех альтернатив функции
5. Квадратные матрицы с действительными числами	1. Обычный двумерный массив 2. Диагональная (на основе одномерного массива) 3. Нижняя треугольная матрица (одномерный массив с формулой пересчета)	Размерность – целое число	

2 часть задания:

23. Переместить в начало контейнера те элементы, для которых значение, полученное с использованием функции, общей для всех альтернатив, больше чем среднее арифметическое для всех элементов контейнера, полученное с использованием этой же функции. Остальные элементы сдвинуть к началу без изменения их порядка.

Выбранные случайные параметры, не очерченные в условии задачи.

В условии задачи чётко прописаны форматы тестовых данных: **файлы с тестовыми данными** могут содержать **не более 20 уникальных элементов** контейнера. Больше 20 элементов генерируется внутри программы случайным образом.

Однако для матриц не заданы граничные значения элементов и размера матрицы. Поэтому их выбор был сделан самостоятельно: **размер матрицы** задаётся **от 2 до 30** элементов в матрице включительно (тут имеется в виду параметр n матрицы. У диагональной матрицы n*n элементов будет n ненулевых элементов, у нижнетреугольной n*(n-1)/2 (по формуле суммы арифметической прогрессии) элементов, у обычной квадратной будет n^2 элементов).

Элементы же **матрицы** генерируются от **-100 до 100**. При чтении с файлов ограничения на читаемое число элементов нет.

Какие входные данные предполагаются программой верными.

Запуск программы предполагается из командной консоли на Линуксе. При этом в командной строке в функцию main программы передаются параметры работы программы — режим заполнения контейнера(с файла, случайно), путь к файлу с элементами контейнера или длина случайно генерируемого контейнера в зависимости от первого

параметра, а также путь файла с выходными данными. Длину контейнера при случайной генерации контейнера можно задать от 1 до 10 000 включительно(согласно условию). Чтобы входной файл с элементами контейнера правильно обрабатывался программой, элементы контейнера должны быть представлены в файле следующий образом:
Для диагональной матрицы:
<1> <Пробел><n>
<a1> <Пробел><a2><Пробел> <a3>...<an>
Для квадратной матрицы:
<2> <Пробел><n>
<a11> <Пробел><a12><Пробел> <a13>...<a1n>
<a21> <Пробел><a22><Пробел> <a23>...<a2n>
….....
<an1> <Пробел><na2><Пробел> <na3>...<ann>
Для диагональной матрицы:
<3> <Пробел><n>
<a11> <Пробел><a12><Пробел> <a13>...<a1(n*(n-1)/2)>

В конце файла с исходными данными не должно быть пустой строки.

Сами элементы (<aij>) должны быть такими, чтобы их смогла считать функция fscanf(file, “%f”, aij).

Какие параметры проверяются на верность при работе программы, а какие не проверяются.

В программе **проверяется количество введённых** из командной консоли в программу **параметров**. Их должно быть 3(в указанном порядке) : Формат заполнения(-f, -n) , Путь к файлу с исходными данными или кол-во генерируемых элементов. Если параметров не хватает, то программа выведет в консоль сообщение с ошибкой и завершит свою работу.

Также **проверяется** на верность введённое **кол-во генерируемых элементов** в контейнере. Если будет число не в диапазоне от 1 до 10 000, то в консоль выведется ошибка,

работа программы завершится.

Проверяется существование входного и директории выходного файла, если любого не существует, то выведется сообщение об ошибке, затем программа завершится.

Входной файл проверяется на пустоту, то есть пустой файл обрабатываться программой не будет. Если входной файл пустой, то программа выведет в консоль сообщение об ошибке и завершит работу программы.

Проверяется верность содержащегося в файле **элемента матрицы** функцией fscanf стандартной библиотеки C. Если функция не может прочитать число, то выводится сообщение об ошибке и программа завершает свою работу.

Проверяется описание матрицы во входном файле: если тип матрицы или параметр n заданы как вещественное, а не целое число, или параметр типа матрицы не в диапазоне от 1 до 3, то в консоль выводится сообщение об ошибке, а программа завершит свою работу.

Не проверяется соответствие строки с описанием матрицы **с элементами матрицы на следующей строке**, т. е. Если в входной файле будет содержаться:

1 3

1 1

То на это программа проверять входной файл не будет, возникнет ошибка исполнения программы, так как диагональная матрица будет ожидать 3 элемента, а введено 2. Такие данные в файле считаются изначально верными.

Среда разработки и инструменты разработки.

Для разработки программы использовалась среда Clion с интегрированным WSL(Ubuntu), язык c++ в стиле c. Операционная система: Windows с установленным WSL(Ubuntu).

Структурная схема изучаемой архитектуры ВС с размещенной на ней разработанной программы.

Таблица типов

Тип	Память(байт)
int	4
float	4
double	8
float *	8
float **	8
FILE *	8
Matrix *	8
Diagonal *	8
Triangular *	8
Square *	8
class Matrix vtptr: void* n:int	12 8[0] 4[8]
Статические поля(не хранятся в классе): static Random rnd100 static Random rnd30	
Методы(не хранятся в классе): public: Matrix() virtual ~Matrix() virtual void In(FILE *, bool&) virtual void InRnd() virtual void Out(FILE *) virtual void Print() virtual double Average() virtual void Delete()	

static Matrix* StaticIn(FILE*, bool&)
static Matrix* StaticRnd()

class Diagonal : Matrix 16
vtptr: void* 8[0]
elements:float * 8[8]

Статические поля(не хранятся в классе):

-

Методы(не хранятся в классе):

public:

Diagonal(int)
~Diagonal() override
void In(FILE *, bool&) override
void InRnd() override
void Out(FILE *) override
void Print() override
double Average() override
void Delete() override

class Triangular : Matrix 16
vtptr: void* 8[0]
elements:float * 8[8]

Статические поля(не хранятся в классе):

-

Методы(не хранятся в классе):

public:

Triangular(int)
~Triangular() override
void In(FILE *, bool&) override
void InRnd() override
void Out(FILE *) override
void Print() override
double Average() override
void Delete() override

class Square : Matrix	16
vtptr: void*	8[0]
elements:float **	8[8]

Статические поля(не хранятся в классе):

-

Методы(не хранятся в классе):

public:

Square(int)

~Square() override

void In(FILE *, bool&) override

void InRnd() override

void Out(FILE *) override

void Print() override

double Average() override

void Delete() override

class Container	80004
len:int	4[0]
storage:Matrix *[]	8 * 10000 = 80000[4]

Статические поля(не хранятся в классе):

-

Методы(не хранятся в классе):

public:

Container()

~Container()

void In(FILE* file, bool& success)

void InRnd(int size)

void Out(FILE* file)

void Print()

double Average()

void Delete()

void Rearrange()

class Random min:int max:int	8 4[0] 4[4]
Статические поля(не хранятся в классе): -	
Методы(не хранятся в классе): public: Random(int, int) int Get() int GetFloat()	

Глобальная память

Переменная	Память
static Random rnd100 static Random rnd3 static random rnd30	8 8 8
Локальные методы	
int Get(Random*) int GetFloat(Random*) void In(Container *, FILE* file, bool& success) void InRnd(Container *, int size) void Out(Container *, FILE* file) void Print(Container *) double Average(Container *) void Delete(Container *) void Rearrange(Container *)	
Статические методы	
static Matrix* StaticIn(FILE* file, bool& success); static Matrix* StaticRnd();	

Виртуальные(+override) методы
~Matrix(Matrix *) void In(Matrix *, FILE * file, bool& success) void InRnd(Matrix *) void Out(Matrix *, FILE * file) void Print(Matrix *) double Average(Matrix *) void Delete(Matrix *) ~Diagonal(Diagonal *) void In(Diagonal *, FILE* file, bool& success) void InRnd(Diagonal *) void Out(Diagonal *, FILE* file) void Print(Diagonal *) double Average(Diagonal *) void Delete(Diagonal *) ~Square(Square *) void In(Square *, FILE* file, bool& success) void InRnd(Square *) void Out(Square *, FILE * file) void Print(Square *) double Average(Square *) void Delete(Square *) ~Triangular(Triangular *) void In(Triangular *, FILE* file, bool& success) void InRnd(Triangular *) void Out(Triangular *, FILE * file) void Print(Triangular *) double Average(Triangular *) void Delete(Triangular *)

Таблицы виртуальных методов(тут хранятся только указатели на методы, которые указаны выше).
Matrix(хранятся указатели на методы, указанные ниже, а не сами методы)
~Matrix(Matrix *) void In(Matrix *, FILE * file, bool& success) void InRnd(Matrix *) void Out(Matrix *, FILE * file) void Print(Matrix *) double Average(Matrix *) void Delete(Matrix *)
Triangular (хранятся указатели на методы, указанные ниже, а не сами методы)
~Triangular(Triangular *) void In(Triangular *, FILE* file, bool& success) void InRnd(Triangular *) void Out(Triangular *, FILE * file) void Print(Triangular *) double Average(Triangular *) void Delete(Triangular *)
Diagonal (хранятся указатели на методы, указанные ниже, а не сами методы).
~Diagonal(Diagonal *) void In(Diagonal *, FILE* file, bool& success) void InRnd(Diagonal *) void Out(Diagonal *, FILE* file) void Print(Diagonal *) double Average(Diagonal *) void Delete(Diagonal *)

Square (хранятся указатели на методы, указанные ниже, а не сами методы).
~Square(Square *) void In(Square *, FILE* file, bool& success) void InRnd(Square *) void Out(Square *, FILE * file) void Print(Square *) double Average(Square *) void Delete(Square *)

Память программы **Куча**

main(int argc, char* argv[])			
argc:int	4[0]	“task”	0
argv:char**	8[4]	“-f”	1
c:Container	80004[12]	…	…
f:FILE *	8[80016]	Diagonal	
size:int	4[80024]	(	
f:FILE *	8[80028]	...	k
		)	…
		…	
		FILE	...
		(	
		...	
		)	n
		…	
		FILE	...
		(	
		...	
		)	s

Собираемые метрики программы.

Ниже представлены основные метрики программы:

Название файла	Размер файла .h	Размер файла .cpp
square	510 байт	2 984 байт
matrix	654 байт	1 544 байт
diagonal	521 байт	2 473 байт
triangular	531 байт	3 110 байт
container	970 байт	3 561 байт
rnd	657 байт	-
main	-	6 577 байт

Заголовочных файлов(.h) = 6. Модулей реализации(.cpp) = 6.
Общий размер файлов: 24092 байт.
Размер исполняемого файла: 38 264 байт.

Тесты

Номер	Название и размер входного файла	Кол-во генерируемых элементов.	Название и размер выходного файла.	Сообщение об ошибке	Time old prog total	Time new prog total	Time new prog user	Time new prog sys
1	-	100	output100 397 736 байт	-	78ms	308 ms	47 ms	63 ms
2	-	1000	output1000 3 357 216 байт	-	516 ms	3299 ms	625 ms	938 ms
3	-	10000	output10000 34 840 348	-	5.563 s	27.2 s	6 s	6.281 s

			байт					
4	input0c 19074 байт	-	output0c 44276 байт	-	78 ms	61 ms	16 ms	16 ms
5	input1c 3961 байт	-	output1c 9690 байт	-	26 ms	29 ms	0 ms	16 ms
6	input2c 3734 байт	-	output2c 8818 байт	-	24 ms	24 ms	0 ms	31 ms
7	input3c 3346 байт	-	output3c 8084 байт	-	24 ms	24 ms	0 ms	16 ms
8	input4c 4655 байт	-	output4c 11720 байт	-	44 ms	34 ms	0 ms	0 ms
9	input5c 3110 байт	-	output5c 7962 байт	-	54 ms	30 ms	0 ms	0 ms
10	input6c 5156 байт	-	output6c 13300 байт	-	143 ms	67 ms	47 ms	16 ms
11	input7c 11186 байт	-	output7c 26562 байт	-	85 ms	50 ms	0 ms	41 ms
12	input8c 7629 байт	-	output8c 18660 байт	-	77 ms	39 ms	0 ms	31 ms
13	input9c 1480 байт	-	output9c 3962 байт	-	53 ms	83 ms	0 ms	16 ms

14	input10c 4570 байт	-	output10c 10886 байт	-	56 ms	32 ms	16 ms	16 ms
15	input11c 13885 байт	-	output11c 32752 байт	-	106 ms	82 ms	0 ms	16 ms
16	Input12w 0 байт	-	-	File is empty	14 ms	16 ms	0 ms	0 ms
17	input13w 16925 байт	-	-	Couldn't read a square matrix.	48 ms	53 ms	0 ms	31 ms
18	input14w 7754 байт	-	-	Couldn't read a square matrix.	27 ms	54 ms	0 ms	16 ms

Сравнение с прошлым дз.

В таблице с тестами представлены результаты тестирования программы 1 дз с тестами из 1 дз, а также программы 2 дз с тестами из 1 дз. При этом время замерялось командой time командной строки(перед вызовом функции пишу time). Команда выдаёт 3 времени: real, system, user. В 1 дз собиралось только real. Сейчас собираются все 3. Анализ времени выполнения на тестовых данных показывает, что на маленьких данных из файлов разница времени выполнения

особенно не заметно. Разница тут можно списать на статистическую погрешность, ведь на компьютере открыты и другие приложения, а это может незначительно сказываться на результатах. Однако на большом количестве данных(первые 3 теста) сразу видно, что ООП оказывается в разы медленнее функционального подхода. Размеры файлов .h в программе с ООП подходом оказались несколько больше, однако меньше стали весить файлы .cpp. Т.о. суммарно весит программа примерно так же, как и в 1 дз, однако разительно больше весит бинарник: в 1 дз — 23760 байт, во 2 — 38264 байта.