Todo list

Программирование

Жуйков Артем

19 декабря 2015 г.

Глава 1

Основные конструкции языка

1.1 Задание 1

1.1.1 Задание

Найти длину отрезка, соединяющего точки на плоскости с координатами (x_1, y_1) и (x_1, y_1).

1.1.2 Теоретические сведения

Для вычисления длины отрезка по заданным координатам концов воспользуемся формулой:

$$l = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \tag{1.1}$$

и используем заголовочный файл math.h.

1.1.3 Проектирование

Введем функцию `void length_of_segment_ui()`, которая будет отвечать за взаимодействие с пользователем. Из консоли сначала считаем координаты (x_1, y_1) первой точки и запишем их в соответствующие переменные `x_1` и `y_1`. Координаты второй точки запишем в переменные `x_2`, `y_2` соответственно. Выделим функцию, которая получает четыре параметра - координаты точек и возвращает расстояние между ними: `float length_of_segment(int a, int b, int c, int, d)`. Будем выводить его в консоль с точностью до двух знаков после запятой.

1.1.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2
Операционная система: Debian GNU/Linux 8.1
Для проверки работы программы использовалось ручное тестирование, а так же автоматическое. Результаты автоматического тестирования .....

1.1.5 Тестовый план и результаты тестирования

Описание хода ручного тестирования:

1. Пусть координаты первой точки равны (0, 0), а координаты второй точки - (3, 4). Тогда длина отрезка, соединяющего эти точки будет равна:

$$l = \sqrt{(3-0)^2 + (4-0)^2} = \sqrt{3^2 + 4^2} = \sqrt{25} = 5 \qquad (1.2)$$

   Для таких исходных данных программа вывела 5.00, что соответствует ожидаемому результату.

2. Пусть координаты первой точки – (-1, -2), а координаты второй точки – (30, 52). Вычислим расстояние между точками в этом случае:

$$l = \sqrt{(30-(-1))^2 + (52-(-2))^2} = \sqrt{31^2 + 54^2} = \sqrt{3877} \approx 62,26656... \qquad (1.3)$$

   Программа вывела число 60.27, что с учетом введенного округления соответствует ожидаемому результату.

3. Пусть координаты первой точки (16, 16), координаты второй точки (-4, -10). Тогда расстояние между точками равно:

$$l = sqrt(16-(-4))^2 + (16-(-10))^2 = \sqrt{1076} \approx 32,80244... \qquad (1.4)$$

   Программа вывела число 32,80, что с учетом округления соответствует ожидаемому результату.

Код проверялся программой cppcheck.c. Утилита вывела следующие ошибки:
(portability) scanf without field width limits can crash with huge input data on some versions of libc.
Это значит, что в форматной строке нужно указать границу вводимой

переменной. Если пользователь введет число, большее максимально допустимого, например, в типе int, это приведет к ошибке. Чтобы такого не произошло, форматная строка должна иметь следующий вид: "%100d".

1.1.6 Выводы

В ходе работы над задачей автор получил новые знания о языке C: структуры, консольный ввод-вывод. Научился пользоваться функциональностью дебага в среде QtCreator. Написанная программа была протестирована. Результаты всех тестов соответствуют ожидаемым результатам, следовательно, можно сделать вывод, что программа работает верно.

1.2 Задание 2

1.2.1 Задание

Найти корни биквадратного уравнения: $y = ax^4 + bx^2 + c$.

1.2.2 Теоретические сведения

Обозначим исходное уравнение (1).
Чтобы найти корни биквадратного уравнения, можно решить его, как квадратное, относительно x^2, т.е

$$a(x^2)^2 + bx^2 + c = 0, \ (2)$$

При этом корнем уравнения (2) будем называть x^2. Уравнение (1) будет иметь корни, если хотя бы один корень уравнения (2) неотрицателен.
Рассмотрим уравнение (2). Уравнение имеет корни (или не имеет их) в зависимости от дискриминанта.
Если дискриминант меньше нуля, то уравнение (2), а следовательно и уравнение (1), не будет иметь решений. В этом случае сразу известен ответ задачи: исходное уравнение не имеет решений.
Если дискриминант равен нулю, то корни уравнения (2) не зависят от него. ($x^2 = \frac{-b \pm \sqrt{D}}{2a}$, где $D = 0$). При этом биквадратное уравнение будет иметь два корня, если $x^2 > 0$; один корень (равный нулю), если $x^2 = 0$ или не иметь корней, если $x^2 < 0$.
Если дискриминант больше нуля, то уравнение (2) имеет два корня. Возможны случаи:

если оба корня отрицательны, то уравнение (1) не имеет решений;
если один корень отрицательный, а другой равен нулю, то исходное уравнение имеет единственный корень, равный нулю;
если один корень равен нулю, а второй положительный, то уравнение (1) имеет три корня, один из которых - нуль;
если оба корня положительны, то уравнение (1) имеет четыре корня.

1.2.3 Проектирование

Пусть a, b, c - коэффициенты биквадратного уравнения.
Создадим структуру Solutions_of_equation, в которую в последующем запишем корни (максимум 4). Помимо значения корней будем хранить в ней и их существование. Для этого у каждого корня будет соответствующее поле равное еденице, если корень существует и нулю в противном случае. Если корня не существует, то он не выведется в ответе. Также корень не будет напечатан, если он равен корню, выведенному ранее.

1. Функция void solution_of_equation_ui() отвечает за взаимодействие с пользователем. Она считывает коэффициенты уравнения из консоли и вызывает функцию int solve_the_equation(int a, int b, int c, struct Solutions_of_equations* f).
   Введем функцию, вычисляющую дискриминант уравнения: int sign_discriminant(int a, int b, int c). Функция вернет нуль, если дискриминант отрицательный и единицу, если дискриминант больше либо равен нулю.

2. Функция int solve_the_equation(int a, int b, int c, struct Solutions_of_equations* f) (при $a \neq 0$) будет анализировать значение, которое вернула предыдущая функция, и, в зависимости от него, вызывать соответствующую функцию. Если дискриминант меньше нуля функция возвращает нуль. В противном случае, функция вызовет функцию void calculating_of_solutions(int, int, int, struct Solutions_of_equation*) и вернет единицу.

3. Функция void calculating_of_solutions(int, int, int, struct Solutions_of_equation*) вычисляет корни уравения, записывает их в структуру, а также обозначает их существование путем присваивания единицы в соответствующее поле структуры. В конце работы функция вызывает функцию void analysis_of_solutions(struct Solutions_of_equation

*).

4. Функция void analysis_of_solutions(struct Solutions_of_equation *) анализирует полученные в структуре корни: если какие-то два корня равны и при этом существуют оба, то функция присвоит нуль в поле структуры, отвечающее за существование корня. Таким образом корень не будет выводиться дважды.

После работы всех функций, мы вернемся в функцию solution_of_equation_ui. Она выведет решения уравнения, если функция solve_the_equation вернула 1 или выведет на экран "Не имеет решений", если последняя функция вернула 0.
Для корней уравнения используем тип float и будем выводить их с точностью до двух знаков после запятой.

1.2.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2 Операционная система: Debian GNU/Linux 8.1
При решении задачи использовалось автоматическое тестирование.
Программа проверялась утилитой cppcheck: (portability) scanf without field width limits can crash with huge input data on some versions of libc.
Ошибка аналогична ошибке в предыдущей задаче, нужно указать максимально допустимое значение переменной в форматной строке scanf.

1.2.5 Тестовый план и результаты тестирования

Результаты автоматического тестирования тестирования:

Входные данные	Результат работы программы	Ожидаемый результат
1 -3 2	1.41; -1,41; 1.00; -1.00	1.41; -1.41; 1; -1
2 -8 0	2.00; -2.00; 0	0; 2; -2
1 0 0	0	0

Таким образом, программа успешно прошла все представленные тесты.

1.2.6 Выводы

Решая эту задачу, я научился разделять одну сложную программу на несколько функций. Одни из них отвечают за математику, другие - за взаимодействие с пользователем.

Глава 2

Циклы

2.1 Задание 1

2.1.1 Задание

Вывести на экран таблицу пересчета миль в километры и обратно до заданного расстояния в километрах, по возрастанию расстояний, как указано в примере (1 миля = 1.609 км). Пример для 5 километров:
мили км
0.62 1.00
1.00 1.61
1.24 2.00
1.86 3.00
2.00 3.22
2.49 4.00
3.00 4.83
3.11 5.00

2.1.2 Теоретические сведения

Для решения задачи воспользуемся циклом с предусловием. Будем выводить значения миль и километров, если хотя бы одно из них целое число. Повторяем это до тех пор, пока километры не станут больше необходимого числа.

2.1.3 Проектирование

За взаимодействие с пользователем будет отвечать функция void km_to_miles_ui(). Она же будет проверять, является ли натуральным введенное число километров. В случае, если введенное число не удовлетворяет данному условию, программа прекращает работу и выводит сообщение об ошибке. Если введенное число километров является натуральным числом, запускается функция void km_to_miles(int km), которая считает и выводит необходимую таблицу до заданного числа километров. Передадим функции это число, как параметр.
Рассмотрим функцию void km_to_miles(int). Начальные значения миль и километров равны 0.62 и 1.00 соответственно.
В цикле будем выполнять следующие действия:
Выведем строчку со значениями миль и километров, при этом число километров всегда будет целое. (Обозначим эту строку "*"). Пока число километров не станет больше нужного числа, будем увеличивать его на единицу. Затем пересчитаем соответствующее число миль по формуле: мили = $\frac{km}{1.609}$, где km - число километров.
Чтобы в таблице были не только целые значения километров, но и целые значения миль, в начале каждой итерации цикла будем проверять, не стало ли текущее значение миль больше целого. Для этого будем сравнивать его с переменной, содержащей следующее целое число миль, которое необходимо вывести с соответствующим значением числа километров. (Пусть это переменная m). Изначально ее значение равно единице, т.к это следующее целое число миль для 0.62.
Если число миль "перескочило"через m, то перед тем, как вывести строку (*), выведем строку для целого числа миль, которое находится в m. Тогда число километров для этой строки будет равно: километры = 1.609 * мили = 1.609 * m. Тут же увеличим m на единицу.
После проверки условия, выведем строку "* увеличим километры на единицу, пересчитаем число миль для нового значения километров и перейдем к следующей итерации цикла.
Будем выводить таблицу в консоль. Числа - с точностью до двух знаков после запятой. Так мы выведем таблицу по образцу, представленном в условии задания.

2.1.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2

Операционная система: Debian GNU/Linux 8.1
При решении задачи использовалось ручное тестирование.

2.1.5 Тестовый план и результаты тестирования

Передадим программе число 11. Программа выводит:
Мили Километры
0.62 1.00
1.00 1.61
1.24 2.00
1.86 3.00
2.00 3.22
2.49 4.00
3.00 4.83
3.11 5.00
3.73 6.00
4.00 6.44
4.35 7.00
4.97 8.00
5.00 8.04
5.59 9.00
6.00 9.65
6.22 10.00
6.84 11.00
Визуально таблица соответствует примеру из условия. Заметим, что до пяти километров программа дает верный ответ. Посчитаем, например, значение миль для 8.00 километров. Мили $= \frac{8.00}{1.609} = 4.9720...$ С учетом округления, то же число напечатано в таблице. Для 11.00 километров число миль равно: $\frac{11.00}{1.609} = 6{,}83654...$, что соответствует табличному.
Программа проверялась утилитой cppcheck. Была найдена единственная ошибка, аналогичная ошибкам в предыдущих заданиях.

2.1.6 Выводы

При написании программы удалось избежать перебора значений километров с шагом 0.001, вычисления соответствующего значения миль и вывода строки, если одно из них целое. Программа корректно выводит на экран таблицу пересчета миль в километры и обратно до заданного расстояния в километрах, по возрастанию расстояний.

Глава 3

Матрицы

3.1 Задание 1

3.1.1 Задание

Каждый элемент a_{ij} матрицы A(m, n) заменить суммой элементов подматрицы A'(i, j), расположенной в левом верхнем углу матрицы A.

3.1.2 Теоретические сведения

Пусть размер исходной матрицы равен m × n. Обозначим исходную матрицу A, а матрицу, которую нужно получить B.
Для обработки двумерного массива воспользуемся динамическим выделением памяти. Функция malloc(N) из библиотеки stdlib.h выделяет ячейки памяти для N байт и возвращает адрес на первую из них. Для начала создадим массив длинной n. Будем хранить в нем n указателей, каждый из которых содержит адрес первой ячейки n-ого массива из m элементов. Таким образом, получим n массивов длинной m, где n-ый массив - n-ый столбец в матрице.
Для нахождения элемента B_{ij}, будем динамично его вычислять. Он равен ans + C, где ans - ответ для предыдущего элемента B_{ij-1}, а C - сумма элементов столбца j до элемента B_{ij}, включая его самого. Для нулевой строки $C = 0$, а для нулевого столбца $ans = 0$.

3.1.3 Проектирование

Используем файловый ввод-вывод. Из файла input.txt считываем исходную матрицу, при этом в первую строку файла поместим числа m и n - число строк и число столбцов исходной матрицы соответственно. Ответ

задачи выводим в файл output.txt. Функция void change_matrix() будет считывать матрицу, выделять память, вычислять ответ и писать новую матрицу в нужный файл.
После выделения памяти "побежим"по исходному двумерному массиву. Для вычисления каждого элемента B_{ij} будем хранить ans и C (см. "Теоретические сведения"). Переменная pre_ans содержит ответ для предыдущего элемента B_{ij-1}. Чтобы найти C введем вспомогательную функцию int calc_column(int line, int col, int **arr). Функция получает три параметра: строку и столбец текущего элемента, а также указатель на двумерный массив. Результатом ее работы является число C. После вычисления элемента B_{ij} напишем его в файл output.txt, затем обновим pre_ans (оно становится равно B_{ij}) и перейдем к следующему элементу. После вычисления всех элементов не забудем закрыть файлы input.txt и output.txt и освободить память функцией free() из библиотеки stdlib.h.
Результат работы программы - двумерный массив, записанный в файл output.txt.

3.1.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2
Операционная система: Debian GNU/Linux 8.1
При решении задачи использовалось ручное тестирование.

3.1.5 Тестовый план и результаты тестирования

1. Файл input.txt имеет следующий вид:
   5 5
   1 2 3 4 5
   1 3 7 6 3
   2 8 1 2 4
   6 1 1 3 4
   5 8 6 1 2
   Результат работы программы в файле output.txt:
   1 3 6 10 15
   2 7 17 27 35
   4 17 28 40 52
   10 24 36 51 67
   15 37 55 71 89
   Рассмотрим элемент A_{23}. В новой матрице он должен быть равен:

$1 + 2 + 3 + 1 + 3 + 7 = 17$, что мы и видим в результате работы программы. Вычислим новое значение элемента A_{34}: $1 + 2 + 3 + 4 + 1 + 3 + 7 + 6 + 2 + 8 + 1 + 2 = 40$. В получившейся матрице элемент [3, 4] равен 40.

2. Пусть теперь в файле input.txt записано следующее:
   4 3
   1 1 1
   1 1 1
   1 1 1
   1 1 1
   После работы программы файл output.txt выглядит следующим образом:
   1 2 3
   2 4 6
   3 6 9
   4 8 12
   Вычислим новое значение элемента A_{43}: $12 * 1 = 12$, что и находится на месте [4, 3] в получившейся матрице.

Утилита cppcheck нашла следующие ошибки:

1. (portability) scanf without field width limits can crash with huge input data on some versions of libc.
   Эту ошибку несложно исправить указанием границы вводимой переменной.

2. (style) The scope of the variable 'pre_ans' can be reduced.
   Ошибка значит, что для переменной pre_ans можно использовать short int вместо int.

3.1.6 Выводы

При решении задачи автор научился использовать файловый ввод-вывод на языке C, выделять динамически память для двумерного массива и работать с ним. Как показало тестирование, написанная программа верно находит ответ задачи.

Глава 4

Строки

4.1 Задание 1

4.1.1 Задание

В текстовом файле представлены в виде таблтцы результаты соревнований по прыжкам в длину. В каждой строке файла записаны фамилия с инициалами спортсмена и через пробелы - его результаты в трех попытках. Найти трех призеров соревнования.
Иванов И.И. 720 732 735
Петров П.П. 722 727 730
Сидоров С.С. 721 733 738

4.1.2 Теоретические сведения

Результатом спортсмена является максимум из трех попыток. Для решения задачи необходимо определить результат каждого участника и выбрать трех человек с наилучшим.
Для чтения даных из файла подключим библиотеку stdlib.h. Для использования строковой функции strcpy(char[], char[]) подключим библиотеку string.h.

4.1.3 Проектирование

Организуем структуру Competitors. В ней будет два поля: имя участника (фамилия и инициалы) - name и его лучший результат из трех попыток - res. Создадим переменные first, second, third и participant этого типа. Присвоим полю res первых трех переменных -1. Получать данные

будем из файла "competitions.txt". Сведения о каждом участнике - это строка в файле.

Функция void results_of_competitions()- основная функция. В ней будут объявляться переменные и структура, а также считываться данные и выводиться ответ. Пока не достигнут конец файла, будем получать имя участника и три его результата. Делать это будем с помощью функций int find_name(char [40], struct Competitors*) и void find_results(int, char [40], int [3]).
Функция int find_name(char [40], struct Competitors*) ищет в строке файла имя и инициалы участника и записывает их в . Имя спортсмена находится в строке до второго пробела. Учитывая этот факт, функция посимвольно записывает имя и инициалы участника в строку participant.name. Как только достигается второй пробел, функция возвращает номер символа, который используется в дальнейшем в функции void find_results(int, char [40], int [3]).
Функция void find_results(int, char [40], int [3]) получает номер символа, который вернула предыдущая функция. Именно с этого символа функция начинает свою работу. Она считывает символы до пробела и записывает их в строку res. Затем, с помощью функции atoi(res), строка res преобразуется в число и записывается в массив результатов данного спортсмена. Функция обрабатывает три числа, разделенных пробелами, т.е. пока не достигнут символ перевода строки.
После разбора полученной из файла строки находим максимум в получившемся массиве результатов спортсмена и запишем его в participant.res. Таким образом в переменной participant записаны все сведения об участнике.
Теперь будем поочередно сравнивать получившуюся переменную participant с переменными first, second и third. Реализовывать это будет функция int compare_structs(struct Competitors*, struct Competitors*) Если participant.res больше first.res, то с помощью функции void assigment_of_structs(struct Competitors *, struct Competitors *) заменим поля переменной first на поля переменной participant. Затем, той же функцией сдвинем тройку лучших: первый станет вторым, второй - третьим).
После сравнения структур перейдем к следующей итерации цикла.
Когда будет достигнут конец файла, цикл завершит свою работу. Закроем файл с исходными данными и перейдем к выводу ответа на экран. Выводить будем переменные first, second и third в том случае, если в поле res переменной не записано -1.

4.1.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2
Операционная система: Debian GNU/Linux 8.1
При решении задачи использовалось ручное тестирование.

4.1.5 Тестовый план и результаты тестирования

При решении задачи было проведено ручное тестирование.

1. Содержание исходного файла: Ivanov I.I. 312 312 321
   Petrov V.B. 444 380 239
   Kozlov K.N. 323 990 234
   Sidorovich A.H. 700 902 492
   Kuznetsov D.L. 865 323 401
   Putin V.V. 3004 4242 4245
   Bereznin A.A. 594 442 689
   Pahomov S.I. 2999 0 0
   Grigorivich G.P. 544 9 234
   Mechenkov K.S. 333 3 3
   Naumenkov I.F. 90 880 353
   Saharov K.I. 390 399 3950
   Lopuhov T.M. 595 445 399

   Результат работы программы:
   I место: Putin V.V., результат: 4245
   II место: Saharov K.I., результат: 3950
   III место: Pahomov S.I., результат: 2999

2. Содержание исходного файла:
   Ivanov I.I. 312 312 321
   Petrov V.B. 444 380 239

   Результат работы программы:
   I место: Petrov V.B., результат: 444
   II место: Ivanov I.I., результат: 321

Таким образом, программа прошла все представленные тесты.
Утилита cppcheck сообщила о стилевой ошибке, аналогичной в предыду-

щей задаче:
(style) The scope of the variable 'i' can be reduced.

4.1.6 Выводы

При решении задачи автор научился пользоваться некоторыми строковыми функциями в языке C. Получившаяся программа верно выводит ответ в каждом тесте.

Глава 5

Инкапсуляция

5.1 Задание1

5.1.1 Задание

Реализовать класс МАТРИЦА. Требуемые методы: конструктор, деструктор, копирование, сложение, вычитание, умножение матриц, умножение на число.

5.1.2 Теоретические сведения

Для выполнения этого задания, используем объектно-ориентированное программирование. Создадим класс Matrix, в котором и будут реализованы требуемые методы. Для того, чтобы сложить две матрицы, необходимо к каждому элементу первой матрицы прибывать соответствующий элемент второй матрицы. Аналогично вычитание матриц. Чтобы умножить матрицу на число, нужно каждый ее элемент умножить на это число. Операция умножения двух матриц выполнима только в том случае, если число столбцов в первом сомножителе равно числу строк во втором.
Для копирования, сложения, вычитания, умножения и умножения на число перегрузим операторы. Если размеры матриц неравны при сложении, копировании и вычитании, введем исключение. Если матрицы при умножении не соответствуют условию, будем "бросать"другое исключение.

5.1.3 Проектирование

В классе Matrix имеются public-методы и private-переменные. Размеры матрицы и ее содержимое будет private. Публичными будут следующие методы: конструктор, конструктор копирования, деструктор, все перегруженные операторы, функции, возвращающие количество столбцов и строк матрицы, а так же функции, устанавливающие и получающие значение элемента матрицы.
В конструкторе сначала инициализируем размеры матрицы числами, с которыми он вызовется. Стандартные размеры будут 5×5, n - количество строк матриц, а m - количество столбцов. Затем, с помощью функции new, выделим память для матрицы: сначала для массива (длинной n) указателей на массивы, потом для самих n массивов, длинны m. После выделения памяти инициализируем матрицу нулями.
Конструктор копирования вызывается для матрицы, которая передается как параметр функции. В нем выделим память под копию матрицы и инициализируем ее.
В деструкторе будем освобождать память в обратном порядке ее выделения. Т.е. сначала очистим память для n массивов, а затем удалим массив указателей.
Перегрузка оператора присваивания. Обозначим матрицу, которой присваивают, - Matrix1; матрица, которую присваивают - Matrix2. Во-первых, учтем тот факт, что мы не можем присваивать друг другу матрицы разных размеров. Поэтому перед реализацией проверим, являются ли размеры матриц равными. Если размеры матриц неравны, возбудим исключение UnequalMatrix. После проверки размеров матриц, освободим память матрицы Matrix1, затем выделим новую память для этой же матрицы и скопируем в нее элементы Matrix2 с помощью метода set(int, int, int).
Метод void set(int i, int j, int value) устанавливает в элемент с индексом [i, j] значение value.
Метод int get(int i, int j) возвращает значение элемента с индексом [i, j].
Перегрузка оператора сложения. Перед суммированием матриц проверим, являются ли их размеры равными. Если нет, "бросим"исключение UnequalMatrix, как в операторе присваивания. После проверки создадим матрицу result с размерами, аналогичными размерам суммируемых матриц. Каждому элементу матрицы result присвоим сумму соответствующих элементов складываемых матриц. Возвращать будем матрицу result. Аналогично перегрузим оператор вычитания.
Для реализации умножения на число number перегрузим оператор умножения. Здесь нам не важен размер матрицы. Создадим матрицу result

таких же размеров, как исходная. Затем каждый элемент исходной матрицы умножим на number и присвоим результат в соответствующий элемент матрицы result. Вернем матрицу result.
Перегрузка оператора умножения матрицы на матрицу. Перед началом умножения проверим условие, которое позволяет умножить матрицу Matrix1 на матрицу Matrix2. Если количество столбцов матрицы Matrix1 равно количеству строк Matrix2, то можно выполнять умножение матрицы Matrix1 на Matrix2. В противном случае возбудим исключение ImpossibleMultiplication. После проверки условия введем матрицу result. Количество ее строк будет равно количеству строк матрицы Matrix1, а число столбцов - числу столбцов матрицы Matrix2. По правилу умножения матриц будем находить каждый элемент матрицы result. После выполнения всех действий, вернем result.
Методы int getNumOfCols() и int getNumOfRows() возвращают количество столбцов и строк матрицы соответственно.

5.1.4 Описание тестового стенда и методики тестирования

Среда разработки: Qt Creator 3.5.0 (opensource)
Компилятор: gcc (Debian 4.9.2-10) 4.9.2
Операционная система: Debian GNU/Linux 8.1
При решении задачи использовалось автоматическое тестирование.

5.1.5 Тестовый план и результаты тестирования

1. Тестирование суммы, разности матриц, а так же методов getNumOfRows и getNumOfCols.
   Введем матрицы matrix1(2, 3), matrix2(2, 3), matrix3(2, 3). Последняя матрица будет содержать результаты действий с первыми двумя.
   $matrix1 = \begin{pmatrix} 2 & 1 & 4 \\ 3 & 2 & 5 \end{pmatrix}$; $matrix2 = \begin{pmatrix} 3 & 4 & 3 \\ 2 & 7 & 1 \end{pmatrix}$.
   Метод getNumOfRows(matrix1) вернул 2, метод getNumOfCols(matrix2) вернул 3, что соответствует действительности. Сложим их и проверим некоторые элементы матрицы matrix3. matrix3.get(0, 0) = 5; matrix3.get(1, 1) = 9; matrix3.get(1, 2) = 6. При "ручном"сложении матриц можно убедиться, что данные элементы имеют именно такие значения.
   Вычтем матрицу matrix2 из matrix1. Так же проверим некоторые элементы. matrix3.get(0, 1) = -3; matrix3.get(1, 0) = 1; matrix3.get(1,

1) = -5. Это не противоречит ожидаемому результату.

2. Тестирование умножения на число. Возьмем матрицу matrix3 из предыдущего пункта. Умножим ее на 4 и проверим некоторые ее элементы. matrix3.get(0, 1) = -12; matrix3.get(1, 1) = -20. Сравним эти элементы с элементами из предыдущего пункта. Очевидно, что матрица умножилась верно.

3. Тестирование умножения матрицы на матрицу. Для этого создадим матрицы matrix4(3, 3) и matrix5(2, 3). Будем умножать matrix1 из первого пункта на matrix4. Результатом умножения является матрица matrix5.
   $matrix1 = \begin{pmatrix} 2 & 1 & 4 \\ 3 & 2 & 5 \end{pmatrix}$; $matrix5 = \begin{pmatrix} 0 & -2 & 3 \\ 1 & 3 & 4 \\ 6 & 2 & -1 \end{pmatrix}$
   После умножения должна получиться матрица: $matrix4 = \begin{pmatrix} 25 & 7 & 6 \\ 32 & 10 & 12 \end{pmatrix}$
   После проверки каждого элемента полученной матрицы, оказалось, что результат умножения оправдал наши ожидания.

Так же в тестировании использовались методы set и get, а так же оператор присваивания. Они работают верно.
Cppcheck не нашел ошибок в этом задании.

5.1.6 Выводы

При решении задачи, автор впервые столкнулся с объектно-ориентированным программированием; узнал, что такое класс, методы класса, какими бывают методы; научился перегружать операторы, возбуждать исключения, а так же пользоваться утилитой для статического анализа кода Cppcheck. Получившийся класс МАТРИЦА содержит все требуемые в условии методы. Результаты их работы соответствуют ожидаемым. Можно сделать вывод, что задание выполнено верно.

Листинги

Функция main

```
1 #include <stdio.h>
2 #include <math.h>
```

```c
#include <stdlib.h>
#include <string.h>
#include "length_oh_segment_ui.h"
#include "equation_ui.h"
#include "km_to_miles_ui.h"
#include "matrix.h"
#include "strings.h"

int main(int argc, char* argv[])
{

     printf("\n argc = %d \n", argc);

     printf("value is %s\n", argv[0]);
     puts("1. Нахождение расстояния между двумя точками");
     puts("2. Решение биквадратного уравнения");
     puts("3. Таблица пересчета миль в километры");
     puts("4. Замена элемента матрицы A[i, j] на сумму");
     puts(" элементов подматрицы A'[i, j]");
     puts("5. Определение победителя в соревнованиях.");
     printf("> ");

     int choice;
     scanf("%5d", &choice);
     switch(choice){
        case 1: {
            length_of_segment_ui();
            break;
        }
        case 2: {
            solution_of_equation_ui();
            break;
        }
        case 3: {
            km_to_miles_ui();
            break;
        }
        case 4: {
            change_matrix();
            break;
        }
        case 5: {
            results_of_competition();
        }
     }

     return 0;
}
```

Задача 1. Длина отрезка

```c
#include <stdio.h>
#include "length_oh_segment_ui.h"
#include "length_of_segment.h"

void length_of_segment_ui(){

    puts("Вычисление длины отрезка по заданным координатам концов");
    puts("Введите кординаты первой точки:");

    struct Point A;
    scanf("%30000d%300000d", &A.x, &A.y);

    puts("Введите координаты второй точки:");
    struct Point B;
    scanf("%30000d%300000d", &B.x, &B.y);

    /// не помню, писала ли я об этом раньше, сейчас не буду искать в истории,
    /// но было бы хорошо сделать структуру для координат, а не передавать четыре штуки по отдельности
    printf("%4.2f \n", length_of_segment(A, B));

}
```

```c
#include <math.h>
#include "length_of_segment.h"

double length_of_segment(struct Point a, struct Point b){

    return sqrt(pow((a.x - b.x), 2) + pow((b.y - a.y), 2));
}
```

Задача 2. Биквадратное уравнение

```c
#include <stdio.h>
#include "equation.h"
#include "equation_ui.h"

void solution_of_equation_ui(){

    struct Solutions_of_equation answer;

    puts("Решение биквадратного уравнения");
    puts("Введите коэффициенты и свободный член");

    int a, b, c;
    scanf("%50000d %50000d %50000d", &a, &b, &c);
    puts("Ответ:");

    if (solve_the_equation(a, b, c, &answer) == 0) {
        puts("Не имеет решений");
    }
    else {

        int i;
        for (i = 0; i < 4; ++i){
            if (answer.existence[i] == 1){
                printf("%4.2f\n", answer.solutions[i]);
            }
        }
    }

}
```

```c
#include <stdio.h>
#include <math.h>
#include "equation.h"

int sign_discriminant(int a, int b, int c){

    int d;
    d = b * b - 4 * a * c;

    if (d < 0) return 0;
    return 1;

}

int solve_the_equation(int a, int b, int c, struct
   Solutions_of_equation* f){

```

```
17      if (a != 0) {
18          switch (sign_discriminant(a, b, c)){
19              case 0: return 0;
20              case 1: calculating_of_solutions(a, b, b * b - 4
                    * a * c, f);
21          }
22        }
23      return 1;
24  }
25
26  void calculating_of_solutions(int a, int b, int D,
27                                struct Solutions_of_equation* f
                                      ){
28
29      int i;
30      for(i = 0; i < 4; ++i){
31          f->existence[i] = 0;
32      }
33
34      double x1_2, x2_2;
35
36      x1_2 = (-b + sqrt(D)) / (2 * a);
37      x2_2 = (-b - sqrt(D)) / (2 * a);
38
39      if (x1_2 >= 0) {
40          f->solutions[0] = sqrt(x1_2);
41          f->existence[0] = 1;
42
43          f->solutions[1] = -sqrt(x1_2);
44          f->existence[1] = 1;
45      }
46      if (x2_2 >= 0) {
47          f->solutions[2] = sqrt(x2_2);
48          f->existence[2] = 1;
49
50          f->solutions[3] = -sqrt(x2_2);
51          f->existence[3] = 1;
52      }
53
54      analysis_of_solutions(f);
55  }
56
57   void analysis_of_solutions(struct Solutions_of_equation *f){
58
59      int i, j;
60
61      for (i = 0; i < 4; ++i){
62          if (f->existence[i] == 1){
63              for (j = 1 + i; j < 4; ++j){
```

```
64                  if ((f->existence[j] == 1) && (f->solutions[i
                        ] == f->solutions[j])) {
65                      f->existence[j] = 0;
66                  }
67              }
68          }
69
70      }
71
72  }
```

5.1.7 Тесты к задаче 1 и задаче 2

```
#include <QString>
#include <QtTest>
#include <math.h>
#include "length_of_segment.h"
#include "equation.h"
#include "stringsx.h"


int double_compare(double a, double res){

    if (fabs(a - res) < 10e-5) return 1;
    return 0;
}

int struct_equality(struct Solutions_of_equation *f, float arr[4]){

    int i;
    for (i = 0; i < 4; ++i){
        if ((f->existence[i] == 1) &&
                (fabs(f->solutions[i] - arr[i]) >= 10e-2))
                    return 0;
    }
    return 1;
}

//int compare_strings(char str1 [], char str2 [])
//{
//    for (int i = 0; i < (int)strlen(str1); i++)
//        if (str1[i] != str2[i]) return 0;
//    return 1;
//}


class TestTest : public QObject
{
    Q_OBJECT

public:
    TestTest();

private Q_SLOTS:
    void length_test();
    void equation_test();
   // void string_test();

};
```

```
46
47 TestTest::TestTest() {}
48
49 void TestTest::length_test()
50 {
51     Point A, B, C, D, E, F;
52     A.x = 0; A.y = 0; B.x = 3; B.y = 4; C.x = 30; C.y = 20;
53     D.x = 1; D.y = 1; E.x = 16; E.y = 16; F.x = -4; F.y =
           -10;
54     QVERIFY2(double_compare(length_of_segment(A, B), 5), "
           Length fail");
55     QVERIFY2(double_compare(length_of_segment(C, D),
           34.66987), "Length fail");
56     QVERIFY2(double_compare(length_of_segment(E, F),
           32.80244), "Length fail");
57 }
58
59 void TestTest::equation_test()
60 {
61     float res1[4], res2[4], res3[4];
62     struct Solutions_of_equation s1, s2, s3;
63     struct Solutions_of_equation *a1, *a2, *a3;
64
65     res1[0] = 1.41; res1[1] = -1.41; res1[2] = 1.00; res1[3]
           = -1.00;
66     res2[0] = 2.00; res2[1] = -2.00; res2[2] = 0;
67     res3[0] = 0.00;
68
69     a1 = &s1; a2 = &s2; a3 = &s3;
70
71     solve_the_equation(1, -3, 2, a1);
72     solve_the_equation(2, -8, 0, a2);
73     solve_the_equation(1, 0, 0, a3);
74
75     QVERIFY2(struct_equality(a1, res1), "Failure");
76     QVERIFY2(struct_equality(a2, res2), "Failure");
77     QVERIFY2(struct_equality(a3, res3), "Failure");
78 }
79
80 //void TestTest::string_test()
81 //{
82 //     struct Competitors comp;
83 //     char str[24], name[12];
84 //     int i, results[3];
85
86 //     strcpy(str, "Ivanov D.L. 500 403 555");
87 //     strcpy(name, "Ivanov D.L. ");
88 //     i = parse_name(str, &comp);
89 //     parse_results(i, str, results);
```

```
90
91 //     QVERIFY2(compare_strings(comp.name, name), "Fail string
       name");
92
93 //}
94
95 QTEST_APPLESS_MAIN(TestTest)
96
97 #include "tst_testtest.moc"
```

Задача 3. Циклы

```c
#include <stdio.h>
#include <stdlib.h>
#include"km_to_miles_ui.h"
#include"km_to_miles.h"

void km_to_miles_ui(){

    puts("Таблица пересчета миль в километры");
    puts("Введите число километров");

    /// конкрно здесь float может быть и уместен,
    /// но по умолчанию лучше брать double, памяти занимает б
        ольше на байты,
    /// а точности дает больше на порядки
    double km;
    scanf("%1000lf", &km);

    ///обычно более вероятную и "нормальную" ветвь исполнения
         делают ветвью истинности,
    /// так проще будет читать
    if (km > 0) km_to_miles(km);
        else {
            puts("Число километров должно быть натуральным!")
                ;
            exit(0);
    }

}
```

```c
#include <stdio.h>
void km_to_miles(float km){

    float mile, kilometers, m;
    puts("Мили   Километры");

    mile = 0.622;
    kilometers = 1;
    m = 1;

    while (kilometers <= km) {

      if (mile > m) {
          printf("%4.2f %4.2f\n", m, m * 1.609);
          m = m + 1;
      }
      printf("%4.2f %4.2f\n", mile, kilometers);
      kilometers = kilometers + 1;
```

```
19        mile = kilometers / 1.609;
20
21      }
22  }
```

Задача 4. Матрицы

```c
#include"matrix.h"
#include <stdlib.h>
#include <stdio.h>

int calc_column(int line, int col, int **arr){

    int u, ans;

    ans = 0;
    for(u = 0; u <= line; ++u){
        ans = ans + arr[u][col];
    }

    return ans;
}

/// изменить матрицу, а выше ф-я поменять строку, хм... что же они должны делать???
// Выше функция посчитать столбец. Она сделана, чтобы не пересчитывать послностью матрицу для элемента,
// а взять ответ для прерыдущего и прибавить сумму элементов, расположенных выше. Эту сумму и считает ф-я выше.
void change_matrix(){

    FILE *file_in, *file_out;
    int m, n, i, j;
    short int pre_ans;
    int ** arr;

    file_in = fopen("input.txt", "r");

    fscanf(file_in, "%200d %200d", &m, &n);
    arr = malloc(m * sizeof(int *));
    for (i = 0; i < m; i++){
        arr[i] = malloc(n * sizeof(int));
    }

    for (i = 0; i < m; i++){
        for (j = 0; j < n; j++){
            fscanf(file_in, "%5000d", &arr[i][j]);
        }
    }

    fclose(file_in);

    file_out = fopen("output.txt", "w");

```

```
45      for (i = 0; i < m; ++i){
46          pre_ans = calc_column(i, 0, arr);
47          fprintf(file_out, "%d ", pre_ans);
48          for (j = 1; j < n; ++j){
49              fprintf(file_out, "%d ", pre_ans + calc_column(i,
                    j, arr));
50              pre_ans = pre_ans + calc_column(i, j, arr);
51          }
52          fprintf(file_out, "\n");
53      }
54
55      fclose(file_out);
56
57      for (i = 0; i < m; ++i){
58          free(arr[i]);
59      }
60      free(arr);
61  }
```

Задача 5. Строки

```c
#include <stdio.h>
#include <stdlib.h>
#include "stringsx.h"
#include "strings.h"

void results_of_competition(){

    FILE* f;
    char str[40];
    int results [3];
    short int i, max;

    struct Competitors participant, first, second, third;

    first.res = -1;
    second.res = -1;
    third.res = -1;
    f = fopen("competition.txt", "r");

    while (fgets(str, 40, f) != NULL) {
        i = parse_name(str, &participant);
        parse_results(i, str, results);

        max = -1;
        for (i = 0; i < 3; ++i){
            if (results[i] > max)  max = results[i];
        }
        participant.res = max;

        place_participants(first, third, second, participant);
    }

    fclose(f);

    if (first.res != -1) printf("I место: %s, результат: %d\n", first.name, first.res);
    if (second.res != -1) printf("II место: %s, результат: %d\n", second.name, second.res);
    if (third.res != -1) printf("III место: %s, результат: %d\n", third.name, third.res);

}
```

```c
#include <string.h>
#include <stdlib.h>
#include "stringsx.h"
```

```
4  #include "strings.h"
5
6  int parse_name(char name [40], struct Competitors* ptr){
7
8      int k, i;
9      k = 0;
10     i = 0;
11
12     while (k < 2) {
13         if (name[i] == ' ') k = k + 1;
14         ptr->name[i] = name[i];
15         i = i + 1;
16     }
17     ptr->name[i] = '\0';
18     return i;
19 }
20
21 void parse_results(int i, char name[40], int results[3]){
22
23     short int j, u;
24     char res [6];
25
26     for (j = 0; j < 3; ++j) {
27         u = 0;
28         while ((name[i] != ' ') && (name[i] != '\n')) {
29             res[u] = name[i];
30             i = i + 1;
31             u = u + 1;
32         }
33         res[u] = '\0';
34         i = i + 1;
35         results[j] = atoi(res);
36     }
37
38 }
39
40 int compare_structs(struct Competitors* k1,
41                     struct Competitors* k2){
42
43     if (k1->res > k2->res) return 1;
44     return 0;
45 }
46
47 void assigment_of_structs(struct Competitors* k1,
48                           struct Competitors* k2){
49
50     strcpy(k2->name, "");
51     strcpy(k2->name, k1->name);
52     k2->res = k1->res;
```

```
53
54 }
55
56 void place_participants(struct Competitors first, struct
       Competitors second, struct Competitors third, struct
       Competitors participant)
57 {
58     if (compare_structs(&participant, &first) == 1) {
59         assigment_of_structs(&second, &third);
60         assigment_of_structs(&first, &second);
61         assigment_of_structs(&participant, &first);
62     }
63     else if (compare_structs(&participant, &second) == 1) {
64             assigment_of_structs(&second, &third);
65             assigment_of_structs(&participant, &second);
66          }
67          else if (compare_structs(&participant, &third) == 1)
                {
68                   assigment_of_structs(&participant, &third);
69               }
70 }
```

Задача 6. Инкапсуляция. МАТРИЦА

```
#ifndef MATRIX_H
#define MATRIX_H

class Matrix
{
public:
    Matrix(int n = 5, int m = 5);
    Matrix(const Matrix&);
    int getNumOfCols() const;
    int getNumOfRows() const;
    void set(int, int, int);
    int get(int, int) const;
    Matrix& operator=(const Matrix& arr);
    Matrix operator+(const Matrix& a);
    Matrix operator-(const Matrix& a);
    Matrix operator*(const int);
    Matrix operator*(const Matrix& a);

    ~Matrix();

private:
    int ** matrix;
    const int n, m;
};

#endif // MATRIX_H
```

```
#include <iostream>
#include <iomanip>
#include "exceptions.h"
#include "matrix.h"

Matrix::Matrix(int a, int b): n(a), m(b)
{
    matrix = new int*[n];
    for (int i = 0; i < n; ++i){
        matrix[i] = new int[m];
        for (int j = 0; j < m; j++)
            matrix[i][j] = 0;
    }

}

Matrix::Matrix(const Matrix& ptr): n(ptr.n), m(ptr.m)
{
    matrix = new int*[n];
    for (int i = 0; i < n; i++){
```

```
21          matrix[i] = new int[m];
22          for (int j = 0; j < m; j++)
23              matrix[i][j] = ptr.matrix[i][j];
24      }
25  }
26
27  int Matrix::getNumOfCols() const
28  {
29      return m;
30  }
31
32  int Matrix::getNumOfRows() const
33  {
34      return n;
35  }
36
37  Matrix&Matrix::operator=(const Matrix &arr)
38  {
39      if (n != arr.getNumOfRows() || m != arr.getNumOfCols())
            throw UnequalMatrix(*this, arr);
40
41      if (this != &arr){
42          int i, j;
43
44          for (i = 0; i < this->getNumOfRows(); i++)
45              /// семен-семеныч...?????????
46              delete[] this->matrix[i];
47          delete[] this->matrix;
48
49          this->matrix = new int*[n];
50          for (i = 0; i < n; i++)
51              matrix[i] = new int[m];
52
53          for (i = 0; i < arr.getNumOfRows(); i++)
54              for (j = 0; j < arr.getNumOfCols(); j++)
55                  this->set(i, j, arr.get(i, j));
56      }
57      return *this;
58  }
59
60  Matrix Matrix::operator+(const Matrix& a)
61  {
62
63      if (n != a.getNumOfRows() || m != a.getNumOfCols()) throw
            UnequalMatrix(*this, a);
64
65      Matrix result(a.getNumOfRows(), a.getNumOfCols());
66      for (int i = 0; i < result.getNumOfRows(); i++)
67          for (int j = 0; j < result.getNumOfCols(); j++)
```

```
68              result.set(i, j, a.get(i, j) + this->get(i, j));
69      return result;
70  }
71
72  Matrix Matrix::operator-(const Matrix& a)
73  {
74      if (n != a.getNumOfRows() || m != a.getNumOfCols()) throw
            UnequalMatrix(*this, a);
75
76      Matrix result(a.getNumOfRows(), a.getNumOfCols());
77      for (int i = 0; i < result.getNumOfRows(); i++)
78          for (int j = 0; j < result.getNumOfCols(); j ++)
79              result.set(i, j, this->get(i, j) - a.get(i, j));
80      return result;
81  }
82
83  Matrix Matrix::operator*(const int number)
84  {
85      Matrix result(n, m);
86      for (int i = 0; i < n; i++)
87          for (int j = 0; j < m; j++)
88              result.set(i, j, matrix[i][j] * number);
89      return result;
90  }
91
92  Matrix Matrix::operator*(const Matrix& a)
93  {
94      if (m != a.getNumOfRows()) throw ImpossibleMultiplication
            ();
95
96      Matrix result(this->getNumOfRows(), a.getNumOfCols());
97      int element = 0;
98      for (int i = 0; i < n; i++)
99          for (int j = 0; j < a.getNumOfCols(); j++){
100             for (int u = 0; u < m; u++)
101                 element = element + (this->get(i, u) * a.get(
                        u, j));
102             result.set(i, j, element);
103             element = 0;
104         }
105     return result;
106
107 }
108
109 void Matrix::set(int i, int j, int val)
110 {
111     if (i < 0 || i > n || j < 0 || j > m) throw
            IndexException(i, j);
112     matrix[i][j] = val;
```

```
113 }
114
115 int Matrix::get(int i, int j) const
116 {
117     if (i < 0 || i > n || j < 0 || j > m) throw
        IndexException(i, j);
118     return matrix[i][j];
119 }
120
121 Matrix::~Matrix()
122 {
123     for (int i = 0; i < n; ++i){
124         delete[] matrix[i];
125     }
126     delete[]matrix;
127
128 }
```

Тесты для МАТРИЦЫ

```
#include <QString>
#include <QtTest>
#include "matrix.h"

class CppTestsTest : public QObject
{
    Q_OBJECT

public:
    CppTestsTest();

private Q_SLOTS:
    void matrix_NumOfCols();
    void matrix_sum();
    void matrix_sub();
    void matrix_multNumber();
    void matrix_mult();
};

CppTestsTest::CppTestsTest()
{
}

/// Не надо лепить все тесты сразу в одном тестовом методе
/// Делайте тестовый метод для каждого public метода (и перег
    руженного оператора) по отдельности
/// тогда, если сломается умножение матриц, упадет только тес
    товый метод связанный с умножением,
/// но будет понятно, что все остальное работает по-прежнему
/// это поможет локализовать ошибку просто глядя на отчет о п
    рохождении тестов
/// а в текущем состоянии будет казаться, что вся матрица к ч
    ертям сломалась
void CppTestsTest::matrix_NumOfCols()
{
    Matrix matrix1(2, 3);

    QCOMPARE(matrix1.getNumOfCols(), 3);
    QCOMPARE(matrix1.getNumOfRows(), 2);

}

void CppTestsTest::matrix_sum()
{
    Matrix matrix1(2, 3), matrix2(2, 3), matrix3(2, 3);
    matrix1.set(0, 0, 2); matrix1.set(0, 1, 1); matrix1.set
        (0, 2, 4);
```

```
43      matrix1.set(1, 0, 3); matrix1.set(1, 1, 2); matrix1.set
            (1, 2, 5);
44      matrix2.set(0, 0, 3); matrix2.set(0, 1, 4); matrix2.set
            (0, 2, 3);
45      matrix2.set(1, 0, 2); matrix2.set(1, 1, 7); matrix2.set
            (1, 2, 1);
46
47      matrix3 = matrix1 + matrix2;
48      QCOMPARE(matrix3.get(0, 0), 5);
49      QCOMPARE(matrix3.get(1, 1), 9);
50      QCOMPARE(matrix3.get(1, 2), 6);
51
52  }
53
54  void CppTestsTest::matrix_sub()
55  {
56      Matrix matrix1(2, 3), matrix2(2, 3), matrix3(2, 3);
57      matrix1.set(0, 0, 2); matrix1.set(0, 1, 1); matrix1.set
            (0, 2, 4);
58      matrix1.set(1, 0, 3); matrix1.set(1, 1, 2); matrix1.set
            (1, 2, 5);
59      matrix2.set(0, 0, 3); matrix2.set(0, 1, 4); matrix2.set
            (0, 2, 3);
60      matrix2.set(1, 0, 2); matrix2.set(1, 1, 7); matrix2.set
            (1, 2, 1);
61
62      matrix3 = matrix1 - matrix2;
63      QCOMPARE(matrix3.get(0, 1), -3);
64      QCOMPARE(matrix3.get(1, 0), 1);
65      QCOMPARE(matrix3.get(1, 1), -5);
66
67  }
68
69  void CppTestsTest::matrix_multNumber()
70  {
71      Matrix matrix3(2, 3);
72      matrix3.set(0, 0, 3); matrix3.set(0, 1, -3); matrix3.set
            (0, 2, 1);
73      matrix3.set(1, 0, 5); matrix3.set(1, 1, -5); matrix3.set
            (1, 2, 5);
74      matrix3 = matrix3 * 4;
75      QCOMPARE(matrix3.get(0, 1), -12);
76      QCOMPARE(matrix3.get(0, 2), 4);
77      QCOMPARE(matrix3.get(1, 1), -20);
78
79  }
80
81  void CppTestsTest::matrix_mult()
82  {
```

```
83      Matrix matrix4(3, 3), matrix5(2, 3), matrix1(2, 3);
84      matrix1.set(0, 0, 2); matrix1.set(0, 1, 1); matrix1.set
            (0, 2, 4);
85      matrix1.set(1, 0, 3); matrix1.set(1, 1, 2); matrix1.set
            (1, 2, 5);
86      matrix4.set(0, 0, 0); matrix4.set(0, 1, -2); matrix4.set
            (0, 2, 3);
87      matrix4.set(1, 0, 1); matrix4.set(1, 1, 3); matrix4.set
            (1, 2, 4);
88      matrix4.set(2, 0, 6); matrix4.set(2, 1, 2); matrix4.set
            (2, 2, -1);
89
90      matrix5 = matrix1 * matrix4;
91      QCOMPARE(matrix5.get(0, 0), 25);
92      QCOMPARE(matrix5.get(0, 1), 7);
93      QCOMPARE(matrix5.get(0, 2), 6);
94      QCOMPARE(matrix5.get(1, 0), 32);
95      QCOMPARE(matrix5.get(1, 1), 10);
96      QCOMPARE(matrix5.get(1, 2), 12);
97
98  }
99
100 QTEST_APPLESS_MAIN(CppTestsTest)
101
102 #include "tst_cppteststest.moc"
```