

Программирование

А. Ю. Ламтев

13 декабря 2015 г.

Оглавление

1	Основные конструкции языка	3
1.1	Задание 1. Размен	3
1.1.1	Задание	3
1.1.2	Теоретические сведения	3
1.1.3	Проектирование	3
1.1.4	Описание тестового стенда и методики тестирования	5
1.1.5	Тестовый план и результаты тестирования	5
1.1.6	Выводы	6
1.2	Задание 2. Ферзи	8
1.2.1	Задание	8
1.2.2	Теоретические сведения	8
1.2.3	Проектирование	8
1.2.4	Описание тестового стенда и методики тестирования	10
1.2.5	Тестовый план и результаты тестирования	10
1.2.6	Выводы	11
2	Циклы	16
2.1	Задание 1. Деление уголком	16
2.1.1	Задание	16
2.1.2	Теоретические сведения	16
2.1.3	Проектирование	16
2.1.4	Описание тестового стенда и методики тестирования	18
2.1.5	Тестовый план и результаты тестирования	19
2.1.6	Выводы	19
3	Матрицы	25
3.1	Задание 1. Нули на главной диагонали	25
3.1.1	Задание	25
3.1.2	Теоретические сведения	25
3.1.3	Проектирование	25
3.1.4	Описание тестового стенда и методики тестирования	27

3.1.5	Тестовый план и результаты тестирования	27
3.1.6	Выводы	28
4	Строки	31
4.1	Задание 1. Отцентрировать текст	31
4.1.1	Задание	31
4.1.2	Теоритические сведения	31
4.1.3	Проектирование	31
4.1.4	Описание тестового стенда и методики тестирования	33
4.1.5	Тестовый план и результаты тестирования	33
4.1.6	Выводы	33
5	Приложение к главам 1 - 4	37
5.1	Листинги	37
6	Введение в классы C++	47
6.1	Задание 1. Инкапсуляция. Таблица-ключ-значение	47
6.1.1	Задание	47
6.1.2	Теоритические сведения	47
6.1.3	Проектирование	47
6.1.4	Описание тестового стенда и методики тестирования	48
6.1.5	Тестовый план и результаты тестирования	48
6.1.6	Выводы	48
7	Классы C++	54
7.1	Задание 1. Реализовать классы для всех приложений	54
7.1.1	Задание	54
7.1.2	Выводы	54

Глава 1

Основные конструкции языка

1.1 Задание 1. Размен

1.1.1 Задание

Пользователь задает сумму денег в рублях, меньшую 100 (например, 16). Определить, как выдать эту сумму монетами по 5, 2 и 1 рубль, израсходовав наименьшее количество монет (например, $3 \times 5p + 0 \times 2p + 1 \times 1p$).

1.1.2 Теоретические сведения

При разработке приложения были задействованы следующие конструкции языка: оператор **switch**, структуры данных **struct**, макросы препроцессора – и были использованы функции стандартной библиотеки *printf*, *scanf* и *puts*, определённые в заголовочном файле *stdio.h*; *atoi*, определённая в *stdlib.h*.

Я решил, что разменять сумму денег монетами номиналом 5, 2 и 1 руб. наиболее оптимально можно следующим образом. Необходимо, чтобы монет большего номинала было больше, чем монет меньшего номинала, насколько это возможно. Это послужило основой для реализации алгоритма.

1.1.3 Проектирование

В ходе проектирования было решено выделить пять функций, одна из которых отвечает за логику, а остальные за взаимодействие с пользователем.

1. Логика

- `change_by_coins`

Эта функция вычисляет результат. Она содержит один целочисленный параметр - сумму денег, которую необходимо разменять. Возвращаемое значение имеет структурный тип, который включает 3 целочисленных поля: число монеток в 5 руб, число монеток в 2 руб и число монеток в 1 руб.

2. Взаимодействие с пользователем

- `exchange_output`

Эта функция выводит в консоль результат функции *change_by_coins*. Она содержит один параметр структурного типа, который включает 3 целочисленных поля: число монеток в 5 руб, число монеток в 2 руб и число монеток в 1 руб. Возвращаемое значение имеет тип *void*.

- `help_exchange`

Эта функция выводит в консоль информацию о том, как запускать приложение **Размен** из параметров командной строки. Она не имеет параметров и возвращает пустое значение.

- `exchange_parameters`

Эта функция отвечает за взаимодействие с пользователем при чтении данных из параметров командной строки. Она содержит 2 параметра: типа *int* - количество аргументов командной строки и типа *char*** - массив, содержащий эти аргументы. Считывает данные из параметров командной строки. Вызывает функцию *exchange_output*, которая в свою очередь выводит в консоль результат. Возвращает пустое значение.

- `exchange`

Эта функция отвечает за взаимодействие с пользователем в интерактивном режиме. Она не имеет параметров. Выводит в консоль сообщение о том, что нужно ввести число. Осуществляет контролируемый ввод данных. Вызывает функцию *exchange_output*, которая уже и выводит в консоль результат. Возвращает пустое значение.

1.1.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило тестирование, ручное и автоматическое. Последнее осуществлялось посредством модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

1.1.5 Тестовый план и результаты тестирования

1. Ручные тесты

I тест

Входные данные: 11

Выходные данные: 2 0 1

Результат: Тест успешно пройден

II тест

Входные данные: 3

Выходные данные: 0 1 1

Результат: Тест успешно пройден

2. Модульные тесты *Qt*

I тест

Входные данные: 28

Выходные данные: 5 1 1

Результат: Тест успешно пройден

II тест

Входные данные: 44

Выходные данные: 8 2 0

Результат: Тест успешно пройден

3. Статический анализ *cprcheck*

Утилита *cprcheck* не выявила ошибок.

1.1.6 Выводы

В ходе выполнения работы я получил опыт создания многомодульного приложения с отделением логики от взаимодействия с пользователем. Укрепил навыки в создании структурных типов. А также научился тестировать программу с помощью модульных тестов и анализировать с помощью утилиты *cprcheck*.

Листинги

exchange.h

```
1 #ifndef EXCHANGE_H
2 #define EXCHANGE_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 struct purse
9 {
10     int ones;
11     int twos;
12     int fives;
13 };
14
15 struct purse change_by_coins(int amount);
16
17 #ifdef __cplusplus
18 }
19 #endif
20
21 #endif // EXCHANGE_H
```

exchange_of_coins_process.c

```
1 #include "exchange.h"
2
3 struct purse change_by_coins(int amount)
4 {
5     struct purse coins;
6     coins.fives = amount / 5;
7     coins.twos = (amount % 5) / 2;
```

```

8     coins.ones = (amount % 5) % 2;
9     return coins;
10 }

```

exchange_ui.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "exchange.h"
5 #include "main.h"
6
7 void exchange_output(struct purse coins)
8 {
9     printf("Пятирублёвых монет: %i\n"
10           "Двухрублёвых монет: %i\n"
11           "Рублёвых монет: %i\n",
12           coins.fives, coins.twos, coins.ones);
13 }
14
15 void exchange(void)
16 {
17     int number;
18     struct purse coins;
19
20     do
21     {
22         puts("Сколько рублей нужно разменять?");
23         scanf("%i", &number);
24     }
25     while (number >= 100);
26
27     coins = change_by_coins(number);
28     exchange_output(coins);
29 }
30
31 void exchange_parameters(int argc, char** argv)
32 {
33     switch (argc)
34     {
35         case 2:
36             exchange();
37             break;
38         case 3:
39             {
40                 int num = atoi(argv[2]);
41                 struct purse coins = change_by_coins(num);
42                 exchange_output(coins);
43                 break;

```



```

44|         }
45|         default:
46|             put_error;
47|             help_exchange();
48|             break;
49|     }
50| }

```

1.2 Задание 2. Ферзи

1.2.1 Задание

На шахматной доске стоят три ферзя (ферзь бьет по вертикали, горизонтали и диагоналям). Найти те пары из них, которые угрожают друг другу. Координаты ферзей вводить целыми числами.

1.2.2 Теоретические сведения

При разработке приложения были задействованы следующие конструкции языка: операторы ветвления **if** и **if-else-if**, оператор **switch**, оператор цикла с постусловием **do-while**, структуры данных *struct* и перечисления *enum* – и были использованы функции стандартной библиотеки *printf*, *scanf*, *puts*, определенные в заголовочном файле *stdio.h*; функции *abs* и *atoi*, определенные в *stdlib.h*.

Сведения о том, что ферзь бьет по вертикали, горизонтали или диагоналям, стали основой для реализации алгоритма. Я понял, что два ферзя бьют друг друга в двух случаях: когда они находятся на одной вертикали или горизонтатали, а значит у них есть общая соответственная координата, или когда они находятся на одной диагонали, т.е расстояние между их соответственными координатами одинаково.

1.2.3 Проектирование

В ходе проектирования было решено выделить семь функций, две из которых отвечают за логику, а остальные за взаимодействие с пользователем.

1. Логика

- `check_for_beating`

Эта функция вычисляет, бьют два ферзя друг друга или нет. Имеет два параметра (2 ферзя) структурного типа, объединяющего два целочисленных поля - две координаты ферзя. Тип возвращаемого значения – *int* – 1, если два ферзя бьют друг друга, и 0 – в противном случае.

- **queens_result**

Эта функция определяет, какой ферзь, кого бьет. Имеет три параметра (3 ферзя) структурного типа, объединяющего два целочисленных поля - две координаты ферзя. Далее она несколько раз вызывает функцию *check_for_beating* и для каждой пары ферзей вычисляет результат. Возвращаемое значение имеет тип *int* – один элемент из перечисления *enum*, название которого характеризует результат.

2. Взаимодействие с пользователем

- **input_with_check**

Эта функция осуществляет контролируемый ввод из консоли координат ферзя. Имеет два параметра типа *int* - две координаты ферзя. Возвращает пустое значение.

- **display_result**

Эта функция выводит в консоль результат функции *queens_result*. Она принимает один параметр типа *int* – один элемент из перечисления *enum*, название которого характеризует результат. Возвращаемое значение имеет тип *void*.

- **help_queens**

Эта функция выводит в консоль информацию о том, как запускать приложение **Ферзи** из параметров командной строки. Она не имеет параметров и возвращает пустое значение.

- **queens_parameters**

Эта функция отвечает за взаимодействие с пользователем при вводе данных через параметры командной строки. Она содержит 2 параметра: типа *int* - количество аргументов командной строки и типа *char*** - массив, содержащий эти аргументы.

Считывает данные из параметров командной строки. Вызывает функцию *display_result*, которая выводит результат в консоль. Возвращаемое значение - *void*.

- **queens**

Эта функция отвечает за взаимодействие с пользователем при запуске приложения в интерактивном режиме. Она не имеет параметров. Считывает данные из консоли с помощью функции *input_with_check*. Затем вызывает функцию *display_result*, которая выводит результат в консоль. Возвращаемое значение - *void*.

1.2.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило тестирование, ручное и автоматическое. Последнее осуществлялось посредством модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

1.2.5 Тестовый план и результаты тестирования

1. Ручные тесты

I тест

Входные данные: 3 1 4 8 2 2

Выходные данные: no_one

Результат: Тест успешно пройден

II тест

Входные данные: 4 4 8 2 7 7

Выходные данные: OneThree

Результат: Тест успешно пройден

2. Модульные тесты *Qt*

I тест

Входные данные: 1 2 3 4 5 6

Выходные данные: everyone

Результат: Тест успешно пройден

II тест

Входные данные: 1 6 2 6 1 3

Выходные данные: OneTwo_OneThree

Результат: Тест успешно пройден

3. Статический анализ *cppcheck*

Утилита *cppcheck* не выявила ошибок.

1.2.6 Выводы

В ходе выполнения работы я получил опыт создания многомодульного приложения с отделением логики от взаимодействия с пользователем. Впервые использовал перечисления *enum*, что оказалось очень удобно. Укрепил навыки в создании структурных типов, тестировании программы с помощью модульных тестов и анализе утилитой *cppcheck*.

Листинги

```
queens.h
1 #ifndef QUEENS_H
2 #define QUEENS_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 struct queen
9 {
10     int x;
11     int y;
12 };
13
14 int check_for_beating(struct queen q1, struct queen q2);
15
```

```

16 enum who_beat {no_one = 0, everyone, OneTwo_OneThree,
17               OneTwo_TwoThree, OneTwo,
18               OneThree_TwoThree, OneThree, TwoThree};
19 int queens_result(struct queen q1, struct queen q2,
20                  struct queen q3);
21 #ifdef __cplusplus
22 }
23 #endif
24
25 #endif // QUEENS_H

```

queens_check_for_beating.c

```

1 #include <stdlib.h>
2
3 #include "queens.h"
4
5 int check_for_beating(struct queen q1, struct queen q2)
6 {
7     return (q1.x == q2.x || q1.y == q2.y) || (abs(q1.x-q2
8         .x) == abs(q1.y-q2.y));
9 }

```

queens_result_for_output.c

```

1 #include "queens.h"
2
3 int queens_result(struct queen q1, struct queen q2,
4                  struct queen q3)
5 {
6     int result = no_one;
7     if (check_for_beating(q1, q2))
8     {
9         if (check_for_beating(q1, q3))
10        {
11            if (check_for_beating(q2, q3))
12            {
13                result = everyone;
14            }
15            else
16            {
17                result = OneTwo_OneThree;
18            }
19        }
20        else
21        {

```

```

21         if (check_for_beating(q2, q3))
22         {
23             result = OneTwo_TwoThree;
24         }
25         else
26         {
27             result = OneTwo;
28         }
29     }
30 }
31 else if (check_for_beating(q1, q3))
32 {
33     if (check_for_beating(q2, q3))
34     {
35         result = OneThree_TwoThree;
36     }
37     else
38     {
39         result = OneThree;
40     }
41 }
42 else if (check_for_beating(q2, q3))
43 {
44     result = TwoThree;
45 }
46 return result;
47 }

```

queens_ui.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "main.h"
5 #include "queens.h"
6
7 void queens(void)
8 {
9     struct queen q1, q2, q3;
10
11     // Считать координаты шахматной доски координатами ма
12     // трицы 8x8 от 1 до 8 !!!
13
14     input_with_check(&q1.x, &q1.y, 1);
15     input_with_check(&q2.x, &q2.y, 2);
16     input_with_check(&q3.x, &q3.y, 3);
17
18     display_result(queens_result(q1, q2, q3));
19 }

```

```

19
20 void queens_parameters(int argc, char** argv)
21 {
22     switch (argc)
23     {
24         case 2:
25             queens();
26             break;
27         case 8:
28             {
29                 struct queen q1, q2, q3;
30                 q1.x = atoi(argv[2]);
31                 q1.y = atoi(argv[3]);
32                 q2.x = atoi(argv[4]);
33                 q2.y = atoi(argv[5]);
34                 q3.x = atoi(argv[6]);
35                 q3.y = atoi(argv[7]);
36
37                 display_result(queens_result(q1, q2, q3));
38                 break;
39             }
40         default:
41             put_error;
42             help_queens();
43             break;
44     }
45 }
46
47 void input_with_check(int* x, int* y, int number)
48 {
49     do
50     {
51         printf("Введите координаты %i-го ферзя\n", number);
52         scanf("%i%i", x, y);
53     }
54     while (*x < 1 || *x > 8 || *y < 1 || *y > 8);
55 }
56
57 void display_result(int result)
58 {
59     switch (result)
60     {
61         case no_one:
62             puts("Никто никого не бьет");
63             break;
64         case everyone:
65             puts("Все ферзи бьют друг друга");
66             break;

```

```

67     case OneTwo_OneThree:
68         printf("1 и 2 ферзи бьют друг друга\n1 и 3 фе
           рзи бьют друг друга\n");
69         break;
70     case OneTwo_TwoThree:
71         printf("1 и 2 ферзи бьют друг друга\n2 и 3 фе
           рзи бьют друг друга\n");
72         break;
73     case OneTwo:
74         puts("1 и 2 ферзи бьют друг друга");
75         break;
76     case OneThree_TwoThree:
77         printf("1 и 3 ферзи бьют друг друга\n2 и 3 фе
           рзи бьют друг друга\n");
78         break;
79     case OneThree:
80         puts("1 и 3 ферзи бьют друг друга");
81         break;
82     case TwoThree:
83         puts("2 и 3 ферзи бьют друг друга");
84         break;
85     }
86 }

```


Глава 2

Циклы

2.1 Задание 1. Деление уголком

2.1.1 Задание

Даны натуральные числа M и N . Вывести на экран процесс их деления с остатком

2.1.2 Теоритические сведения

При разработке приложения были задействованы следующие конструкции языка: оператор выбора **switch**, операторы ветвления **if** и **if-else-if**, оператор цикла с предусловием **while** и оператор цикла со счётчиком **for** – и были использованы функции стандартной библиотеки *scanf* и *puts*, определённые в заголовочном файле *stdio.h*; *atoi*, *calloc*, *free*, определённые в *stdlib.h*.

При реализации алгоритма решения задачи, я воспользовался методом деления в столбик целых чисел. Конкретно в таком виде алгоритм используется в России, Франции, Бельгии и других странах.

2.1.3 Проектирование

В ходе проектирования было решено выделить 9 функций, 6 из которых отвечают за логику, а остальные – за взаимодействие с пользователем.

1. Логика

- `numlen`

Эта функция вычисляет длину числа - количество цифр в записи числа. Она имеет один целочисленный параметр - число, длину которого нужно найти. Возвращаемое значение типа `int` - длина числа.

- `power`

Эта функция возводит целое число, переданное как первый аргумент, в целую степень - число переданное, как второй аргумент. Возвращает целое число - результат.

- `n_th_dig_of_num`

Эта функция возвращает *n*-ую цифру числа `number`, где *n* - первый аргумент функции, а `number` - второй. Обращается к функциям `power` и `numlen`.

- `put_number_char_by_char_to_array_with_counter`

Эта функция помещает в массив символов, который является ее первым аргументом, посимвольно число, которое является вторым аргументом. При всем этом есть третий аргумент типа *`int*`* - указатель на счетчик, который считает, сколько в массиве заполнено ячеек. Обращается к функциям `numlen` и `n_th_dig_of_num`.

- `put_n_symbols_to_array_with_counter`

Эта функция помещает в массив символов, который является ее первым аргументом, *n* - второй целочисленный аргумент функции - символов, которые являются третьим аргументом функции. При всем этом есть четвертый аргумент типа *`int*`* - указатель на счетчик, который считает, сколько в массиве заполнено ячеек.

- `put_result_to_array`

Эта функция вычисляет результат - символьную последовательность и помещает его в массив символов. Имеет три аргумента: массив символов, в который помещается результирующая символьная последовательность; и два аргумента типа *`int`* - делимое и делитель соответственно. Возвращает пустое значение. Обращается к функциям

`numlen`;

`power`;

```
n_th_dig_of_num;  
put_number_char_by_char_to_array_with_counter;  
put_n_symbols_to_array_with_counter.
```

2. Взаимодействие с пользователем

- **help_quotient**

Эта функция выводит в консоль информацию о том, как запускать приложение **Деление уголком** из параметров командной строки. Она не имеет аргументов и возвращает пустое значение.

- **quotient_parameters**

Эта функция отвечает за взаимодействие с пользователем при вводе данных через параметры командной строки. Она содержит 2 параметра: типа *int* - количество аргументов командной строки и типа *char*** - массив, содержащий эти аргументы. Считывает данные из параметров командной строки. Динамически выделяет память под массив, в который будет помещен результат. Вызывает функцию *put_result_to_array*. Выводит полученный массив символов в консоль. Освобождает выделенную память. Возвращаемое значение - *void*.

- **quotient**

Эта функция отвечает за взаимодействие с пользователем при запуске приложения в интерактивном режиме. Она не содержит аргументов. Считывает данные из консоли. Динамически выделяет память под массив, в который будет помещен результат. Вызывает функцию *put_result_to_array*. Выводит полученный массив символов в консоль. Освобождает выделенную память. Возвращаемое значение - *void*.

2.1.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило автоматическое тестирование с помощью модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

2.1.5 Тестовый план и результаты тестирования

1. Модульные тесты *Qt*

I тест

Входные данные: 128 2

Выходные данные:

"128|2\n12 64\n--\n 08\n 8\n --\n 0"

Результат: Тест успешно пройден

2. Статический анализ *cppcheck*

Утилита *cppcheck* выдала следующее предупреждение: "на 57-й строке в документе `quotient_process.c` совершается операция над неинициализированной переменной". В данной ситуации мной было предусмотрено, чтобы в итерации цикла, в которой переменная еще не инициализировалась, она не использовалась. То есть к ошибке в выполнении программы это не приводило. Тем не менее, я исправил этот недочет, и теперь *cppcheck* больше не предупреждает о чем-то плохом.

2.1.6 Выводы

В ходе выполнения работы я получил опыт в использовании циклов, обработке массивов и динамическом выделении памяти.

Листинги

`quotient.h`

```
1 #ifndef QUOTIENT_H
2 #define QUOTIENT_H
3
4 #ifdef __cplusplus
5 extern "C" {
```

```

6 #endif
7
8 void put_number_char_by_char_to_array_with_counter(char*
    array, int number, int *index);
9 void put_n_symbols_to_array_with_counter(char* array, int
    n, char symbol, int *index);
10 void put_result_to_array(char* array, int first_number,
    int second_number);
11 int numlen(int num);
12 int n_th_dig_of_num(int n, int num);
13 int power(int a, int b);
14
15 #ifdef __cplusplus
16 }
17 #endif
18
19 #endif // QUOTIENT_H

```

quotient_process.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "quotient.h"
5
6 void put_n_symbols_to_array_with_counter(char* array, int
    n, char symbol, int *index)
7 {
8     for (int i = 0; i < n; ++i)
9         array[++*index] = symbol;
10 }
11
12 void put_number_char_by_char_to_array_with_counter(char*
    array, int number, int *index)
13 {
14     for (int i = 1; i <= numlen(number); ++ i)
15         array[++*index] = n_th_dig_of_num(i, number) +
            48;
16 }
17
18 void put_result_to_array(char* array, int first_number,
    int second_number)
19 {
20     int dividend, residue, result, product;
21     result = first_number / second_number;
22     dividend = first_number / power(10, numlen(result) -
        1);
23     residue = first_number % power(10, numlen(result) -
        1);

```

```

24     int indent = dividend;
25     int crutch = 1; //нужен для правильного числа черточ
        ек в случаях, когда разность равна 0 :)
26     int index = -1;
27     int num_of_additional_spaces;
28     for (int i = 1; i <= numlen(result); ++i)
29     {
30         if (i == 1)
31         {
32             put_number_char_by_char_to_array_with_counter
                (array, dividend, &index);
33             if (residue != 0)
34                 put_number_char_by_char_to_array_with_counter
                    (array, residue, &index);
35             put_n_symbols_to_array_with_counter(array, 1,
                ' ', &index);
36             put_number_char_by_char_to_array_with_counter
                (array, second_number, &index);
37             put_n_symbols_to_array_with_counter(array, 1,
                '\n', &index);
38         }
39
40         product = second_number * n_th_dig_of_num(i,
            result);
41         put_n_symbols_to_array_with_counter(array, numlen
            (indent) - numlen(product), ' ', &index);
42         put_number_char_by_char_to_array_with_counter(
            array, product, &index);
43
44         if (i != 1)
45             put_n_symbols_to_array_with_counter(array, 1,
                '\n', &index);
46
47         if (i == 1)
48         {
49             put_n_symbols_to_array_with_counter(array,
                numlen(first_number) - numlen(dividend) +
                1, ' ', &index);
50
51             put_number_char_by_char_to_array_with_counter
                (array, result, &index);
52             put_n_symbols_to_array_with_counter(array, 1,
                '\n', &index);
53         }
54
55         if (i != 1)
56             put_n_symbols_to_array_with_counter(array,
                num_of_additional_spaces, ' ', &index);
57

```

```

58     if (crutch == 0)
59         put_n_symbols_to_array_with_counter(array,
        numlen(dividend) + 1, '-', &index);
60     else
61         put_n_symbols_to_array_with_counter(array,
        numlen(dividend), '-', &index);
62
63     put_n_symbols_to_array_with_counter(array, 1, '\n',
        &index);
64
65     num_of_additional_spaces = numlen(indent) -
        numlen(dividend - product);
66
67     put_n_symbols_to_array_with_counter(array,
        num_of_additional_spaces, ' ', &index);
68
69     put_number_char_by_char_to_array_with_counter(
        array, dividend - product, &index);
70
71     if (i != numlen(result))
72     {
73         put_number_char_by_char_to_array_with_counter
        (array, n_th_dig_of_num(i, residue), &
        index);
74         put_n_symbols_to_array_with_counter(array, 1,
        '\n', &index);
75     }
76
77     crutch = dividend - product;
78     dividend = (dividend - product) * 10 +
        n_th_dig_of_num(i, residue);
79     indent = indent * 10 + n_th_dig_of_num(i, residue
        );
80
81 }
82 }
83
84 int numlen(int num)
85 {
86     int count;
87
88     if (num)
89     {
90         count = 0;
91         while (num)
92         {
93             ++count;
94             num /= 10;
95         }

```

```

96     }
97     else
98     {
99         count = 1;
100    }
101    return count;
102 }
103
104 int power(int a, int b)
105 {
106     int result = 1;
107     for (int i = 0; i < b; ++i)
108     {
109         result *= a;
110     }
111     return result;
112 }
113
114 int n_th_dig_of_num(int n, int num)
115 {
116     return (num / power(10, numlen(num) - n)) % 10;
117 }

```

quotient_ui.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3
4 #include "quotient.h"
5 #include "main.h"
6
7 void quotient(void)
8 {
9     int m, n;
10    puts("Введите 2 числа");
11    scanf("%i%i", &m, &n);
12    puts("");
13    char* buffer = (char*) calloc(1000, sizeof(char));
14
15    put_result_to_array(buffer, m, n);
16
17    puts(buffer);
18
19    free(buffer);
20 }
21
22 void quotient_parameters(int argc, char** argv)
23 {
24     switch (argc)

```



```
25     {
26         case 2:
27             quotient();
28             break;
29
30         case 4:
31         {
32             int m = atoi(argv[2]), n = atoi(argv[3]);
33             char* buffer = (char*) calloc(1000, sizeof(
34                 char));
35
36             put_result_to_array(buffer, m, n);
37
38             puts(buffer);
39
40             free(buffer);
41             break;
42         }
43         default:
44             put_error;
45             help_quotient();
46             break;
47     }
```

Глава 3

Матрицы

3.1 Задание 1. Нули на главной диагонали

3.1.1 Задание

В каждом столбце и каждой строке матрицы $P(n,n)$ содержится ровно один нулевой элемент. Перестановкой строк добиться расположения всех нулей по главной диагонали матрицы.

3.1.2 Теоритические сведения

При разработке приложения были задействованы следующие конструкции языка: оператор выбора **switch**, оператор ветвления **if**, оператор цикла со счётчиком **for** – и были использованы функции стандартной библиотеки *fopen*, *fclose*, *fscanf*, *fprintf* и *puts*, определённые в заголовочном файле *stdio.h*; *atoi*, *malloc*, *free*, определённые в *stdlib.h*.

Алгоритм решения задачи был реализован благодаря знанию такого понятия, как сортировка массива.

3.1.3 Проектирование

В ходе проектирования было решено выделить 5 функций, 2 из которых отвечают за логику, а остальные – за взаимодействие с пользователем.

1. Логика

- `level_of_null`

Эта функция определяет, в каком столбце в конкретной строке матрицы находится ноль. Имеет 2 аргумента: *int*** - целочисленную матрицу, *int* - ее размерность и *int* - номер строки. Возвращает целое число - номер столбца.

- `sort_nulls_to_the_main_diagonal`

Эта функция сортирует в матрице нули на главную диагональ. Имеет 2 аргумента: *int*** - целочисленную матрицу и *int* - ее размерность. Обращается к функции *level_of_null*. Возвращает пустое значение.

2. Взаимодействие с пользователем

- `matrix`

Эта функция отвечает за взаимодействие с пользователем при запуске приложения в интерактивном режиме. Она содержит два аргумента типа *char** - названия файлов для ввода и вывода. Динамически выделяет память под массив, в который будет помещен результат. Вызывает функцию *sort_nulls_to_the_main_diagonal*. Выводит полученный массив символов в консоль. Освобождает выделенную память. Возвращаемое значение - *void*.

- `help_matrix`

Эта функция выводит в консоль информацию о том, как запускать приложение **Матрица** из параметров командной строки. Она не имеет аргументов и возвращает пустое значение.

- `matrix_parameters`

Эта функция отвечает за взаимодействие с пользователем при вводе данных через параметры командной строки. Она содержит 2 параметра: типа *int* - количество аргументов командной строки и типа *char*** - массив, содержащий эти аргументы. Считывает данные из параметров командной строки - названия файлов, из которых осуществляется ввод массива, и в которые осуществляется вывод обработанного массива. Вызывает функцию *matrix*. Возвращаемое значение - *void*.

3.1.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило автоматическое тестирование с помощью модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

3.1.5 Тестовый план и результаты тестирования

1. Модульные тесты *Qt*

I тест

Входные данные:

```
5
1 1 1 0 1
1 0 3 4 5
1 3 5 7 0
0 4 7 10 13
1 5 0 13 17
```

Выходные данные:

```
0 4 7 10 13
1 0 3 4 5
1 5 0 13 17
1 1 1 0 1
1 3 5 7 0
```

Результат: Тест успешно пройден

2. Статический анализ *cppcheck*

Утилита *cppcheck* не выдала никаких предупреждений.

3.1.6 Выводы

В ходе выполнения работы я получил опыт в обработке матрицы и в работе с файлами.

Листинги

matrix.h

```
1 #ifndef MATRIX_H
2 #define MATRIX_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 int level_of_null(int** P, int n, int number_of_column);
9 void sort_nulls_to_the_main_diagonal(int** P, int n);
10
11 #ifdef __cplusplus
12 }
13 #endif
14
15 #endif // MATRIX_H
```

matrix_processing.c

```
1 int level_of_null(int** P, int n, int number_of_line)
2 {
3     int result = 0;
4     for (int i = 0; i < n; ++i)
5         if (P[number_of_line][i] == 0)
6         {
7             result = i;
8             break;
9         }
10    return result;
11 }
12
13
14 void sort_nulls_to_the_main_diagonal(int** P, int n)
15 {
16     int* t;
17     int i, j;
18     for (i = n-1; i >= 0; --i)
19         for (j = 0; j < i; ++j)
20             if (level_of_null(P, n, j) > level_of_null(P,
21                 n, j+1))
```

```

21         {
22             t = P[j];
23             P[j] = P[j+1];
24             P[j+1] = t;
25         }
26     }

```

matrix_ui.c

```

1  #include <stdlib.h>
2  #include <stdio.h>
3
4  #include "matrix.h"
5  #include "main.h"
6
7  void matrix(char* input_file_name, char* output_file_name
8  )
9  {
10     FILE* in;
11     FILE* out;
12     in = fopen(input_file_name, "r");
13     out = fopen(output_file_name, "w");
14     int** P;
15     int n, i, j;
16     fscanf(in, "%i", &n);
17
18     P = (int**) malloc(n * sizeof(int*));
19     for (i = 0; i < n; ++i)
20         P[i] = (int*) malloc(n * sizeof(int));
21
22     for (i = 0; i < n; ++i)
23         for (j = 0; j < n; ++j)
24             fscanf(in, "%i\n", &P[i][j]);
25
26     sort_nulls_to_the_main_diagonal(P, n);
27
28     for (i = 0; i < n; ++i)
29     {
30         for (j = 0; j < n; ++j)
31             fprintf(out, "%i ", P[i][j]);
32         fprintf(out, "\n");
33     }
34
35     for (i = 0; i < n; ++i)
36         free(P[i]);
37     free(P);
38     fclose(in);
39     fclose(out);
40     puts("Программа успешно выполнена!");

```

```
40|}  
41|  
42|void matrix_parameters(int argc, char** argv)  
43|{  
44|    switch (argc)  
45|    {  
46|        case 2:  
47|            matrix("matrix.in", "matrix.out");  
48|            break;  
49|        case 4:  
50|            matrix(argv[2], argv[3]);  
51|            break;  
52|        default:  
53|            put_error;  
54|            help_matrix();  
55|            break;  
56|    }  
57|}
```

Глава 4

Строки

4.1 Задание 1. Отцентрировать текст

4.1.1 Задание

Каждую строку заданного текста вывести на экран симметрично относительно его центра.

4.1.2 Теоритические сведения

При разработке приложения были задействованы следующие конструкции языка: оператор выбора **switch**, оператор ветвления **if**, оператор цикла со счётчиком **for**, оператор цикла с предусловием **while** – и были использованы функции стандартной библиотеки *fopen*, *fclose*, *fgets*, *fputs* и *puts*, определённые в заголовочном файле *stdio.h*; *atoi*, *calloc*, *free*, определённые в *stdlib.h*; *strlen*, *memset* и *strcat*, определённые в *string.h*.

Мне не раз приходилось центровать текст в текстовых редакторах, поэтому именно этот опыт стал основой для реализации алгоритма решения задачи.

4.1.3 Проектирование

В ходе проектирования было решено выделить 5 функций, 2 из которых отвечают за логику, а остальные – за взаимодействие с пользователем.

1. Логика

- `determine_file_proportions`

Эта функция открывает файл, проходит по тексту, лежащему в нем, и определяет число строк и максимальную длину строки в этом тексте, а затем закрывает файл. Она имеет три аргумента: *char** - имя файла, *int** - указатель на число строк и *int** - указатель на максимальную длину строки. Возвращает значение типа *void*.

- **symmetrize_line**

Эта функция добавляет в начало строки необходимое число пробелов. Она имеет 3 аргумента: *char** - строка, в которую помещается результат; *char** - исходная строка; *int* - максимальная длина строки в тексте. Возвращает пустое значение.

2. Взаимодействие с пользователем

- **help_lines_symmetrization**

Эта функция выводит в консоль информацию о том, как запускать приложение **Центрирование строк** из параметров командной строки. Она не имеет аргументов и возвращает пустое значение.

- **lines_symmetrization_parameters**

Эта функция отвечает за взаимодействие с пользователем при вводе данных через параметры командной строки. Она содержит 2 параметра: типа *int* - количество аргументов командной строки и типа *char*** - массив, содержащий эти аргументы. Считывает данные из параметров командной строки - названия файлов, из которых осуществляется ввод строк текста, и в которые осуществляется вывод обработанных строк текста. Вызывает функции *determine_file_proportions* и *symmetrize_line*. Возвращаемое значение - *void*.

- **lines_symmetrization**

Эта функция отвечает за взаимодействие с пользователем при запуске приложения в интерактивном режиме. Она содержит два аргумента типа *char** - названия файлов для ввода и вывода. Динамически выделяет память под строки, в которые будет происходить запись с файла, и в которые будут записываться обработанные строки. Вызывает функции *determine_file_proportions*

и *symmetrize_line*. Выводит полученный текст построчно в консоль. Освобождает выделенную память. Возвращаемое значение - *void*.

4.1.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило автоматическое тестирование с помощью модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

4.1.5 Тестовый план и результаты тестирования

1. Модульные тесты *Qt*

I тест

Входные данные: "sdfjl sfvslk ! asdf" 30

Выходные данные: " sdfjl sfvslk ! asdf"

Результат: Тест успешно пройден

2. Статический анализ *cppcheck*

Утилита *cppcheck* не выдала никаких предупреждений.

4.1.6 Выводы

В ходе работы я получил опыт в обработке строк, а также укрепил навык работы с файлами.

Листинги

lines_symmetrization.h

```
1 #ifndef LINES_SYMMETRIZATION_H
2 #define LINES_SYMMETRIZATION_H
3
4 #ifdef __cplusplus
5 extern "C" {
6 #endif
7
8 void symmetrize_line(char* final_line, char* initial_line
9 , int max_length_of_line);
10 void determine_file_proportions(char* input_file_name,
11 int* number_of_lines, int* max_length_of_line);
12
13 #ifdef __cplusplus
14 }
15 #endif // LINES_SYMMETRIZATION_H
```

lines_symmetrization_processing.c

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4
5 #include "lines_symmetrization.h"
6
7 void determine_file_proportions(char* input_file_name,
8 int* number_of_lines, int* max_length_of_line)
9 {
10     const int maximum_length_of_line = 256;
11     FILE *in;
12     in = fopen(input_file_name, "r");
13     char *str;
14     str = (char *) calloc(maximum_length_of_line, sizeof(
15 char));
16     int count = 0;
17     *max_length_of_line = 0;
18     while (!feof(in))
19     {
20         fgets(str, maximum_length_of_line, in);
21         if ((int) strlen(str) > *max_length_of_line)
22             *max_length_of_line = strlen(str);
23         ++count;
24     }
25     *number_of_lines = count;
```

```

24     free(str);
25     fclose(in);
26 }
27
28 void symmetrize_line(char* final_line, char* initial_line
    , int max_length_of_line)
29 {
30     int left_indent = (max_length_of_line - strlen(
        initial_line)) / 2;
31
32     memset(final_line, ' ', left_indent);
33     final_line[left_indent] = '\\0';
34     strcat(final_line, initial_line);
35 }

```

lines_symmetrization_ui.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  #include "lines_symmetrization.h"
5  #include "main.h"
6
7  void lines_symmetrization(char* input_file_name, char*
    output_file_name)
8  {
9      FILE* in;
10     FILE* out;
11     int number_of_lines, max_length_of_line;
12     determine_file_proportions(input_file_name, &
        number_of_lines, &max_length_of_line);
13
14     char* initial_line = (char*) calloc (
        max_length_of_line, sizeof(char));
15     char* final_line = (char*) calloc (max_length_of_line
        , sizeof(char));
16     in = fopen(input_file_name, "r");
17     out = fopen(output_file_name, "w");
18
19     for (int i = 0; i < number_of_lines; ++i)
20     {
21         fgets(initial_line, max_length_of_line, in);
22         symmetrize_line(final_line, initial_line,
            max_length_of_line);
23         fputs(final_line, out);
24     }
25
26     free(initial_line);
27     free(final_line);

```

```

28     fclose(in);
29     fclose(out);
30     puts("Программа успешно выполнена!");
31 }
32
33 void lines_symmetrization_parameters(int argc, char**
    argv)
34 {
35     switch (argc)
36     {
37         case 2:
38             lines_symmetrization("lines.in", "lines.out")
39             ;
40             break;
41         case 4:
42             lines_symmetrization(argv[2], argv[3]);
43             break;
44         default:
45             put_error;
46             help_lines_symmetrization();
47             break;
48     }

```

Глава 5

Приложение к главам 1 - 4

5.1 Листинги

```
main.h
1 #ifndef MAIN
2 #define MAIN
3
4 #define put_error puts("Неправильный ввод параметров!!!")
5 ;
6 void exchange(void);
7 void exchange_parameters(int argc, char** argv);
8
9 void help_exchange(void);
10 void help_queens(void);
11 void help_matrix(void);
12 void help_quotient(void);
13 void help_lines_symmetrization(void);
14 void help(void);
15 void help_parameters(int argc, char** argv);
16
17 void queens(void);
18 void input_with_check(int* x, int* y, int number);
19 void display_result(int result);
20 void queens_parameters(int argc, char** argv);
21
22 void lines_symmetrization(char* input_file_name, char*
    output_file_name);
23 void lines_symmetrization_parameters(int argc, char**
    argv);
24
25 void matrix(char* input_file_name, char* output_file_name
    );
```

```

26 void matrix_parameters(int argc, char** argv);
27
28 void main_menu(void);
29 void menu_no_parameters(void);
30
31 void quotient(void);
32 void quotient_parameters(int argc, char** argv);
33
34 #endif // MAIN

```

help_ui.c

```

1 #include <stdio.h>
2 #include <string.h>
3
4 #include "main.h"
5
6 #define put_equally puts
7 ("===== "\
8 "=====");
9 #define put_exch puts("Параметр --exchange:\n\
10 --exchange                запуск программ
   ы Размен в автоматическом режиме\n\
11 --exchange number        запуск программ
   ы Размен с аргументом number (number - натуральное чис
   ло)");
12 #define put_quens puts("Параметр --queens:\n\
13 --queens                запуск программ
   ы Ферзи в автоматическом режиме\n\
14 --queens x1 y1 x2 y2 x3 y3 запуск программ
   ы Ферзи с аргументами x1, y1, x2, y2, x3, y3 (натураль
   ные числа)");
15
16 #define put_quot puts("Параметр --quotient:\n\
17 --quotient                запуск программ
   ы Деление уголком в автоматическом режиме\n\
18 --quotient dividend divider запуск программ
   ы Деление уголком, где dividend - делимое, а divider -
   делитель");
19
20 #define put_matr puts("Параметр --matrix:\n\
21 --matrix                запуск программ
   ы Матрица в автоматическом режиме\n\
22 --matrix <input> <output> запуск программ
   ы Матрица с входными данными из файла input и с выводом
   в файл output");
23
24 #define put_str puts("Параметр --centered_lines:\n\

```

```

25 --lines_symmetrization                                запуск программ
    ы Симметрирование строк в автоматическом режиме\n\
26 --lines_symmetrization <input> <output>              запуск программ
    ы Симметрирование строк с входными данными из файла
    input и с выводом в файл output");
27
28
29 void help_exchange(void)
30 {
31     put_equally;
32     put_exch;
33     put_equally;
34 }
35
36 void help_queens(void)
37 {
38     put_equally;
39     put_quens;
40     put_equally;
41 }
42 void help_matrix(void)
43 {
44     put_equally;
45     put_matr;
46     put_equally;
47 }
48
49 void help_quotient(void)
50 {
51     put_equally;
52     put_quot;
53     put_equally;
54 }
55
56 void help_lines_symmetrization(void)
57 {
58     put_equally;
59     put_str;
60     put_equally;
61 }
62
63 void help(void)
64 {
65     put_equally;
66     puts("Информация о параметрах командной строки\n"
67         "Параметр --interactive:\n"
68         "--interactive                                запус
        к приложения в интерактивном режиме");
69     puts("");

```


70	<code>puts("Параметр --help:\n"</code>	
71	<code> "--help</code>	ПОМОЩ
	<code> Ъ\n"</code>	
72	<code> "--help <--name_of_task></code>	ПОМОЩ
	<code> Ъ с --name_of_task");</code>	
73	<code>puts("");</code>	
74	<code>put_exch;</code>	
75	<code>puts("");</code>	
76	<code>put_quens;</code>	
77	<code>puts("");</code>	
78	<code>put_quot;</code>	
79	<code>puts("");</code>	
80	<code>put_matr;</code>	
81	<code>puts("");</code>	
82	<code>put_str;</code>	
83	<code>puts("");</code>	
84	<code>put_equally;</code>	
85	<code>}</code>	
86		
87	<code>void help_parameters(int argc, char** argv)</code>	
88	<code>{</code>	
89	<code> switch (argc)</code>	
90	<code> {</code>	
91	<code> case 2:</code>	
92	<code> help();</code>	
93	<code> break;</code>	
94	<code> case 3:</code>	
95	<code> if (!strcmp(argv[2], "--exchange"))</code>	
96	<code> help_exchange();</code>	
97	<code> else if (!strcmp(argv[2], "--queens"))</code>	
98	<code> help_queens();</code>	
99	<code> else if (!strcmp(argv[2], "--matrix"))</code>	
100	<code> help_matrix();</code>	
101	<code> else if (!strcmp(argv[2], "--quotient"))</code>	
102	<code> help_quotient();</code>	
103	<code> else if (!strcmp(argv[2], "--</code>	
	<code> lines_symmetrization"))</code>	
104	<code> help_lines_symmetrization();</code>	
105	<code> break;</code>	
106	<code> default:</code>	
107	<code> put_error;</code>	
108	<code> help();</code>	
109	<code> break;</code>	
110	<code> }</code>	
111	<code>}</code>	

menu.c

1	<code>#include <stdio.h></code>
---	---------------------------------------

```

2 #include <stdlib.h>
3
4 #include "main.h"
5
6 void main_menu(void)
7 {
8     char key;
9     do
10    {
11        system("clear");
12        printf("Выберите программу!\n1)Размен\n2)Ферзи\n3
13        )Деление уголком\n4)Матрица\n"
14        "5)Симметрирование строк\n6)Завершить рабо
15        ту\n");
16        scanf("%c", &key);
17    }
18    while (key < '1' || key > '6');
19
20    switch (key)
21    {
22        case '1':
23            exchange();
24            break;
25        case '2':
26            queens();
27            break;
28        case '3':
29            quotient();
30            break;
31        case '4':
32            matrix("matrix.in", "matrix.out");
33            break;
34        case '5':
35            lines_symmetrization("lines.in", "lines.out")
36            ;
37            break;
38        case '6':
39            exit(0);
40            break;
41    }
42 }
43
44 void menu_no_parameters(void)
45 {
46     char key;
47     do
48    {
49        system("clear");
50        puts("Вы запустили прогамму без параметров!!!");

```

```

48     puts("Выберите вариант продолжения");
49     puts("1)Получить информацию об эксплуатации\n2)За
        вершить программу");
50     scanf("%c", &key);
51 }
52 while (key < '1' || key > '2');
53 switch (key)
54 {
55     case '1':
56         help();
57         break;
58     case '2':
59         exit(0);
60         break;
61 }
62 }

```

main.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <string.h>
4
5  #include "lines_symmetrization.h"
6
7  void determine_file_proportions(char* input_file_name,
    int* number_of_lines, int* max_length_of_line)
8  {
9      const int maximum_length_of_line = 256;
10     FILE *in;
11     in = fopen(input_file_name, "r");
12     char *str;
13     str = (char *) calloc(maximum_length_of_line, sizeof(
        char));
14     int count = 0;
15     *max_length_of_line = 0;
16     while (!feof(in))
17     {
18         fgets(str, maximum_length_of_line, in);
19         if ((int) strlen(str) > *max_length_of_line)
20             *max_length_of_line = strlen(str);
21         ++count;
22     }
23     *number_of_lines = count;
24     free(str);
25     fclose(in);
26 }
27

```

```

28 void symmetrize_line(char* final_line, char* initial_line
    , int max_length_of_line)
29 {
30     int left_indent = (max_length_of_line - strlen(
        initial_line)) / 2;
31
32     memset(final_line, ' ', left_indent);
33     final_line[left_indent] = '\\0';
34     strcat(final_line, initial_line);
35 }

```

tst_qt_teststest.cpp

```

1  #include <QString>
2  #include <QtTest>
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <string.h>
6  #include "exchange.h"
7  #include "queens.h"
8  #include "quotient.h"
9  #include "matrix.h"
10 #include "lines_symmetrization.h"
11
12 class Qt_testsTest : public QObject
13 {
14     Q_OBJECT
15
16 public:
17     Qt_testsTest();
18
19 private Q_SLOTS:
20     void exchange_test();
21     void queens_test();
22     void quotient_test();
23     void matrix_test();
24     void lines_simmetrization_test();
25 };
26
27 Qt_testsTest::Qt_testsTest()
28 {
29 }
30
31 void Qt_testsTest::exchange_test()
32 {
33     struct purse coins_actual;
34
35     coins_actual = change_by_coins(28);
36     QCOMPARE(coins_actual.fives, 5);

```

```

37     QCOMPARE(coins_actual.twos, 1);
38     QCOMPARE(coins_actual.ones, 1);
39
40     coins_actual = change_by_coins(44);
41     QCOMPARE(coins_actual.fives, 8);
42     QCOMPARE(coins_actual.twos, 2);
43     QCOMPARE(coins_actual.ones, 0);
44
45 }
46
47 void Qt_testsTest::queens_test()
48 {
49     struct queen q1, q2, q3;
50
51     q1.x = 1;
52     q1.y = 2;
53     q2.x = 3;
54     q2.y = 4;
55     q3.x = 5;
56     q3.y = 6;
57     QCOMPARE(queens_result(q1, q2, q3), (int) everyone);
58
59     q1.x = 1;
60     q1.y = 6;
61     q2.x = 2;
62     q2.y = 6;
63     q3.x = 1;
64     q3.y = 3;
65     QCOMPARE(queens_result(q1, q2, q3), (int)
66               OneTwo_OneThree);
67 }
68
69 void Qt_testsTest::quotient_test()
70 {
71     char* actual;
72     const char* expected;
73
74     actual = (char*) calloc(38, sizeof(char));
75     put_result_to_array(actual, 128, 2);
76     actual[37] = '\0';
77     expected = "128|2\n12 64\n--\n 08\n 8\n --\n 0";
78     QCOMPARE(strcmp(actual, expected), 0);
79     free(actual);
80 }
81
82 void Qt_testsTest::matrix_test()
83 {
84     int** actual = (int**) malloc(5 * sizeof(int*));

```

```

85     for (int i = 0; i < 5; ++i)
86     {
87         actual[i] = (int*) malloc(5 * sizeof(int));
88         for (int j = 0; j < 5; ++j)
89         {
90             i[actual][j] = i * j + 1; //по приколу так сде
лал :) Страуструп сказал, что так можно.
А я решил проверить.
91         } //Логично, если чер
ез оператор разыменовывания записать. Но в гол
ову это не приходило
92     }
93
94     actual[0][3] = 0; actual[1][1] = 0; actual[2][4] = 0;
95     actual[3][0] = 0; actual[4][2] = 0;
96
97     sort_nulls_to_the_main_diagonal(actual, 5);
98
99     int expected[5][5] = {{0, 4, 7, 10, 13},
100                          {1, 0, 3, 4, 5},
101                          {1, 5, 0, 13, 17},
102                          {1, 1, 1, 0, 1},
103                          {1, 3, 5, 7, 0}};
104
105     for (int i = 0; i < 5; ++i)
106     {
107         for(int j = 0; j < 5; j++)
108         {
109             QCOMPARE(actual[i][j], expected[i][j]);
110         }
111         free(actual[i]);
112     }
113
114     free(actual);
115 }
116
117 void Qt_testsTest::lines_simmetrization_test()
118 {
119     char str[] = "sdfjl sfvslk ! asdf";
120     char* actual = (char*) calloc (30, sizeof(char));
121
122     symmetrize_line(actual, str, 30);
123
124     char expected[] = "      sdfjl sfvslk ! asdf";
125
126     QCOMPARE(strcmp(actual, expected), 0);
127     free(actual);
128 }

```

```
129  
130  
131 QTEST_APPLESS_MAIN(Qt_testsTest)  
132  
133 #include "tst_qt_teststest.moc"
```

Глава 6

Введение в классы C++

6.1 Задание 1. Инкапсуляция. Таблица-ключ-значение

6.1.1 Задание

Реализовать класс ТАБЛИЦА КЛЮЧ-ЗНАЧЕНИЕ (хранит строки, каждой из которых соответствует уникальный целый ключ). Требуемые методы: конструктор, деструктор, копирование, индексация по ключу, добавление нового элемента.

6.1.2 Теоритические сведения

При разработке приложения была задействована объектная ориентированность языка C++.

6.1.3 Проектирование

В ходе проектирования было решено выделить 2 класса, 1 из которых отвечает за логику, а другой – за взаимодействие с пользователем.

1. Класс с логикой `Table`

Были выделены методы: `Table()` - конструктор, `~Table()`- деструктор, `Table(const Table&)` - конструктор копирования, `put(string*, int*)` - добавление нового элемента и `operator[] (int)` - перегруженный оператор индексирования для индексирования по ключу. Также были выделены вспомогательные методы.

2. Класс взаимодействия с пользователем `TableApp`

Были выделены методы *putCellKey()* - интерфейс для добавления нового элемента, *findCellByKey()* - интерфейс для индексирования по ключу, *copyObject()* - интерфейс для копирования объекта.

6.1.4 Описание тестового стенда и методики тестирования

Интегрированная среда разработки: Qt Creator 3.5.0 (opensource)

Компилятор: GCC 4.9.1 20140922 (Red Hat 4.9.1-10)

Операционная система: Debian GNU/Linux 8 (jessie) 32-бита (version 3.14.1)

На всех стадиях разработки приложения проходило автоматическое тестирование с помощью модульных тестов *Qt*, основанных на библиотеке *QTestLib*.

На финальной стадии был проведён статический анализ с помощью утилиты *cppcheck*

6.1.5 Тестовый план и результаты тестирования

1. Модульные тесты *Qt*

Модульными тестами была протестирована работоспособность методов. Все требуемые методы - добавление элемента, копирование объекта, индексирование по ключу, конструктор и деструктор - работают.

2. Статический анализ *cppcheck*

Утилита *cppcheck* не выдала никаких предупреждений.

6.1.6 Выводы

Я познакомился с языком C++. Познакомился с новой парадигмой программирования - ООП.

Листинги

`table.h`

```

1 #ifndef TABLE_H
2 #define TABLE_H
3
4 #include <iostream>
5 #include <new>
6
7 using namespace std;
8
9 class Table
10 {
11     string* cell;
12     int* key;
13     int index;
14     int currentSize;
15     void allocateMoreMemory();
16     string indexByKey(const int keyValue);
17     const char* ERROR_CELL_KEY = "ERROR: there are not
18         any cells with typed key";
19     const int ADDITIONAL_SIZE = 10;
20 public:
21     Table(int tableSize = 5);
22     Table(const Table& object);
23     ~Table();
24     void put(const string value, const int key);
25     string operator[](const int keyValue);
26     //не нужны по заданию, но нужны для удобного тестиров
27     ания
28     string getLastElement();
29     int getKeyOfLastElement();
30 };
31 #endif // TABLE_H

```

table.cpp

```

1 #include "table.h"
2
3 Table::Table(int tableSize)
4 {
5     currentSize = tableSize;
6     index = -1;
7     cell = new string[tableSize];
8     key = new int[tableSize];
9 }
10
11 Table::Table(const Table &object)
12 {

```

```

13     cell = new string[currentSize = object.currentSize];
14     key = new int[currentSize];
15     for (int i = 0; i <= (index = object.index); ++i)
16     {
17         cell[i] = object.cell[i];
18         key[i] = object.key[i];
19     }
20 }
21
22 Table::~Table()
23 {
24     delete[] cell;
25     delete[] key;
26 }
27
28 void Table::allocateMoreMemory()
29 {
30     string* tempCell = new string[currentSize +
31         ADDITIONAL_SIZE];
32     int* tempKey = new int[currentSize + ADDITIONAL_SIZE
33         ];
34     for (int i = 0; i < currentSize; ++i)
35     {
36         tempCell[i] = cell[i];
37         tempKey[i] = key[i];
38     }
39     delete[] cell;
40     delete[] key;
41
42     cell = tempCell;
43     key = tempKey;
44
45     currentSize += ADDITIONAL_SIZE;
46 }
47
48 void Table::put(const string cellValue, const int
49     keyValue)
50 {
51     if (index == currentSize - 1)
52     {
53         allocateMoreMemory();
54     }
55     cell[++index] = cellValue;
56     key[index] = keyValue;
57 }
58 string Table::operator[](const int keyValue)

```

```

59 {
60     for (auto i = 0; i <= index; ++i)
61     {
62         if (key[i] == keyValue)
63         {
64             return cell[i];
65         }
66     }
67     throw ERROR_CELL_KEY;
68 }
69
70 string Table::getLastElement()
71 {
72     return cell[index];
73 }
74
75 int Table::getKeyOfLastElement()
76 {
77     return key[index];
78 }

```

tableApp.h

```

1  #ifndef TABLEAPP_H
2  #define TABLEAPP_H
3
4  #include "table.h"
5
6  using namespace std;
7
8  class TableApp
9  {
10     Table table;
11     void putCellKey();
12     void findCellByKey();
13     void copyObject();
14     const int EXIT_CODE = 9;
15 public:
16     TableApp();
17     ~TableApp();
18     void menu();
19 };
20
21 #endif // TABLEAPP_H

```

tableApp.cpp

```

1  #include "tableApp.h"

```

```

2 #include "table.h"
3
4 TableApp::TableApp()
5 {
6
7 }
8
9 TableApp::~TableApp()
10 {
11
12 }
13
14 void TableApp::menu()
15 {
16     int key;
17     do
18     {
19         cout << "Выберите вариант\n";
20         cout << "1) Положить в таблицу строковое значение и
                целочисленный ключ\n"
21                "2) Найти строку по ключу\n"
22                "3) Копировать текущий объект в другой\n"
23                "9) Завершить работу программы\n";
24         cin >> key;
25         switch (key)
26         {
27             case 1:
28             {
29                 putCellKey();
30                 break;
31             }
32             case 2:
33             {
34                 findCellByKey();
35                 break;
36             }
37             case 3:
38             {
39                 copyObject();
40                 break;
41             }
42         }
43     }
44     while (key != EXIT_CODE);
45 }
46
47
48 void TableApp::putCellKey()
49 {

```

```

50     string str;
51     int k;
52     cout << "Введите строковое значение\n";
53     getline(cin >> ws, str);
54     cout << "Введите целочисленный ключ\n";
55     cin >> k;
56     cin.ignore();
57     table.put(str, k);
58 }
59
60 void TableApp::findCellByKey()
61 {
62     int k;
63     cout << "Введите целочисленный ключ\n";
64     cin >> k;
65     try
66     {
67         cout << table[k];
68     }
69     catch(const char* e)
70     {
71         cout << e;
72     }
73
74     cout << endl;
75 }
76
77 void TableApp::copyObject()
78 {
79     Table newTable(table);
80 }

```

Глава 7

Классы C++

7.1 Задание 1. Реализовать классы для всех приложений

7.1.1 Задание

Реализовать классы для всех приложений. Поработать с потоками.

7.1.2 Выводы

Получил опыт создания классов. Получил опыт в работе с потоками.

Листинги

```
exchange.h
1 #ifndef EXCHANGE_H
2 #define EXCHANGE_H
3
4 struct Coins
5 {
6     int ones;
7     int twos;
8     int fives;
9 };
10
11 class Exchange
12 {
13 public:
14     Exchange();
15     ~Exchange();
16     Coins exchangeMoney(const int moneyAmount) const;
```

```

17 };
18
19 #endif // EXCHANGE_H

```

exchange.cpp

```

1 #include "exchange.h"
2
3 Exchange::Exchange()
4 {
5 }
6
7 Exchange::~Exchange()
8 {
9 }
10
11 Coins Exchange::exchangeMoney(const int moneyAmount)
12     const
13 {
14     Coins coins;
15     coins.fives = moneyAmount / 5;
16     coins.twos = (moneyAmount % 5) / 2;
17     coins.ones = (moneyAmount % 5) % 2;
18     return coins;
19 }

```

queens.h

```

1 #ifndef QUEENS_H
2 #define QUEENS_H
3
4 #include <cstdlib>
5
6 struct Queen
7 {
8     int x;
9     int y;
10 };
11
12 enum Who_beat {NO_ONE = 0, EVERYONE, OneTwo_OneThree,
13               OneTwo_TwoThree, OneTwo,
14               OneThree_TwoThree, OneThree, TwoThree};
15
16 class Queens
17 {
18 public:
19     Queens();
20     ~Queens();

```



```

20     int getResult(const Queen q1, const Queen q2, const
        Queen q3) const;
21 private:
22     int checkForBeating(const Queen q1, const Queen q2)
        const;
23 };
24
25 #endif // QUEENS_H

```

queens.cpp

```

1 #include "queens.h"
2
3 Queens::Queens()
4 {
5 }
6
7 Queens::~~Queens()
8 {
9 }
10
11 int Queens::checkForBeating(const Queen q1, const Queen
    q2) const
12 {
13     return (q1.x == q2.x || q1.y == q2.y) || (abs(q1.x-q2
        .x) == abs(q1.y-q2.y));
14 }
15
16 int Queens::getResult(const Queen q1, const Queen q2,
    const Queen q3) const
17 {
18     int result = NO_ONE;
19     if (checkForBeating(q1, q2))
20     {
21         if (checkForBeating(q1, q3))
22         {
23             if (checkForBeating(q2, q3))
24             {
25                 result = EVERYONE;
26             }
27             else
28             {
29                 result = OneTwo_OneThree;
30             }
31         }
32         else
33         {
34             if (checkForBeating(q2, q3))
35             {

```

```

36         result = OneTwo_TwoThree;
37     }
38     else
39     {
40         result = OneTwo;
41     }
42 }
43 }
44 else if (checkForBeating(q1, q3))
45 {
46     if (checkForBeating(q2, q3))
47     {
48         result = OneThree_TwoThree;
49     }
50     else
51     {
52         result = OneThree;
53     }
54 }
55 else if (checkForBeating(q2, q3))
56 {
57     result = TwoThree;
58 }
59 return result;
60 }

```

longDivision.h

```

1  #ifndef LONGDIVISION_H
2  #define LONGDIVISION_H
3
4  #include <iostream>
5
6  class LongDivision
7  {
8  public:
9      LongDivision();
10     ~LongDivision();
11     void putResultToArray(char*& array, const int
        firstNumber, const int secondNumber) const;
12 private:
13     void putNumberCharByCharToArrayWithIndexation(char*&
        array, const int number, int& index) const;
14     void putNSymbolsToArrayWithIndexation(char*& array,
        const int n, const char symbol, int& index) const;
15     int numlen(int number) const;
16     int nthDigOfNumber(const int n, const int number)
        const;
17     int power(const int a, const int b) const;

```

```

18 };
19
20 #endif // LONGDIVISION_H

```

longDivision.cpp

```

1 #include "longDivision.h"
2
3 LongDivision::LongDivision()
4 {
5 }
6
7 LongDivision::~~LongDivision()
8 {
9 }
10
11 void LongDivision::putNSymbolsToArrayWithIndexation(char
    *& array, const int n, const char symbol, int& index)
    const
12 {
13     for (int i = 0; i < n; ++i)
14         array[++index] = symbol;
15 }
16
17 void LongDivision::
    putNumberCharByCharToArrayWithIndexation(char*& array,
        const int number, int& index) const
18 {
19     for (int i = 1; i <= numlen(number); ++ i)
20         array[++index] = nThDigOfNumber(i, number) + 48;
21 }
22
23 void LongDivision::putResultToArray(char*& array, const
    int firstNumber, const int secondNumber) const
24 {
25     int dividend, residue, result, product;
26     result = firstNumber / secondNumber;
27     dividend = firstNumber / power(10, numlen(result) -
        1);
28     residue = firstNumber % power(10, numlen(result) - 1)
        ;
29     int indent = dividend;
30     int crutch = 1;
31     int index = -1;
32     for (int i = 1; i <= numlen(result); ++i)
33     {
34         if (i == 1)
35         {

```

```

36         putNumberCharByCharToArrayWithIndexation(
37             array, dividend, index);
38         if (residue != 0)
39             putNumberCharByCharToArrayWithIndexation(
40                 array, residue, index);
41         putNSymbolsToArrayWithIndexation(array, 1, '|'
42             , index);
43         putNumberCharByCharToArrayWithIndexation(
44             array, secondNumber, index);
45         putNSymbolsToArrayWithIndexation(array, 1, '\
46             n', index);
47     }
48
49     product = secondNumber * nthDigOfNumber(i,
50         result);
51     putNSymbolsToArrayWithIndexation(array, numlen(
52         indent) - numlen(product), ' ', index);
53     putNumberCharByCharToArrayWithIndexation(array,
54         product, index);
55
56     if (i != 1)
57         putNSymbolsToArrayWithIndexation(array, 1, '\
58             n', index);
59
60     if (i == 1)
61     {
62         putNSymbolsToArrayWithIndexation(array,
63             numlen(firstNumber) - numlen(dividend) +
64             1, ' ', index);
65
66         putNumberCharByCharToArrayWithIndexation(
67             array, result, index);
68         putNSymbolsToArrayWithIndexation(array, 1, '\
69             n', index);
70     }
71
72     int numberOfAdditionalSpaces;
73
74     if (i != 1)
75         putNSymbolsToArrayWithIndexation(array,
76             numberOfAdditionalSpaces, ' ', index);
77
78     if (crutch == 0)
79         putNSymbolsToArrayWithIndexation(array,
80             numlen(dividend) + 1, '-', index);
81     else
82         putNSymbolsToArrayWithIndexation(array,
83             numlen(dividend), '-', index);

```

```

69         putNSymbolsToArrayWithIndexation(array, 1, '\n',
70             index);
71         numberOfAdditionalSpaces = numlen(indent) -
72             numlen(dividend - product);
73         putNSymbolsToArrayWithIndexation(array,
74             numberOfAdditionalSpaces, ' ', index);
75         putNumberCharByCharToArrayWithIndexation(array,
76             dividend - product, index);
77         if (i != numlen(result))
78         {
79             putNumberCharByCharToArrayWithIndexation(
80                 array, nThDigOfNumber(i, residue), index);
81             putNSymbolsToArrayWithIndexation(array, 1, '\n', index);
82         }
83         crutch = dividend - product;
84         dividend = (dividend - product) * 10 +
85             nThDigOfNumber(i, residue);
86         indent *= 10;
87         indent += nThDigOfNumber(i, residue);
88     }
89 }
90
91 int LongDivision::numlen(int number) const
92 {
93     int count;
94
95     if (number)
96     {
97         count = 0;
98         while (number)
99         {
100             ++count;
101             number /= 10;
102         }
103     }
104     else
105     {
106         count = 1;
107     }
108     return count;
109 }
110

```

```

111 int LongDivision::power(const int a, const int b) const
112 {
113     int result = 1;
114     for (int i = 0; i < b; ++i)
115     {
116         result *= a;
117     }
118     return result;
119 }
120
121 int LongDivision::nthDigOfNumber(const int n, const int
    number) const
122 {
123     return (number / power(10, numlen(number) - n)) % 10;
124 }

```

matrix.h

```

1 #ifndef MATRIX_H
2 #define MATRIX_H
3
4
5 class Matrix
6 {
7 public:
8     Matrix();
9     ~Matrix();
10    void sortNullsToTheMainDiagonal(int**& P, const int
        size) const;
11 private:
12    int levelOfNull(int**& P, const int size, const int
        numberOfColumn) const;
13 };
14
15 #endif // MATRIX_H

```

matrix.cpp

```

1 #include "matrix.h"
2
3 Matrix::Matrix()
4 {
5 }
6
7 Matrix::~~Matrix()
8 {
9 }
10 int Matrix::levelOfNull(int**& P, const int size, const
    int numberOfLine) const

```

```

11 {
12     int result = 0;
13     for (int i = 0; i < size; ++i)
14         if (P[numberOfLine][i] == 0)
15             {
16                 result = i;
17                 break;
18             }
19     return result;
20 }
21
22
23 void Matrix::sortNullsToTheMainDiagonal(int**& P, const
    int size) const
24 {
25     int* t;
26     int i, j;
27     for (i = size - 1; i >= 0; --i)
28         for (j = 0; j < i; ++j)
29             if (levelOfNull(P, size, j) > levelOfNull(P,
                size, j+1)){
30                 t = P[j];
31                 P[j] = P[j+1];
32                 P[j+1] = t;
33             }
34 }

```

text.h

```

1 #ifndef TEXT_H
2 #define TEXT_H
3
4 #include <iostream>
5
6 using namespace std;
7
8 class Text
9 {
10 public:
11     Text();
12     ~Text();
13     void symmetrizeLine(string& finalLine, string&
        initialLine, int maxLengthOfLine);
14 };
15
16 #endif // TEXT_H

```

text.cpp

```

1 #include "text.h"
2
3 Text::Text()
4 {
5
6 }
7
8 Text::~~Text()
9 {
10
11 }
12
13 void Text::symmetrizeLine(string& finalLine, string&
    initialLine, int maxLengthOfLine)
14 {
15     int leftIndent = (maxLengthOfLine - initialLine.
        size()) / 2;
16     finalLine.append(leftIndent, ' ');
17     finalLine += initialLine;
18 }

```

matrixapp.h

```

1 #ifndef MATRIXAPP_H
2 #define MATRIXAPP_H
3
4 void matrixApp();
5
6 #endif // MATRIXAPP_H

```

matrixapp.cpp

```

1 #include <fstream>
2 #include <iostream>
3
4 #include "matrix.h"
5
6 using namespace std;
7
8 void matrixApp()
9 {
10     ifstream fileInput;
11     fileInput.open("matrix.in");
12
13     int matrixSize;
14     fileInput >> matrixSize;
15
16     int** matrix;

```



```

17     matrix = new int* [matrixSize];
18     for (int i = 0; i < matrixSize; ++i)
19     {
20         matrix[i] = new int[matrixSize];
21     }
22     for (int i = 0; i < matrixSize; ++i)
23     {
24         for (int j = 0; j < matrixSize; ++j)
25         {
26             fileInput >> matrix[i][j];
27         }
28     }
29
30     fileInput.close();
31
32     Matrix app;
33     app.sortNullsToTheMainDiagonal(matrix, matrixSize);
34
35     ofstream fileOutput;
36     fileOutput.open("matrix.out");
37     for (int i = 0; i < matrixSize; ++i)
38     {
39         for (int j = 0; j < matrixSize; ++j)
40         {
41             fileOutput << matrix[i][j] << " ";
42         }
43         fileOutput << endl;
44     }
45
46     fileOutput.close();
47
48     for (int i = 0; i < matrixSize; ++i)
49     {
50         delete[] matrix[i];
51     }
52     delete[] matrix;
53 }

```

main.cpp

```

1 #include <iostream>
2 #include <cstdlib>
3
4 #include "tableApp.h"
5 #include "matrixapp.h"
6
7 using namespace std;
8
9 int main()

```

```

10| {
11|     char key;
12|     do
13|     {
14|         system("clear");
15|         cout << "1) Таблица-ключ-значение\n"
16|                "2) Сортировка нулей матрицы на главную диагональ\n"
17|                "3) Выход\n";
18|         cin >> key;
19|     }
20|     while (key < '1' || key > '3');
21|
22|     switch (key)
23|     {
24|         case '1':
25|         {
26|             TableApp app;
27|             app.menu();
28|             break;
29|         }
30|         case '2':
31|             matrixApp();
32|             break;
33|         case '3':
34|             exit(0);
35|             break;
36|     }
37|     return 0;
38| }

```

testcpp.cpp

```

1| #include <QString>
2| #include <QtTest>
3| #include <iostream>
4| #include <new>
5| #include <fstream>
6|
7|
8| #include "exchange.h"
9| #include "queens.h"
10| #include "longDivision.h"
11| #include "matrix.h"
12| #include "text.h"
13| #include "table.h"
14|
15| class TestCpp : public QObject
16| {

```

```

17     Q_OBJECT
18
19 public:
20     TestCpp();
21
22 private Q_SLOTS:
23     void testCase1();
24     void testExchange();
25     void testQueens();
26     void testLongDivision();
27     void testMatrix();
28     void testText();
29     void testTable();
30 };
31
32 TestCpp::TestCpp()
33 {
34 }
35
36 void TestCpp::testCase1()
37 {
38 }
39
40 void TestCpp::testExchange()
41 {
42     Exchange exchange;
43     Coins coins;
44
45     coins = exchange.exchangeMoney(12);
46     QCOMPARE(coins.fives, 2);
47     QCOMPARE(coins.twos, 1);
48     QCOMPARE(coins.ones, 0);
49
50     coins = exchange.exchangeMoney(98);
51     QCOMPARE(coins.fives, 19);
52     QCOMPARE(coins.twos, 1);
53     QCOMPARE(coins.ones, 1);
54
55     coins = exchange.exchangeMoney(99);
56     QCOMPARE(coins.fives, 19);
57     QCOMPARE(coins.twos, 2);
58     QCOMPARE(coins.ones, 0);
59
60     coins = exchange.exchangeMoney(6);
61     QCOMPARE(coins.fives, 1);
62     QCOMPARE(coins.twos, 0);
63     QCOMPARE(coins.ones, 1);
64 }
65

```

```

66 void TestCpp::testQueens()
67 {
68     Queens queens;
69     Queen q1, q2, q3;
70
71     q1 = {1, 2};
72     q2 = {3, 4};
73     q3 = {5, 6};
74     QCOMPARE(queens.getResult(q1, q2, q3), (int) EVERYONE
75             );
76
77     q1 = {1, 2};
78     q2 = {2, 4};
79     q3 = {5, 5};
80     QCOMPARE(queens.getResult(q1, q2, q3), (int) NO_ONE);
81
82     q1 = {1, 8};
83     q2 = {1, 4};
84     q3 = {8, 1};
85     QCOMPARE(queens.getResult(q1, q2, q3), (int)
86             OneTwo_OneThree);
87 }
88
89 void TestCpp::testLongDivision()
90 {
91     LongDivision longDivision;
92     const char* expected;
93     char* actual;
94
95     actual = new char[37];
96     longDivision.putResultToArray(actual, 128, 2);
97     expected = "128|2\n12 64\n--\n 08\n 8\n --\n 0";
98     QCOMPARE((std::string) actual == (std::string)
99             expected, true);
100     delete[] actual;
101 }
102
103 void TestCpp::testMatrix()
104 {
105     Matrix matrix;
106     int** actual = new int* [5];
107     for (int i = 0; i < 5; ++i)
108     {
109         actual[i] = new int[5];
110         for (int j = 0; j < 5; ++j)
111         {
112             actual[i][j] = i * j + 1;
113         }
114     }

```

```

112
113     actual[0][3] = 0; actual[1][1] = 0; actual[2][4] = 0;
        actual[3][0] = 0; actual[4][2] = 0;
114
115     matrix.sortNullsToTheMainDiagonal(actual, 5);
116
117     int expected[5][5] = {{0, 4, 7, 10, 13},
118                           {1, 0, 3, 4, 5},
119                           {1, 5, 0, 13, 17},
120                           {1, 1, 1, 0, 1},
121                           {1, 3, 5, 7, 0}};
122
123
124     for (int i = 0; i < 5; ++i)
125     {
126         for(int j = 0; j < 5; j++)
127         {
128             QCOMPARE(actual[i][j], expected[i][j]);
129         }
130         delete[] actual[i];
131     }
132
133     delete[] actual;
134 }
135
136 void TestCpp::testText()
137 {
138     Text text;
139     string finalLine;
140     string initialLine = "hfsdfsg deq";
141     text.symmetrizeLine(finalLine, initialLine, 25);
142     string expected = "          hfsdfsg deq";
143     QCOMPARE(finalLine == expected, 1);
144 }
145
146 void TestCpp::testTable()
147 {
148     Table table;
149     table.put("hello world!", 128);
150     table.put("dfsdg", 256);
151     QCOMPARE(table[128] == "hello world!", 1);
152     QCOMPARE(table.getLastElement() == "dfsdg", 1);
153     QCOMPARE(table.getKeyOfLastElement() == 256, 1);
154 }
155
156 QTEST_APPLESS_MAIN(TestCpp)
157
158 #include "testcpp.moc"

```