Программирование

А. Ю. Ламтев

13 декабря 2015 г.

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 x 5р + 0 x 2р + 1 x 1р).

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 *cppcheck***

   Утилита *cppcheck* 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 *cppcheck*.

Листинги

exchange.h

```h
#ifndef EXCHANGE_H
#define EXCHANGE_H

#ifdef __cplusplus
extern "C" {
#endif

struct purse
{
     int ones;
     int twos;
     int fives;
};

struct purse change_by_coins(int amount);

#ifdef __cplusplus
}
#endif

#endif // EXCHANGE_H
```

exchange_of_coins_process.c

```c
#include "exchange.h"

struct purse change_by_coins(int amount)
{
    struct purse coins;
    coins.fives = amount / 5;
    coins.twos = (amount % 5) / 2;
```

```
    coins.ones = (amount % 5) % 2;
    return coins;
}
```

exchange_ui.c

```
#include <stdio.h>
#include <stdlib.h>

#include "exchange.h"
#include "main.h"

void exchange_output(struct purse coins)
{
    printf("Пятирублёвых монет: %i\n"
           "Двухрублёвых монет: %i\n"
           "Рублёвых монет: %i\n",
           coins.fives, coins.twos, coins.ones);
}

void exchange(void)
{
    int number;
    struct purse coins;

    do
    {
        puts("Сколько рублей нужно разменять?");
        scanf("%i", &number);
    }
    while (number >= 100);

    coins = change_by_coins(number);
    exchange_output(coins);
}

void exchange_parameters(int argc, char** argv)
{
    switch (argc)
    {
        case 2:
            exchange();
            break;
        case 3:
        {
            int num = atoi(argv[2]);
            struct purse coins = change_by_coins(num);
            exchange_output(coins);
            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 горизонатали, а значит у них есть общая соответственная координата, или когда они находятся на одной диагоняли, 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 резултат. 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,
       OneTwo_TwoThree, OneTwo,
17                 OneThree_TwoThree, OneThree, TwoThree};
18
19 int queens_result(struct queen q1, struct queen q2,
       struct queen q3);
20
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
          .x) == abs(q1.y-q2.y));
8 }
```

queens_result_for_output.c

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

```
            if (check_for_beating(q2, q3))
            {
                result = OneTwo_TwoThree;
            }
            else
            {
                result = OneTwo;
            }
        }
    }
    else if (check_for_beating(q1, q3))
    {
        if (check_for_beating(q2, q3))
        {
            result = OneThree_TwoThree;
        }
        else
        {
            result = OneThree;
        }
    }
    else if (check_for_beating(q2, q3))
    {
        result = TwoThree;
    }
    return result;
}
```

queens_ui.c

```
#include <stdio.h>
#include <stdlib.h>

#include "main.h"
#include "queens.h"

void queens(void)
{
    struct queen q1, q2, q3;

    // Считать координаты шахматной доски координатами ма
       трицы 8x8 от 1 до 8 !!!

    input_with_check(&q1.x, &q1.y, 1);
    input_with_check(&q2.x, &q2.y, 2);
    input_with_check(&q3.x, &q3.y, 3);

    display_result(queens_result(q1, q2, q3));
}
```

```
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. Возвращат 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 аргуметом, 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 аргуметом, 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" {
```

```
#endif

void put_number_char_by_char_to_array_with_counter(char* array, int number, int *index);
void put_n_symbols_to_array_with_counter(char* array, int n, char symbol, int *index);
void put_result_to_array(char* array, int first_number, int second_number);
int numlen(int num);
int n_th_dig_of_num(int n, int num);
int power(int a, int b);

#ifdef __cplusplus
}
#endif

#endif // QUOTIENT_H
```

quotient_process.c

```
#include <stdio.h>
#include <stdlib.h>

#include "quotient.h"

void put_n_symbols_to_array_with_counter(char* array, int n, char symbol, int *index)
{
    for (int i = 0; i < n; ++i)
        array[++*index] = symbol;
}

void put_number_char_by_char_to_array_with_counter(char* array, int number, int *index)
{
    for (int i = 1; i <= numlen(number); ++ i)
        array[++*index] = n_th_dig_of_num(i, number) + 48;
}

void put_result_to_array(char* array, int first_number, int second_number)
{
    int dividend, residue, result, product;
    result = first_number / second_number;
    dividend = first_number / power(10, numlen(result) - 1);
    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(
                    char));
34
35              put_result_to_array(buffer, m, n);
36
37              puts(buffer);
38
39              free(buffer);
40              break;
41          }
42          default:
43              put_error;
44              help_quotient();
45              break;
46      }
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

```h
#ifndef MATRIX_H
#define MATRIX_H

#ifdef __cplusplus
extern "C" {
#endif

int level_of_null(int** P, int n, int number_of_column);
void sort_nulls_to_the_main_diagonal(int** P, int n);

#ifdef __cplusplus
}
#endif

#endif // MATRIX_H
```

matrix_processing.c

```c
int level_of_null(int** P, int n, int number_of_line)
{
    int result = 0;
    for (int i = 0; i < n; ++i)
        if (P[number_of_line][i] == 0)
        {
            result = i;
            break;
        }
    return result;
}


void sort_nulls_to_the_main_diagonal(int** P, int n)
{
    int* t;
    int i, j;
    for (i = n-1; i >= 0; --i)
        for (j = 0; j < i; ++j)
            if (level_of_null(P, n, j) > level_of_null(P,
                n, j+1))
```

```
            {
                    t = P[j];
                    P[j] = P[j+1];
                    P[j+1] = t;
            }
}
```

matrix_ui.c

```
#include <stdlib.h>
#include <stdio.h>

#include "matrix.h"
#include "main.h"

void matrix(char* input_file_name, char* output_file_name
    )
{
    FILE* in;
    FILE* out;
    in = fopen(input_file_name, "r");
    out = fopen(output_file_name, "w");
    int** P;
    int n, i, j;
    fscanf(in, "%i", &n);

    P = (int**) malloc(n * sizeof(int*));
    for (i = 0; i < n; ++i)
        P[i] = (int*) malloc(n * sizeof(int));

    for (i = 0; i < n; ++i)
        for (j = 0; j < n; ++j)
            fscanf(in, "%i\n", &P[i][j]);

    sort_nulls_to_the_main_diagonal(P, n);

    for (i = 0; i < n; ++i)
    {
        for (j = 0; j < n; ++j)
            fprintf(out, "%i ", P[i][j]);
        fprintf(out, "\n");
    }

    for (i = 0; i < n; ++i)
        free(P[i]);
    free(P);
    fclose(in);
    fclose(out);
    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 }
```