Программирование

А. Ю. Ламтев

12 декабря 2015 г.

Глава 1

Основные конструкции языка

1.1 Задание 1

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 четыре функции, 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***.

- ***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. **Модульное тестирование Qt**

   I тест :

   Входные данные: 28
   Выходные данные: 5 2 1
   Результат: Тест успешно пройден

   II тест :

   Входные данные: 44
   Выходные данные: 8 0 2
   Результат: Тест успешно пройден

2. **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

```
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  #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  }
```

1.2 Задание 2

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
textbftextitif 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 шесть 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***.

- ***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 с помощью 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. **Модульное тестирование 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

2. **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. Укрепил навыки в создании структурных типов. Впервые использовал перечисления enum, что оказалось очень удобно. А также научился тестировать программу с помощью модульных тестов и анализировать с помощью утилиты **cppcheck**.

Листинги

```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 матрицы 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));
}

void queens_parameters(int argc, char** argv)
{
    switch (argc)
    {
        case 2:
            queens();
            break;
        case 8:
        {
            struct queen q1, q2, q3;
            q1.x = atoi(argv[2]);
            q1.y = atoi(argv[3]);
            q2.x = atoi(argv[4]);
            q2.y= atoi(argv[5]);
            q3.x = atoi(argv[6]);
            q3.y = atoi(argv[7]);

            display_result(queens_result(q1, q2, q3));
            break;
        }
        default:
            put_error;
            help_queens();
            break;
    }
}

```

```
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 ферзи
                   бьют друг друга\n");
69             break;
70         case OneTwo_TwoThree:
71             printf("1 и 2 ферзи бьют друг друга\n2 и 3 ферзи
                   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 ферзи
                   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 }
```

```
1 #include <stdlib.h>
2
3 #include "queens.h"
4
5 int check_for_beating(struct queen q1, struct queen q2)
```

```
{
    return (q1.x == q2.x || q1.y == q2.y) || (abs(q1.x-q2.x)
       == abs(q1.y-q2.y));
}
```

```
#include "queens.h"

int queens_result(struct queen q1, struct queen q2, struct
   queen q3)
{
    int result = no_one;
    if (check_for_beating(q1, q2))
    {
        if (check_for_beating(q1, q3))
        {
            if (check_for_beating(q2, q3))
            {
                result = everyone;
            }
            else
            {
                result = OneTwo_OneThree;
            }
        }
        else
        {
            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))
    {
```

```
44          result = TwoThree;
45      }
46      return result;
47  }
```

Глава 2

Циклы

2.1 Задание 1

2.1.1 Задание

2.1.2 Теоритические сведения

2.1.3 Проектирование

2.1.4 Описание тестового стенда и методики тестирования

2.1.5 Тестовый план и результаты тестирования

2.1.6 Выводы