Программирование

Корсков А. В.

22 декабря 2015 г.

Глава 1

Основные конструкции языка

1.1 Задание 1

1.1.1 Задание

В морской миле 2000 ярдов или 6000 футов. Задано некоторое расстояние в футах, например, 9139 футов. Вывести то же расстояние в милях, ярдах и футах, например, 9139 = 1 миля 1046 ярдов и 1 фут.

1.1.2 Теоретические сведения

Для реализации данного алгоритмы были использованы функции библеотеки stdio.h для ввода и вывода информации. Также для передачи значений из функции мы использовали указатели. В ходе доработки было принято решение использовать структуры.

Морская миля — единица измерения расстояния, применяемая в мореплавании и авиации. Ярд (англ. yard) — британская и американская единица измерения расстояния. Фут — единица измерения длины в английской системе мер. Мы знаем, что в 1 миле содержится 2000 ярдов или 6000 футов. А в 1 ярде содержится 3 фута.

1.1.3 Проектирование

В функции main вызывается функция convert_UI. Взаимодействие с пользователем реализовано в фунции convert_UI. Она считывает ввёдные пользователем значения из консоли, вызывает функцию convert, а после выполнения функции convert, выводит результат в консоль. Для реализации алгоритма мы выделили функцию convert. Функция принимает один указатель на структуру. Сначала в функции вычисляет-

ся колличество миль. Далее находится колличество футов без миль. На следующем шаге вычисляется колличество ярдов. И на последнем шаге находится колличество футов. Функция ничего не возвращает, так как для передачи значения мы пользуемся указателями.

1.1.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator. Пользователь может сам ввести его данные или воспользоваться автоматическими тестами, запустив их отдельно.

1.1.5 Тестовый план и результаты тестирования

В программе предусмотрены тесты, которые проверяют правильность исполнения программы. В тесте вызывается функция, ей передаётся колличество футов, равное 9139. Далее каждая из трёх переменных проверяется на соответствие с ожидаемым правильным результатом. При вызове теста ошибок не обнаружено.

1.1.6 Выводы

В ходе работы мы научились структурировать проект, разбивая задачи на мелкие подзадачи. Получили опыт создания тестов. Также мы научились работать с системой контроля версий и приобрели навыки работы с указателями. Также получили опыт работы со структурами.

Листинги

```
1  #include <stdio.h>
2  #include "convert_UI.h"
3  #include "check_UI.h"
4  #include "removal_UI.h"
5  #include "square_UI.h"
6  #include "removing_words_UI.h"
7
8  int main(void)
9  {
10     puts("1. Перевод футов в мили, ярды и футы.");
11     puts("2. Проверка допустимости треугольника.");
12     puts("3. Удаление из числа нечётных чисел.");
```

```
13      puts("4. Проверить является ли двумерный массив латинским
            квадратом.");
14      puts("5. Удаление повторяющихся слов.");
15      int choice;
16      scanf("%d", &choice);
17      switch(choice){
18          case 1:
19              convert_UI();
20          break;
21          case 2:
22              check_UI();
23          break;
24          case 3:
25              removal_UI();
26          break;
27          case 4:
28              square_UI();
29          case 5:
30              removing_words_UI();
31      }
32      return 0;
33  }
```

```
1   #include <stdio.h>
2   #include "convert.h"
3
4   void convert_UI(){
5       puts("Введите количество футов.");
6       struct measure_units st1;
7       scanf("%d", &st1.ft);
8       convert(&st1);
9       printf("Колличество миль: %d, колличество ярдов: %d колли
            чество футов: %d \n", st1.m, st1.yd, st1.ft);
10  }
```

```
1   #include <stdio.h>
2   #include"convert.h"
3
4   void convert(struct measure_units *arg){
5       arg->m = arg->ft / 6000;
6       arg->ft = arg->ft - arg->m * 6000;
7       arg->yd = arg->ft / 3;
8       arg->ft = arg->ft - arg->yd * 3;
9   }
```

1.2 Задание 2

1.2.1 Задание

Заданы три целых числа: a, b, c. Определить, могут ли они быть длинами сторон треугольника, и если да, определить, является ли он равнобедренным либо равносторонним.

1.2.2 Теоритические сведения

Треугольник существует только тогда, когда сумма любых двух его сторон больше третьей. У равнобедренного треугольника две стороны равны. У равностороннего все стороны равны. Для реализации данного алгоритмы были использованы функции библеотеки stdio.h для ввода и вывода информации. Также использовалась конструкция if...else.

1.2.3 Проектирование

В функции main вызывается функция check_UI. Взаимодействие с пользователем реализовано в фунции check_UI. Она считывает ввёдные пользователем значения из консоли, вызывает функцию check, а после выполнения функции check, выводит результат в консоль. Для реализации алгоритма мы выделили функцию check. Функция принимает три переменные типа int. Каждая из этих трёх переменных задаёт одну из сторон треугольника. Сначала входные данные проверяются на корректность. Далее проверяем возможен ли данный треугольник или нет. Затем смотрим является ли данный треугольник равнобедренним. Если это условие выполняется, проверяем будет ли данный треугольник равносторонним.

1.2.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator. Пользователь может сам ввести его данные или воспользоваться автоматическими тестами, запустив их отдельно.

1.2.5 Тестовый план и результаты тестирования

В программе предусмотрены тесты, которые проверяют правильность исполнения программы. В тесте вызывается функция, ей передаётся три

стороны треугольника, которые все равны 3. Далее полученное число сверяется с ожидаемым правильным ответом. При вызове теста ошибок не обнаружено.

1.2.6 Выводы

В ходе работы мы научились работать с конструкцией if...else. Получили опыт создания вложенных операторов if...else.

```c
#include <stdio.h>
#include "check.h"

void check_UI(){
    puts("Введите три стороны треугольника.");
    int a,b,c,result_check;
    scanf("%d%d%d", &a, &b, &c);
    result_check = check(a,b,c);
    switch(result_check){
        case 0:
            printf("Данные некорректны.\n");
        break;
        case 1:
            printf("Данный треугольник возможен.\n");
        break;
        case 2:
            printf("Данный треугольник является равнобедренны
               м.\n");
        break;
        case 3:
            printf("Данный треугольник является равносторонни
               м.\n");
        break;
        case 4:
            printf("Данный треугольник не возможен.\n");
    }
}
```

```c
#include <stdio.h>
#include "check.h"

int check(int a, int b, int c){
    enum types_of_triangles {incorrectly, possible, isosceles
       , equilateral, impossible};
    if (a <= 0 || b <= 0 || c <= 0)
        return incorrectly;
    else{
        if (a + b > c && b + c > a && a + c > b){
            if (a == b && b == c){
```

```
                return equilateral;
            }
            if ((a == b) || (a == c) || (b == c)){
                return isosceles;
            }
            return possible;
        }
        else{
            return impossible;
        }
    }
}
```

Глава 2

Циклы

2.1 Задание 1

2.1.1 Задание

Найти число, полученное из данного выбрасыванием нечётных цифр.

2.1.2 Теоритические сведения

Для решения данной задачи ипользовалась конструкция if..else. А также цикл с предусловием while и функции библеотеки stdio.h для ввода и вывода информации.

2.1.3 Проектирование

В функции main вызывается функция removal_UI. Взаимодействие с пользователем реализовано в фунции removal_UI. Она считывает ввёдные пользователем значения из консоли, вызывает функцию removal, а после выполнения функции removal, выводит результат в консоль. Для реализации алгоритма мы выделили функцию removal. Функция принимает одно значение типа int. Это значение задаёт число, в котором нужно удалить цифры. Далее пока число не будет равняться 0 оно будет делиться на 10. От числа отрезается последняя цифра и если она чётная записать её в результат.

2.1.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator. Пользователь может сам ввести его данные или воспользоваться автоматическими тестами, запустив их отдельно.

2.1.5 Тестовый план и результаты тестирования

В программе предусмотрены тесты, которые проверяют правильность исполнения программы. В тесте вызывается функция, ей передаётся число равное 2597. Далее полученное число сверяется с ожидаемым правильным ответом. При вызове теста ошибок не обнаружено.

2.1.6 Выводы

В этом задании мы научились работать с циклом while.

```c
#include <stdio.h>
#include "removal.h"

void removal_UI(){
    puts("Введите число.");
    int number,result_removal;
    scanf("%d", &number);
    result_removal = removal(number);
    printf("Результат: %d \n", result_removal);
}
```

```c
#include<stdio.h>
#include"removal.h"

int removal(int number){
    int numeral, result = 0, digit = 1;
    while(number > 0){
        numeral = number % 10;
        number = number / 10;
        if (numeral % 2 == 0){
            result = result + digit * numeral;
            digit = digit * 10;
        }
    }
    return result;
}
```

Глава 3

Матрицы

3.1 Задание 1

3.1.1 Задание

Латинским квадратом порядка n называется квадратная таблица размером n x n, каждая строка и каждый столбец которой содержат все числа от 1 до n. Проверить, является ли заданная целочисленная матрица латинским квадратом.

3.1.2 Теоритические сведения

Двумерный массив является латинским квадратом, если во всех его строках и столбцах есть все цифры от 1 до n.

Для решения данной задачи ипользовалась конструкция if..else, цикл for функции библеотеки stdio.h для ввода и вывода информации. Также в задаче понадобилось использовать двумерный массив и выделять для него память.

3.1.3 Проектирование

В функции main вызывается функция square_UI. Взаимодействие с пользователем реализовано в фунции square_UI. Она считывает ввёдные пользователем значения из файла, вызывает функцию square, а после выполнения функции square, выводит результат в файл. Для реализации алгоритма мы выделили функцию square. Функция принимает одно значение типа int и указатель на двумерный массив. Далее мы проходим по всем столбцам и строкам и ищем все цифры от 1 до n, которое мы

передали функции square. Если какой-либо цифры нет мы выходим из функции с помощью переменной flag.

3.1.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator. Пользователь может сам ввести его данные или воспользоваться автоматическими тестами, запустив их отдельно.

3.1.5 Тестовый план и результаты тестирования

В программе предусмотрены тесты, которые проверяют правильность исполнения программы. В тесте вызывается функция, ей передаётся двумерный массив размером 2x2. Далее полученный результат сверяется с ожидаемым правильным ответом. При вызове теста ошибок не обнаружено.

3.1.6 Выводы

В этом задании мы научились создавать двумерный массив, выделять под него память, а также манипулировать с его содержимым.

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

void square_UI(){
    int n,result_square;
    FILE * iFile;
    iFile = fopen("input", "r");
    fscanf(iFile, "%d", &n);
    int** two_dim;
    two_dim = (int **) malloc(sizeof(int*) * n);
    int i,j;
    for (i = 0; i < n; i++){
        two_dim[i] = (int *) malloc(sizeof(int) * n);
    }
    for (i = 0; i < n; i++){
        for (j = 0; j < n; j++){
            fscanf(iFile, "%d", &two_dim[i][j]);
        }
    }
    fclose(iFile);
```

```
    result_square = square(two_dim,n);
    for (i = 0; i < n; i++){
        free(two_dim[i]);
    }
    free(two_dim);
    FILE * oFile;
    oFile = fopen("output", "w");
    if (result_square == 1){
        fprintf(oFile, "Матрица является латинским квадратом.
            ");
    }
    else{
        fprintf(oFile, "Матрица не является латинским квадрат
            ом.");
    }
    fclose(oFile);
}
```

```
int square(int** two_dim, int n){
    int i,j,flag,l;
    for (i = 0; i < n; i++){
        for (j = 1; j < n+1; j++){
            flag = 0;
            for (l = 0; l < n; l++){
                if (two_dim[i][l] == j){
                    flag = 1;
                }
            }
            if (flag == 0){
                return flag;
            }
        }
    }
    for (i = 0; i < n; i++){
        for (j = 1; j < n+1; j++){
            flag = 0;
            for (l = 0; l < n; l++){
                if (two_dim[l][i] == j){
                    flag = 1;
                }
            }
            if (flag == 0){
                return flag;
            }
        }
    }
    return 1;
}
```

Глава 4

Строки

4.1 Задание 1

4.1.1 Задание

Часто встречающаяся ошибка начинающих наборщиков – дважды записанное слово. Обнаружить и исправить такие ошибки в тексте.

4.1.2 Теоритические сведения

Для решения данной задачи ипользовалась конструкция if..else, циклы for и while функции библеотеки stdio.h для ввода и вывода информации. Также в задаче понадобилось использовать функции библиотеки string.h для работы со строками.

4.1.3 Проектирование

В функции main вызывается функция removing_words_UI. Взаимодействие с пользователем реализовано в фунции removing_words_UI. Она считывает ввёдные пользователем значения из файла, вызывает функцию removing_words. Для реализации алгоритма мы выделили функцию removing_words. Функция принимает одно значение типа массив char. Мы отрезаем каждое слово до пробела и сравниваем со следующим словом и если слова различны записываем данное слово.

4.1.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator.

4.1.5 Тестовый план и результаты тестирования

4.1.6 Выводы

В этом задании мы научились работать со строками.

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

void removing_words_UI(){
    FILE * iFile;
    iFile = fopen("input_removing_words", "r");
    char string[100];
    fgets(string, 100, iFile);
    fclose(iFile);
    removing_words(string);
}
```

```
#include "removing_words.h"
#include <string.h>
#include <stdlib.h>
#include <stdio.h>

void removing_words(char string[]){
    FILE * oFile;
    oFile = fopen("output_removing_words", "w");
    int n,i = 0,j,l,k,flag;
    n = strlen(string);
    i = 0;
    char word[100];
    while(i <= n){
        l=0;
        while ((string[i] != ' ') && (i < n)){
            word[l] = string[i];
            i++;
            l++;
        }
        word[l]='\0';
        j = i + 1;
        k = 0;
        flag = 1;
```

```
24          for (j=i+1;j <= i + l; j++){
25              if (word[k] != string[j]){
26                  flag = 0;
27              }
28              k++;
29          }
30          if (flag == 0){
31              fprintf(oFile, "%s ", word);
32          }
33          i++;
34      }
35      fclose(oFile);
36  }
```

Глава 5

Инкапсуляция

5.1 Задание 1

5.1.1 Задание

Реализовать класс ВЕКТОР (многомерный). Требуемые методы: конструктор, деструктор, сложение, вычитание, копирование, скалярное произведение, умножение на число, модуль.

5.1.2 Теоритические сведения

Два вектора имеют координаты x_1, y_1, x_2, y_2. В результате их сложения получится вектор с координатами $x_1 + x_2$ и $y_1 + y_2$. В результате их вычитания получится вектор с координатами $x_1 - x_2$ и $y_1 - y_2$. В результате их скалярного произведения получится число равное $x_1 * x_2 + y_1 * y_2$. В результате умножения вектора на число z получится вектор с координатами $x_1 * z$ и $y_1 * z$. В результате взятия модуля получится число равное $\sqrt{x_1 * x_1 + y_1 * y_1}$.

Для решения данной задачи мы создали специальный класс. Его методы будут задавать действия над векторами.

5.1.3 Проектирование

В функции main вызывается функция works_with_vectors. Взаимодействие с пользователем реализовано в фунции works_with_vectors. С помощью неё пользователь может производить действия над векторами. Для реализации алгоритма мы выделили класс vectors. Его методы будут задавать действия над векторами. А также будут два поля x и y задающие координаты.

5.1.4 Описание тестового стенда и методики тестирования

Для решения задачи использовалась вертуальная машина с операционной системой Debian, в которой установлен QtCreator.

5.1.5 Тестовый план и результаты тестирования

В программе предусмотрены тесты, которые проверяют правильность исполнения программы. В тесте проверяются все методы класса. Для теста сложения задаются два вектора с координатами (3,4) и (5,6). Для теста вычитания задаются два вектора с координатами (5,6) и (3,4). Для теста копирования задаются два вектора с координатами (3,4) и (5,6). Для теста скалярного произведения задаются два вектора с координатами (3,4) и (5,6). Для теста умножения вектора на число задаётся вектор с координатами (3,4) и число 10. Для теста взятия модуля задаётся вектор с координатами (3,4). Все тесты проходят успешно.

5.1.6 Выводы

В этом задании мы научились создавать классы. Задавать методы и поля класса. А также производить действия над объектами одного класса.

```
1  #include <iostream>
2  #include "vectors.h"
3
4  using namespace std;
5
6  void work_with_vectors(){
7      int choice;
8      cin >> choice;
9      int x1,x2,y1,y2;
10     vector vec1,vec2;
11     switch (choice){
12         case 1:
13             cout << "Введите координаты двух векторов." <<
                   endl;
14             cin >> x1 >> y1 >> x2 >> y2;
15             vec1.set_x(x1);
16             vec1.set_y(y1);
17             vec2.set_x(x2);
18             vec2.set_y(y2);
19             vec1.addition(vec2);
20             cout << vec1.get_x() << ' ' << vec1.get_y() <<
                   endl;
```

```
21          break;
22          case 2:
23              cout << "Введите координаты двух векторов." <<
                    endl;
24              cin >> x1 >> y1 >> x2 >> y2;
25              vec1.set_x(x1);
26              vec1.set_y(y1);
27              vec2.set_x(x2);
28              vec2.set_y(y2);
29              vec1.subtraction(vec2);
30              cout << vec1.get_x() << ' ' << vec1.get_y() <<
                    endl;
31          break;
32          case 3:
33              cout << "Введите координаты двух векторов." <<
                    endl;
34              cin >> x1 >> y1 >> x2 >> y2;
35              vec1.set_x(x1);
36              vec1.set_y(y1);
37              vec2.set_x(x2);
38              vec2.set_y(y2);
39              vec1.copy(vec2);
40              cout << vec1.get_x() << ' ' << vec1.get_y() <<
                    endl;
41          break;
42          case 4:
43              int result;
44              cout << "Введите координаты двух векторов." <<
                    endl;
45              cin >> x1 >> y1 >> x2 >> y2;
46              vec1.set_x(x1);
47              vec1.set_y(y1);
48              vec2.set_x(x2);
49              vec2.set_y(y2);
50              result = vec1.scalar_product(vec2);
51              cout << result << endl;
52          break;
53          case 5:
54              int number;
55              cout << "Введите координаты вектора." << endl;
56              cin >> x1 >> y1 >> number;
57              vec1.set_x(x1);
58              vec1.set_y(y1);
59              vec1.multiply(number);
60              cout << vec1.get_x() << ' ' << vec1.get_y() <<
                    endl;
61          break;
62          case 6:
63              double result_module;
```

```
64              cout << "Введите координаты вектора." << endl;
65              cin >> x1 >> y1;
66              vec1.set_x(x1);
67              vec1.set_y(y1);
68              result_module = vec1.module();
69              cout << result_module << endl;
70          break;
71      }
72  }
```

```
1  #include "vectors.h"
2  #include <iostream>
3  #include <math.h>
4
5  using namespace std;
6
7  vector::vector()
8  {
9      x = 0;
10     y = 0;
11 }
12
13 vector::vector(int arg_x, int arg_y){
14     x = arg_x;
15     y = arg_y;
16 }
17
18 vector::~vector(){
19
20 }
21
22 void vector::addition(vector sum){
23     x += sum.x;
24     y += sum.y;
25 }
26
27 void vector::subtraction(vector sub){
28     x -= sub.x;
29     y -= sub.y;
30 }
31
32 void vector::copy(vector arg){
33     x = arg.x;
34     y = arg.y;
35 }
36
37 int vector::scalar_product(vector arg1){
38     return arg1.x * x + arg1.y * y;
39 }
```

```
40
41 void vector::multiply(int z){
42     x *= z;
43     y *= z;
44 }
45
46 double vector::module(){
47     return sqrt(x * x + y * y);
48 }
49
50 int vector::get_x(){
51     return x;
52 }
53 int vector::get_y(){
54     return y;
55 }
56 void vector::set_x(int xx){
57     x = xx;
58 }
59 void vector::set_y(int yy){
60     y = yy;
61 }
```