Министерство образования и науки Российской Федерации

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

федеральное государственное бюджетное образовательное учреждение

высшего образования

"Санкт-Петербургский государственный технологический институт

(технический университет)"

Кафедра САПР и У Факультет 4

Курс 2

Группа 4291

Учебная дисциплина: «Разработка программных систем»

Отчёт по контрольным работам

Вариант №4

Студент Бурдыгин И. К.

Личная подпись

Санкт-Петербург

2025

**Контрольная работа №1**

*Цель:*

Ознакомление со средой программирования, приобретение навыков разработки алгоритмов ветвления, декомпозиции задач и составления тест-планов.

*Задание:*

Даны две параболы: первая с вершиной в точке (4,8), вторая с вершиной в точке (4,4). Обе параболы пересекают ось Ox в точках (0,0) и (8,0). Прямая пересекает ось Ox в точке (8,0) и ось Oy в точке (0,8).

*Графическая модель:*

Построим фигуры и пронумеруем области. Графики с указанием номеров областей представлены на рисунке 1:

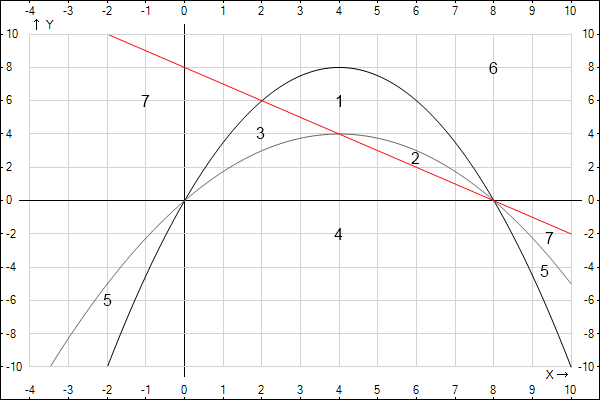


Рисунок 1 - Графики с указанием номеров областей

*Математическая модель:*

Получим математические уравнения кривых и линий, образующих фигуры на плоскости:

Таблица 1 - математические уравнения кривых и линий, образующих

|  |  |  |
| --- | --- | --- |
| 1 | первая парабола |  |
| 2 | вторая парабола |  |
| 3 | прямая |  |

Составим условия проверки попадания точки с координатами (xz , yz) в каждую из фигур, изображенных на рисунке 1:

Таблица 2 - условия проверки попадания точки с координатами (xz , yz)

|  |  |  |
| --- | --- | --- |
| 1 | внутри первой параболы |  |
| 2 | снаружи первой параболы |  |
| 3 | внутри второй параболы |  |
| 4 | снаружи второй параболы |  |
| 5 | под прямой |  |
| 6 | под прямой |  |

Составим условия проверки попадания точки в каждую из областей, изображенных на рисунке 1:

Таблица 3 - условия проверки попадания точки в каждую из областей

|  |  |  |  |
| --- | --- | --- | --- |
| № | Первая парабола | Вторая парабола | Прямая |
| 1 | внутри | снаружи | над |
| 2 | внутри | внутри | над |
| 3 | внутри | снаружи | под |
| 4 | внутри | внутри | под |
| 5 | снаружи | внутри | под |
| 6 | снаружи | снаружи | над |
| 7 | снаружи | снаружи | под |

*Тест-план:*

Для тестирования программы необходимо выбрать по точке в каждой из областей, используя следующие данные:

Таблица 4 - Координаты точек, в каждой из областей

|  |  |  |  |
| --- | --- | --- | --- |
| Тест | xz | yz | Результат |
| 1 | 4 | 6 | Область № 1 |
| 2 | 6,5 | 2 | Область № 2 |
| 3 | 2 | 4 | Область № 3 |
| 4 | 4 | -2 | Область № 4 |
| 5 | -2 | -6 | Область № 5 |
| 6 | 8 | 8 | Область № 6 |
| 7 | -1 | 6 | Область № 7 |
| 8 | ab | 1 | Incorrect y-value (повтор ввода) |
| 9 | 1 | ab | Incorrect x-value (повтор ввода) |

*Декомпозиция задач:*

Исходя из условий задания, в основной задаче Main можно выделить самостоятельную задачу CheckArea – проверка попадания точки в одную из областей на плоскости. Эту задачу можно разбить на три независимые подзадачи:

IsParabola1 – проверка попадания точки внутрь первой параболы,

IsParabola2 – проверка попадания точки внутрь второй параболы,

UnderLine – проверка попадания точки под прямую.

Дерево декомпозиции задач следующее:



Рисунок 2 - Дерево декомпозиции задач

*Блок-схемы и описание алгоритмов:*

Алгоритм IsParabola1 получает координаты точки, как исходные данные, проверяет условия 1 и 2, и возвращает True, если точка лежит внутри параболы, или False, если точка лежит снаружи параболы.

Алгоритм IsParabola2 получает координаты точки, как исходные данные, проверяет условия 3 и 4, и возвращает True, если точка лежит внутри параболы, или False, если точка лежит снаружи параболы.

Алгоритм UnderLine получает координаты точки, как исходные данные, проверяет условия 5 и 6, и возвращает True, если точка лежит под прямой, или False, если точка лежит над прямой.





Рисунок 3 - Алгоритмы IsParabola1, IsParabola2, UnderLine

Алгоритм CheckArea получает координаты точки, как исходные данные, и использует алгоритмы IsParabola1, IsParabola2, UnderLine для проверки попадания точки в каждую из фигур по отдельности и, затем, осуществляет блок последовательных проверок в соответствии с таблицей условий по областям для проверки попадания точки в каждую из областей. Область 7 определяется как исключение, если ни одно из условий не выполнилось.



Рисунок 4 - Алгоритм CheckArea

**Листинг программы:**

#include <stdio.h>

// Функция проверки попадания точки внутрь первой параболы

// Уравнение первой параболы: y = -x^2 + 8x

int IsParabola1(double x, double y) {

// Вычисляем значение функции параболы в точке x

double parabola1\_y = -x \* x + 8 \* x;

// Проверяем, находится ли точка ниже параболы

return y <= parabola1\_y;

}

// Функция проверки попадания точки внутрь второй параболы

// Уравнение второй параболы: y = -x^2 + 4x

int IsParabola2(double x, double y) {

// Вычисляем значение функции параболы в точке x

double parabola2\_y = -x \* x + 4 \* x;

// Проверяем, находится ли точка ниже параболы

return y <= parabola2\_y;

}

// Функция проверки попадания точки под прямую

// Уравнение прямой: y = -x + 8

int UnderLine(double x, double y) {

// Вычисляем значение функции прямой в точке x

double line\_y = -x + 8;

// Проверяем, находится ли точка под прямой

return y <= line\_y;

}

// Основная функция для проверки попадания точки в область

int CheckArea(double x, double y) {

// Если точка под первой параболой, но выше второй

if (IsParabola1(x, y) && !IsParabola2(x, y)) {

return 1; // Область 1

}

// Если точка между второй параболой и прямой

if (IsParabola2(x, y) && UnderLine(x, y)) {

return 2; // Область 2

}

// Если точка под первой параболой и прямой, но не под второй параболой

if (IsParabola1(x, y) && UnderLine(x, y) && !IsParabola2(x, y)) {

return 3; // Область 3

}

// Если точка под второй параболой, но не под прямой

if (IsParabola2(x, y) && !UnderLine(x, y)) {

return 4; // Область 4

}

// Если точка выше прямой и парабол

return 0; // Точка вне областей

}

int main() {

double x, y;

int continue\_program = 1;

while (continue\_program) {

// Запрос на ввод X

printf("Vvedite X: ");

scanf("%lf", &x);

// Запрос на ввод Y

printf("Vvedite Y: ");

scanf("%lf", &y);

// Проверка области

int area = CheckArea(x, y);

// Вывод номера области

if (area == 0) {

printf("Tochka ne popadaet ni v odnu oblast'.\n");

} else {

printf("Tochka nahodits'a v oblasti %d.\n", area);

}

// Запрос на продолжение

printf("Hotite proverit' druguyu tochku? (1 - Da, 0 - Viyti): ");

scanf("%d", &continue\_program);

}

printf("Programma zavershena.\n");

return 0;

}

**Контрольная работа №2**

*Цель:*

Изучение среды программирования, отладочных средств, циклических конструкций языка, ознакомление со свойствами рядов целых чисел, приобретение навыков разработки циклических алгоритмов, преобразования данных, декомпозиции задач и составления тест-планов.

*Задание:*

Числа, сумма цифр которых является пятиугольным числом и разность между соседними цифрами равна k. Например:

23 → 2+3=5 (k=1); 25852→ 2+5+8+5+2=22 (k=3), …

*Математическая модель:*

Числа степени двойки получаются по формуле: 

Последовательность чисел для n = 1, 2, … начинается так:

1, 5, 12, 22, 35, 51, 70, 92, 117, 145, …

*Тестовые планы:*

Тестирование программы предлагается разбить на три части:

1. Тестирование печати чисел в заданном диапазоне

Вызвать функцию печати чисел, обладающих свойством, для заданного диапазона и использовать следующие данные:

Таблица 5 - Тестирование печати чисел в заданном диапазоне

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Тест | От | До | K | Результат |
| 1 | 10 | 100 | 2 | 57, 75 |
| 2 | 10 | 100 | 4 | 48, 84 |
| 3 | 10 | 100 | 6 | 39, 93 |
| 4 | 10 | 100 | 8 | - |
| 5 | 100 | 10000 | 3 | 147, 363, 525, 741, 3036, 3630, 4747, 6303, 7474 |
| 6 | 100 | 10000 | 4 | 840, 1515, 5151 |
| 7 | 100 | 10000 | 5 | 949, 3838, 8383 |
| 8 | 100 | 10000 | 6 | 282, 606, 6060 |
| 9 | 10000 | 900000 | 3 | 25258, 25852, 63030, 85252 |
| 10 | 10000 | 900000 | 6 | 28282 |
| 11 | 10000 | 900000 | 8 | - |
| 12 | -500 | -10 | 4 | -84, -48 |
| 13 | 100 | 10 | 6 | 39, 93 |
| 14 | ab | 100 | 2 | Incorrect lower bound value |
| 15 | 10 | ab | 2 | Incorrect upper bound value |
| 16 | 10 | 100 | А | Incorrect digit difference value |
| 17 | 10 | 100 | 9 | Incorrect digit difference value |
| 18 | 10 | 100 | -1 | Incorrect digit difference value |

2. Тестирование последовательности ввода чисел

В качестве тестовых последовательностей использовать следующие данные (жирным выделены числа, обладающие свойством):

Таблица 6 - Тестовые последовательности

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тест | K | Последовательность чисел | | | | | | | | | | | N | M |
| 1 | 2 | 22 | 34 | 57 | 66 | 75 | 10 | 100 | 32 | 45 | 84 | 0 | 2 | 8 |
| 2 | 4 | 47 | 78 | 48 | 62 | 15 | 37 | 848 | 84 | 519 | 95 | 0 | 2 | 8 |
| 3 | 3 | 14 | 25 | 36 | 37 | 38 | 47 | 58 | 69 | 10 | 369 | 0 | 1 | 9 |
| 4 | 1 | 1 | 10 | 12 | 23 | 34 | 45 | 46 | 47 | 48 | 98 | 0 | 3 | 7 |
| 5 | 1 | а | 10 | 12 | 23 | 34 | 45 | 46 | 47 | 48 | 98 | 0 | 2 | 7 |
| 6 | 8 | 0 | | | | | | | | | | | 0 | 0 |

Тесты 1 и 2 – последовательности, содержащие числа, обладающие и не обладающие свойством. Тест 3 – последовательность, не содержащая чисел, обладающих свойством. Тест 4 – последовательность, содержащая только числа, обладающие свойством. Тест 5 – последовательность, содержащая ошибочные данные. Тест 6 – пустая последовательность (сразу два нуля).

3. Тестирование печати ряда степени двойки

Вызвать функцию печати чисел степени двойки. Результат:

*Декомпозиция задач:*

Исходя из условий задания, главную задачу Main можно разбить на две основные задачи:

CheckNumber – проверка свойств чисел,

Entry&Print – ввод данных и печать результатов.

Задачу CheckNumber можно разбить на три независимые подзадачи:

1 – является ли число пятиугольным,

2 – проверка разности между цифрами,

3 – вычисление суммы цифр.

Задачу Entry&Print можно разбить на три независимые подзадачи:

1 – печать пятиугольных чисел,

2 – печать в заданном диапазоне чисел, удовлетворяющих свойствам,

3 – определение и подсчёт чисел, удовлетворяющих свойствам, при вводе, а условием прекращения ввода является ввод двух нулей подряд.

Для CheckNumber предлагается использовать алгоритмы:

IsFive – для проверки, является ли число числом пятиугольным,

IsDifference– для проверки, равна ли разность между цифрами числа заданному значению,

SumDigits –для вычисления суммы цифр числа.

CheckNumber – для проверки, является сумма цифр числа пятиугольным числом и разность между соседними цифрами равна k.

Для решения Entry&Print предлагается использовать алгоритмы:

PrintFive – для печати ряда пятиугольных чисел,

PrintNumbers – для печати в заданном диапазоне тех чисел, которые обладают заданными свойствами,

InputNumbers – для ввода чисел с консоли, проверки их на заданные свойства, подсчёта чисел, пока не будут введены два нуля подряд.

Дерево декомпозиции задач следующее:



Рисунок 5 - Дерево декомпозиции задач

*Блок-схемы и описание алгоритмов:*



Рисунок 6 - Алгоритм IsFive

Алгоритм IsFive получает число для проверки, как исходные данные. Затем, он последовательно в цикле вычисляет следующее пятиугольное число и сравнивает его с числом, переданным как исходные данные. Если текущее пятиугольное число равно заданному числу, то цикл прекращается, и алгоритм возвращает True, как результат. Если в процессе вычислений, число степени двойки стало больше заданного числа, то вычисления можно прекратить, и алгоритм возвращает False, как результат.



Рисунок 7 - Алгоритм SumDigits

Алгоритм SumDigits получает число для проверки, как исходные данные. Затем, он последовательно в цикле разбивает число на цифры и суммирует их. Чтобы отделить очередную цифру, алгоритм использует операцию вычисления остатка от деления на цело. Чтобы получить число без отделённой цифры, алгоритм использует обычную операцию деления и отбрасывает дробную часть. Цикл выполняется до тех пор, пока в числе не останется цифр. Алгоритм возвращает сумму цифр, как результат.

Алгоритм IsDifference получает число для проверки и разность между цифрами, как исходные данные. Затем, он последовательно в цикле разбивает число на цифры, вычисляет разность между соседними цифрами и сравнивает её с заданной разностью. Чтобы отделить очередную цифру, алгоритм использует операцию вычисления остатка от деления на цело. Чтобы получить число без отделённой цифры, алгоритм использует обычную операцию деления и отбрасывает дробную часть. Цикл выполняется до тех пор, пока в числе не останется цифр. Если на очередном шаге вычислений разность оказывается больше или меньше заданной, то вычисления можно прекратить, и алгоритм возвращает False, как результат. Если в числе больше не осталось цифр, то цикл прекращается, и алгоритм возвращает True, как результат.



Рисунок 8 - Алгоритм CheckNumber

Алгоритм CheckNumber получает число для проверки и разность между цифрами, как исходные данные. Затем, он последовательно использует алгоритмы IsFive, IsDifference и SumDigits, описанные выше, для проверки, является ли сумма цифр числом степени двойки при заданной разности между соседними цифрами. Алгоритм возвращает True, если результаты всех проверок оказались True, или False, если хотя бы одна проверка не выполнилась.



Рисунок 9 - Алгоритм IsDifference

Алгоритм InputNumbers запрашивает только разность между цифрами, количество вводимых чисел ему не известно. Алгоритм организует цикл по вводу чисел с проверкой каждого нового числа до пор, пока пользователь не введет два нуля подряд. Для проверки введенного числа он использует алгоритм CheckNumber. Если введенное число удовлетворяет свойствам, то алгоритм увеличивает счётчик проверяемых чисел (N) на единицу и печатает информацию о числе, если нет, то увеличивает счётчик прочих чисел (M). Чтобы проверить ввод двух нулей подряд, алгоритм использует переменную Zero, которая получает значение True, всякий раз, когда был введен ноль, и значение False, всякий раз, когда было введено ненулевое значение. Если был введен первый ноль, т.е. Zero еще False, то алгоритм присваивать введенному числу ненулевое значение, чтобы не прекращать цикл.



Рисунок 10 - Алгоритм InputNumbers

Алгоритм PrintFive печатает все числа степени двойки, используя цикл от 1 до предельного значения используемого типа данных (long). В цикле он вычисляет значение чисел по номерам. Цикл выполняется, пока текущее вычисленное значение меньше предела и является положительным.

Алгоритм PrintNumbers запрашивает нижнее и верхнее значение диапазона чисел и разность между цифрами. Алгоритм организует цикл от нижнего значения до верхнего значения и использует в цикле алгоритмы IsFive, IsDifference и SumDigits, описанные выше, для проверки, является ли сумма цифр очередного числа числом степени двойки при заданной разности между соседними цифрами.



Рисунок 11 - Алгоритмы PrintFive и PrintNumbers

**Листинг программы:**

#include <stdio.h>

// Функция для вычисления абсолютного значения числа без использования abs

int MyAbs(int num) {

if (num < 0) {

return -num; // Если число отрицательное, делаем его положительным

}

return num; // Если число положительное или ноль, возвращаем его как есть

}

// Функция для вычисления суммы цифр числа

int SumDigits(int num) {

int sum = 0;

while (num > 0) {

sum += num % 10; // Получаем последнюю цифру числа и добавляем к сумме

num /= 10; // Убираем последнюю цифру

}

return sum;

}

// Функция для проверки, является ли число пятиугольным

// Пятиугольные числа: Pn = n \* (3n - 1) / 2, где n = 1, 2, 3, ...

int IsFive(int sum) {

for (int n = 1; ; n++) {

int pentagonal = n \* (3 \* n - 1) / 2;

if (pentagonal == sum) {

return 1; // Число пятиугольное

}

if (pentagonal > sum) {

return 0; // Число не является пятиугольным

}

}

}

// Функция для проверки, равна ли разность между соседними цифрами числу k

int IsDifference(int num, int k) {

int prev\_digit = num % 10;

num /= 10;

while (num > 0) {

int curr\_digit = num % 10;

if (MyAbs(curr\_digit - prev\_digit) != k) { // Используем MyAbs вместо abs

return 0; // Разность между соседними цифрами не равна k

}

prev\_digit = curr\_digit;

num /= 10;

}

return 1; // Разность между соседними цифрами равна k

}

// Основная функция для проверки чисел

int CheckNumber(int num, int k) {

int sum = SumDigits(num); // Сумма цифр числа

if (IsFive(sum) && IsDifference(num, k)) {

return 1; // Число удовлетворяет условиям

}

return 0; // Число не удовлетворяет условиям

}

// Функция для печати ряда пятиугольных чисел

void PrintFive(int limit) {

int n = 1;

while (1) {

int pentagonal = n \* (3 \* n - 1) / 2;

if (pentagonal > limit) break;

printf("%d ", pentagonal);

n++;

}

printf("\n");

}

// Функция для печати чисел в заданном диапазоне, обладающих заданными свойствами

void PrintNumbers(int n, int m, int k) {

for (int num = n; num <= m; num++) {

if (CheckNumber(num, k)) {

printf("%d ", num);

}

}

printf("\n");

}

// Функция для ввода чисел с консоли и подсчёта чисел, обладающих заданными свойствами

void InputNumbers(int k) {

int num, count = 0;

while (1) {

printf("Vvedite chislo (0 0 dlya togo, chtoby zavershit'): ");

scanf("%d", &num);

if (num == 0) {

int second\_num;

scanf("%d", &second\_num);

if (second\_num == 0) {

break; // Завершаем программу, если введены два нуля

}

}

if (CheckNumber(num, k)) {

count++;

}

}

printf("Obshcheye kollichestvo: %d\n", count);

}

// Функция для ввода данных и печати результатов

void EntryAndPrint() {

int choice, n, m, k;

printf("Vvedite:\n1 - Dlya vyvoda nuzhnih chisel v diapazone,\n2 - Dlya vvedeniya svoih chisel v konsoli:\n");

scanf("%d", &choice);

if (choice == 1) {

printf("Vvedite diapazon [n, m]: ");

scanf("%d %d", &n, &m);

printf("Vvedite k: ");

scanf("%d", &k);

printf("Chisla v diapazone [%d, %d], kotorye obladayut svoistvami:\n", n, m);

PrintNumbers(n, m, k);

} else if (choice == 2) {

printf("Vvedite k: ");

scanf("%d", &k);

InputNumbers(k);

}

}

int main() {

int continue\_program = 1;

while (continue\_program) {

EntryAndPrint();

// Запрос на продолжение

printf("Proverit’ drugie chisla? (1 - Da, 0 - Viyti): ");

scanf("%d", &continue\_program);

}

printf("Programma zavershena.\n");

return 0;

}

**Контрольная работа №3**

*Цель:*

Изучение приемов работы с одномерными статическими и динамическими массивами, способов передачи параметров и массивов в функции, конструкций множественного выбора. Изучение работы с файлами. Приобретение навыков декомпозиции задач, модульного проектирования программ, разработки циклических алгоритмов, алгоритмов ветвления, составления тест-планов и написания программной документации.

*Задание:*

Непрерывная последовательность чисел имеет либо минимальное, либо максимальное (должно выбираться опционально) среднеарифметическое значение элементов по абсолютному значению.

*Математическая модель:*

Для решения задачи предлагается использовать метод перебора всех возможных непрерывных последовательностей элементов массива длины 1<M<Size–1 с вычислением суммы каждой последовательности, которое сравнивается с критериальным среднеарифметическим значением элементов по абсолютному значению.

Пусть имеется массив:

right

left4

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2.1 | 0.1 | 1.2 | -4.5 | 3.1 | 8.5 | 9.3 | -9.8 | 0.2 | -1 | 8.2 | -5.4 |

min∑

max∑

Для перебора последовательностей необходимо завести два индекса Left и Right. Индекс Left обозначает начало очередной последовательности и сдвигается все время влево к концу массива. Индекс Right обозначает конец последовательности и сдвигается от индекса (Left+1) влево не более длины последовательности M. Длины последовательностей перебираются, начиная от 2 до (Size – Left). Для каждой последовательности вычисляется критерий как .

*План работы:*

Таблица 7 - План работы

|  |  |  |
| --- | --- | --- |
| № | Вид работы | Время, ч |
| 1 | Изучение предметной области, анализ интернет-публикаций,  составление технического задания | 8 |
| 2 | Определение функциональности, декомпозиция задач и проектирование  архитектуры модулей | 8 |
| 3 | Разработка и написание спецификаций (SRS, AMS, FFS) | 6 |
| 4 | Разработка алгоритмов решения задач, составление блок-схем | 20 |
| 5 | Анализ ситуаций, подбор входных данных, разработка тестовых планов | 16 |
| 6 | Кодирование алгоритмов, написание C++ кода | 24 |
| 7 | Тестирование программы в соответствии с тестовыми планами, отладка  модулей, исправление ошибок | 10 |
| 8 | Документирование: API reference, руководство пользователя,  комментарии к коду, оформление отчёта | 10 |

*Техническое задание:*

Разработать программу поиска в массиве вещественных чисел непрерывной последовательности, удовлетворяющей критерию: максимум или минимум среднеарифметического значения элементов по абсолютному значению.

Требования к функциональности:

* + консольный ввод-вывод массива чисел,
  + файловый ввод-вывод массива чисел,
  + обработка массива чисел для поиска последовательности,
  + диагностика ошибок.

Требования к интерфейсу:

* диалоговое консольное меню ко всем функциям,
* наличие подсказок при вводе данных и выборе команд,
* запрос подтверждений от пользователя в неоднозначных ситуациях,
* вывод понятной информации об ошибках.

Требования к тестированию:

* все функции программы должны иметь тестовые планы,
* программа должна хранить готовые тестовые массивы,
* отработать тестовые случаи при работе с файлами.

Требования к форматам хранения данных:

* разработать формат хранения массива в файле,
* составить спецификацию хранения данных.

Требования к архитектуре:

* программа должна строиться из четырёх модулей, каждый из которых отвечает за определённую функциональность программы,
* составить спецификацию на содержание и назначение модулей.

Требования к комплектности и поставке:

* программа должна поставляться в виде zip-архива,
* в поставку должны входить тестовые файлы,
* в поставку должно входить руководство пользователя.

Требования к техническим и программным средствам:

* PC Pentium, не менее 128М ОЗУ,
* OC Microsoft Windows XP SP1/SP2/SP3/ или выше

*Декомпозиция задач:*

Исходя из условий задания, главную задачу Main можно разбить на четыре основные задачи:

Console I/O – Организация консольного ввода-вывода массива,

File I/O – Организация файлового ввода-вывода массива,

Solve – Обработка массива в соответствии с заданием,

Interface – Организация интерфейса с пользователем.

Задачу Console I/O можно разбить на три независимые подзадачи:

ShowArray – вывод массива на консоль,

InputArray – вывод массива с консоли,

Error – вывод информации об ошибках.

Задача InputArray дополнительно требует решения задачи диагностики ошибок при вводе размера и элементов массива (InputChecking)

Задачу File I/O можно разбить на две независимые подзадачи:

SaveArray – сохранение массива в файле,

LoadArray – загрузка массива из файла.

Задача LoadArray дополнительно требует решения задачи диагностики ошибок при чтении размера и элементов массива из файла (FileChecking). Обе задачи дополнительно требуют решения задачи проверки корректности имени файла (CheckName), задачи проверки существования файла (CheckExist) и доступности файла для чтения-записи (CheckAccess).

Задачу Solve можно разбить на две независимые подзадачи:

Search – поиск непрерывной последовательности элементов массива, которая имеет минимальную или максимальную сумму элементов,

TestArray – создание тестового массива.Задача Search дополнительно требует решения задачи вычисления суммы последовательности элементов массива (Criteria).

Задача Interface можно разбить на подзадачу разработки консольного меню (MainMenu) и подзадачу обращений к пользователю по ситуации. В рамках данной работы выделяется одна ситуация – необходимость перезаписи файла при сохранении массива (IsRewrite).

Дерево декомпозиции задач следующее:



Рисунок 12 - Дерево декомпозиции задач

Таблица 8 - Спецификация требований к задачам (System Requirement Specification)

|  |  |  |  |
| --- | --- | --- | --- |
| Требования | | Задачи | |
| 1 | Консольный ввод-вывод | 1 | Вывод массива чисел на экран |
| 2 | Ввод чисел с консоли и сохранение их в массиве |
| 3 | Диагностика ошибок при вводе размера массива и  чисел |
| 4 | Вывод информации об ошибках пользователя |
| 2 | Файловый ввод-вывод | 1 | Сохранение массива в файле |
| 2 | Загрузка массива из файла |
| 3 | Диагностика ошибок работы файлом при чтении-записи данных |
| 4 | Проверка корректности имени файла |
| 5 | Обработка ситуаций существования файла |
| 3 | Обработка массива | 1 | Вычисление критерия |
| 2 | Поиск последовательности, удовлетворяющей критерию |
| 3 | Загрузка тестового массива |
| 4 | Интерфейс пользователя | 1 | Разработка интерактивного консольного меню для  выбора всех команд |

Таблица 9 - Спецификация архитектуры модулей (Architectural Module Specification)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Модуль | Тип | Функция | Назначение | SRS |
| INOUT | CPP | ShowArray | Вывод массива на консоль. Массив передается через аргументы. | 1.1 |
| InputArray | Ввод размера массива, резервирование массива, ввод чисел и сохранение их в массиве с проверкой ошибок при вводе. Новый массив передается через аргументы. | 1.2  1.3 |
| Error | Вывод информации об ошибке на консоль. | 1.4 |
| FILE | CPP | SaveArray | Сохранение массива в файле с диагностикой возникающих ошибок. Массив передается через аргументы. | 2.1  2.3 |
| LoadArray | Загрузка массива из файла с диагностикой возникающих ошибок. Корректно загруженный массив передается через аргументы. | 2.2  2.3 |
| CheckFileName | Проверка корректности имени файла. | 2.4 |
| SOLVE | CPP | Search | Поиск последовательности чисел, удовлетворяющей критерию. | 3.2 |
| Criteria | Расчёт суммы чисел от одного индекса до другого. | 3.1 |
| TestArray | Копирование тестового массива на место текущего массива | 3.3 |
| LAB3 | CPP | MainMenu | Организация диалога с пользователем в виде консольного меню. | 4.1 |
| IsRewrite | Запрос подтверждения на перезапись файла. | 2.5 |
| Main | Главная функция |  |
| TASK | H | enum errCode | Коды ошибок |  |
| Прототипы всех функций | |



Рисунок 13 - Диаграмма взаимосвязи модулей

*Спецификация формата файла (File Format Specification)*

Назначение:

Разработанный формат файла предназначен для хранения массива вещественных чисел, обрабатываемых программой LAB3.

Основные требования:

Программа использует текстовые файлы формата ASCII. Файл содержит последовательность чисел, разделенных символами пробела, табуляции или переноса строки. При записи чисел допускаются цифры 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 и знаки +, –, точка. Точка разделяет целую и дробную части числа. В файле не допускаются иные символы, кроме указанных.

Внутренняя структура:

Первое число в файле хранит количество чисел в массиве. Количество последущих чисел должно быть не меньше указанного количества. Каждое число отделяется хотя бы одним пробельным символом. Первое число присутствует обязательно и не может иметь точку или знак минус.

N Element1 Element2 … Elementi … ElementN

где N – количество чисел – размер массива

Elementi – числа – элементы массива

Пример файла:

4

1.25 3.45 –0.56 10

Таблица 10 - Диагностика ошибок формата файла

|  |  |
| --- | --- |
| Код ошибки | Описание ошибки |
| errArraySize | Если первое число – размер массива – указано не правильно. Например, если использован алфавитный символ, число отрицательно, или выходит за допустимый предел. |
| errArrayElement | Если число – элемент массива – указано не правильно. Например, если использован алфавитный символ. |
| errAbsentElements | Если количество чисел в последовательности меньше, чем размер массива, указанный первым числом. |
| errCanNotSaveValue | Если значение не может быть сохранено в файле из-за системных ошибок работы с файлом. |

Таблица 11 - Диагностика ошибок работы с файлом

|  |  |
| --- | --- |
| Код ошибки | Описание ошибки |
| errFileName | Если имя файла записано некорректно. |
| errFileExists | Если файл существует с указанным именем, а должен отсутствовать. |
| errFileNotFound | Если файл не существует с указанным именем, а должен присутствовать. |
| errCanNotOpenFile | Если файл не может быть открыт. |
| errFileIsReadOnly | Если файл предназначен только для чтения. |
| errFileIsWriteOnly | Если файл предназначен только для записи. |

*Блок-схемы и описание алгоритмов:*

Алгоритм CalcCriteria рассчитывает сумму последовательности и получает ссылку на массив, в котором находится последовательность, индекс начала и индекс конца последовательности как исходные данные. Алгоритм обходит все элементы последовательности и суммирует их в переменную sum. Алгоритм обнуляет сумму перед вычислениями. Алгоритм возвращает вычисленное значение суммы.



Рисунок 14 - Алгоритм CalcCriteria

Алгоритм Search осуществляет поиск в массиве чисел такой непрерывной последовательности элементов, которая имеет максимальную или минимальную сумму. Алгоритм получает ссылку на массив, размер массива и опцию поиска (True – максимум, False – минимум) как исходные данные. Алгоритм организует перебор последовательностей длины от 2 до (Size – 1). Для этого он использует счётчик I, который проходит элементы от первого до предпоследнего. Для каждого значения счётчика I алгоритм перебирает для него возможные длины последовательностей с помощью счётчика M. Для I-ого положения можно перебрать длины от 2 до (Size – I), так как последовательности больших длин уже будут перебраны на предыдущих шагах. Для каждой последовательности длины M, начинающейся от элемента I, алгоритм вычисляет конечный элемент как (I + M – 1) и использует алгоритм вычисления критерия CalcCriteria, описанный выше, чтобы рассчитать сумму.

Полученное значение суммы алгоритм Search сравнивает со значением критерия. В зависимости от значения опции поиска, чтобы запомнить границы последовательности, полученное значение суммы либо должно быть больше значения критерия при поиске максимума, либо должно быть меньше значения критерия при поиске максимума. Если условие критерия выполняется, то алгоритм запоминает начало последовательности как I,

конец последовательности как (I + M – 1), и значение суммы как новое значение критерия.

Перед началом вычислений алгоритм Search инициализирует критериальное значение суммы Criteria либо минимально возможным значением для используемого типа данных (double) при поиске максимума суммы, либо максимально возможным значением для используемого типа данных (double) при поиске минимума суммы в зависимости от опции поиска. Так как алгоритм использует предельное значение типа данных в качестве стартового значения критерия, то при первом же вычислении критерия в циклах алгоритма, он получит значение соответственно либо большее предельно минимального, либо меньшее предельно максимального, и первое сравнение полученного значения со стартовым значением критерия приведёт к корректировке значения критерия. Это позволяет избежать контроля номеров итераций для установки значения критерия.

Алгоритм возвращает начало последовательности и конец последовательности. Поскольку алгоритм возвращает два значения, то при практической реализации рекомендуется использовать передачу аргументов по ссылке.

Алгоритм ShowArray получает ссылку на массив и размер массива как исходные данные. Алгоритм последовательно выводит все элементы массива на консоль.

Алгоритм InputArray получает ссылку на массив, который должен быть перезаписан, и ссылку на переменную, хранящую размер массива как исходные данные. Алгоритм запрашивает размер массива, резервирует память под массив, и последовательно вводит элементы массива с консоли. Алгоритм проверяет вводимые данные. Так, размер массива не может быть отрицательным числом и должен быть в диапазоне от 1 до 1000 элементов. Чтобы не испортить текущий массив, находящийся в главной функции, алгоритм InputArray сначала запоминает ссылку и размер нового массива во временных переменных. Если в процессе ввода произойдут ошибки, то алгоритм освободит зарезервированную память, не изменяя массив в главной программе. Если ввод нового массива будет успешным, то алгоритм освободит память текущего массива, используя ссылку на него, и сделает новый массив текущим.



Рисунок 15 - Алгоритм Search

Алгоритм SaveArray получает ссылку на массив, размер массива, имя файла для сохранения, и признак перезаписи как исходные данные. Сначала алгоритм осуществляет проверку ситуаций работы с файлами. Если имя файла задано неправильно, то алгоритм возвращает ошибку errFileName.Если файл существует, но имеет атрибут Read-Only, то алгоритм возвращает ошибку errFileIsReadOnly. Если файл существует, то алгоритм использует признак перезаписи для выбора действий. Если признак перезаписи False, т.е. перезапись запрещена, то алгоритм возвращает ошибку errFileExists. Если признак перезаписи True, т.е. перезапись разрешена, об алгоритм запрашивает подтверждение на выполнение перезаписи, используя функцию IsRewrite, определенную в интерфейсе. Если файл существует, но не может быть открыт для записи, например, файл используется другим приложением, то алгоритм возвращает ошибку errCanNotOpenFile. Если ошибок работы с файлом не произошло, то алгоритм последовательно записиывает все элементы массива в файл.

Алгоритм LoadArray получает ссылку на массив, который должен быть перезаписан, и ссылку на переменную, хранящую размер массива, и имя файла для загрузки как исходные данные. Алгоритм работает также как InputArray, но дополни-тельно осуществляет проверку ситуаций работы с файлами. Если имя файла задано неправильно, то алгоритм возвращает ошибку errFileName. Если файл не существует или путь к файлу неправильный, то алгоритм возвращает ошибку errFileNotFound. Если файл существует, но не может быть открыт для чтения, то алгоритм возвращает ошибку errCanNotOpenFile. При чтении данных алгоритм проверяет корректность размера массива, соответствие размера массива и количества элементов в файле, правильность задания чисел и соответствие спецификации формата.

Алгоритм TestArray использует статические массивы для хранения тестовых массивов. Алгоритм освобождает память из под текущего массива, резервирует память под тестовый массив, копирует элементы из статического массива, и делает тестовый массив текущим. Чтобы определить размер статического массива, он использует операцию sizeof.

Алгоритм CheckFileName получает имя файла как исходные данные и проверяет его на наличие недопустимых символов <, >, |, \*, ?, “, символ двоеточия может быть только на вторым символом в строке, имя не должно содержать двух и более обратных подряд, и т.д.

*Тестовые планы:*

Тестирование программы предлагается разбить на четыре части:

1. Тестирование поиска последовательности чисел в массиве

В качестве тестового массива использовать следующие данные:

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
| 2.1 | 0.1 | 1.2 | -4.5 | 3.1 | 8.5 | 9.3 | -9.8 | 0.2 | -1 | 8.2 | -5.4 |

Результат поиска последовательности с максимальной суммой:

Left Index = 6

Right Index = 7

Результат поиска последовательности с минимальной суммой:

Left Index = 8

Right Index = 9

1. Тестирование ввода массива с консоли

Для тестирования ввода размера массива необходимо использовать следующие данные:

Таблица 12 - Данные для тестирования ввода размера массива

|  |  |  |
| --- | --- | --- |
| № | Размер | Ожидаемый результат |
| 1 | A1 | Вывод ошибки: Incorrect array size |
| 2 | 0 | Вывод ошибки: Incorrect array size |
| 3 | -10 | Вывод ошибки: Incorrect array size |
| 4 | пробелы | Повтор ввода |
| 5 | 3 | Правильный ввод. Резервирование трёх элементов. |

Для тестирования ввода элемента массива необходимо использоватьследующие данные:

Таблица 13 - Данные для тестирования ввода элемента массива

|  |  |  |
| --- | --- | --- |
| № | Элемент | Ожидаемый результат |
| 1 | A1 | Вывод ошибки: Incorrect value |
| 2 | пробелы | Повтор ввода |
| 3 | 3.2 | Ввод числа 3.2 |
| 4 | –4.89 | Ввод числа –4.89 |
| 5 | 1.2.3 | Ввод числа 1.2 |
| 6 | 00091 | Ввод числа 91 |
| 7 | 1.20000 | Ввод числа 1.2 |

Для тестирования функций ввода, отображения и создания тестового массива выполнить следующие действия:

1. Вызвать функцию ввода массива и ввести пять элементов:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 00004 | –.1 | 9.4 | 3.1.5 | 1.2000 |

2) Вызвать функцию отображения массива. Результат на экране:

4 –0.1 9.4 3.1 1.2

1. Вызвать функцию ввода массива и ввести три элемента:

|  |  |  |
| --- | --- | --- |
| 14 | 35 | –1.4 |

4) Вызвать функцию отображения массива. Результат на экране:

14 35 –1.4

5) Вызвать функцию ввода массива и ввести некорректные данные.

6) Вызвать функцию отображения массива. Результат на экране:

14 35 –1.4

7) Вызвать функцию создания тестового массива.

6) Вызвать функцию отображения массива. Результат на экране:

2.1 0.1 1.2 –4.5 3.1 8.5 9.3 –9.8 0.2 –1.0 8.2 –5.4

1. Тестирование загрузки массива из файла

Создать тестовые файлов со следующим содержанием:

|  |  |  |
| --- | --- | --- |
| Файл | Содержание | Ожидаемый результат |
| 1.txt | u3 1.1 –2.3 4.0 | Вывод ошибки: Incorrect array size |
| 2.txt | 0 | Вывод ошибки: Incorrect array size |
| 3.txt | –10 1.1 4.0 | Вывод ошибки: Incorrect array size |
| 4.txt | 3 1.A 1/0 e4 | Вывод ошибки: Incorrect element value |
| 5.txt | 3 1.1 –2.3 | Вывод ошибки: There are absent elements |
| 6.txt | 3 2.1 –1.4 6.0 | Правильный ввод: Array is loaded successfully |

Для тестирования загрузки вместе с сохранением выполнить

следующие действия:

1) Вызвать функцию создания тестового массива.

2) Вызвать функцию сохранения массива в файле. Ввести имя 0.txt.

Вывод сообщения: Array is saved successfully

3) Открыть файл 0.txt в программе «Блокнот». Результат в файле:

12 2.1 0.1 1.2 –4.5 3.1 8.5 9.3 –9.8 0.2 –1.0 8.2 –5.4

4) Вызвать функцию загрузки массива из файла. Ввести имя 6.txt.

5) Вызвать функцию отображения массива. Результат на экране:

2.1 –1.4 6.0

6) Вызвать функцию загрузки массива из файла. Ввести имя 0.txt.

5) Вызвать функцию отображения массива. Результат на экране:

2.1 0.1 1.2 –4.5 3.1 8.5 9.3 –9.8 0.2 –1.0 8.2 –5.4

1. Тестирование работы с файлами

Для тестирования функции проверки имени файла использовать следующие данные:

|  |  |  |
| --- | --- | --- |
| № | Имя файла | Ожидаемый результат |
| 1 | 1?.txt | Вывод ошибки: Incorrect filename |
| 2 | 0\*1.txt | Вывод ошибки: Incorrect filename |
| 3 | 0.t:x | Вывод ошибки: Incorrect filename |
| 4 | C:\\1.txt\\ | Вывод ошибки: Incorrect filename |
| 5 | 11\2.txt\\ | Вывод ошибки: Incorrect filename |
| 6 | C:\0.txt | Корректное сохранение или загрузка файла |

Для тестирования ситуаций при сохранении данных в файл выполнить следующие действия:

1) Скопировать файл 6.txt в файл 7.txt и в файл 8.txt.

2) В проводнике установить атрибут Read-Only для файла 7.txt.

3) Вызвать функцию создания тестового массива.

4) Вызвать функцию сохранения массива в файле. Ввести имя 7.txt.

Вывод ошибки: File is accessed for read-only

5) Вызвать функцию сохранения массива в файле. Ввести имя 8.txt.

Вывод сообщения: File already exists. Do you want to overwrite it [Y/N]?

6) Ввести [Y].

Вывод сообщения: Array is saved successfully

3) Открыть файл 8.txt в программе «Блокнот». Результат в файле:

12 2.1 0.1 1.2 –4.5 3.1 8.5 9.3 –9.8 0.2 –1.0 8.2 –5.4

5) Вызвать функцию сохранения массива в файле. Ввести имя 6.txt.

Вывод сообщения: File already exists. Do you want to overwrite it [Y/N]?

6) Ввести [N].

7) Открыть файл 6.txt в программе «Блокнот». Результат в файле:

3 2.1 –1.4 6.0

8) Открыть файл 6.txt в программе Word.

9) Вызвать функцию сохранения массива в файле. Ввести имя 6.txt.

Вывод сообщения: File can not be opened

Для тестирования ситуаций при загрузке данных из файла выполнить следующие действия:

1) Вызвать функцию загрузки массива из файла. Ввести имя 100.txt.

Вывод сообщения: File not found

API reference основных функций:

Прототип:

errCode LoadArray(double\* &pArray, int &Size, char \*Name)

Описание:

Функция загружает массив вещественных чисел из файла.

Аргументы:

pArray – ссылка к указателю на массив вещественных чисел,

Size – ссылка к переменной, которая хранит размер массива,

Filename – указатель на строку имени файла с путем.

Возвращаемое значение: Код:

errOK – загрузка была успешна, ошибок нет,

errFileName – некорректное имя файла,

errFileNotFound – файл не найден или неверный путь,

errCanNotOpenFile– невозможно открыть,

errArraySize – файл хранит некорректный размер массива,

errNotMemory – недостаточно памяти для резервирования массива,

errArrayElement – файл хранит некорректные элементы массива,

errAbsentElements – количество элементов массива меньше его размера

Прототип:

errCode SaveArray(double\* pArray, int Size, char \*Name, bool rw)

Описание:

Функция записывает массив вещественных чисел в файл.

Аргументы:

pArray – указатель на массив вещественных чисел,

Size – размер массива,

Name – указатель на строку имени файла с путем,

rw – режим перезаписи в случае существования файла:

true – запросить подтверждение на перезапись,

false – отменить сохранение с выдачей ошибки.

Возвращаемое значение: Код:

errOK – сохранение было успешено, ошибок нет,

errFileName – некорректное имя файла,

errFileExists – файл существует и перезапись отменена,

errFileIsReadOnly – файл существует и доступен только для чтения,

errCanNotOpenFile– невозможно открыть или создать файл.

Прототип:

void ShowArray(double \* pArray, int Size)

Описание:

Функция выводит массив вещественных чисел на консоль.

Аргументы:

pArray – указатель на массив вещественных чисел,

Size – размер массива.

Возвращаемое значение: нет.

Прототип:

errCode InputArray(double\* &pArray, int &Size)

Описание:

Функция вводит массив вещественных чисел с консоли.

Аргументы:

pArray – ссылка к указателю на массив вещественных чисел,

Size – ссылка к переменной, которая хранит размер массива.

Возвращаемое значение: Код:

errOK – ввод был успешен, ошибок нет,

errArraySize – был введен некорректный размер массива,

errNotMemory – недостаточно памяти для резервирования массива,

errArrayElement– элементы были введены некорректно.

Прототип:

Double CalcCriteria(double \* pArray, int Left, int Right)

Описание:

Функция вычисляет сумму последовательности элементов массива.

Аргументы:

pArray – указатель на массив вещественных чисел,

Left – индекс начала последовательности,

Right – индекс конца последовательности.

Возвращаемое значение:

Сумма элементов последовательности.

Прототип:

void Search(double \*Arr, int Size, bool Opt, int& Begin, int& End)

Описание:

Функция ищет в массиве вещественных чисел такую непрерывную последовательность элементов, которая имеет либо максимальную, либо минимальную сумму элементов.

Аргументы:

Arr – указатель на массив вещественных чисел,

Size – размер массива,

Opt – опция поиска:

true – максимум суммы,

false – минимум суммы.

Begin – ссылка на переменную, в которую помещается индекс начала последовательности,

End – ссылка на переменную, в которую помещается индекс конца последовательности.

Возвращаемое значение:

Функция возвращает индексы начала и конца последовательности через свои аргументы.

Прототип:

bool CheckFileName(char \*name)

Описание:

Функция проверяет корректность имени файла.

Аргументы:

Filename – указатель на строку имени файла с путем.

Возвращаемое значение:

true – если имя файла корректно,

false – если имя файла некорректно.

Прототип:

errCode TestArray(double\* &pArray, int &Size)

Описание:

Функция копирует тестовый массив.

Аргументы:

pArray – ссылка к указателю на массив вещественных чисел,

Size – ссылка к переменной, которая хранит размер массива.

Возвращаемое значение: всегда errOK.

Листинг программы

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <stdbool.h>

#define errOK 0

#define errFileName 1

#define errFileNotFound 2

#define errCanNotOpenFile 3

#define errArraySize 4

#define errNotMemory 5

#define errArrayElement 6

#define errAbsentElements 7

#define errFileExists 8

#define errFileIsReadOnly 9

#define errCanNotSaveValue 10

// Прототипы функций

typedef int errCode;

errCode LoadArray(double\*\* pArray, int\* Size, char\* Name);

errCode SaveArray(double\* pArray, int Size, char\* Name, bool rw);

void ShowArray(double\* pArray, int Size);

errCode InputArray(double\*\* pArray, int\* Size);

double CalcCriteria(double\* pArray, int Left, int Right);

void Search(double\* Arr, int Size, bool Opt, int\* Begin, int\* End);

bool CheckFileName(char\* name);

errCode TestArray(double\*\* pArray, int\* Size);

int main() {

double\* array = NULL;

int size = 0;

int begin = 0, end = 0;

bool opt = true;

int choice;

char filename[256];

while (1) {

printf("\n----- MENYU -----\n");

printf("1. Sozdat testoviy massiv\n");

printf("2. Vvod massiva s konsoli\n");

printf("3. Pokazat massiv\n");

printf("4. Sohranit massiv v fayl\n");

printf("5. Zagrusit massiv iz fayla\n");

printf("6. Poisk maksimal'noy/ minimal'noy sumy\n");

printf("7. Vykhod\n");

printf("Vyberite komandu: ");

scanf("%d", &choice);

switch (choice) {

case 1: {

// Создание тестового массива

if (TestArray(&array, &size) != errOK) {

printf("Oshibka pri sozdanii testovogo massiva!\n");

} else {

printf("Testovyy massiv sozdan uspekhom!\n");

}

break;

}

case 2: {

// Ввод массива с консоли

if (InputArray(&array, &size) != errOK) {

printf("Oshibka pri vvedenii massiva!\n");

} else {

printf("Massiv vveden uspekhom!\n");

}

break;

}

case 3: {

// Показать массив

if (array == NULL) {

printf("Massiv ne sozdann!\n");

} else {

ShowArray(array, size);

}

break;

}

case 4: {

// Сохранить массив в файл

printf("Vvedite imya fayla dlya sohraneniya: ");

scanf("%s", filename);

if (SaveArray(array, size, filename, true) != errOK) {

printf("Oshibka pri sohranenii massiva v fayl!\n");

} else {

printf("Massiv uspesno sohranen!\n");

}

break;

}

case 5: {

// Загрузить массив из файла

printf("Vvedite imya fayla dlya zagruzki: ");

scanf("%s", filename);

if (LoadArray(&array, &size, filename) != errOK) {

printf("Oshibka pri zagruzke massiva iz fayla!\n");

} else {

printf("Massiv uspesno zagruzhen!\n");

}

break;

}

case 6: {

// Поиск максимальной/минимальной суммы

if (array == NULL) {

printf("Massiv ne sozdann!\n");

} else {

printf("Vvedite kriteriy poiska (1 - max, 0 - min): ");

scanf("%d", &opt);

Search(array, size, opt, &begin, &end);

printf("Naydennaya posledovatel'nost':\n");

printf("Nachalo: %d, Konets: %d\n", begin, end);

}

break;

}

case 7: {

// Выход из программы

free(array);

printf("Vykhod iz programmy.\n");

return 0;

}

default:

printf("Nekorrektnyy vvod. Povtorite\n");

}

}

return 0;

}

// Функции

// Загрузка массива из файла

errCode LoadArray(double\*\* pArray, int\* Size, char\* Name) {

FILE\* file = fopen(Name, "r");

if (!file) {

return errCanNotOpenFile;

}

// Чтение размера массива из файла

if (fscanf(file, "%d", Size) != 1) {

fclose(file);

return errArraySize;

}

// Выделение памяти для массива

\*pArray = (double\*)malloc(\*Size \* sizeof(double));

if (\*pArray == NULL) {

fclose(file);

return errNotMemory;

}

// Чтение элементов массива из файла

for (int i = 0; i < \*Size; i++) {

if (fscanf(file, "%lf", &(\*pArray)[i]) != 1) {

free(\*pArray);

fclose(file);

return errArrayElement;

}

}

fclose(file);

return errOK;

}

// Сохранение массива в файл

errCode SaveArray(double\* pArray, int Size, char\* Name, bool rw) {

// Открытие файла для записи (всегда перезаписываем файл)

FILE\* file = fopen(Name, "w");

if (!file) {

return errCanNotOpenFile;

}

// Записываем размер массива в файл

fprintf(file, "%d\n", Size);

// Записываем элементы массива

for (int i = 0; i < Size; i++) {

fprintf(file, "%lf\n", pArray[i]);

}

fclose(file); // Закрываем файл

return errOK;

}

// Показать массив

void ShowArray(double\* pArray, int Size) {

printf("Massiv: ");

for (int i = 0; i < Size; i++) {

printf("%lf ", pArray[i]);

}

printf("\n");

}

// Ввод массива с консоли

errCode InputArray(double\*\* pArray, int\* Size) {

printf("Vvedite razmer massiva: ");

scanf("%d", Size);

\*pArray = (double\*)malloc(\*Size \* sizeof(double)); // Выделение памяти

if (\*pArray == NULL) {

return errNotMemory; // Ошибка выделения памяти

}

// Ввод элементов массива

for (int i = 0; i < \*Size; i++) {

printf("Vvedite element %d: ", i);

scanf("%lf", &(\*pArray)[i]);

}

return errOK;

}

// Расчет среднего арифметического для подмассива

double CalcCriteria(double\* pArray, int Left, int Right) {

double sum = 0;

for (int i = Left; i <= Right; i++) {

sum += pArray[i];

}

return sum / (Right - Left + 1);

}

// Поиск максимальной/минимальной суммы

void Search(double\* Arr, int Size, bool Opt, int\* Begin, int\* End) {

double bestValue = Opt ? -1e10 : 1e10; // Начальное значение для поиска

for (int i = 0; i < Size - 1; i++) {

for (int j = i + 1; j < Size; j++) {

double current = CalcCriteria(Arr, i, j); // Среднее значение для подмассива

if ((Opt && current > bestValue) || (!Opt && current < bestValue)) {

bestValue = current;

\*Begin = i;

\*End = j;

}

}

}

}

// Проверка имени файла

bool CheckFileName(char\* name) {

return (strlen(name) > 0); // Проверка, что имя файла непустое

}

// Создание тестового массива

errCode TestArray(double\*\* pArray, int\* Size) {

\*Size = 5;

\*pArray = (double\*)malloc(\*Size \* sizeof(double)); // Выделение памяти для массива

if (\*pArray == NULL) {

return errNotMemory; // Ошибка выделения памяти

}

// Инициализация тестового массива

(\*pArray)[0] = 1.0;

(\*pArray)[1] = 2.0;

(\*pArray)[2] = -1.0;

(\*pArray)[3] = 3.0;

(\*pArray)[4] = -4.0;

return errOK;

}