# Лабораторная работа № 4 Работа с текстовыми файлами

# и реализация математических алгоритмов

**Цель лабораторной работы**

Закрепление теоретических знаний считывания-записи данных в текстовые файлы, реализация вычислительных алгоритмов на примере расчета определенных интегралов.

**Постановка задачи**

**Работа с текстовыми файлами**

Работу с файловой системой начнем с самого верхнего уровня - дисков. Для представления диска в пространстве имен System.IO имеется класс **DriveInfo**.

Этот класс имеет статический метод GetDrives, который возвращает имена всех логических дисков компьютера. Также он предоставляет ряд полезных свойств:

* **AvailableFreeSpace**: указывает на объем доступного свободного места на диске в байтах
* **DriveFormat**: получает имя файловой системы
* **DriveType**: представляет тип диска
* **IsReady**: готов ли диск (например, DVD-диск может быть не вставлен в дисковод)
* **Name**: получает имя диска
* **TotalFreeSpace**: получает общий объем свободного места на диске в байтах
* **TotalSize**: общий размер диска в байтах
* **VolumeLabel**: получает или устанавливает метку тома

Для работы с каталогами в пространстве имен System.IO предназначены сразу два класса: **Directory** и **DirectoryInfo**.

**Класс Directory**

Класс Directory предоставляет ряд статических методов для управления каталогами. Некоторые из этих методов:

* **CreateDirectory(path)**: создает каталог по указанному пути path
* **Delete(path)**: удаляет каталог по указанному пути path
* **Exists(path)**: определяет, существует ли каталог по указанному пути path. Если существует, возвращается true, если не существует, то false
* **GetDirectories(path)**: получает список каталогов в каталоге path
* **GetFiles(path)**: получает список файлов в каталоге path
* **Move(sourceDirName, destDirName)**: перемещает каталог
* **GetParent(path)**: получение родительского каталога

**Класс DirectoryInfo**

Данный класс предоставляет функциональность для создания, удаления, перемещения и других операций с каталогами. Во многом он похож на Directory. Некоторые из его свойств и методов:

* **Create()**: создает каталог
* **CreateSubdirectory(path)**: создает подкаталог по указанному пути path
* **Delete()**: удаляет каталог
* Свойство **Exists**: определяет, существует ли каталог
* **GetDirectories()**: получает список каталогов
* **GetFiles()**: получает список файлов
* **MoveTo(destDirName)**: перемещает каталог
* Свойство **Parent**: получение родительского каталога
* Свойство **Root**: получение корневого каталога
* Свойство **Exists**: указывает, существует ли файл
* Свойство **Length**: получает размер файла
* Свойство **Extension**: получает расширение файла
* Свойство **Name**: получает имя файла
* Свойство **FullName**: получает полное имя файла

Класс File реализует похожую функциональность с помощью статических методов:

* **Copy()**: копирует файл в новое место
* **Create()**: создает файл
* **Delete()**: удаляет файл
* **Move**: перемещает файл в новое место
* **Exists(file)**: определяет, существует ли файл

**Получение информации о файле**

string path = @"C:\apache\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

Console.WriteLine("Имя файла: {0}", fileInf.Name);

Console.WriteLine("Время создания: {0}", fileInf.CreationTime);

Console.WriteLine("Размер: {0}", fileInf.Length);

}

**Удаление файла**

string path = @"C:\apache\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.Delete();

// альтернатива с помощью класса File

// File.Delete(path);

}

**Перемещение файла**

string path = @"C:\apache\hta.txt";

string newPath = @"C:\SomeDir\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.MoveTo(newPath);

// альтернатива с помощью класса File

// File.Move(path, newPath);

}

**Копирование файла**

string path = @"C:\apache\hta.txt";

string newPath = @"C:\SomeDir\hta.txt";

FileInfo fileInf = new FileInfo(path);

if (fileInf.Exists)

{

fileInf.CopyTo(newPath, true);

// альтернатива с помощью класса File

// File.Copy(path, newPath, true);

}

Метод CopyTo класса FileInfo принимает два параметра: путь, по которому файл будет копироваться, и булевое значение, которое указывает, надо ли при копировании перезаписывать файл (если true, как в случае выше, файл при копировании перезаписывается). Если же в качестве последнего параметра передать значение false, то если такой файл уже существует, приложение выдаст ошибку.

Метод Copy класса File принимает три параметра: путь к исходному файлу, путь, по которому файл будет копироваться, и булевое значение, указывающее, будет ли файл перезаписываться.

Класс **FileStream** представляет возможности по считыванию из файла и записи в файл. Он позволяет работать как с текстовыми файлами, так и с бинарными.

Рассмотрим наиболее важные его свойства и методы:

* Свойство **Length**: возвращает длину потока в байтах
* Свойство **Position**: возвращает текущую позицию в потоке
* Метод **Read**: считывает данные из файла в массив байтов. Принимает три параметра: int Read(byte[] array, int offset, int count) и возвращает количество успешно считанных байтов. Здесь используются следующие параметры:
  + array - массив байтов, куда будут помещены считываемые из файла данные
  + offset представляет смещение в байтах в массиве array, в который считанные байты будут помещены
  + count - максимальное число байтов, предназначенных для чтения. Если в файле находится меньшее количество байтов, то все они будут считаны.
* Метод **long Seek(long offset, SeekOrigin origin)**: устанавливает позицию в потоке со смещением на количество байт, указанных в параметре offset.
* Метод **Write**: записывает в файл данные из массива байтов. Принимает три параметра: Write(byte[] array, int offset, int count)
  + array - массив байтов, откуда данные будут записываться в файла
  + offset - смещение в байтах в массиве array, откуда начинается запись байтов в поток
  + count - максимальное число байтов, предназначенных для записи

FileStream представляет доступ к файлам на уровне байтов, поэтому, например, если вам надо считать или записать одну или несколько строк в текстовый файл, то массив байтов надо преобразовать в строки, используя специальные методы. Поэтому для работы с текстовыми файлами применяются другие классы.

В то же время при работе с различными бинарными файлами, имеющими определенную структуру FileStream может быть очень даже полезен для извлечения определенных порций информации и ее обработки.

Рассмотрим примеры работы с FileStream.

Запись в файл:

string path = "example.txt";

using (FileStream fs = new FileStream(path, FileMode.Create, FileAccess.Write))

{

byte[] info = new UTF8Encoding(true).GetBytes("Hello, World!");

fs.Write(info, 0, info.Length);

}

Чтение из файла:

string path = "example.txt";

if (File.Exists(path))

{

using (FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read))

{

byte[] buffer = new byte[fs.Length];

fs.Read(buffer, 0, buffer.Length);

string content = System.Text.Encoding.UTF8.GetString(buffer);

Console.WriteLine("Содержимое файла: " + content);

}

}

else

{

Console.WriteLine("Файл не найден.");

}

Дополнение файла:

string path = "example.txt";

using (FileStream fs = new FileStream(path, FileMode.Append, FileAccess.Write))

{

byte[] info = new UTF8Encoding(true).GetBytes("\nAppending some text.");

fs.Write(info, 0, info.Length);

}

Чтение файла построчно с использованием StreamReader:

string path = "example.txt";

using (StreamReader sr = new StreamReader(path))

{

string line;

while ((line = sr.ReadLine()) != null)

{

Console.WriteLine(line);

}

}

Запись массива байтов в файл:

string path = "binaryData.bin";

byte[] data = new byte[] { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };

using (FileStream fs = new FileStream(path, FileMode.Create, FileAccess.Write))

{

fs.Write(data, 0, data.Length);

}

Чтение массива байтов из файла:

string path = "binaryData.bin";

if (File.Exists(path))

{

using (FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read))

{

byte[] data = new byte[fs.Length];

fs.Read(data, 0, data.Length);

Console.WriteLine("Данные из файла: " + BitConverter.ToString(data));

}

}

else

{

Console.WriteLine("Файл не найден.");

}

**Реализация математических алгоритмов**

Для некоторых подынтегральных функций интеграл можно вычислить аналитически или найти в справочниках. Однако в общем случае первообразная функции:

* может быть *неопределенной*;
* может не иметь выражения через элементарные функции.

Кроме того, сами подынтегральные функции в отдельных случаях не являются элементарными. В конечном счёте это приводит к необходимости разработки приближенных методов вычисления определенных интегралов. Наиболее простыми среди них являются так называемые *классические методы численного интегрирования*:

* прямоугольников;
* трапеций;
* парабол.

Каждый из этих методов основан на суммировании элементарных площадей, на которые разбивается вся площадь под функцией. Так в методе прямоугольников площадь под графиком функции (а значит, и определенный интеграл от *a* до *b*) может быть определен по одной из формул:

а) для входящих прямоугольников



б) для выходящих прямоугольников



где *n* – кратность (количество шагов) интегрирования функции  в точке, определяющей либо вписанный, либо описанный по отношению к графику интегрируемой функции прямоугольник.

Для уточнения значения интеграла, полученного по формулам суммирования площадей прямоугольников, существует формула остаточного члена

,

где ξ – максимум значения второй производной на рассматриваемом

интервале интегрирования, .

В методе трапеций площадь криволинейной трапеции и интеграл могут быть вычислены по формуле

.

Формула остаточного члена для метода трапеций имеет вид

.

В методе парабол (Симпсона) определение площади под графиком интегрируемой функции основано на замене двух смежных фрагментов участком параболы второго порядка (параболической трапецией)



Остаточный член для этого метода находится по формуле



где ξ – максимум значения четвёртой производной на рассматриваемом интервале интегрирования, .

**Задание на лабораторную работу**

1. Разработать консольное приложение.
2. Реализовать меню для выбора задания с возможностью неограниченного количества запусков заданий.
3. Выполнить задания согласно варианту.
4. Предусмотреть меню выбора файла из списка.
5. Подготовить входные данные для проверки работы приложения.

**Варианты заданий на лабораторную работу**

**Задание 1.**

1. **Анализ данных о погоде**: Считайте данные о температуре и влажности из CSV файла. Рассчитайте среднюю температуру и влажность за неделю и запишите результаты в новый файл.
2. **Обработка данных о продажах**: Прочитайте файл с данными о продажах (товар, количество, цена). Рассчитайте общую выручку для каждого товара и запишите результаты в новый CSV файл.
3. **Статистический анализ оценок**: Считайте файл с оценками студентов. Рассчитайте среднее, медиану и стандартное отклонение для каждого студента и запишите результаты в новый файл.
4. **Анализ данных о трафике**: Прочитайте лог-файл с данными о посещениях веб-сайта (время, IP-адрес). Определите количество уникальных посетителей и запишите это значение в новый файл.
5. **Обработка данных о финансах**: Считайте данные о транзакциях (дата, сумма, категория). Рассчитайте общие расходы по каждой категории и запишите результаты в новый файл.
6. **Анализ временных рядов**: Прочитайте файл с временными рядами (время, значение). Выполните скользящее среднее и запишите сглаженные данные в новый файл.

**** .

1. **Обработка данных о фитнесе**: Считайте данные из файла с информацией о тренировках (дата, продолжительность, калории). Рассчитайте общие калории за месяц и запишите результаты в новый файл.
2. **Анализ данных о клиентах**: Прочитайте файл с данными о клиентах (имя, возраст, покупки). Определите средний возраст клиентов и запишите результат в новый файл.
3. **Обработка данных о транспорте**: Считайте данные о поездках (начало, конец, расстояние). Рассчитайте общее расстояние, пройденное за день, и запишите в новый файл.
4. **Анализ данных о продуктах**: Прочитайте файл с данными о продуктах (название, цена, количество на складе). Рассчитайте общую стоимость запасов и запишите результат в новый файл.
5. **Анализ данных о сотрудниках**: Считайте данные о сотрудниках (имя, отдел, зарплата). Рассчитайте среднюю зарплату по каждому отделу и запишите результаты в новый файл.
6. **Обработка данных о заказах**: Прочитайте файл с данными о заказах (клиент, товар, количество). Определите наиболее популярный товар и запишите его в новый файл.
7. **Анализ данных о посещаемости**: Считайте данные о посещаемости студентов (имя, дата, статус). Определите процент посещаемости для каждого студента и запишите в новый файл.
8. **Обработка данных о расходах**: Прочитайте файл с данными о расходах (дата, сумма, категория). Найдите день с наибольшими расходами и запишите его в новый файл.
9. **Анализ данных о производительности**: Считайте данные о производительности сотрудников (имя, задачи, время выполнения). Рассчитайте среднее время выполнения задач и запишите результаты в новый файл.
10. **Обработка данных о транспортных средствах**: Прочитайте данные о транспортных средствах (тип, год выпуска, пробег). Определите средний пробег для каждого типа и запишите в новый файл.
11. **Анализ данных о потреблении энергии**: Считайте данные о потреблении энергии (дата, количество). Рассчитайте среднее потребление за месяц и запишите результаты в новый файл.
12. **Обработка данных о клиентах**: Прочитайте файл с данными о клиентах (имя, покупки, сумма). Определите клиента с наибольшими покупками и запишите его в новый файл.
13. **Анализ данных о производстве**: Считайте данные о производстве (дата, количество, дефекты). Рассчитайте процент дефектов и запишите результаты в новый файл.
14. **Обработка данных о здоровье**: Прочитайте файл с данными о здоровье (имя, вес, рост). Рассчитайте индекс массы тела (ИМТ) для каждого человека и запишите в новый файл.
15. **Анализ данных о продажах недвижимости**: Считайте данные о продажах (адрес, цена, площадь). Рассчитайте среднюю цену за квадратный метр и запишите результаты в новый файл.
16. **Обработка данных о бронированиях**: Прочитайте файл с данными о бронированиях (гость, дата, номер). Определите наиболее популярные даты и запишите их в новый файл.
17. **Анализ данных о доходах**: Считайте данные о доходах (источник, сумма, дата). Рассчитайте общий доход за год и запишите результаты в новый файл.
18. **Обработка данных о поставках**: Прочитайте файл с данными о поставках (товар, количество, дата). Определите товар с наибольшими поставками и запишите его в новый файл.
19. **Анализ данных о занятиях спортом**: Считайте данные о занятиях (вид спорта, время, калории). Рассчитайте общее количество сожжённых калорий и запишите результаты в новый файл.
20. **Обработка данных о путешествиях**: Прочитайте файл с данными о путешествиях (место, дата, расходы). Определите самое дорогое путешествие и запишите его в новый файл.
21. **Анализ данных о потребителях**: Считайте данные о потребителях (возраст, покупки, предпочтения). Определите наиболее популярную категорию товаров и запишите её в новый файл.
22. **Обработка данных о транспорте**: Прочитайте файл с данными о рейсах (номер рейса, время, задержка). Рассчитайте среднюю задержку и запишите результаты в новый файл.
23. **Анализ данных о производительности сайта**: Считайте данные о времени загрузки страниц (URL, время). Определите страницы с наибольшим временем загрузки и запишите их в новый файл.
24. **Обработка данных о звонках**: Прочитайте файл с данными о звонках (номер, длительность, дата). Определите самый долгий звонок и запишите его в новый файл.

**Задание 2.**

|  |  |  |
| --- | --- | --- |
| № варианта | Интеграл | Метод |
| 1 |  | Трапеций |
| 2 |  | Входящих прямоугольников |
| 3 |  | Выходящих прямоугольников |
| 4 |  | Трапеций |
| 5 |  | Входящих прямоугольников |
| 6 |  | Выходящих прямоугольников |
| 7 |  | Трапеций |
| 8 |  | Входящих прямоугольников |
| 9 |  | Выходящих прямоугольников |
| 10 |  | Трапеций |
| 11 |  | Входящих прямоугольников |
| 12 |  | Выходящих прямоугольников |
| 13 |  | Трапеций |
| 14 |  | Входящих прямоугольников |
| 15 |  | Выходящих прямоугольников |
| 16 |  | Трапеций |
| 17 |  | Входящих прямоугольников |
| 18 |  | Выходящих прямоугольников |
| 19 |  | Трапеций |
| 20 |  | Входящих прямоугольников |
| 21 |  | Выходящих прямоугольников |
| 22 |  | Трапеций |
| 23 |  | Входящих прямоугольников |
| 24 |  | Выходящих прямоугольников |
| 25 |  | Трапеций |
| 26 |  | Входящих прямоугольников |
| 27 |  | Выходящих прямоугольников |
| 28 |  | Трапеций |
| 29 |  | Входящих прямоугольников |
| 20 |  | Выходящих прямоугольников |

**Пример разработки приложения**

Задание 1.

Обработка данных о звонках: Прочитайте файл с данными о звонках (номер, длительность, дата). Определите среднюю длительность звонков и запишите все звонки, превышающие эту среднюю длительность, в новый файл.

Оформим код в виде классов и методов, запишем класс, представляющий данные о звонке:

using System;

using System.Collections.Generic;

using System.IO;

public class Call

{

public string PhoneNumber { get; set; } // Номер телефона

public int Duration { get; set; } // Длительность звонка в секундах

public DateTime Date { get; set; } // Дата звонка

// Конструктор класса Call

public Call(string phoneNumber, int duration, DateTime date)

{

PhoneNumber = phoneNumber;

Duration = duration;

Date = date;

}

}

Запишем класс, для обработки звонков:

public class CallProcessor

{

private List<Call> calls = new List<Call>(); // Список звонков

// Метод для чтения звонков из файла

public void ReadCallsFromFile(string filePath)

{

// Читаем каждую строку из файла

foreach (var line in File.ReadLines(filePath))

{

var parts = line.Split(','); // Разделяем строку на части

string phoneNumber = parts[0]; // Получаем номер телефона

int duration = int.Parse(parts[1]); // Получаем длительность звонка

DateTime date = DateTime.Parse(parts[2]); // Получаем дату звонка

// Добавляем новый объект Call в список

calls.Add(new Call(phoneNumber, duration, date));

}

}

// Метод для вычисления средней длительности звонков

public double CalculateAverageDuration()

{

// Если список звонков пуст, возвращаем 0

if (calls.Count == 0) return 0;

int totalDuration = 0; // Переменная для хранения общей длительности

// Суммируем длительности всех звонков

foreach (var call in calls)

{

totalDuration += call.Duration;

}

// Возвращаем среднее значение длительности звонков

return (double)totalDuration / calls.Count;

}

// Метод для получения звонков, превышающих среднюю длительность

public List<Call> GetCallsAboveAverage(double averageDuration)

{

List<Call> callsAboveAverage = new List<Call>(); // Список для хранения звонков выше среднего

// Проверяем каждый звонок

foreach (var call in calls)

{

// Если длительность звонка больше средней, добавляем его в новый список

if (call.Duration > averageDuration)

{

callsAboveAverage.Add(call);

}

}

// Возвращаем список звонков, превышающих среднюю длительность

return callsAboveAverage;

}

// Метод для записи звонков в новый файл

public void WriteCallsToFile(string filePath, List<Call> callsAboveAverage)

{

// Создаем StreamWriter для записи в файл

using (StreamWriter writer = new StreamWriter(filePath))

{

// Записываем каждый звонок в файл

foreach (var call in callsAboveAverage)

{

writer.WriteLine($"{call.PhoneNumber},{call.Duration},{call.Date}");

}

}

}

}

Запишем Основной класс программы:

public class Program

{

public static void Main(string[] args)

{

// Путь к файлу с данными о звонках

string inputFilePath = "calls.txt";

// Путь к выходному файлу

string outputFilePath = "calls\_above\_average.txt";

// Создаем экземпляр CallProcessor

CallProcessor processor = new CallProcessor();

// Шаг 1: Чтение звонков из файла

processor.ReadCallsFromFile(inputFilePath);

// Шаг 2: Вычисление средней длительности звонков

double averageDuration = processor.CalculateAverageDuration();

Console.WriteLine($"Средняя длительность звонков: {averageDuration}");

// Шаг 3: Получение звонков, превышающих среднюю длительность

List<Call> callsAboveAverage = processor.GetCallsAboveAverage(averageDuration);

// Шаг 4: Запись звонков в новый файл

processor.WriteCallsToFile(outputFilePath, callsAboveAverage);

// Выводим сообщение о завершении записи

Console.WriteLine($"Звонки, превышающие среднюю длительность, записаны в файл: {outputFilePath}");

}

}

Задание 2.

Метод трапеции

.

Запишем метод для расчета подынтегральной функции:

using System;

double Function(double x)

{

return x \* x; // f(x) = x^2

}

Запишем метод для расчета первообразной функции:

double Antiderivative(double x)

{

return (x \* x \* x) / 3; // F(x) = x^3 / 3

}

Запишем метод трапеций для численного интегрирования:

double TrapezoidalIntegration(double a, double b, int n)

{

double h = (b - a) / n; // Шаг

double sum = 0.5 \* (Function(a) + Function(b)); // Начальная сумма

// Считаем сумму значений функции в промежуточных точках

for (int i = 1; i < n; i++)

{

double x = a + i \* h;

sum += Function(x);

}

return sum \* h; // Возвращаем результат

}

Запишем основную часть программы:

// Определяем границы интегрирования и количество подынтервалов

double a = 0; // Начало интервала

double b = 1; // Конец интервала

int n = 100; // Количество подынтервалов

// Численное интегрирование методом трапеций

double numericalResult = TrapezoidalIntegration(a, b, n);

// Аналитическое решение через первообразные

double analyticalResult = Antiderivative(b) - Antiderivative(a);

// Вывод результатов

Console.WriteLine($"Численное интегрирование (метод трапеций): {numericalResult}");

Console.WriteLine($"Аналитическое решение (формула Ньютона-Лейбница): {analyticalResult}");

Console.WriteLine($"Ошибка: {Math.Abs(numericalResult - analyticalResult)}");

**Контрольные вопросы**

1. Что такое пространство имен System.IO и какие классы оно включает?
2. Как открыть файл для чтения и записи в C#?
3. Как проверить, существует ли файл в C#?
4. Как прочитать все строки из текстового файла в массив строк?
5. Как записать массив строк в текстовый файл?
6. Что такое метод в C# и как его объявить?
7. Как передать параметры в метод и вернуть значение из метода?
8. Что такое перегрузка методов? Приведите пример.
9. Что такое класс в C# и как его объявить?
10. Что такое объект и как его создать из класса?
11. Как создать методы в классе?
12. Как передать объект класса в метод?
13. Какова формула для численного интегрирования методом трапеций?
14. Что такое первообразная функции и как она связана с интегрированием?
15. Как можно оценить ошибку численного интегрирования?