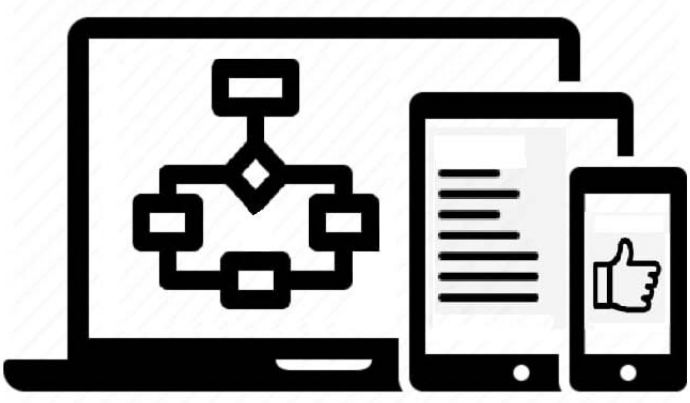
МЕЖГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛОРУССКО-РОССИЙСКИЙ УНИВЕРСИТЕТ»

Кафедра «Программное обеспечение информационных технологий»

ПРАКТИКА НАПИСАНИЯ ПРОГРАММНОГО КОДА

Методические рекомендации к лабораторным работам для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения



Могилев 2023

УДК 004.42
ББК 32.973-018
П78

Рекомендовано к изданию
учебно-методическим отделом
Белорусско-Российского университета

Одобрено кафедрой «Программное обеспечение информационных технологий» «28» марта 2023 г., протокол № 9

Составители: ст. преподаватель О. В. Сергиенко;
канд. техн. наук, доц. Н. Н. Горбатенко

Рецензент канд. техн. наук, доц. И. В. Лесковец

Методические рекомендации к лабораторным работам предназначены для студентов направлений подготовки 09.03.04 «Программная инженерия» и 09.03.01 «Информатика и вычислительная техника» дневной формы обучения.

Учебное издание

ПРАКТИКА НАПИСАНИЯ ПРОГРАММНОГО КОДА

Ответственный за выпуск	В. В. Кутузов
Корректор	А. А. Подошевко
Компьютерная верстка	Н. П. Полевничая

Подписано в печать . Формат 60×84/16. Бумага офсетная. Гарнитура Таймс.
Печать трафаретная. Усл. печ. л. . Уч.-изд. л. . Тираж 21 экз. Заказ №

Издатель и полиграфическое исполнение:
Межгосударственное образовательное учреждение высшего образования
«Белорусско-Российский университет».
Свидетельство о государственной регистрации издателя,
изготовителя, распространителя печатных изданий
№ 1/156 от 07.03.2019.
Пр-т Мира, 43, 212022, г. Могилев.

© Белорусско-Российский
университет, 2023

Содержание

Введение

При изучении дисциплины «Практика написания программного кода» студенты выполняют лабораторные работы, варианты которых приведены в данных методических рекомендациях.

Целью учебной дисциплины является приобретение студентами навыков написания программного кода на основе принципов процедурной декомпозиции решаемой задачи.

Каждая лабораторная работа соответствует темам рабочей программы и содержит в себе десять вариантов индивидуальных заданий.

Варианты заданий выдаются студентам заранее с тем, чтобы они имели возможность подготовиться к выполнению лабораторной работы: просмотреть теоретический материал по теме работы и продумать алгоритмы решения задач.

Программы пишутся на языке C#. Каждую программу в работающем виде (после отладки и тестирования) студент показывает преподавателю, после чего лабораторная работа подлежит защите.

К защите работы студент подготавливает отчет, включающий в себя титульный лист, формулировку задания, описание исходных, результирующих данных и вспомогательных переменных, текст программы и результаты ее тестирования. Отчет оформляется в соответствии с ГОСТ 2.105–2019.

Защита лабораторной работы состоит из двух частей: практической и теоретической. В практической части студент объясняет принципы работы представленной им программы, в теоретической – отвечает на вопросы по теме лабораторной работы.

1 Лабораторная работа № 1. Практика написания методов

Цель работы: приобретение студентами навыков написания методов на основе принципов процедурной декомпозиции решаемой задачи.

Теоретические сведения

Некоторые принципы построения методов в программах на языке C#, которые применяются в IT, включают в себя.

1 Принцип единственной ответственности (Single Responsibility Principle, SRP): каждый метод должен отвечать только за одну задачу. Это упрощает понимание и поддержку кода, позволяет избегать дублирования кода и повышает его повторное использование.

2 Принцип открытости/закрытости (Open/Closed Principle, OCP): методы должны быть открыты для расширения, но закрыты для изменения. Это означает, что изменение поведения метода должно происходить только через добавление нового функционала, а не через изменение существующего кода.

3 Принцип подстановки Барбары Лисков (Liskov Substitution Principle, LSP): объекты должны быть заменяемыми на своих наследников без изменения корректности программы. Это означает, что методы должны работать с абстрактными типами данных, а не с конкретными реализациями.

4 Принцип инверсии зависимостей (Dependency Inversion Principle, DIP): зависимости должны зависеть от абстракций, а не от конкретных классов. Это позволяет создавать более гибкие и расширяемые системы.

5 Принцип разделения интерфейса (Interface Segregation Principle, ISP): интерфейсы должны быть разделены на маленькие и специфичные для каждой задачи. Это позволяет избежать ненужной зависимости между различными компонентами системы.

6 Принцип композиции/агрегации (Composition/Aggregation Principle): объекты должны создаваться с помощью композиции или агрегации других объектов. Это позволяет создавать более сложные объекты из более простых компонентов, что упрощает их понимание и поддержку.

7 Принцип минимального знания (Law of Demeter, LoD): объекты должны иметь минимальное знание о других объектах. Это означает, что объект должен взаимодействовать только с объектами, с которыми он непосредственно связан, а не с объектами, которые находятся далеко от него в иерархии.

Эти принципы помогают создавать более гибкие, расширяемые и легко поддерживаемые программы на языке C#.

Задания для самостоятельного выполнения

Создать программу, которая использует рассмотренные принципы.

Варианты заданий

1 Дана целочисленная прямоугольная матрица. Определить количество строк, не содержащих ни одного нулевого элемента.

2 Дана целочисленная прямоугольная матрица. Определить количество столбцов, не содержащих ни одного нулевого элемента.

3 Дана целочисленная квадратная матрица. Определить произведение элементов в тех строках, которые не содержат отрицательных элементов.

4 Дана целочисленная квадратная матрица. Определить сумму элементов в тех столбцах, которые не содержат отрицательных элементов.

5 Вычислить сумму элементов каждого столбца матрицы X(4×3), определить наименьшее значение этих сумм и номер соответствующего столбца.

6 Найти наибольший элемент матрицы A(5×3) и номер строки и столбца, в котором он находится.

7 Вычислить сумму элементов каждой строки матрицы X(4×3), определить наименьшее значение этих сумм и номер соответствующей строки.

8 Найти наибольшие элементы каждой строки матрицы X(4×5) и записать их в массив Y.

9 Найти наибольший элемент главной диагонали матрицы A(4×4) и вывести на экран все строку, в которой он находится.

10 В массиве M[6×4] все числа различны. В каждой строке находится минимальный элемент, затем среди этих чисел выбирается максимальное. Напечатать номер строки массива M, в которой расположено выбранное число.

Контрольные вопросы

1 Напишите метод, который принимает два целых числа и возвращает их сумму. Также объясните свою реализацию.

2 Напишите метод, который принимает строку и возвращает ее длину. Также объясните свою реализацию.

3 Напишите метод, который принимает массив целых чисел и возвращает среднее значение. Также объясните свою реализацию.

4 Напишите метод, который принимает две строки и возвращает true, если они идентичны, и false в противном случае. Также объясните свою реализацию.

5 Напишите метод, который принимает список объектов и возвращает количество объектов, у которых определено некоторое свойство. Также объясните свою реализацию.

2 Лабораторная работа № 2. Обработка исключительных ситуаций

Цель работы: приобретение студентами навыков корректной обработки исключительных ситуаций.

Теоретические сведения

Обработка исключительных ситуаций в языке C# осуществляется с помощью механизма try-catch-finally.

Блок try содержит код, который может вызвать исключительную ситуацию. Если исключительная ситуация происходит в блоке try, то управление передается в блок catch, который содержит код для обработки исключения. В блоке catch указывается тип исключения, которое нужно обработать. Если тип исключения не указан явно, то обрабатываются все исключения.

Блок finally содержит код, который будет выполнен независимо от того, произошло исключение или нет. Этот блок используется для освобождения ресурсов, например, закрытия файлов или соединений с базой данных.

Пример обработки исключительной ситуации в C#:

```
try
{
   int a = 10;
   int b = 0;
   int c = a / b; // попытка деления на ноль
}
catch (DivideByZeroException ex)
{
   Console.WriteLine("Ошибка деления на ноль: " + ex.Message);
}
finally
{
   Console.WriteLine("Выполнение блока finally");
}
```

В данном примере, если происходит деление на ноль, то управление передается в блок catch, где выводится сообщение об ошибке. Затем управление передается в блок finally, где выводится сообщение о выполнении блока finally. Если исключительная ситуация не происходит, то блок catch не выполняется, но блок finally все равно будет выполнен.

Правильное расположение кода в блоках try-catch-finally может повлиять на корректность обработки исключительных ситуаций.

В блоке try следует размещать код, который может привести к возникновению исключительной ситуации. Если исключительная ситуация не возникает,

код в блоке try будет выполнен без ошибок.

В блоках catch следует обрабатывать исключительные ситуации, которые могут возникнуть в блоке try. Каждый блок catch должен обрабатывать определенный тип исключительной ситуации. Например:

```
try
{
   // Код, который может привести к возникновению исключительной ситуации
}
catch (DivideByZeroException ex)
{
   // Обработка исключительной ситуации деления на ноль
}
catch (Exception ex)
{
   // Обработка исключительных ситуаций, которые не были предусмотрены в других блоках catch
}
```

В блоке finally следует размещать код, который должен быть выполнен независимо от того, возникла исключительная ситуация или нет. Например, в блоке finally можно закрыть открытые ресурсы, такие как файлы или соединения с базой данных.

```
try
{
   // Код, который может привести к возникновению исключительной ситуации
}
catch (DivideByZeroException ex)
{
   // Обработка исключительной ситуации деления на ноль
}
finally
{
   // Код, который должен быть выполнен независимо от того, возникла исключительная ситуация или нет
}
```

Важно помнить, что блок finally будет выполнен в любом случае, даже если было вызвано исключение и не было обработано в блоках catch. Поэтому в блоке finally следует размещать только тот код, который необходимо выполнить независимо от наличия исключительных ситуаций.

Задания для самостоятельного выполнения

Изменить программу из лабораторной работы № 1. Добавить ввод размера массива и параметров выполнения задачи (номер столбца и т. д.). Добавить обработку возможных исключительных ситуаций.

Контрольные вопросы

1 Что такое исключительная ситуация и почему она может возникнуть в программе? Каким образом можно обработать исключительную ситуацию?

2 Какие способы обработки исключительных ситуаций вы знаете? Какие преимущества и недостатки у каждого из них?

3 Почему использование исключительных ситуаций может быть полезно в программировании? Какие принципы следует соблюдать при выборе типов исключений?

4 Каким образом можно создать собственный тип исключения? В каких случаях это может быть полезно?

5 Как можно протестировать обработку исключительных ситуаций в программе? Какие методы тестирования обработки исключений вы знаете?

3 Лабораторная работа № 3. Форматирование кода

Цель работы: приобретение студентами навыков форматирования кода.

Теоретические сведения

В C# существует несколько стандартов и соглашений по форматированию кода, которые помогают улучшить читабельность и поддерживаемость кода. Некоторые из них описаны в следующем перечне.

1 Используйте каркасный стиль фигурных скобок (C# стиль) для определения блочных конструкций кода. Фигурная скобка должна быть на отдельной строке, а не на той же строке, что и конструкция, которую она определяет. Например:

```
Copy
if (condition)
{
   // Код
}
else
{
   // Код
}
```

2 Используйте отступы в четыре пробела для каждого уровня вложенности

кода. Не используйте символ табуляции (Tab) для отступов. Например:

```
Copy
public void MyMethod()
{
   if (condition)
   {
      // Код
   }
   else
   {
      // Код
   }
}
```

3 Используйте CamelCase для именования переменных, методов и свойств. Например:

```
arduino
Copy
public class MyClass
{
   public string myStringVariable;
   public int myIntVariable;

   public void myMethod()
   {
      // Код
   }

   public string MyProperty { get; set; }
}
```

4 Используйте PascalCase для именования классов, интерфейсов и перечислений. Например:

```
java
Copy
public class MyClass
{
   // Код
}

public interface IMyInterface
{
```

```
    // Код
}

public enum MyEnum
{
    // Код
}
```

5 Используйте пробелы между операторами и операндами, чтобы улучшить читабельность кода. Например:

```
Copy
int result = variable1 + variable2;
if (condition1 && condition2)
{
    // Код
}
```

6 Используйте комментарии для документирования кода, объяснения сложных алгоритмов или указания на особенности реализации. Комментарии должны быть на английском языке и использоваться только там, где это необходимо. Например:

```
Copy
// Этот метод выполняет сложение двух чисел
public int Add(int a, int b)
{
    return a + b;
}
```

7 Избегайте использования слишком длинных строк кода. Рекомендуется ограничивать длину строки 80 символами.

8 Старайтесь использовать константы вместо магических чисел и строк. Например:

```
Copy
public const int MAX_COUNT = 100;
public const string DEFAULT_NAME = "John";
```

9 Используйте using для управления ресурсами, такими как файлы, сокеты или базы данных. Например:

```
Copy
using (FileStream fs = new FileStream("file.txt", FileMode.Open))
{
```

```
    // Код
}
```

Это не полный список правил и соглашений по форматированию кода в C#, однако следуя им, вы можете улучшить читабельность и поддерживаемость своего кода. Кроме того, в Visual Studio есть функциональность автоматического форматирования кода, которая может помочь соблюдать эти правила и соглашения.

В C# существует несколько дополнительных соглашений по форматированию кода, которые помогают улучшить читабельность и поддерживаемость кода. Некоторые из них описаны в следующем перечне.

1 Размещайте модификаторы доступа перед ключевым словом, которое они модифицируют. Например:

```
Copy
public class MyClass
{
    private int myIntVariable;

    public void MyMethod()
    {
        // Код
    }
}
```

2 Используйте var для объявления переменных, когда тип переменной очевиден из контекста. Например:

```
Copy
var myVariable = "Hello World";
var myIntVariable = 42;
```

3 Используйте скобки для улучшения читабельности, даже если они не обязательны в конкретной ситуации. Например:

```
Copy
if (condition)
{
    // Код
}

while (condition)
{
    // Код
}
```

```
for (int i = 0; i < count; i++)
{
   // Код
}
```

4 Используйте явные типы при создании массивов. Например:

```
Copy
int[] myIntArray = new int[10];
string[] myStringArray = new string[5];
```

5 Используйте стандартную нотацию для имен файлов и папок в проекте. Например, название файла должно быть в PascalCase и заканчиваться на .cs, а название папки должно быть в lowercase.

6 Используйте #region и #endregion для группировки связанных секций кода внутри класса. Например:

```
csharp
Copy
public class MyClass
{
   #region Private Fields

   private int myIntVariable;

   #endregion

   #region Public Methods

   public void MyMethod()
   {
      // Код
   }

   #endregion
}
```

7 Используйте свойства (properties) вместо открытых поля (public fields), когда это возможно. Например:

```
Copy
public class MyClass
{
   private int myIntVariable;
```

```
    public int MyIntVariable
    {
        get { return myIntVariable; }
        set { myIntVariable = value; }
    }
}
```

8 Используйте оператор ? для проверки на null вместо if (variable == null). Например:

```
Copy
if (myObject?.MyProperty == null)
{
    // Код
}
```

Эти соглашения не являются обязательными, но следование им улучшает читабельность и поддерживаемость кода в C#.

Задания для самостоятельного выполнения

1 Изменить программу из лабораторной работы № 2 в соответствии с описанными выше требованиями.

2 Разработать программу, используя правила оформления кода, приведенные выше.

Варианты заданий

1 Напишите метод, который принимает на вход два аргумента – строку и символ – и возвращает количество вхождений символа в строку.

2 Напишите метод, который принимает на вход целое число и возвращает true, если оно является простым числом, и false в противном случае.

3 Напишите метод, который принимает на вход массив целых чисел и возвращает среднее арифметическое значение элементов массива.

4 Напишите метод, который принимает на вход два массива целых чисел и возвращает новый массив, содержащий элементы обоих массивов (в том порядке, в котором они идут в исходных массивах).

5 Напишите метод, который принимает на вход два аргумента – строку и целое число – и возвращает новую строку, состоящую из символов исходной строки, начиная с указанной позиции и до конца строки.

6 Напишите метод, который принимает на вход массив целых чисел и возвращает новый массив, содержащий только уникальные элементы исходного массива.

7 Напишите метод, который принимает на вход строку и возвращает новую строку, в которой все слова записаны в обратном порядке.

8 Напишите метод, который принимает на вход массив целых чисел и воз-

вращает новый массив, содержащий элементы исходного массива в обратном порядке.

9 Напишите метод, который принимает на вход два аргумента – целое число и массив целых чисел – и возвращает новый массив, содержащий только те числа, которые больше заданного числа.

10 Напишите метод, который принимает на вход два аргумента – строку и целое число – и возвращает новую строку, состоящую из первых указанных символов исходной строки.

Контрольные вопросы

1 Что такое форматирование кода и почему оно важно для программистов? Какие принципы следует соблюдать при форматировании кода?

2 Какие инструменты для форматирования кода вы знаете? Какие преимущества и недостатки у каждого из них?

3 Какие основные элементы стиля кодирования существуют? Какие стандарты форматирования кода существуют в различных языках программирования?

4 Каким образом можно автоматизировать форматирование кода в проекте? Какие инструменты и плагины можно использовать для этого?

5 Каким образом форматирование кода может повлиять на производительность программы? Какие методы оптимизации форматирования кода вы знаете?

4 Лабораторная работа № 4. Написание комментариев

Цель работы: приобретение студентами навыков написания комментариев, облегчающих чтение программы.

Теоретические сведения

Правила составления комментариев к коду, применяемых в промышленном программировании, могут варьироваться в зависимости от конкретных требований компании или проекта. Вот несколько общих рекомендаций.

1 Комментируйте только те участки кода, которые сложны для понимания или имеют особую важность. Избегайте комментирования очевидных вещей.

2 Используйте информативные и точные названия переменных, функций, классов и методов. В идеале, название должно само по себе давать представление о том, что делает соответствующий элемент.

3 При описании функций или методов комментарий должен содержать информацию о том, что функция делает, что она принимает в качестве параметров и что возвращает.

4 Избегайте описания того, как код работает. Вместо этого, комментируйте, почему код работает так, как он работает, и как он связан с другими частями системы.

5 Используйте ясный и лаконичный язык. Избегайте слишком длинных

комментариев, которые могут запутать читателя.

6 Пишите комментарии на английском языке, если это требуется в проекте. Это поможет обеспечить международную понятность кода.

7 Проверяйте свои комментарии на грамматические ошибки и опечатки.

8 Обновляйте комментарии при изменении кода. Если вы вносите изменения в код, убедитесь, что соответствующие комментарии тоже обновлены.

9 Используйте специальные теги комментариев, такие как TODO, FIXME или HACK, чтобы обозначить, что нужно еще что-то сделать или что в коде есть некоторые проблемы, которые нужно решить.

10 Не забывайте, что комментарии – это не замена хорошо структурированному и чистому коду. Хороший код должен говорить сам за себя, а комментарии должны дополнять его и улучшать понимание его работы.

Варианты заданий для самостоятельного выполнения

1 Напишите приложение, которое принимает на вход строку и определяет, является ли она палиндромом (т. е. читается одинаково как слева направо, так и справа налево).

2 Разработайте приложение, которое позволяет пользователю выбирать из двух вариантов заданий (например, играть в крестики-нолики или угадывать загаданное число) и запускает соответствующую игру.

3 Создайте приложение, которое позволяет пользователю выбирать из нескольких вариантов цветов и изменять цвет фона окна.

4 Напишите приложение, которое читает данные из текстового файла и выводит их на экран в отсортированном порядке.

5 Разработайте конвертер валют, который позволяет пользователю вводить сумму в одной валюте и переводить ее в другую валюту по текущему курсу.

6 Создайте приложение, которое позволяет пользователю добавлять, редактировать и удалять записи в текстовом файле.

7 Напишите приложение, которое генерирует рандомные пароли заданной длины и сложности.

8 Разработайте приложение, которое позволяет пользователю выбирать из нескольких вариантов графических элементов (например, круг, квадрат, треугольник) и рисовать их на экране.

9 Создайте приложение, которое позволяет пользователю выбирать из нескольких вариантов алгоритмов сортировки и сортировать заданный массив данных.

10 Напишите приложение, которое позволяет пользователю выбирать из нескольких вариантов заданий (например, решать математические задачи или играть в слова) и запускает соответствующую задачу.

Контрольные вопросы

1 Зачем нужны комментарии в коде? Какие принципы следует соблюдать при написании комментариев?

2 Какие виды комментариев существуют? Какие из них наиболее полезны для программистов и почему?

3 Как можно определить, когда нужно добавлять комментарии в код? Какие ситуации требуют особого внимания со стороны программиста?

4 Каким образом можно управлять объемом комментариев в коде? Какие методы сокращения объема комментариев можно использовать?

5 Какие инструменты и плагины существуют для автоматизации написания комментариев? Какие преимущества и недостатки у каждого из них?

5 Лабораторная работа № 5. Отладка и тестирование программного кода

Цель работы: приобретение студентами навыков отладки и тестирования программного кода.

Теоретические сведения

Отладка и тестирование программного кода – это важная часть процесса разработки программного обеспечения. Правильное тестирование и отладка помогают обнаруживать и исправлять ошибки, улучшать качество и надежность программы.

Правила отладки программного кода.

Используйте инструменты отладки: отладчик, логирование, проверку типа данных, проверку граничных условий, чтобы быстро обнаружить и исправить ошибки.

Используйте методы инкрементальной разработки: тестируйте код после каждого изменения, чтобы быстро выявить и исправить ошибки.

Изучайте сообщения об ошибках: читайте сообщения об ошибках и записывайте их, чтобы быстро исправить ошибки.

Используйте систему контроля версий: создавайте резервные копии и сохраняйте версии программного кода, чтобы быстро вернуться к предыдущей версии в случае ошибки.

Правила тестирования программного кода.

Планируйте тестирование: определите цели тестирования, создайте план тестирования и убедитесь, что все тесты покрывают все возможные случаи использования программы.

Используйте различные методы тестирования: проводите модульное тестирование, функциональное тестирование, интеграционное тестирование, системное тестирование, чтобы проверить работу программы на всех уровнях.

Используйте автоматизированные тесты: создавайте тесты, которые можно автоматически запускать, чтобы быстро обнаруживать и исправлять ошибки.

Используйте реалистичные данные: используйте реальные данные или случайно созданные данные, чтобы проверить работу программы на всех воз-

можных случаях использования.

Проверяйте на безопасность: проверьте программу на уязвимости и возможные атаки, чтобы убедиться, что программа безопасна и защищена от взлома.

Задания для самостоятельного выполнения

Создайте приложение для решения задачи. Составьте различные виды тестов для проверки корректности работы. Продемонстрируйте работу приложения в режиме отладки и в обычном режиме.

Варианты заданий

1 Напишите программу для решения одномерного уравнения теплопроводности методом конечных разностей. Протестируйте программу на простых тестовых задачах.

2 Реализуйте метод Гаусса – Зейделя для решения систем линейных уравнений. Протестируйте программу на матрицах различной размерности и сравните скорость сходимости с другими методами.

3 Напишите программу для численного интегрирования функций методом прямоугольников. Протестируйте программу на простых тестовых задачах.

4 Реализуйте метод Ньютона для решения нелинейных уравнений. Протестируйте программу на функциях различной сложности и сравните скорость сходимости с другими методами.

5 Напишите программу для численного решения систем обыкновенных дифференциальных уравнений методом Рунге – Кутты. Протестируйте программу на простых тестовых задачах.

6 Реализуйте метод минимизации функций на основе градиентного спуска. Протестируйте программу на простых тестовых задачах.

7 Напишите программу для численного решения краевых задач для обыкновенных дифференциальных уравнений методом стрельбы. Протестируйте программу на простых тестовых задачах.

8 Реализуйте метод секущих для решения нелинейных уравнений. Протестируйте программу на функциях различной сложности и сравните скорость сходимости с другими методами.

9 Напишите программу для численного решения уравнения Бюргерса методом конечных объемов. Протестируйте программу на простых тестовых задачах.

10 Реализуйте метод Брента для решения нелинейных уравнений. Протестируйте программу на функциях различной сложности и сравните скорость сходимости с другими методами.

Контрольные вопросы

1 Что такое отладка и тестирование программного кода? Каковы их основные цели и принципы работы?

2 Какие виды тестирования программного кода существуют? Какие из них наиболее эффективны для выявления ошибок в коде?

3 Какие инструменты и технологии могут быть использованы для отладки и тестирования программного кода? Какие преимущества и недостатки у каждого из них?

4 Каким образом можно улучшить качество тестирования программного кода? Какие методы и подходы существуют для повышения эффективности тестирования?

5 Каким образом можно эффективно отлаживать программный код? Какие методы и инструменты существуют для поиска и устранения ошибок в коде?

6 Лабораторная работа № 6. Применение стратегий оптимизации кода

Цель работы: приобретение студентами навыков оптимизации кода.

Теоретические сведения

Оптимизация кода – это процесс улучшения производительности и эффективности программного кода. Существует множество стратегий оптимизации кода, вот некоторые из них.

1 Использование эффективных алгоритмов: при написании программного кода необходимо выбирать наиболее эффективные алгоритмы для решения задач, чтобы ускорить работу программы.

2 Уменьшение количества операций: чем меньше операций выполняется в программном коде, тем быстрее он работает. Поэтому необходимо стараться уменьшить количество операций до минимума.

3 Использование более эффективных структур данных: выбор правильных структур данных может значительно ускорить работу программы. Например, использование хэш-таблицы может ускорить поиск элемента в несколько раз по сравнению с обычным списком.

4 Использование многопоточности: при использовании многопоточности можно распределять задачи между несколькими ядрами процессора, что ускоряет работу программы.

5 Избегание избыточных проверок: некоторые проверки могут быть избыточными и замедлять работу программы. Необходимо избегать избыточных проверок и использовать только необходимые.

6 Использование локальных переменных: использование локальных переменных вместо глобальных может ускорить работу программы.

7 Использование компилятора с оптимизацией: современные компиляторы могут проводить оптимизацию кода при компиляции, что может значительно ускорить работу программы.

8 Использование кэшей: использование кэшей может ускорить работу программы, т. к. данные могут быть быстрее доступны из кэша, чем из оперативной памяти.

Это лишь некоторые из стратегий оптимизации кода. Важно помнить, что оптимизация кода не всегда является необходимой, и ее не следует проводить без необходимости. Также важно найти баланс между производительностью и читаемостью кода, чтобы код был понятным и легко поддерживаемым.

Задания для самостоятельного выполнения

Создать структуру данных. Создать массив из элементов структуры. Осуществить чтение данных из файла. При чтении предусмотреть возможность не корректных данных. Произвести анализ полученных данных, выполнить вывод итогов (средние значения, максимальное, минимальное и т. д) в новый файл.

При разработке программы использовать наиболее оптимальные алгоритмы и структуры данных. Обосновать свой выбор.

Варианты заданий

1 Разработать структуру данных для хранения информации о студентах, включая фамилию, имя, номер группы, средний балл и список оценок по каждому предмету.

2 Разработать структуру данных для хранения информации о заказах в интернет-магазине, включая номер заказа, дату заказа, список товаров и стоимость каждого товара.

3 Разработать структуру данных для хранения информации о клиентах банка, включая фамилию, имя, номер счета, баланс и список операций по счету.

4 Разработать структуру данных для хранения информации о книгах в библиотеке, включая название книги, автора, год издания, количество экземпляров и список выданных экземпляров.

5 Разработать структуру данных для хранения информации о поставках товаров на склад, включая название товара, количество, дату поставки и поставщика.

6 Разработать структуру данных для хранения информации о компьютерных играх, включая название игры, жанр, платформу, год выпуска и рейтинг.

7 Разработать структуру данных для хранения информации о пассажирах авиакомпании, включая фамилию, имя, номер билета, маршрут полета и время вылета.

8 Разработать структуру данных для хранения информации о товарах на складе, включая название товара, количество, цену и дату поступления.

9 Разработать структуру данных для хранения информации о покупателях в интернет-магазине, включая фамилию, имя, адрес доставки, список купленных товаров и стоимость каждого товара.

10 Разработать структуру данных для хранения информации о фильмах, включая название фильма, режиссера, актерский состав, длительность и рейтинг.

Контрольные вопросы

1 Что такое оптимизация кода? Каковы основные принципы и цели оптимизации?

2 Какие стратегии оптимизации кода существуют? Какие из них наиболее эффективны для улучшения производительности программного кода?

3 Каким образом можно измерить производительность программного кода? Какие инструменты и методы существуют для оценки производительности?

4 Каким образом можно улучшить производительность программного кода, не ухудшая его читаемость и поддерживаемость? Какие методы и подходы существуют для достижения этих целей?

5 Каким образом можно оптимизировать работу баз данных и сетевых вызовов в программном коде? Какие методы и инструменты существуют для ускорения работы с базами данных и сетью?

7 Лабораторная работа № 7. Применение методик оптимизации логики кода

Цель работы: приобретение студентами навыков оптимизации логики кода.

Теоретические сведения

Существует множество методик оптимизации логики кода программы. Ниже представлены некоторые из наиболее распространенных методик.

Рефакторинг – это процесс изменения кода, не меняя его функциональности. Он может быть использован для улучшения качества кода, повышения производительности и снижения сложности. Рефакторинг может включать в себя различные техники, такие как выделение методов, объединение методов, избавление от дублирования кода, улучшение структуры данных и т. д.

Использование алгоритмов с меньшей сложностью – выбор подходящего алгоритма может существенно повлиять на производительность программы. Иногда оптимальным выбором может быть использование алгоритма с меньшей сложностью, даже если он требует некоторых дополнительных усилий для его реализации.

Уменьшение числа обращений к памяти – обращение к памяти является одной из самых затратных операций в программировании. Уменьшение числа обращений к памяти может привести к существенному увеличению производительности программы. Это может достигаться, например, путем использования локальных переменных, буферизации данных и т. д.

Оптимизация циклов – циклы могут быть критической частью программы, особенно если они выполняются многократно. Оптимизация циклов может включать в себя использование более быстрых операций, уменьшение числа итераций, использование предварительного вычисления значений и т. д.

Использование многопоточности и асинхронного программирования – использование нескольких потоков или асинхронных операций может увеличить производительность программы, особенно если она работает с большим объемом данных или выполняет длительные операции ввода-вывода.

Использование кэширования – кэширование может существенно повысить производительность программы, особенно если она часто обращается к одним и тем же данным. Кэширование может включать в себя сохранение данных в оперативной памяти или на диске, использование механизмов кэширования веб-серверов и т. д.

Использование JIT-компиляции – JIT-компиляция может увеличить производительность программы, особенно для динамических языков программирования. JIT-компиляция позволяет компилировать код в момент выполнения программы, что может уменьшить время загрузки и увеличить скорость выполнения кода.

Использование профайлера – профайлеры позволяют анализировать производительность программы и находить узкие места. Использование профайлера может помочь выявить проблемы в логике кода программы и оптимизировать ее.

Задания для самостоятельного выполнения

Решить задачу. Привести доказательства оптимальности ее логики.

Варианты заданий

1 Написать программу, которая сортирует массив целых чисел методом пузырька, и оптимизировать ее логику.

2 Написать программу, которая реализует игру «Крестики-нолики» и оптимизировать ее логику.

3 Написать программу, которая принимает на вход строку и проверяет, является ли она палиндромом, и оптимизировать ее логику.

4 Написать программу, которая вычисляет факториал числа, используя рекурсию, и оптимизировать ее логику.

5 Написать программу, которая принимает на вход два отсортированных массива и объединяет их в один отсортированный массив, и оптимизировать ее логику.

6 Написать программу, которая принимает на вход строку и удаляет из нее все дубликаты символов, и оптимизировать ее логику.

7 Написать программу, которая принимает на вход два массива целых чисел и находит их пересечение, и оптимизировать ее логику.

8 Написать программу, которая принимает на вход два отсортированных массива и находит их объединение, и оптимизировать ее логику.

9 Написать программу, которая принимает на вход два массива целых чисел и находит их разность, и оптимизировать ее логику.

10 Написать программу, которая принимает на вход строку и находит в ней все подстроки заданной длины, и оптимизировать ее логику.

Контрольные вопросы

1 Что такое методики оптимизации логики кода? Каковы основные принципы и цели оптимизации логики кода?

2 Какие методики оптимизации логики кода существуют? Какие из них наиболее эффективны для улучшения качества программного кода?

3 Каким образом можно выявить и устранить недостатки в логике программного кода? Какие инструменты и методы существуют для этой цели?

4 Каким образом можно сделать код более читаемым и понятным? Какие методы и подходы существуют для улучшения читаемости программного кода?

5 Каким образом можно оптимизировать работу алгоритмов в программном коде? Какие методы и инструменты существуют для ускорения работы алгоритмов и повышения их эффективности?

8 Лабораторная работа № 8. Решение и тестирование задач

Цель работы: приобретение студентами навыков тестирование задач.

Теоретические сведения

Составьте советы по тестированию задачи для студентов.

Конечно, в зависимости от задачи, подход к тестированию может отличаться. Однако, в целом, можно дать несколько советов по тестированию задач для студентов.

1 Начните с тестирования крайних случаев. Это означает, что необходимо протестировать программу на минимальном и максимальном значении, а также на каких-либо необычных или экстремальных кейсах. Например, если программа считает факториал числа, то следует протестировать ее на вводе нуля или на вводе очень большого числа.

2 Проверьте правильность обработки ввода. Очень важно убедиться, что программа обрабатывает входные данные корректно. Необходимо провести тесты на разные типы входных данных, такие как пустые строки, некорректные символы, нулевые значения и т. д.

3 Протестируйте все возможные пути выполнения программы. Это означает, что нужно проверить все возможные варианты исполнения программы и убедиться, что они работают корректно. Например, если программа решает задачу нахождения наибольшего общего делителя двух чисел, то нужно протестировать ее на парах чисел, которые имеют общий делитель и на парах чисел, которые не имеют общего делителя.

4 Проверьте выходные данные. Необходимо убедиться, что программа выдает корректный результат. Например, если программа считает сумму элементов массива, то нужно протестировать ее на массивах разных размеров и с разными значениями элементов.

5 Используйте автоматические тесты. Автоматические тесты позволяют

быстро и эффективно протестировать программу на большом количестве тестовых данных. Использование автоматических тестов позволяет сэкономить время и снизить вероятность ошибок.

6 Проверьте программу на ошибки и исключения. Необходимо убедиться, что программа корректно обрабатывает ошибки и исключения, такие как деление на ноль, выход за границы массива, некорректный ввод данных и т. д.

7 Проверьте программу на производительность. Если программа должна работать с большими объемами данных, то необходимо проверить ее производительность. Например, если программа сортирует массив, то нужно протестировать ее на большом массиве и убедиться, что она работает достаточно быстро.

8 Не забывайте про случайные тесты. Случайные тесты могут помочь выявить неожиданные ошибки и проблемы в программе. Например, если программа генерирует случайные числа, то нужно протестировать ее на разных наборах случайных чисел и убедиться, что результаты корректны.

9 Проверьте программу на многократное использование. Необходимо убедиться, что программа работает корректно при многократном использовании. Например, если программа работает с файлами, то нужно протестировать ее на открытие и закрытие файлов несколько раз.

10 Не забывайте о документации. Не забывайте документировать тесты и результаты их выполнения.

Задания для самостоятельного выполнения

Составьте наборы тестов. Решите задачу и произведите тестирование.

Варианты заданий

1 Реализовать метод бисекции для нахождения корня уравнения f(x) = 0 на заданном интервале [a, b].

2 Реализовать метод Ньютона для нахождения корня уравнения f(x) = 0 с заданной точностью epsilon.

3 Реализовать метод Эйлера для численного решения обыкновенного дифференциального уравнения первого порядка y' = f(x, y) с начальным условием y(x0) = y0.

4 Реализовать метод Рунге – Кутты четвертого порядка для численного решения обыкновенного дифференциального уравнения первого порядка y' = f(x, y) с начальным условием y(x0) = y0.

5 Реализовать метод трапеций для численного вычисления определенного интеграла на заданном интервале [a, b].

6 Реализовать метод Симпсона для численного вычисления определенного интеграла на заданном интервале [a, b].

7 Реализовать метод Гаусса для решения системы линейных уравнений Ax = b.

8 Реализовать метод простой итерации для решения системы линейных уравнений Ax = b.

9 Реализовать метод Якоби для решения системы линейных уравнений Ax = b.

10 Реализовать метод Гаусса – Зейделя для решения системы линейных уравнений Ax = b.

Контрольные вопросы

1 Что такое задача? Каковы основные шаги решения задачи?

2 Каким образом можно разработать алгоритм решения задачи? Какие методы и инструменты могут помочь в этом процессе?

3 Каким образом можно проверить правильность решения задачи? Какие методы и инструменты используются для тестирования программного кода?

4 Каким образом можно оптимизировать алгоритм решения задачи? Какие методы и подходы существуют для улучшения производительности программного кода?

5 Каким образом можно документировать решение задачи? Какие элементы должны входить в документацию решения задачи, и зачем они нужны?

9 Лабораторная работа № 9. Выбор оптимальной стратегии и решение задачи

Цель работы: приобретение студентами навыков выбора оптимальной стратегии и решение задачи.

Теоретические сведения

Выбор оптимальной стратегии решения задачи в программировании - это процесс выбора наилучшего пути решения задачи, который будет иметь наилучшую производительность, скорость и точность. Для выбора оптимальной стратегии решения задачи в программировании можно использовать следующие тактики.

1 Анализ задачи: необходимо понять, что должно быть достигнуто, каким образом, какими данными и алгоритмами можно использовать для достижения желаемых результатов.

2 Изучение алгоритмов: необходимо изучить различные алгоритмы, которые могут быть применены для решения данной задачи.

3 Оценка производительности: необходимо оценить производительность каждого алгоритма, который может быть использован для решения задачи. Это может включать в себя оценку сложности алгоритма, используемой памяти и времени выполнения.

4 Тестирование: необходимо протестировать каждую реализацию алгоритма, чтобы убедиться, что она работает правильно и эффективно.

5 Оптимизация: после тестирования можно определить, какие алгоритмы работают лучше, а какие – хуже. Необходимо оптимизировать лучшие алгоритмы для достижения наилучшей производительности.

6 Компромисс: иногда выбор наилучшей стратегии может быть компромиссом между производительностью, точностью и эффективностью реализации.

7 Учет особенностей задачи: необходимо учитывать особенности задачи при выборе стратегии, например, если задача имеет большой объем данных, то может потребоваться использование алгоритмов с более высокой сложностью, но при этом с меньшим объемом использования памяти.

8 Обратная связь: после реализации стратегии необходимо получить обратную связь от пользователей и проанализировать ее, чтобы определить, какие улучшения можно внести в стратегию.

9 В целом, выбор оптимальной стратегии решения задачи в программировании – это процесс, который требует тщательного анализа и тестирования различных вариантов решения, а также учета особенностей задачи и обратной связи от пользователей.

Задания для самостоятельного выполнения

Опишите стратегию решения и решите задачу. В отчете приведите доказательства оптимальности решения.

Варианты заданий

1 Написать программу, которая шифрует текст методом Цезаря (сдвиг на определенное количество символов) с заданным ключом.

2 Написать программу, которая шифрует текст методом перестановки (меняет местами символы в строке) с заданным ключом.

3 Написать программу, которая шифрует текст методом замены (заменяет символы на другие символы из заданного алфавита) с заданным ключом.

4 Написать программу, которая шифрует текст методом XOR (использует операцию исключающего ИЛИ для каждого символа сообщения и ключа) с заданным ключом.

5 Написать программу, которая шифрует текст методом повторных блоков (разбивает сообщение на блоки и повторяет каждый блок по несколько раз) с заданным ключом.

6 Написать программу, которая шифрует текст методом решетки (использует матрицу, чтобы скрыть символы сообщения) с заданным ключом.

7 Написать программу, которая шифрует текст методом перебора (пробует все возможные ключи, пока не найдет правильный) с заданным алгоритмом шифрования.

8 Написать программу, которая шифрует текст методом забывчивого шифра (использует таблицу, чтобы заменить символы в сообщении) с заданным ключом.

9 Написать программу, которая шифрует текст методом ROT13 (сдвигает каждый символ на 13 позиций в алфавите) без ключа.

10 Написать программу, которая шифрует текст методом базового 64 (использует 64 символа для кодирования) без ключа.

Контрольные вопросы

1 Что такое стратегия решения задачи? Каким образом можно выбрать оптимальную стратегию для решения задачи?

2 Какие методы и инструменты используются для анализа задачи и определения оптимальной стратегии для ее решения?

3 Каким образом можно проверить правильность выбранной стратегии решения задачи? Какие методы и инструменты используются для тестирования программного кода?

4 Каким образом можно оптимизировать выбранную стратегию решения задачи? Какие методы и подходы существуют для улучшения производительности программного кода?

5 Каким образом можно документировать выбранную стратегию решения задачи? Какие элементы должны входить в документацию выбранной стратегии, и зачем они нужны?

10 Лабораторная работа № 10. Анализ условия, постановка задания и решение задачи

Цель работы: приобретение студентами навыков анализа условия и постановка задания необходимых для решения задачи.

Теоретические сведения

Анализ предметной области и составление технического задания на разработку программы включает в себя несколько правил.

1 Изучение предметной области. Необходимо изучить все аспекты предметной области, включая бизнес-процессы, потребности пользователей, конкурентов и т. д.

2 Определение требований. Определите требования к программе, основываясь на изученной предметной области. Важно определить функциональные и нефункциональные требования, а также требования к качеству и безопасности.

3 Определение архитектуры. На основе требований определите архитектуру программы, включая выбор платформы, языка программирования, базы данных и т. д.

4 Разработка дизайна. Разработайте дизайн пользовательского интерфейса, который будет интуитивно понятен и удобен для пользователей.

5 Разработка тест-плана. Разработайте тест-план, который будет использоваться для проверки программы на соответствие требованиям.

6 Разработка технической документации. Разработайте техническую документацию, включая описание архитектуры, дизайна, требований, тест-плана и т. д.

7 Проверка и утверждение. Проверьте техническое задание на соответствие требованиям и утвердите его.

8 Следите за изменениями. Следите за изменениями в предметной области и вносите изменения в техническое задание при необходимости.

Следуя этим правилам, можно создать качественное техническое задание на разработку программы, которое будет соответствовать требованиям заказчика и потребностям пользователей.

Задания для самостоятельного выполнения

Произвести анализ предметной области. Составить техническое задание для своего варианта.

Варианты заданий

1 Создание программы-органайзера, которая будет показывать задачи на текущий день и позволять добавлять новые.

2 Создание программы для загрузки данных из внешних источников и их обработки, например, для получения погоды или курсов валют.

3 Создание программы для расчета и вывода финансовых отчетов на основе введенных пользователем данных.

4 Создание программы для генерации случайных паролей и их сохранения в файл.

5 Создание программы для создания и редактирования текстовых файлов с возможностью сохранения изменений.

6 Создание программы для генерации случайных чисел и их сортировки по возрастанию или убыванию.

7 Создание программы для рисования графиков на основе введенных пользователем данных.

8 Создание программы для автоматического создания резервных копий файлов и папок.

9 Создание программы для генерации идентификаторов, таких как номера заказов или идентификаторы продуктов.

10 Создание программы для управления файлами и папками на компьютере, включая копирование, перемещение, удаление и переименование.

Контрольные вопросы

1 Что такое анализ условия и постановка задания? Какой процесс следует при этом для успешного решения задачи?

2 Как можно провести анализ условия задачи? Какие методы и инструменты используются для этого?

3Каким образом можно определить подходящий алгоритм для решения задачи? Какие методы и инструменты используются для выбора алгоритма?

4 Каким образом можно реализовать выбранный алгоритм для решения задачи? Какие языки программирования и инструменты могут использоваться для этого?

5 Каким образом можно проверить правильность решения задачи? Какие методы и инструменты используются для тестирования программного кода и проверки корректности решения задачи?

11 Лабораторная работа № 11. Решение комплексной задачи

Цель работы: приобретение студентами навыков решение комплексной задачи.

Теоретические сведения

Данное решение подразумевает выполнение всех аспектов, рассмотренных выше. Код программы должен быть оформлен и структурирован грамотно и сопровождаться комментариями.

Принципы построения кода должны соответствовать принципам оптимальности и качества.

Задание для самостоятельного выполнения

На основании результатов лабораторной работы № 10, разработать приложение в соответствии с техническим заданием.

Контрольные вопросы

1 Как была спроектирована архитектура программы? Какие решения были приняты для обеспечения ее оптимальной работы?

2 Какие алгоритмы и структуры данных использовались для решения поставленной задачи? Были ли проведены исследования и оптимизации для улучшения производительности программы?

3 Как был организован процесс тестирования программы? Какие тесты были проведены и какие результаты были получены?

4 Как было организовано взаимодействие с пользователем? Были ли рассмотрены и решены вопросы, связанные с удобством использования программы?

5 Какие инструменты были использованы для разработки программы? Были ли выбраны оптимальные инструменты для решения поставленной задачи? Какие инструменты использовались для управления версиями и совместной работы над проектом?

Список литературы

1 **Мякишев, Д. В.** Разработка программного обеспечения АСУ ТП на основе объектно-ориентированного подхода : методическое пособие / Д. В. Мякишев. – Москва; Вологда : Инфра-Инженерия, 2019. – 128 с.

2 **Павловская, Т. А.** C#. Программирование на языке высокого уровня: учебник для вузов / Т. А. Павловская. – Санкт-Петербург: Питер, 2019. – 432 с.

3 **Васильев, А. В.** Программирование на C# для начинающих. Основные сведения / А. В. Васильев. – Москва: Эксмо, 2018. – 592 с.

4 **Шилд, Г.** C# 4.0: полное руководство: пер. с англ. / Г. Шилд. – Москва: И. Д. Вильямс, 2011. – 1056 с.

5 **Подбельский, В. В.** Язык Си#. Базовый курс: учебное пособие / В. В. Подбельский. – Москва: Финансы и статистика, 2011. – 384 с.

6 **Рихтер, Дж**. CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C# / Дж. Рихтер. – Санкт-Петербург: Питер, 2013. – 896 с.

7 Объектно-ориентированное программирование на языке C#: методические рекомендации к выполнению лабораторных работ / Сост. Л. Е. Потапова, Т. Г. Алейникова. – Витебск: ВГУ им. П. М. Машерова, 2016. – 50 с.

8 Справочная онлайн-библиотека для разработчиков на платформе Microsoft.NET [Электронный ресурс]. – Режим доступа: http://msdn.microsoft.com. – Дата доступа: 20.06.2023.