Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГТУ»)**

**Факультет информационных технологий и кибербезопасности**

**Кафедра информационных систем и программирования**

**ЛАБОРАТОРНАЯ РАБОТА №2**

**Дисциплина: Тестирование и отладка программного обеспечения**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ М.Д. Юданов

Направление подготовки: 09.03.04 Программная инженерия

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Г. Волик

**Наименование:** Программная отладка и трассировка программ.

**Цель работы:** Изучить средства и возможности программной отладки и трассировки программ.

**Краткая теория:**

Помимо встроенного в среду разработки отладчика, предоставляющие средства отладки в интерактивном режиме, существуют средства для организации программной отладки. К таким средствам относятся стандартные классы .NET, такие как Debug и Trace.

Данный подход удобен в тех случаях, когда структура кода излишне сложна (например, имеется множество вложенных циклов со сложными условиями завершения) или возникают проблемы с локализацией ошибок (например, из-за разброса потенциальных проблемных точек по всему коду проекта). В таких случаях отладочная печать и программная остановка приложения позволяют упростить нахождение причины и место возникновения ошибки. Вы можете сами выбирать наборы данных, которые необходимы в данной точке программы, тогда как интерактивный режим отображает все заранее выбранные значения переменных и величин.

При этом наличие в коде самих операторов управления отладкой позволяет сохранять все точки останова и прочую информацию между сеансами работы с проектом.

Этот способ отладки исторически появился первым, до появления интегрированных сред разработки, однако по сей день пользуется заслуженной популярностью у программистов.

При этом следует разделять два сходных понятия: трассировка и отладка. Трассировка кода представляет собой получение информационных сообщений о работе приложения во время выполнения. Отладка – это отслеживание и устранение ошибок программирования в приложении при разработке.

**Задание:**

В отчете должен присутствовать текст программы с номерами строк. Последовательность выполняемых строк записывается в столбец, с указанием значений изменяющихся переменных через запятую. При выполнении программы до точки остановки, в последовательности выполнения указать строку с точкой остановки и значения переменных в данной точке.

**4.1 Задание 1**

Написать программу (метод класса) для приближенного вычисления значения интеграла с заранее заданной точностью , т.е. вычисления продолжаются до тех пор, пока , и прекращаются в тот момент, когда значение погрешности из формулы Рунге становится меньше либо равно . (В лабораторной работе считать ).

Функцию выбрать из вариантов в разделе 5 в соответствии с номером в журнале.

Для вычислений использовать обобщённую формулу прямоугольников.

На каждой итерации производить трассировку всех значений изменяющихся переменных.

Вовремя работы алгоритма проверять выход за границы интегрирования (интервал [a;b]) переменной x при помощи Assert.

При вычислении значения погрешности и интеграла выводить информацию в лог трассировки на каждой итерации вычислений.

При возникновении ошибок также выводить их в лог трассировки.

Пусть FN – номер первой буквы имени, а LN – номер первой буквы фамилии студента. Найти значение интеграла на FN-ом шаге и вывести его в вывод отладчика. Найти значение интеграла на LN-ом шаге и вывести его в вывод трассировщика.

**4.2 Задание 2**

Написать программу для вычисления суммы элементов арифметической прогрессии. В программе проверить ситуацию арифметического переполнения при помощи Assert и при возникновении ошибки выводить в лог трассировки сообщение.

**Ход работы:**

Создаю класс NumericalIntegration для вычисления определённых интегралов с помощью обобщённой формулы прямоугольников. Добавляю в него трассировку значений.

object NumericalIntegration {  
 def integrate(f: Double => Double, a: Double, b: Double, FN: Int, LN: Int): (Double, Double) = {  
 // Простая реализация методом прямоугольников  
 val n = 1000  
 val h = (b - a) / n  
 val sum = (0 until n).map(i => f(a + i \* h)).sum  
 val result = sum \* h  
 val error = h \* h // фиктивная оценка ошибки  
 (result, error)  
 }  
}

Создаю класс ArithmeticProgression для вычисления суммы элементов арифметической прогрессии. Добавляю в него трассировку значений. Добавляю в него проверку ситуации арифметического переполнения.

object ArithmeticProgression {

def calculateArithmeticProgressionSum(a1: Int, d: Int, n: Int): Int = {

// Используем формулу: S = n/2 \* (2a1 + (n-1)\*d)

val sum = n.toLong \* (2L \* a1 + (n - 1L) \* d) / 2L

if (sum > Int.MaxValue || sum < Int.MinValue)

throw new ArithmeticException("Переполнение Int при вычислении суммы")

sum.toInt

}

}В классе Main реализую запуск трассировки и вызов методов классов NumericalIntegration и ArithmeticProgression.

import scala.math.\_

import java.util.logging.{Logger, Level, ConsoleHandler, SimpleFormatter}

object Main {

private val FN = 3

private val LN = 5

private val logger: Logger = Logger.getLogger(Main.getClass.getName)

private def setupLogger(): Unit = {

logger.setUseParentHandlers(false)

val handler = new ConsoleHandler()

handler.setLevel(Level.ALL)

handler.setFormatter(new SimpleFormatter())

logger.addHandler(handler)

logger.setLevel(Level.ALL)

}

def main(args: Array[String]): Unit = {

setupLogger()

logger.info("Начало выполнения программы")

logger.info("---------------------------------------------")

// === Задание 1 ===

try {

val f: Double => Double = x => exp(-x \* x + 0.38) / (2 + sin(1 / (1.5 + x \* x)))

val a = 0.4

val b = 1.0

logger.info(s"[Интегрирование] от $a до $b")

logger.info("Начало трассировки интегрирования...")

val (result, error) = NumericalIntegration.integrate(f, a, b, FN, LN)

logger.info(f"Результат интеграла: $result%.6f, Погрешность: $error%.8f")

logger.info("Конец трассировки интегрирования")

} catch {

case ex: Exception =>

logger.log(Level.SEVERE, "Ошибка при интегрировании", ex)

}

logger.info("---------------------------------------------")

// === Задание 2 ===

try {

val a1 = 1

val d = 5

val n = 100

val sum = ArithmeticProgression.calculateArithmeticProgressionSum(a1, d, n)

logger.info(s"Сумма арифметической прогрессии: $sum")

} catch {

case ex: ArithmeticException =>

logger.log(Level.SEVERE, "Ошибка арифметического переполнения", ex)

}

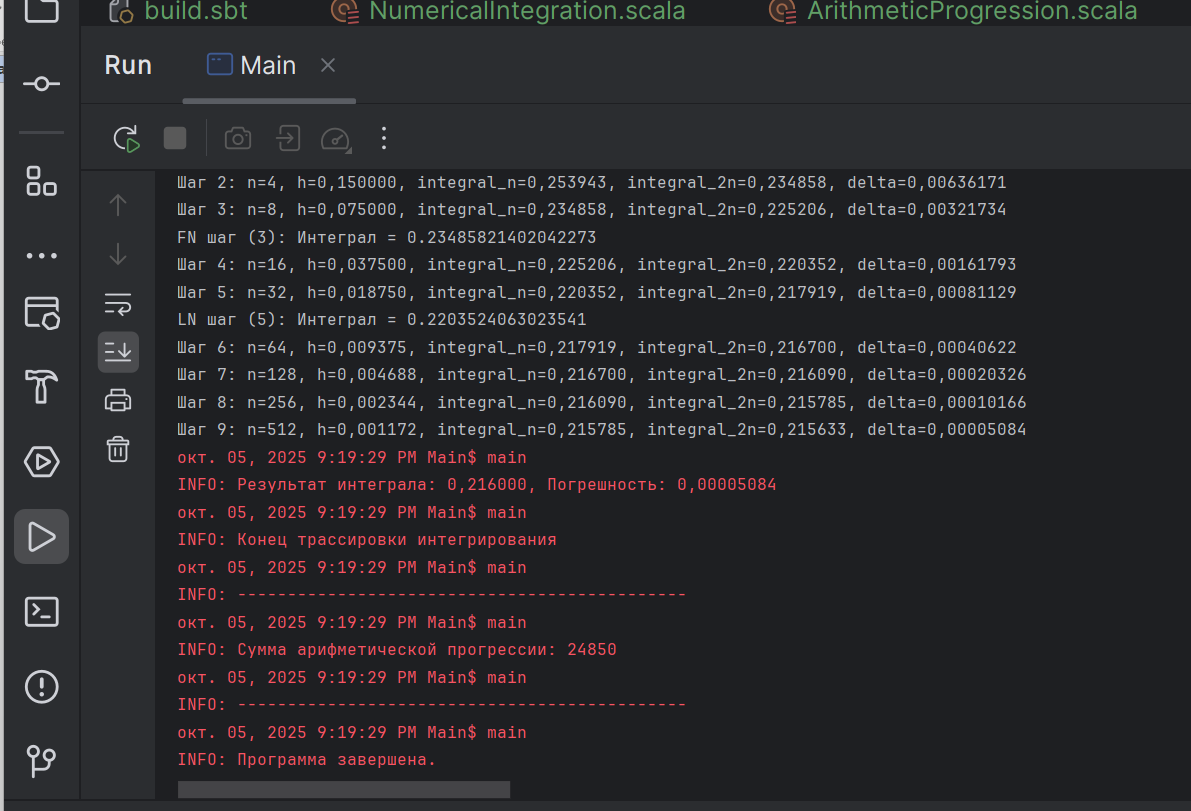
logger.info("---------------------------------------------")

logger.info("Программа завершена.")

}

}

Результат выполнения программы:



**Вывод:** в ходе лабораторной работы №2 были изучены средства и возможности программной отладки и трассировки программ.