Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ   
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

по лабораторной работе №2

на тему:

**ЦИКЛЫ, ДИАПАЗОНЫ И МАССИВЫ**

Проверил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ф.В. Усенко

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А. Н. Кубицкая

(подпись) гр. 214302

Минск, 2024

**Задание:** Реализуйте программу, которая решает квадратное уравнение, а затем находит все целые решения (если таковые существуют) в заданном диапазоне. Программа должна учитывать комплексные корни и их дискретизацию.

**Решение:**

import kotlin.math.sqrt  
import kotlin.math.round  
  
var *imaginaryPart*: Double = 0.0  
  
*// Функция для решения квадратного уравнения*fun solveQuadratic(a: Double, b: Double, c: Double): Pair<Any, Any> { *// пара значений* val discriminant = b \* b - 4 \* a \* c  
  
 return if (discriminant > 0) {  
 *// Два действительных корня* val root1 = (-b + *sqrt*(discriminant)) / (2 \* a)  
 val root2 = (-b - *sqrt*(discriminant)) / (2 \* a)  
 Pair(root1, root2)  
 } else if (discriminant == 0.0) {  
 *// Один действительный корень* val root = -b / (2 \* a)  
 Pair(root, root)  
 } else {  
 *// Комплексные корни* val realPart = -b / (2 \* a)  
 *imaginaryPart* = *sqrt*(-discriminant) / (2 \* a)  
 Pair(complex(realPart, *imaginaryPart*), complex(realPart, -*imaginaryPart*))  
 }  
}  
  
*// Класс для представления комплексных чисел*data class complex(val real: Double, val imaginary: Double) {*// data тк есть метод toString* override fun toString(): String { *// переопределяет стандартный метод toString() из класса Any, который используется для представления объекта в виде строки.* return if (imaginary >= 0) "$real + ${imaginary}i" else "$real - ${-imaginary}i"  
 }  
}  
  
*// Функция для поиска целых решений в диапазоне*fun findIntegerSolutions(a: Double, b: Double, c: Double, p: Int, q: Int): IntArray{  
 val roots = *solveQuadratic*(a, b, c)  
 *println*("Корни уравнения: $roots")  
 val integerSolutions = *mutableListOf*<Int>()  
  
 val firstRoot = roots.first  
 val secondRoot = roots.second  
  
 if (firstRoot is Double && secondRoot is Double) {  
 val root1 = *round*(firstRoot).toInt() *//округление* val root2 = *round*(secondRoot).toInt()  
  
 val rootList = *listOf*(root1, root2)  
 for (r in rootList) { *//Начинаем цикл по элементам списка rootList* if (r in p..q && *isValidIntegerRoot*(a, b, c, r)) {  
 integerSolutions.add(r)  
 }  
 }  
 } else if (firstRoot is complex && secondRoot is complex) {  
 if (*imaginaryPart*.toInt() ==0 ){  
 val realParts = *listOf*(  
 *round*((firstRoot).real).toInt(),*//real-извлечение действительной части  
 round*((secondRoot).real).toInt()  
 )  
  
 for (r in realParts) {  
 if (r in p..q && *isValidIntegerRoot*(a, b, c, r)) {  
 integerSolutions.add(r)  
 }  
 }  
 }  
 }  
 return integerSolutions.*toIntArray*()  
}  
  
*// Проверка, является ли значение x целым решением уравнения*fun isValidIntegerRoot(a: Double, b: Double, c: Double, x: Int): Boolean {  
 return a \* x \* x + b \* x + c == 0.0  
}  
  
*// Функция для считывания числа с консоли с проверкой*fun readDouble(world: String): Double {  
 *print*(world)  
 val input = *readLine*()?.*toDoubleOrNull*()  
  
 if (input != null) {  
 return input  
 }  
  
 *println*("Пожалуйста, введите корректное число.")  
 return *readDouble*(world) *// Рекурсивный вызов для повторного ввода*}  
  
  
*// Функция для считывания целого числа с консоли с проверкой*fun readInt(world: String): Int {  
 *print*(world)  
 val input = *readLine*()?.*toIntOrNull*()  
 if (input != null) {  
 return input  
 }  
  
 *println*("Пожалуйста, введите корректное целое число.")  
 return *readInt*(world)  
}  
  
fun main() {  
 val a = *readDouble*("Введите коэффициент a: ")  
 val b = *readDouble*("Введите коэффициент b: ")  
 val c = *readDouble*("Введите коэффициент c: ")  
  
 var p: Int  
 var q: Int  
  
 while (true) {  
 p = *readInt*("Введите начальное значение диапазона p: ")  
 q = *readInt*("Введите конечное значение диапазона q: ")  
  
 if (p <= q) {  
 break  
 } else {  
 *println*("Ошибка: начальное значение диапазона не может быть больше конечного. Пожалуйста, введите значения заново.")  
 }  
 }  
  
  
 val solutions = *findIntegerSolutions*(a, b, c, p, q)  
  
 if (solutions.*isNotEmpty*()) {  
 *println*("Целые решения в диапазоне [$p, $q]: ${solutions.*joinToString*(", ")}") *//joinToString(", ") преобразует массив в строку, где элементы разделены запятыми* } else {  
 *println*("Целых решений в диапазоне [$p, $q] нет.")  
 }  
}

**Контрольные вопросы:**

1. Какие типы циклов существуют в Kotlin, и как они используются?

**Цикл for:** Используется для выполнения набора инструкций заданное количество раз или для итерации по элементам коллекции.

**Цикл while:** Выполняет блок кода до тех пор, пока условие истинно. Если условие становится ложным, выполнение цикла прекращается.

**Цикл do...while:** Отличается от while тем, что сначала выполняется тело цикла, а затем проверяется условие.

1. Чем отличается цикл while от цикла for?

while выполняется, пока условие истинно, а for проходит по элементам коллекции или диапазона.

1. Какую роль выполняет цикл do...while, и в чем его отличие от while?

Цикл do...while в программировании используется для выполнения блока кода хотя бы один раз, а затем повторения этого блока, пока условие истинно. Основное отличие от цикла while заключается в том, что проверка условия в do...while происходит после выполнения тела цикла, а не до.

1. Что такое диапазон в kotlin, и как его создать?

Диапазон в Kotlin представляет собой последовательность значений, определенную начальным и конечным значениями. Диапазоны могут быть использованы для итерации в циклах и проверки принадлежности значений.

val range = 1..5

val reverseRange = 5 downTo 1

val stepRange = 1..10 step 2