Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерного проектирования

Кафедра проектирования информационно-компьютерных систем

Отчёт

К лабораторной работе № 2

на тему:

**ЦИКЛЫ, ДИАПАЗОНЫ И МАССИВЫ**

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

(подпись)

Выполнил \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ П.А. Барбук

(подпись) 214302

Минск 2024

**Вариант 1: Создание магического квадрата.**

Условие: Напишите программу, которая создает магический квадрат заданного размера (матрица, где суммы чисел в каждой строке, столбце и диагонали равны). Программа должна проверять, что введенное число является допустимым для создания магического квадрата

**Код программы:**

fun main() {

println("Введите размер магического квадрата:")

val n = readLine()?.toIntOrNull()

if (n == null || n <= 0) {

println("Некорректный ввод. Введите положительное число.")

return

}

val magicSquare = when {

n % 2 == 1 -> generateOddMagicSquare(n)

n % 4 == 0 -> generateDoublyEvenMagicSquare(n)

else -> generateSinglyEvenMagicSquare(n)

}

println("Магический квадрат размером $n:")

for (row in magicSquare) {

println(row.joinToString(" "))

}

isMagicSquare(magicSquare)

}

fun generateOddMagicSquare(n: Int): Array<IntArray> {

val magicSquare = Array(n) { IntArray(n) }

var number = 1

var i = 0

var j = n / 2

while (number <= n \* n) {

magicSquare[i][j] = number

number++

val newi = (i - 1 + n) % n

val newj = (j + 1) % n

if (magicSquare[newi][newj] != 0) {

i = (i + 1) % n

} else {

i = newi

j = newj

}

}

return magicSquare

}

fun generateDoublyEvenMagicSquare(n: Int): Array<IntArray> {

val magicSquare = Array(n) { IntArray(n) }

for (i in 0 until n) {

for (j in 0 until n) {

val cellNumber = i \* n + j + 1

magicSquare[i][j] = if (i % 4 == j % 4 || (i % 4 + j % 4) == 3) n \* n + 1 - cellNumber else cellNumber

}

}

return magicSquare

}

fun generateSinglyEvenMagicSquare(n: Int): Array<IntArray> {

val halfN = n / 2

val subSquareSize = halfN \* halfN

val subSquare = generateOddMagicSquare(halfN)

val magicSquare = Array(n) { IntArray(n) }

for (i in 0 until halfN) {

for (j in 0 until halfN) {

magicSquare[i][j] = subSquare[i][j]

magicSquare[i + halfN][j + halfN] = subSquare[i][j] + subSquareSize

magicSquare[i][j + halfN] = subSquare[i][j] + 2 \* subSquareSize

magicSquare[i + halfN][j] = subSquare[i][j] + 3 \* subSquareSize

}

}

val shift = halfN / 2

for (i in 0 until halfN) {

for (j in 0 until shift) {

if (i == shift) {

if (j == 0) {

swap(magicSquare, i, j, i + halfN, j)

} else {

swap(magicSquare, i, n - j - 1, i + halfN, n - j - 1)

}

} else {

swap(magicSquare, i, j, i + halfN, j)

}

}

}

return magicSquare

}

fun swap(magicSquare: Array<IntArray>, i1: Int, j1: Int, i2: Int, j2: Int) {

val temp = magicSquare[i1][j1]

magicSquare[i1][j1] = magicSquare[i2][j2]

magicSquare[i2][j2] = temp

}

fun isMagicSquare(square: Array<IntArray>): Boolean {

val n = square.size

val targetSum = (n \* (n \* n + 1)) / 2

for (i in 0 until n) {

val rowSum = square[i].sum()

if (rowSum != targetSum) {

println("Матрица не является магическим квадратом. Строка $i не совпадает с целевым значением.")

return false

}

}

for (j in 0 until n) {

var colSum = 0

for (i in 0 until n) {

colSum += square[i][j]

}

if (colSum != targetSum) {

println("Матрица не является магическим квадратом. Столбец $j не совпадает с целевым значением.")

return false

}

}

var mainDiagonalSum = 0

for (i in 0 until n) {

mainDiagonalSum += square[i][i]

}

if (mainDiagonalSum != targetSum) {

println("Матрица не является магическим квадратом. Главная диагональ не совпадает с целевым значением.")

return false

}

var secondaryDiagonalSum = 0

for (i in 0 until n) {

secondaryDiagonalSum += square[i][n - i - 1]

}

if (secondaryDiagonalSum != targetSum) {

println("Матрица не является магическим квадратом. Побочная диагональ не совпадает с целевым значением.")

return false

}

println("Матрица является магическим квадратом.")

return true

}

**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 (например, массиву).

– 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. Как работает цикл *for* для итерации по элементам массива? Приведите пример.

– Данный цикл последовательно перебирает все элементы массива и выполняет заданную инструкцию. Пример:

val numbers = arrayOf(1, 2, 3, 4, 5)

for (number in numbers) {

println(number)

}

1. Чем отличается цикл *while* от цикла *for*?

– Цикл *while* выполняется, пока не будет нарушено условие выполнения (*Boolean*). Цикл *for* выполняется, пока не будет выполнено арифметическое условие.

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

– Данный цикл выполняет ту же роль, что и цикл *while*, только, в отличие от while, цикл выполнится в любом случае хотя бы один раз.

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

– Диапазоном называется набор последовательных значений [*a,b*]. Диапазоны определяются с помощью оператора ... Диапазоны могут использоваться для итераций в циклах или для проверки принадлежности значению. Пример:

val x = 3

if (x in 1..5) {

println("$x is in the range")

}

1. Как использовать оператор *in* для проверки принадлежности значения диапазону?

Оператор *in* используется для проверки, находится ли значение в заданном диапазоне. Например, чтобы проверить, находится ли число 5 в диапазоне от 1 до 10*: if (5 in 1..10)*

1. Оператор !*in* используется для проверки, что значение не находится в заданном диапазоне. Например, чтобы проверить, что число 15 не находится в диапазоне от 1 до 10: *if (15 !in 1..10)*
2. Реверсированные диапазоны создаются с помощью метода *downTo*. Например, чтобы создать диапазон от 10 до 1*: for (i in 10 downTo 1)*
3. Шаг диапазона задается с помощью метода *step*. Например, чтобы создать диапазон от 1 до 10 с шагом 2: *for (i in 1..10 step 2)*
4. Массив чисел создается с помощью функции *arrayOf*. Например: *val* массив = *arrayOf*(1, 2, 3)
5. Обращение к элементу массива по индексу выполняется с помощью квадратных скобок. Например, чтобы получить первый элемент массива: массив[0]. Если индекс выйдет за пределы массива, будет выброшено исключение *ArrayIndexOutOfBoundsException*.
6. Значение элемента массива изменяется с помощью квадратных скобок. Например, чтобы изменить значение второго элемента массива: массив[1] = 42
7. Перебор элементов массива выполняется с помощью цикла *for*. Например: *for* (элемент *in* массив) { *println*(элемент) }
8. Массив фиксированного размера — это массив, размер которого задается при создании и не может быть изменен. Например, чтобы создать массив из 5 элементов: *val* массив = *Array*(5) { 0 }
9. Массив чисел сортируется с помощью метода *sort*. Например: массив.*sort*()
10. Для удаления элемента из массива нужно создать новый массив, не содержащий этот элемент. Например: *val* новыйМассив = массив.*filter* { *it* != 2 }.*toTypedArray*()
11. Метод *joinToString* создает строку из элементов массива с заданным разделителем. Метод *sort* сортирует массив. Например: массив.*joinToString*(", ") и массив.*sort*()
12. Два массива объединяются с помощью оператора +. Например: *val* объединенныйМассив = массив1 + массив2
13. Упорядоченность массива проверяется с помощью метода *zipWithNext*. Например: val упорядочен *= массив.zipWithNext().all { it.first <= it.second* }
14. Количество элементов, соответствующих условию, подсчитывается с помощью метода *count*. Например: *val* количество = массив.count { *it* % 2 == 0 }