Лабораторна робота №3  
Розгалуження

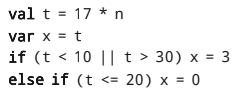
* 1. Мета:

Отримання практичних навичок з розробки програм, що реалізують алгоритми з розгалуженням.

* 1. Завдання:

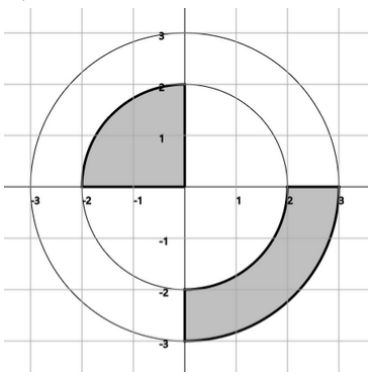
1. Представити математичний запис фрагмента програми та обчислити значення змінної X після його виконання. Позначення: n – номер варіанту.

Завдання для мого (другого) варіанту зображено нижче:



Завдання для другого варіанту

1. Написати функцію, що виводить на екран значення true, якщо точка А з координатами х, у належить до заштрихованої області, та false в іншому випадку. Рисунок з заштрихованими областями зображено на рисунку нижче.



Зображення заштрихованих і не заштрихованих областей

1. Скласти програму, що містить опис функції, яка задана графічно. Доповнити програму функцією main, яка викликає створену функцію для значення аргументу x, що треба вводити з клавіатури, та виводить результат обчислень на консоль. Зображення функції див. нижче:



Графічно задана функція для 2 варіанту

* 1. Теоретична частина

У програмуванні часто виникають ситуації, коли обчислення результату функцій вимагає розгляду різних варіантів або умов. На відміну від простіших випадків, де виконання коду відбувається без альтернативних шляхів, значна частина завдань потребує гнучкості, щоб програма могла реагувати на різні вхідні дані чи стани. Для вирішення таких завдань у мовах програмування існують спеціальні конструкції розгалуження.

Однією з основних конструкцій розгалуження в Kotlin є оператор if..else. Його назва перекладається з англійської як "якщо..інакше". Після ключового слова if в дужках зазначається умова розгалуження. Якщо ця умова виявляється істинною, тобто true, то в якості результату використовується вираз, що стоїть одразу за умовою розгалуження. У випадку, коли умова хибна, тобто false, застосовується вираз, розташований після ключового слова else. Наприклад, функція, що визначає максимум з двох чисел, може бути записана як

fun max(m: Int, n: Int) = if (m > n) m else n

Це читається як "Якщо m > n, (то) m, інакше n". Важливо зазначити, що у мові Kotlin така функція max вже існує у стандартній бібліотеці. Для вирішення більш складних завдань, що передбачають кілька послідовних перевірок умов, конструкцію if..else можна використовувати в так званому "каскадному" записі у вигляді if..else if..else if..else. Це дозволяє додати необмежену кількість проміжних елементів else if. Прикладом такого застосування є обчислення кількості коренів квадратного рівняння

Кількість коренів залежить від значення дискримінанта d: якщо d > 0.0, то коренів два; якщо d == 0.0, то корінь один; в іншому випадку коренів немає. В Kotlin це реалізується виразом

return if (d > 0.0) 2 else if (d == 0.0) 1 else 0

Це означає "повернути: якщо d > 0.0, (то) 2, інакше, якщо d = 0.0, (то) 1, інакше 0".

Альтернативою каскадному if..else та більш витонченим способом представлення розгалужень у "табличній" формі є конструкція when. Вона складається з послідовності записів виду умова -> результат. В останньому записі умова замінюється на ключове слово else, що діє як випадок "інакше" або значення за замовчуванням. when особливо зручна, коли один і той же вираз потрібно послідовно порівняти на рівність з кількома іншими значеннями. Наприклад, функція:

fun gradeNotation(grade: Int): String = when (grade) {

5 -> "відмінно"

4 -> "добре"

3 -> "задовільно"

2 -> "незадовільно"

else -> "неіснуюча оцінка $grade"

}

Ця функція використовує when для перетворення числової оцінки на словесну нотацію (наприклад, "5" як "відмінно", "4" як "добре" тощо). У виразі when (grade) значення grade послідовно порівнюється з 5, 4, 3 та 2. Наявність гілки else у when є обов'язковою у багатьох випадках, оскільки функція має повертати результат для будь-якого припустимого вхідного значення. У прикладі з оцінками гілка else може обробляти помилкові ситуації, формуючи рядок "неіснуюча оцінка", до якої додається значення переданої оцінки. Конструкція when також може бути використана для реалізації логіки, подібної до каскадного if..else, як у прикладі з quadraticRootNumber, що робить код більш читабельним:

return when { d > 0.0 -> 2 d == 0.0 -> 1 else -> 0 }

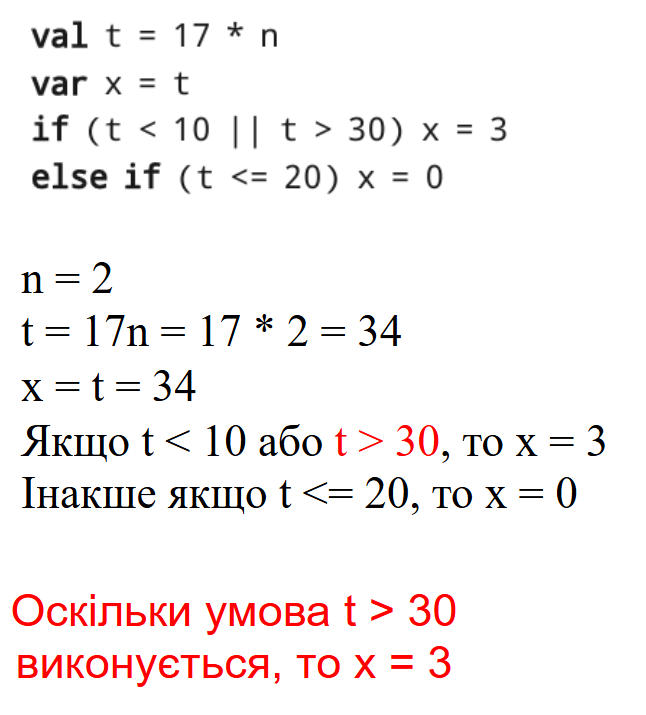
Умови в конструкціях розгалуження формуються за допомогою операцій порівняння. У мові Kotlin існує вісім таких операцій: > (строго більше), >= (більше або дорівнює, аналог математичного ≥), < (строго менше), <= (менше або дорівнює, аналог математичного ≤). Також існують операції для перевірки належності до інтервалу: x in a..b (означає, що x належить інтервалу від a до b, що є аналогом математичного a ≤ x ≤ b) та x !in a..b (означає, що x НЕ належить інтервалу). Для перевірки рівності та нерівності використовуються оператори == (дорівнює, з двома знаками рівності, щоб уникнути плутанини з оператором присвоєння =) та != (не дорівнює, аналог математичного ≠). Операції == та != є універсальними і можуть застосовуватися для порівняння аргументів довільних типів в Kotlin, включаючи рядки (рядки вважаються рівними, якщо вони мають однакову довжину і їхні відповідні символи збігаються, наприклад, "abc" != "cba"). Решта операцій порівняння наразі застосовні переважно тільки для числових типів, оскільки їхня робота залежить від спеціальної функції порівняння compareTo, яка розглядатиметься в наступних роботах. Результатом усіх операцій порівняння завжди є значення типу Boolean, яке може бути або true (істина), або false (хибність).

Часто умова в операторі if сама обчислюється за допомогою функцій, що повертають результат типу Boolean. Такі функції називаються логічними функціями. Прикладом є функція pointInsideCircle, яка визначає, чи лежить точка з координатами (x, y) всередині кола з центром (x0, y0) та радіусом r. Ця функція базується на нерівності кола:

І повертає true, якщо точка всередині, і false в іншому випадку. Тип результату функції pointInsideCircle є Boolean. Для тестування таких функцій зручно використовувати готові функції assertTrue (призводить до невдалого тесту, якщо аргумент false) та assertFalse (призводить до невдалого тесту, якщо аргумент true). Функцію pointInsideCircle можна комбінувати з логічними операціями для розв'язання складніших завдань, наприклад, для визначення приналежності точки до перетину або об'єднання двох кіл. Логічні операції включають: && (логічне І, результат true, якщо обидва аргументи true), || (логічне АБО, результат true, якщо хоча б один з аргументів true), та ! (логічне НЕ, результат true, якщо аргумент false). Наприклад, для перевірки належності точки перетину двох кіл використовується &&, а для об'єднання ||.

* 1. Практична частина

1. Фрагмент програми і його математичний запис, а також обчислення значення змінної X після його виконання зображено на рисунку знизу:



Виконане завдання 1

1. Функція, що виводить на екран значення true, якщо точка А з координатами х, у належить до заштрихованої області, та false в іншому випадку для 2 варіанта:

fun var2z2(x: Double, y: Double): Boolean{

val distanceFromCenter = sqrt(x.pow(2) + y.pow(2))

val inSmallRing = distanceFromCenter in 0.0..2.0

val inBigRing = distanceFromCenter in 2.0..3.0

val inUpperLeftQuarter = (x <= 0 && y >= 0)

val inLowerRightQuarter = (x >= 0 && y <= 0)

return (inSmallRing && inUpperLeftQuarter) || (inBigRing && inLowerRightQuarter)

}

1. Нижче наведено код функції main, що викликає функцію (ця функція – це та, що повторює поведінку графічно заданої функції) з параметром, який користувач ввів у консоль.

fun main() {

print("Введіть x: ")

val x = readlnOrNull()?.toDoubleOrNull()

if (x == null) {

println("Помилка: введіть коректну абсцису")

}

val y = var2z3(x!!)

println("Значення функції в точці x = $x дорівнює y = $y")

}

fun var2z3(x: Double) : Double {

return when {

x < 0.0 -> {

-1 \* x + 0

}

x in 0.0..1.0 -> {

1 \* x + 0

}

x in 1.0..2.0 -> {

0 \* x + 1

}

x > 2.0 -> {

-2 \* x + 5

}

else -> throw IllegalArgumentException("Значення x не входить в ОДЗ")

}

}

* 1. Висновки

У ході виконання лабораторної роботи було отримано практичні навички з розробки програм, що реалізують алгоритми з розгалуженням.