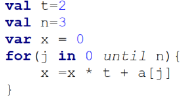
Лабораторна робота №5  
Списки і Масиви

* 1. Мета:

Отримання практичних у використанні складених типів даних, зокрема списків, для розв'язання прикладних задач програмування.

* 1. Завдання:

1. Представити математичний запис фрагмента програми та обчислити значення змінної x після його виконання. Елементи масиву обчислюються за формулою . Значення a[0] дорівнює номеру варіанта за списком групи. Фрагмент програми для другого варіанту:



Фрагмент програми

1. Скласти програму обчислення наступних величин та виконати її у середовищі програмування. Елементи списку (масиву) визначаються за формулою ; де . p[0] дорівнює n – номеру варіанта за списком групи, кількість елементів у списку дорівнює 50. Для варіанту 2: сума елементів масиву a, значення яких кратні N.
2. Завдання складається з трьох пунктів:
   1. Протабулювати функцію із завдання 4.2. Значення x та y занести у списки.
   2. Знайти найбільше, та найменше значення у списку y. Вивести їх та відповідні їм значення з масиву x у наступному вигляді:

yMin = ... при x = ...

yMax = ... при x = ...

* 1. Обчислити суму та середнє арифметичне значення елементів масиву y. Результати вивести на екран.
  2. Теоретична частина

Починаючи з цієї роботи, ми розпочинаємо вивчення складених типів даних, які складаються з декількох елементів простих типів і часто є необхідними в програмуванні. Одним з найбільш поширених складених типів є список. Список може містити будь-яку кількість елементів, від нуля до нескінченності, а його кількість елементів називається розміром. Важливо, що всі елементи списку повинні мати один і той же тип, який може бути як простим (наприклад, список дійсних чисел), так і складеним (наприклад, список рядків або список списків цілих чисел). В іншому, елементи списку незалежні один від одного.

У Kotlin тип List представляє список. У кутових дужках < > вказується типовий аргумент, що визначає тип елементів списку, наприклад, List<Double> означає список дійсних чисел. Для створення списків зручно використовувати функцію listOf(), яка приймає будь-яку кількість аргументів, включаючи нуль, що стають елементами нового списку. У випадках, коли функція повинна повернути порожній список, наприклад, коли біквадратне рівняння не має коренів, використовується listOf() без аргументів. При цьому для порожнього списку без аргументів Kotlin вимагає явної вказівки типу, як у listOf<Double>(), оскільки порожній список рядків і порожній список цілих чисел розглядаються як різні. Однак у багатьох ситуаціях Kotlin може вивести тип списку; наприклад, якщо результат функції biRoots вказаний як List<Double>, то всі списки, що повертаються операторами return у цій функції, повинні мати такий самий тип.

Два списки в Kotlin вважаються рівними тоді й лише тоді, коли збігаються їхні розміри та відповідні елементи. Це означає, що списки, які містять одні й ті ж елементи, але в іншому порядку, вважаються різними. Тому програміст має вирішити, чи важливий порядок елементів для конкретного завдання; якщо так, то списки слід впорядкувати перед порівнянням. Для сортування списку можна використати функцію .sorted(), яка створює новий список того ж розміру, але з елементами, впорядкованими за зростанням. Аналогічно, list.sortedDescending() створює список, відсортований за спаданням.

Серед поширених операцій над списками є додавання двох списків (list1 + list2), що створює новий список з усіма елементами обох. Також можна додати один елемент до списку (list + element). Для отримання розміру списку використовується list.size. Перевірити, чи є список порожнім, можна за допомогою list.isEmpty() або list.isNotEmpty(). Доступ до окремих елементів здійснюється за їх цілочисельним індексом (list[i]). У Kotlin елементи списку мають індекси, що починаються з нуля: перший елемент має індекс 0, а останній елемент списку з n елементів має індекс n - 1. Використання індексу за межами допустимого діапазону призведе до помилки виконання програми. Інші операції включають створення підсписку (list.sublist(from, to)), перевірку наявності елемента в списку (element in list), ітерацію по всіх елементах списку за допомогою циклу for (element in list). Можна отримати перший (list.first()) або останній (list.last()) елемент списку, але ці операції викличуть помилку, якщо список порожній. Функція list.indexOf(element) шукає індекс елемента, повертаючи -1, якщо елемент відсутній. Також доступні функції для пошуку мінімального (list.min()) та максимального (list.max()) елементів, а також для обчислення їх суми (list.sum()).

У таких керуючих структурах, як if..else if..else та when, результат гілки формує її останній оператор. Це дозволяє виражам if та when безпосередньо повертати значення.

Мутуючий список (MutableList<ElementType>) є різновидом звичайного списку, який може змінюватися під час виконання програми або функції. На додаток до можливостей звичайних списків, мутуючий список дозволяє змінювати свій вміст за допомогою операторів list[i] = element. Можна додавати елементи в кінець списку за допомогою list.add(element), що збільшує його розмір на один. Елементи можна видаляти за значенням (list.remove(element)) або за індексом (list.removeAt(index)), що зменшує розмір списку, якщо елемент був знайдений. Також можлива вставка елементів у середину списку за допомогою list.add(index, element), що зсуває наступні елементи. Мутуючі списки створюються за допомогою функції mutableListOf(), яка працює аналогічно listOf().

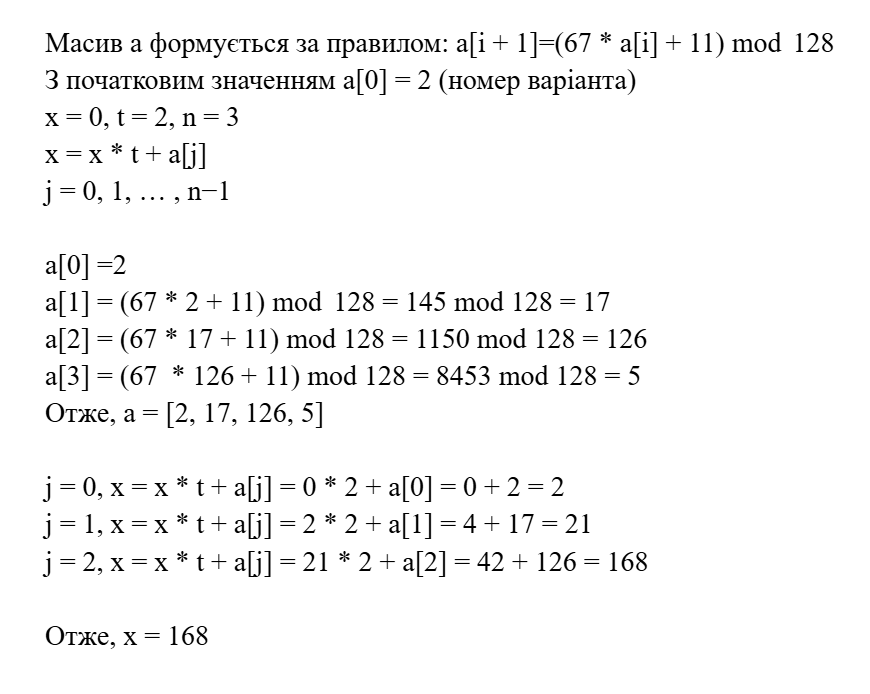
Важливою концепцією є те, що тип MutableList<Int> є підтипом типу List<Int). Це означає, що будь-який мутуючий список також є просто списком, хоча зворотне твердження невірне — не кожен список є мутуючим. Математично, це означає, що область допустимих значень типу MutableList<Int> є підмножиною області допустимих значень типу List<Int>.

Функції, які переважно змінюють свої вхідні параметри, є прикладами функцій з побічним ефектом. Такі функції можуть не повертати явного результату, оскільки їхнє основне призначення полягає в модифікації переданого їм мутуючого об'єкта. При роботі з мутуючими списками, коли необхідно змінити елементи, часто доводиться перебирати їхні індекси, а не самі елементи, використовуючи конструкції на кшталт for (i in 0 until list.size). Прямий перебір елементів (for (element in list)) не дозволяє змінювати element, оскільки параметр циклу for є незмінним, а індекс i недоступний. Більш складний, але можливий спосіб — це одночасна ітерація за елементами та їхніми індексами за допомогою list.withIndex(), яка створює список пар (індекс, елемент). Загалом, рекомендується обмежувати використання функцій, основне призначення яких полягає у зміні їхніх параметрів, оскільки це може ускладнити розуміння та тестування програми.

Масив (Array) – це ще один тип даних для зберігання та маніпулювання однотипними елементами. За функціональністю масиви схожі на мутуючі списки (MutableList), але їхня головна відмінність полягає у фіксованому розмірі: масиви не підтримують операції add або remove для зміни своєї довжини. До елементів масиву можна звертатися та модифікувати їх за допомогою оператора індексації (array[i]). Масиви зазвичай створюються за допомогою функції arrayOf(), яка аналогічна listOf() для списків. Більшість операцій, доступних для списків, також застосовні до масивів, за винятком функцій для створення підсписків (sublist). Однак масиви не слід порівнювати на рівність за допомогою array1 == array2, оскільки це часто дає невірний результат. Масив можна перетворити на звичайний список за допомогою array.toList() або на мутуючий список за допомогою array.toMutableList(). Зворотне перетворення списку на масив виконується за допомогою list.toTypedArray(). Загалом, у програмуванні на Kotlin рідко виникає необхідність використання масивів; одним з небагатьох прикладів є параметр головної функції типу Array<String>, який використовується для передачі аргументів командного рядка, хоча у нових версіях Kotlin цей параметр не є обов'язковим.

* 1. Практична частина

1. Математичний запис фрагмента програми та обчислене значення змінної x після його виконання можна знайти у скриншоті в репозиторії.



Скриншот з репозиторію

1. Під цим описом є частина функції main, що викликає іншу функцію, яка обчислює суму елементів масиву a, значення яких кратні n. N – номер варіанту за списком групи (у моєму випадку – 2). Елементи списку (масиву) визначаються за формулою ; де . p[0] дорівнює n, кількість елементів у списку дорівнює 50.

// Частина функції main

// Завдання 5.2

println("Сума елементів масиву a, значення яких кратні N, введіть N: ")

val N = readln().toInt()

println("Сума елементів масиву a, які кратні $N: ${var2z2(N)}")

// Сума елементів масиву a, значення яких кратні N.

fun var2z2(N: Int) : Int {

val n = 2

val size = 50

val p = IntArray(size)

val a = IntArray(size)

p[0] = n

for (i in 1 until size) {

p[i] = (p[i - 1] \* 67 + 11) % 128

}

for (i in p.indices) {

a[i] = p[i] - 64

}

val sumElements = a.filter { it % N == 0 }.sum()

return sumElements

}

1. Нижче наведений фрагмент функції main, що викликає функцію, що табулює функцію з завдання 4.2; заносить значення у списки x і y; зі списку y знаходить найбільше та найменше значення та виводить відповідні їм значення x; обчислює суму та середнє арифметичне значення елементів масиву y. Результати виводить на екран.

// Завдання 5.3

println("Протабулювати функцію")

println("Start X: ")

val start = readln().toDouble()

println("End X: ")

val end = readln().toDouble()

println("Step: ")

val step = readln().toDouble()

var2z3(start, end, step)

// Табулювання функції y = cbrt(x)

fun var2z3(startX: Double, endX: Double, step: Double) {

val xValues = mutableListOf<Double>()

val yValues = mutableListOf<Double>()

var x = startX

while (x <= endX) {

val y = cbrt(x)

xValues.add(x)

yValues.add(y)

x += step

}

val yMin = yValues.minOrNull()

val yMax = yValues.maxOrNull()

val xMin = xValues[yValues.indexOf(yMin)]

val xMax = xValues[yValues.indexOf(yMax)]

println("yMin = $yMin при x = $xMin")

println("yMax = $yMax при x = $xMax")

val sumY = yValues.sum()

val averageY = yValues.average()

println("Сума елементів масиву y: $sumY")

println("Середнє арифметичне значення елементів масиву y: $averageY")

}

* 1. Висновки

Набув практичних навичок у використанні складених типів даних, зокрема списків, для розв'язання прикладних задач програмування.