Крос-платформне програмування

ЛАБОРАТОРНА РОБОТА № 3

Програмування алгоритмів циклічної структури.

МЕТОДИЧНІ ВКАЗІВКИ
до виконання лабораторної роботи
.

Львів

2024

Мета роботи: Ознайомлення з циклічними алгоритмічними конструкціями Java та Kotlin розгалуження та її використанням в програмах.

Завдання до лабораторної роботи.

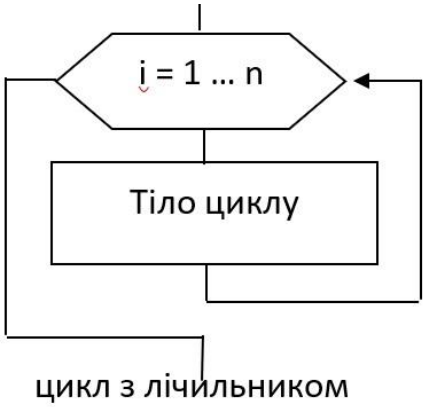
1. Для обидвох задач (Завдання 1 та Завдання 2), відповідно до варіанта, розробити алгоритм розв'язування, та зобразити його графічно у вигляді блок-схеми.
2. Запрограмувати розроблений алгоритм мовою ***Java***.
3. Запрограмувати розроблений алгоритм мовою ***Kotlin***.
4. Оформити письмовий звіт про виконання роботи.

Звіт повинен містити:

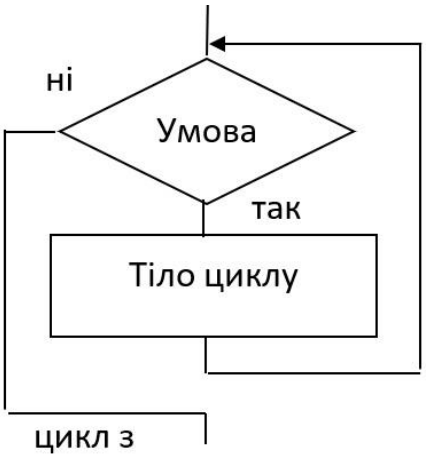
- титульний аркуш:
- для кожного завдання:
  - ✓ умову задачі відповідного варіанту;
  - ✓ блок-схему алгоритму;
  - ✓ програму на Java;
  - ✓ результати виконання програми для деякого набору вхідних даних;
  - ✓ програму на Kotlin;
  - ✓ результати її виконання для того ж набору вхідних даних.

Короткі теоретичні відомості

Цикл це повторення деякої послідовності інструкцій алгоритму. Кількість повторів може задаватися явно, або регламентуватися виконанням деякої умови. Послідовність команд, які повторюються в циклі прийнято називати ***тілом циклу***, а одне виконання ***тіла циклу***, зазвичай, називають ***ітерацією***. Цикл разом з розгалуженням та лінійною алгоритмічною конструкцією, розглянутими у попередніх розділах, складають базовий набір конструкцій для побудови алгоритмів будь-якої складності.



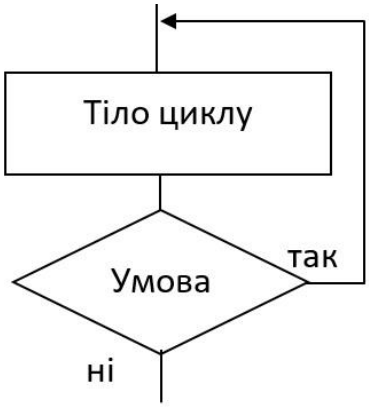
цикл з лічильником



цикл з умовою на початку

Залежно від способу організації ітерацій виділяють три базові конструкції циклу:

- ✓ *цикл з лічильником (параметром) виконує послідовність команд для кожного значення які пробігає деяка змінна-лічильник;*
- ✓ *цикл з умовою на початку перевіряє потребу виконання тіла циклу перед кожною ітерацією, якщо умова виконання хибна вже на початку циклу, то тіло не виконується жодного разу;*
- ✓ *цикл з умовою після тіла виконує перевірку умови після кожної ітерації і вирішує чи потрібно повернутися і повторити інструкції ще раз. Тут команди тіла циклу виконаються обов'язково хоча б один раз.*



цикл з умовою після тіла

Такий поділ суто теоретичним, реалізації зазначених концепцій в різних технологіях програмування, як можна бачити далі, доволі розмаїті. Окремі синтаксичні конструкції, як, наприклад, оператор ***for*** в С-подібних мовах мають засоби для реалізації одразу двох концепцій (цикл з лічильником та цикл з умовою на початку). З іншого боку, цикли з умовами за рахунок допоміжних змінних та контролю їх значень дозволяють реалізовувати цикли з лічильником. І, взагалі, до кожної практичної задачі, що розв'язується за допомогою циклу, можна застосувати будь-яку із зазначених форм циклу. Розглянемо ці конструкції на конкретних прикладах.

В ***Java*** оператор циклу з умовою на початку має вигляд:

```
while(<умова>){
  <послідовність команд>;
}
```

а оператор з умовою після ітерації:

```
do {
  <послідовність команд>;
} while(<умова>);
```

В обидвох конструкціях:

- ✓ *виконання команди повторюється до тих пір, поки умова не набуде хибного значення;*
- ✓ *після того, як умова перестає виконуватися, програма переходить до виконання наступного за* **`while`** *оператора;*
- ✓ *в конструкції з умовою на початку фігурних дужок можна не ставити, якщо послідовність команд складається лише з однієї команди.*

для реалізації циклу з лічильником тут використовується конструкція:

```
for ( <ініціалізація >; <умова>; <дії після ітерації>)
{
    <команди>
}
```

Така конструкція охоплює багато різних можливих алгоритмічних конструкцій, оскільки кожна з трьох частин заголовку цього оператора є необов'язковою. Наприклад, реалізувавши лише середню з них можна отримати цикл ***`for`*** у вигляді циклу з умовою на початку:

```
for ( ; <умова>; ){
    <команди тіла циклу>
}
```

ідентичного до циклу з передумовою ***`while`***:

```
while (<умова>){
    <команди тіла циклу>
}
```

Основні правила запису та функціонування циклу ***`for`*** в ***Java***:

- ✓ *перша частина заголовку оператора* **`for`** *(ініціалізація) оголошує дії, що виконуються перед початком циклу, тут можна оголошувати нові змінні, але їх область існування буде обмежена тілом циклу;*
- ✓ *ініціалізація циклу може містити декілька команд, тоді їх записують підряд, розділяючи комами;*
- ✓ *у другій частині заголовку* **`for`** *записують умову, яка перевіряється перед кожним виконанням тіла циклу (ітерацією), якщо результат перевірки хибний, то ітерація не виконується і програма переходить до наступного після циклу оператора;*

- ✓ *третя частина заголовку описує дії, що потрібно виконати після завершення окремої ітерації, якщо їх декілька, то їх записують підряд, розділяючи комами;*
- ✓ *кожна з трьох частин заголовку за потреби може бути пропущена, зокрема, конструкція* **`for( ; ; ){<інструкції>}`** *є синтаксично коректною і виконуватиме "вічний цикл";*
- ✓ *тіло циклу, що складається з однієї команди можна записувати після заголовку без фігурних дужок.*

Перелічені вище правила декларують різні можливі способи застосування оператора ***`for`***. Реалізація простого циклу з лічильником, що повторює тіло циклу задану кількість разів за допомогою оператора ***`for`*** набагато простіша, пропонуємо розглянути її безпосередньо в прикладі програми для розв'язування задачі.

В ***Kotlin*** реалізовано аналоги усіх трьох ***C***-подібних циклів (***`for`***, ***`while`*** та ***`do-while`***), але оператор ***`for`*** тут має дещо простішу структуру, більше схожу до його реалізації в ***Python***:

```
for (<елемент> in <колекція>) {
    <команди тіла циклу>
}
```

Синтаксис та дія цього оператора такі:

- ✓ *оператор* **`for`** *виконує команди тіла циклу для кожного з елементів вказаної колекції;*
- ✓ *в якості колекції можна використовувати масив, чи екземпляри відповідних класів;*
- ✓ *колекцію для лічильника циклу можна сформувати у вигляді діапазону значень за допомогою знака* **`«..»`** *чи ключових слів* **`until`**, **`downTo`** *та* **`step`***;*
- ✓ *тіло циклу, що складається з однієї команди можна записувати без фігурних дужок але в тому ж самому рядку.*

Подібна конструкція циклу ***`for`*** реалізована також і в ***Java***:

```
for (<тип> <елемент> : <колекція>){
    <команди тіла циклу>
}
```

Усе сказане вище про оператори ***`while`*** та ***`do-while`*** в ***Java*** стосується і операторів оператор циклу в ***Kotlin***.

Завдання 1.

Скласти програму для обчислення значення елементарної функції в заданій точці за допомогою степеневого ряду з точністю до члена, меншого за 0.00001. Визначити кількість доданків. Порівняти отримане значення суми зі значенням вказаної функції, обчисленим за допомогою бібліотечних функцій.

Варіанти завдань.

1. $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$;

2. $sin^3 x = \frac{1}{4} \cdot \sum_{k=1}^{\infty} (-1)^{k+1} * \frac{3^{2k+1}-3}{(2k+1)!} * x^{2k+1}$;

3. $\ln x = 2 \sum_{k=1}^{\infty} \frac{1}{2k-1} \left(\frac{x-1}{x+1}\right)^{2k-1}$;

4. $e^{-x^2} = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{k!}$;

5. $\cos^2 x = 1 - \sum_{k=1}^{\infty} (-1)^{k+1} \frac{2^{2k-1} x^{2k}}{(2k)!}$

6. $ln \frac{1+x}{1-x} = 2 \sum_{k=0}^{\infty} \frac{x^{2k+1}}{2k+1}$;

7. $(1+x)^{-\frac{1}{2}} = 1 - \frac{1}{2}x + \frac{1*3}{2*4}x^2 - \frac{1*3*5}{2*4*6}x^3 + \cdots$;

8. $arcctg\ x = \frac{\pi}{2} - \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k}}{2k+1}$;

9. $\cos^3 x = \frac{1}{4} \sum_{k=0}^{\infty} (-1)^k \frac{(3^{2k}+3)}{(2k)!} x^{2k}$;

10. $sh\ x = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$;

11. $\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$;

12. $(1+x)^{-\frac{1}{3}} = 1 - \frac{1}{3}x + \frac{1*4}{3*6}x^2 - \frac{1*4*7}{3*6*9}x^3 + \cdots$;

13. $\sin^2 x = \sum_{k=1}^{\infty} (-1)^{k+1} \frac{2^{2k-1} x^{2k}}{(2k)!}$;

14. $ch\ x = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$;

15. $e^x = \sum_{k=0}^{\infty} \frac{x^k}{k!}$;

16. $sh\ x = \sum_{k=0}^{\infty} \frac{x^{2k+1}}{(2k+1)!}$;

17. $\sin x = \sum_{k=0}^{\infty} (-1)^k \frac{x^{2k+1}}{(2k+1)!}$;

18. $(1+x)^{\frac{1}{4}} = 1 + \frac{1}{4}x - \frac{1*3}{4*8}x^2 + \frac{1*3*7}{4*8*12}x^3 - \frac{1*3*7*11}{4*8*12*16}x^4 + \cdots$.

19. $e^{-x^2} = \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{k!}$;

20. $\cos^2 x = 1 - \sum_{k=1}^{\infty}(-1)^{k+1} \frac{2^{2k-1}x^{2k}}{(2k)!}$

21. $\cos^3 x = \frac{1}{4}\sum_{k=0}^{\infty}(-1)^k \frac{(3^{2k}+3)}{(2k)!}x^{2k}$;

22. $(1+x)^{-\frac{1}{3}} = 1 - \frac{1}{3}x + \frac{1*4}{3*6}x^2 - \frac{1*4*7}{3*6*9}x^3 + \cdots$;

23. $\sin^2 x = \sum_{k=1}^{\infty}(-1)^{k+1} \frac{2^{2k-1}x^{2k}}{(2k)!}$;

24. $ch\, x = \sum_{k=0}^{\infty} \frac{x^{2k}}{(2k)!}$;

25. $ln\frac{1+x}{1-x} = 2\sum_{k=0}^{\infty} \frac{x^{2k+1}}{2k+1}$;

26. $(1+x)^{-\frac{1}{2}} = 1 - \frac{1}{2}x + \frac{1*3}{2*4}x^2 - \frac{1*3*5}{2*4*6}x^3 + \cdots$;

27. $arcctg\, x = \frac{\pi}{2} - \sum_{k=0}^{\infty}(-1)^k \frac{x^{2k}}{2k+1}$;

28. $sin^3 x = \frac{1}{4} \cdot \sum_{k=1}^{\infty}(-1)^{k+1} * \frac{3^{2k+1}-3}{(2k+1)!} * x^{2k+1}$;

29. $\ln x = 2\sum_{k=1}^{\infty} \frac{1}{2k-1}\left(\frac{x-1}{x+1}\right)^{2k-1}$;

30. $(1+x)^{\frac{1}{4}} = 1 + \frac{1}{4}x - \frac{1*3}{4*8}x^2 + \frac{1*3*7}{4*8*12}x^3 - \frac{1*3*7*11}{4*8*12*16}x^4 + \cdots$.

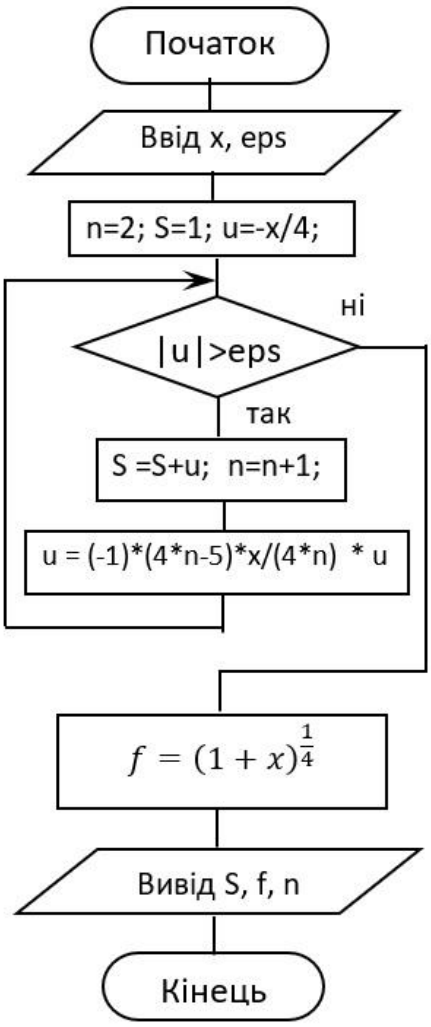
Приклад розв'язування задачі 30

30. Скласти програму для обчислення значення елементарної функції в заданій точці за допомогою степеневого ряду з точністю до члена, меншого за 0.00001. Визначити кількість доданків. Порівняти отримане значення суми зі значенням вказаної функції, обчисленим за допомогою бібліотечних функцій.

$$(1+x)^{\frac{1}{4}} = 1 + \frac{1}{4}x - \frac{1*3}{4*8}x^2 + \frac{1*3*7}{4*8*12}x^3 - \frac{1*3*7*11}{4*8*12*16}x^4 + \cdots .$$

Алгоритму розв'язування задачі:

- *користувач вводить значення x, для якого обчислюється функція за допомогою суми ряду;*
- *програма обчислює початковий доданок суми (або кілька перших доданків);*
- *на кожній ітерації виконується перевірка того, чи знайдений черговий доданок достатньо малий за абсолютною величиною, щоб забезпечити відповідну точність обчислення суми ряду;*
- *якщо абсолютна величина чергового доданка не менша від заданої точності, програма додає його значення до суми, обчислює наступний доданок та повторює перевірку;*
- *при досягненні потрібної точності циклічний процес завершується і програма виводить отриманий результат разом і значенням функції, отриманим за допомогою бібліотечних функцій.*



За складністю ця задача перевищує попередню лише в способі отримання наступного значення члена степеневого ряду. Обчислення загального члена ряду безпосередньо за формулою під знаком суми в умові не завжди можливе через певні обмеження математичного апарату обраної мови програмування. Наприклад, функція піднесення до степеня не працює з від'ємною основою степеня, а потрібно обчислити $(-1)^n$, при великих n значення n! виходить цілочисельного типу, і т.п. В таких випадках для знаходження значення наступного члена ряду A_n зручно використовувати рекурентне співвідношення вигляду $A_n = M_n(x)A_{n-1}$, що пов'язує його із значенням попереднього. Множник $M_n(x)$ завди можна знайти, формально розділивши вираз для A_n на вираз для A_{n-1}: $M_n(x) = \frac{A_n}{A_{n-1}}$. Деколи, як у випадку цієї задачі, $M_n(x)$ можна побудувати просто проаналізувавши формулу загального члена ряду.

В умові нашої задачі

$$1+\frac{1}{4}x-\frac{1*3}{4*8}x^2+\frac{1*3*7}{4*8*12}x^3-\frac{1*3*7*11}{4*8*12*16}x^4+\cdots$$

перші два доданки не зовсім відповідають загальній формулі члена ряду, тому перший доданок включимо безпосередньо в суму, і обчислення суми починемо з $S=1$, а не з $S=0$. У змінну поточного доданка записуватимемо члени ряду, починаючи з другого ($u=\frac{1}{4}x$). Наступні доданки вже вкладаються в загальну схему і дозволяють побудувати рекурентну формулу.

Для побудови згаданого вище коефіцієнта $M_n(x)$ зауважимо, що у нашому знакопочерговому ряді кожен наступний член відрізняється від попереднього протилежним знаком, та на одиницю більшим степенем змінної x. У знаменнику кожного доданка маємо добуток послідовних натуральних чисел, кратних 4, котрі можна записати як $4n$, а в чисельнику, починаючи з другого множника бачимо подібні множники: 3 = 4 – 1, 7=8 – 1 = 4*2 – 1, 11 = 12-1 =4*3 – 1, і т.д. Враховуючи порядок, множник в чисельнику можна задати відповідно виразом $4(n-1)-1=4n-5$. Тобто, $M_n(x)=-\frac{4n-5}{4n}x$ і, починаючи з третього, члени ряду можна шукати за рекурентною формулою

$$A_n=-\frac{4n-5}{4n}x\cdot A_{n-1}.$$

Враховуючи, що синтаксичні конструкції, потрібні для реалізації описаного алгоритму, вже розглянуті в у відповідних пунктах розв’язування задачі 165, далі подаємо програмний код без зайвих пояснень. Заважимо лише, що нумеруючи доданки з 0 та записавши значення $A_0=1$ безпосередньо в початкове значення суми ми використовуємо для початку ітераційного процесу член ряду $A_1=-\frac{1}{4}x$.

Значення $n=2$, яким ініціалізується змінна для визначення кількості доданків вказує, що після проходження першої ітерації в сумі буде враховано два доданки A_0 та A_1. Але в самій ітерації значення n збільшується на 1 і по завершенні усього циклу змінна n міститиме кількість доданків, на одиницю більшу за справжню кількість доданків у сумі. Насправді ж $n-$ий доданок буде обчислено в змінній u, але не додано до суми S. Тому для вказання кількості доданків потрібно виводити значення $n-1$. Змінити ініціалізацію $n=2$ не можна, бо ця змінна бере участь в обчисленні наступного члена ряду.

Під час тестування створених програм слід звернути увагу на математичну специфіку задач – степеневі ряди мають, зазвичай, обмежену область збіжності. Задаючи вхідні дані для змінної x за її межами, ми будемо отримувати зациклення програми. У нашій задачі область збіжності ряду

$$x\in(-1;1)$$

Програма на Java.

```
package com.tasks.part3;
import java.util.Scanner;
public class Task180 {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        double x, u, s = 1, eps = 0.00001;
        System.out.println("Введіть значення x");
        x = scanner.nextDouble();
        u = x/4;  int n = 2;
        while(Math.abs(u)>eps) {
            s += u;  u *= -x*(4*n-5)/(4*n);
            n++;
        }
        System.out.printf("Значення суми s = %.6f\n", s);
        System.out.printf("Значення функції: %.6f\n",
            Math.pow(1+x,0.25));
      System.out.println("Кількість доданків " + (n - 1));
    }
}
```

Програма на Kotlin.

```
package com.tasks.part3

import java.lang.Math.pow
import kotlin.math.abs

fun main(args: Array<String>) {
    println("Введіть значення x")
    val x: Double = readln().toDouble()
    var u: Double = x / 4
    var s = 1.0
```

```
    val eps = 0.00001
    var n = 2
    while (abs(u) > eps) {
        s += u
        u *= -x * (4 * n - 5) / (4 * n)
        n++
    }
    println("Значення суми s = %.6f".format(s))
    println("Значення функції: %.6f"
        .format(pow(1 + x, 0.25)))
    println("Кількість доданків " + (n - 1))
}
```

Завдання 2.

Побудувати таблицю значень заданої функції двох змінних у заданому користувачем прямокутнику $[a;b]\times[c;d]$. Крок зміни значень змінних $x\in[a;b]$ та $y\in[c;d]$ вибрати таким чином, щоб на кожному з відрізків розміщувалося по 8 точок. Результат обчислень вивести на екран у вигляді прямокутної таблиці, в якій по горизонталі вказані різні значення змінної x, по вертикалі – значення змінної y, а на перехресті стовпця зі значенням змінної x та рядка зі значенням змінної y – відповідне значення функції.

Варіанти завдань.

1. $u=2x^2+3y^2;$

2. $u=x^3y+3y^3x;$

3. $u=\sin y+\cos\frac{x}{y};$

4. $u=4x^3-3y^2;$

5. $u=\sin x+\cos y;$

6. $u=5\,lg(x+y);$

7. $u=\sqrt{x^2+4y^2};$

8. $u=x^2+y^4x;$

9. $u=x^3+y^2x;$

10. $u=10\sin(x-y);$

11. $u=5\sin(x+y);$

12. $u=xy+3y^3x;$

13. $u=2x^3+3y^2;$

14. $u=5\,tg(x+y);$

15. $u=\sin x\,y+\cos\frac{x}{y};$

16. $u=2x^3-3y^2;$

17. $u=7\,lg(x+y);$

18. $u=\sqrt{3x^2+2y^2};$

19. $u=4x^3-3y^2;$

20. $u=5\,lg(x+y);$

21. $u=\sqrt{3xy+2y^3}.$

22. $u=\sin^2 x+\cos y;$

23. $u=7\,tg(x-y);$

24. $u=4\sin(x+3y);$

25. $u=x^{\frac{2}{3}}+x^2y;$

26. $u=2x^3+3y^2;$

27. $u=10\sin(x-y);$

28. $u=\sin y+\cos\frac{y}{x};$

29. $u=xy^3+3yx;$

30. $u=\sqrt{2xy+4y^3}.$

Приклад розв'язування задачі 30

30. Побудувати таблицю значень заданої функції двох змінних у заданому користувачем прямокутнику $[a;b]\times[c;d]$. Крок зміни значень змінних $x\in[a;b]$ та $y\in[c;d]$ вибрати таким чином, щоб на кожному з відрізків розміщувалося по 8 точок. Результат обчислень вивести на екран у вигляді прямокутної таблиці, в якій по горизонталі вказані різні значення змінної x, по вертикалі – значення змінної y, а на перехресті стовпця зі значенням змінної x та рядка зі значенням змінної y – відповідне значення функції.

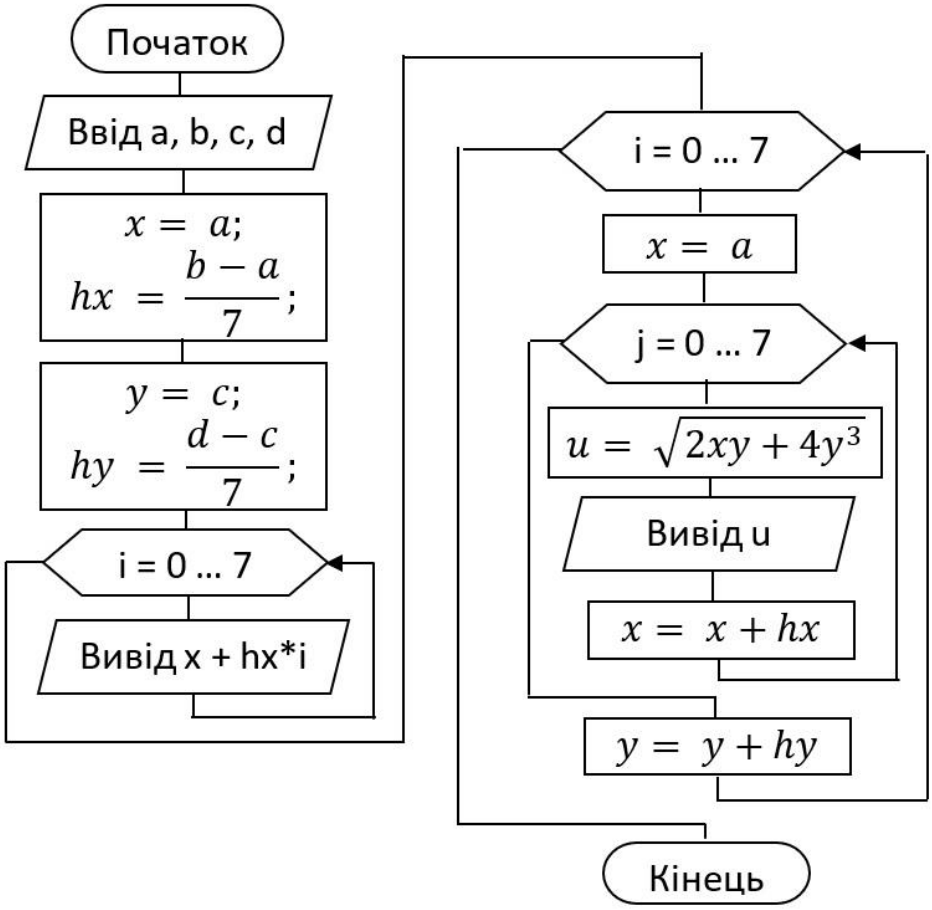
$$u=\sqrt{2xy+4y^3}.$$

Задачі відрізняється від попередньої тим, що потребує виконання «циклу в циклі». Тут кожна ітерація одного циклу передбачає в ході свого виконання повний цикл ітерацій ще одного циклу. Така ситуація в програмуванні отримала назву ***вкладення циклів***. Її реалізація в коді програм не потребує якихось специфічних конструкцій, – просто в блоці команд (тілі) ***зовнішнього циклу*** записують ще один оператор циклу – ***внутрішній цикл***. Вкладення циклів може продовжуватися «вглиб», – кожен внутрішній цикл може містити ще один або декілька внутрішніх циклів. Глибина вкладення циклів, взагалі кажучи, нічим не обмежується, окрім, обчислювальних потужностей техніки, бо при кожному вкладенні циклів потреба в ресурсах зростає на цілий порядок.

Для розв'язування нашої задачі достатньо пари вкладених циклів, – зовнішній цикл повинен переглянути усі значення одного аргумента, наприклад, x, а внутрішній, – до поточного значення цієї змінної поставити кожне із значень другого аргумента, та для кожної, утвореної в такий спосіб точки (x,y) обчислити та вивести значення функції $u(x,y)$.

Щодо типу циклів, то в задачах про табулювання функції за відомим відрізком, та кроком нескладно визначити кількість точок (значень аргумента) ще до початку циклу. Тому будемо використовувати конструкцію циклу з лічильником, який часто називають просто ***цикл for*** за назвою відповідного оператора. Тим більше, що умовою задачі кількість точок на відрізку задана наперед, а крок зміни аргументів ми визначаємо відповідно до неї.

Щоб сформулювати алгоритм тут скористаємося блок-схемою (див. малюнок), а словесне формулювання подавати не будемо. Для виводу значень функції у вигляді прямокутної таблиці доведеться використати перед основним циклом додатковий цикл, щоб вивести у заголовок стовпців відповідні значення одного з аргументів.



Програма на Java.

Програмування вкладених циклів на ***Java*** потребує уважного ставлення до блоків коду в тілі циклів, взятих у фігурні дужки. Компілятор завжди контролює кількість відкритих-закритих дужок і, якщо кількість символів “{” у коді не співпадає з кількістю символів “}”, повідомляє про синтаксичну помилку та не компілює такий код. Сучасні інтегровані системи розробки, зокрема й Intellij Idea, повідомляють про подібні ситуації вже на етапі написання коду і навіть підказують як виправити помилку. Попри те, якщо ви охопите фігурними дужками групу команд, невідповідно до логіки алгоритму розв’язування задачі, жоден компілятор чи IDE не врятує вас від помилкових результатів роботи програми.

Для форматного виводу дійсних чисел в Java також використовується метод ***printf*** класу ***PrintStream*** з відповідними форматними вказівниками, що викликається від об’єкта вихідного потоку консолі System.in.

```
import java.util.Scanner;

public class Task255 {

    public static void main(String[] args) {
```

```
        Scanner scanner = new Scanner(System.in);
        double a, b, c, d, u;
        System.out.println("Введіть межі для x");
        a = scanner.nextDouble();
        b = scanner.nextDouble();
        double x = a, hx = (b - a)/7 ;
        System.out.println("Введіть межі для y");
        c = scanner.nextDouble();
        d = scanner.nextDouble();
        double y = c, hy = (d - c)/7 ;
        System.out.print("y\\x  ");
        for (int i = 0; i < 8; i++)
            System.out.printf("%8.2f",x + hx*i);
        System.out.println();
        for (int i = 0; i < 8; i++){
            System.out.printf("%5.2f",y);
            x = a;
            for (int j = 0; j < 8; j++) {
                u = Math.sqrt(2*x*y + 4*y*y*y);
                System.out.printf("%8.2f", u);
                x += hx;
            }
            System.out.println();
            y += hy;
        }
    }
}
```

Програма на Kotlin.

Програмування вкладених циклів на ***Kotlin,*** за винятком особливостей синтаксису циклів з параметром, має таку ж специфіку, що й в ***Java.*** Тут вкладеність блоків коду також позначається за допомогою фігурних дужок.

Для форматування таблиці значень функції в ***Kotlin*** можна скористатися методом ***format*** класу ***String***, подібно до того, як цей же механізм використовується у ***Python*** (див. код нижче).

З іншого боку, ***Kotlin*** дозволяє в повному обсязі використовувати класи пакету JDK (доступні за посередництвом віртуальної машини Java). А отже ми маємо можливість використуовуати метод ***printf*** класу ***PrintStream***. До прикладу, в коді нижче замість ***Python***-подібного

```
print("%8.2f".format(u))
```

можна використовувати ***Java***-код:

```
System.out.printf("%8.2f", u)
```

Решта особливостей застосування ***Kotlin*** до розв'язування задачі 195 пропонуємо розглянути в коді далі.

```
package com.tasks.part3

import kotlin.math.sqrt


fun main(args: Array<String>) {
    println("Введіть межі для x")
    val a: Double = readln().toDouble()
    val b: Double = readln().toDouble()
    var x = a
    val hx = (b - a) / 7
    println("Введіть межі для y")
    val c: Double = readln().toDouble()
    val d: Double = readln().toDouble()
    var y = c
    val hy = (d - c) / 7
    var u: Double
    print("y\\x  ")
    for (i in 0..7) {
```

```
        print("%8.2f".format(x + hx * i))
    }
    println()
    for (i in 0..7) {
        print("%5.2f".format(y))
        x = a
        for (j in 0..7) {
            u = sqrt(2 * x * y + 4 * y * y * y)
            print("%8.2f".format(u))
            x += hx
        }
        println()
        y += hy
    }
}
```