Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Дослідження ітераційних циклічних алгоритмів»

Варіант 30

Виконав студент ІП-12 Тарасюк Євгеній Сергійович

Перевірив ____________________________________

Київ 2021

Лабораторна робота 3.

Дослідження ітераційних циклічних алгоритмів.

Мета: дослідити подання операторів повторення дій та набути практичних навичок їх використання під час складання циклічних програмних специфікацій.

Задача 30 (варіант 30). Обчислити x = a^(1/p), використовуючи принцип розв'язання (фото 1) з точністю, заданою користувачем. Значення a, p (p != 1, p != 2) ввести з клавіатури.

Фото 1:

Приближенное вычисление корней производится с помощью биноминального ряда

$$(1+x)^m = 1 + mx + \frac{m(m-1)}{2!}x^2 + \ldots + \frac{m(m-1)\ldots(m-n+1)}{n!}x^n + \ldots (|x| \leq 1).$$

Имеем

$$\sqrt[10]{1027} = \left(2^{10}+3\right)^{\frac{1}{10}} = 2\left(1+\frac{3}{2^{10}}\right)^{\frac{1}{10}} =$$

$$= 2 + \frac{3}{10\cdot 2^9} + \frac{\frac{1}{10}\left(\frac{1}{10}-1\right)\cdot 3^2}{2!\cdot 2^{19}} + \frac{\frac{1}{10}\left(\frac{1}{10}-1\right)\left(\frac{1}{10}-2\right)\cdot 3^3}{3!\cdot 2^{29}} + \ldots$$

Полученный ряд является рядом Лейбница, и значит, погрешность от отбрасывания членов, начиная с третьего, по абсолютной величине меньше: $\frac{3^4}{10^2\cdot 2^{20}} < 0{,}0001$.

Сохраняя поэтому только два члена разложения, будем иметь

$$\sqrt[10]{1027} = 2 + \frac{3}{10\cdot 2^9} = 2{,}0006.$$

<u>Розв’язок.</u>

1. **Постановка задачі.**

Початкові дані - це два дійсні числа та одне ціле, додаткових чисел для розв’язку не потрібно. Результатом розв’язку є дійсне число. Використовуватимемо стандартні логічні та арифметичні операції, а також функції для запобігання нагромадженням однакових операцій у коді.

2. **Побудова математичної моделі**

Таблиця змінних та функцій:

Змінні та функції	Тип	Ім’я	Призначення
Число num (в умові - a)	**Ціле число**	**num**	**Збереження початкових даних (число, з якого потрібно знайти корінь)**
Число p	**Ціле число**	**p**	**Збереження початкових даних (показник кореня)**
Точність	**Дійсне число**	**eps**	**Збереження початкових даних (точність)**
“стеля” числа num	**Ціле число**	**a**	**Збереження числа num, округленого вгору**
Число p_2	**Ціле число**	**p2**	**Допоміжна змінна для позначення степеня двійки при бінарному**

			пошуку “стелі” числа num на множині цілих чисел
Фунція ceil_pos	**Функція, повертає одне з трьох значень (“більше”, “менше”, “правильно”)**	**ceil_pos(a, p, num)**	**Допоміжна функція для пошуку “стелі” числа num**
Число b	**Дійсне число**	**b**	**Збереження даних про різницю числа num та числа a^p**
Число n	**Ціле число**	**n**	**Порядковий номер члена біноміального ряду**
Число preans	**Дійсне число**	**preans**	**Збереження суми доданків біноміального ряду**
Число res	**Дійсне число**	**res**	**Допоміжна змінна для збереження проміжних значень функцій**
Функція fact	**Функція, ціле число**	**fact(n)**	**Функція, що розраховує факторіал числа**
Функція mult	**Функція, дійсне число**	**mult(p,n)**	**Функція, що розраховує добуток для членів біноміального ряду**

Кінцевий результат	**Змінна логічного типу**	**Eq**	**Збереження даних про результат**

3. Псевдокод алгоритму

Початок

Введення num, p, eps

p2 = -1

a = 0

Поки *ceil_pos(a, p, num)* = “менше”:

p2 += 1

a = 2^p2

Все поки

Поки *ceil_pos(a, p, num)* != “правильно”:

p2 -= 1

Якщо *ceil_pos(a, p, num)* = “більше”:

a -= 2^p2

Інакше:

a += 2^p2

Все якщо

b = num - a^p

n = 1

preans = 1

Поки |*add(a, p, b, n)*| >= eps:

ans += *add(a, p, b, n)*

n += 1

Виведення ans*a

Кінець.

Функція *ceil_pos(a, p, num):*

Початок

Якщо ((a-1)^p < num) та (a^p >= num):

res = “правильно”

Інакше якщо :

res = “більше”

Інакше:

res = “менше”

Повернути res

Кінець.

Функція *mult(p, n)***:**

Початок

res = 1

Для i на проміжку [0,n):

res *= 1/(p-i)

Кінець циклу

Повернути res

Кінець.

Функція *fact(n)***:**

Початок

res = 1

Для i на проміжку [1,n]:

res *= i

Кінець циклу

Повернути res

Кінець.

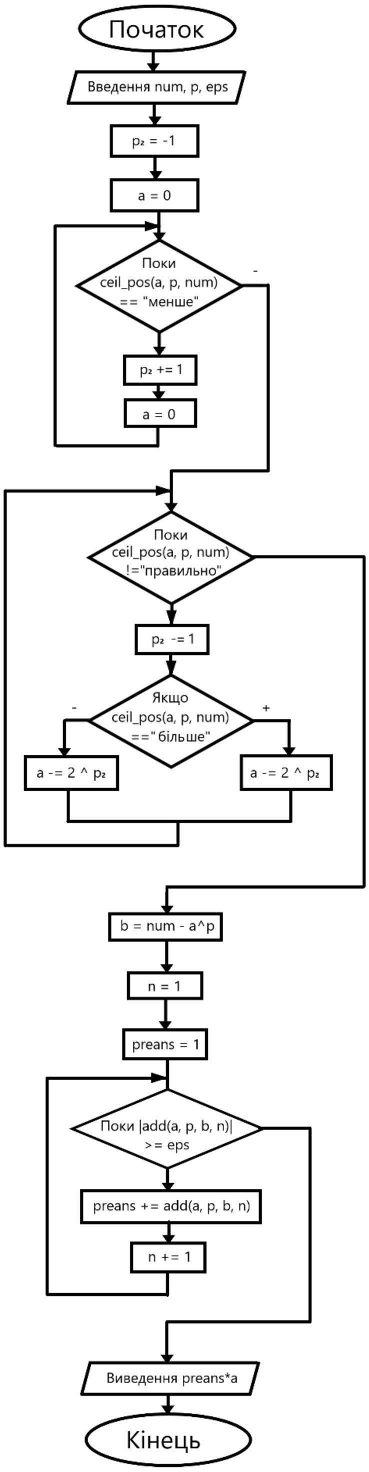
Функція *add(a, p, b, n)***:**

Початок

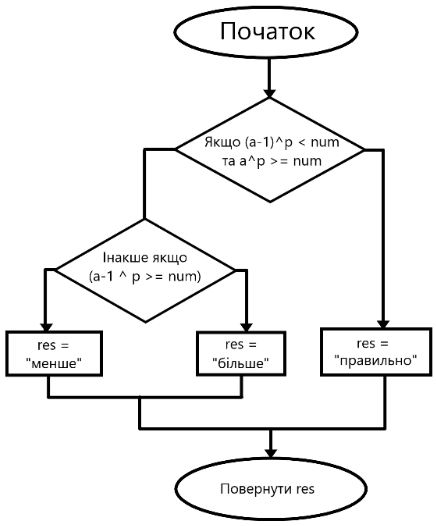
Повернути (b/a^p)^n**mult(p,n)*/*fact(n)*

Кінець.

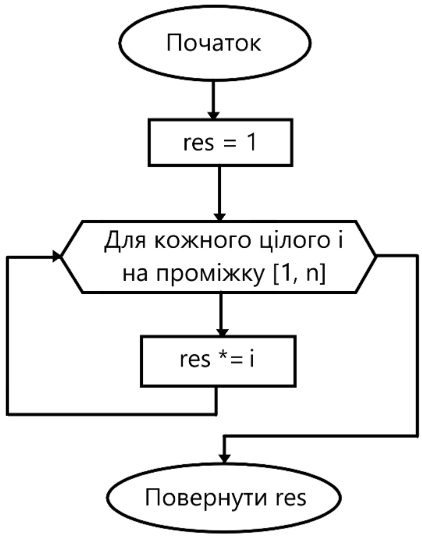
4. Блок схема алгоритму



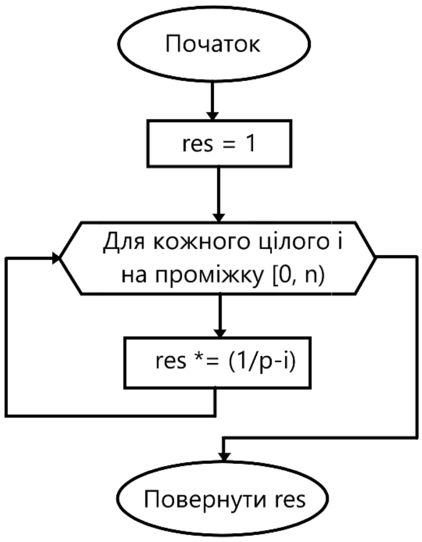
Функція ceil_pos(a, p, num)



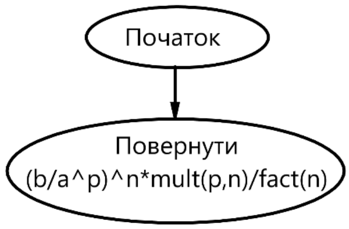
Функція fact(n)



Функція mult (p, n)



Функція add (a, p, b, n)



5. Випробування алгоритму.

Перевіримо правильність алгоритму для різних вхідних даних:

Початок

Введення num = 234, p = 5, eps = 10^(-5)

p2 = -1

a = 0

Поки *ceil_pos(a, p, num)* = “менше”:

p2 += 1

a = 2^p2

Все поки

Поки *ceil_pos(a, p, num)* != “правильно”:

p2 -= 1

Якщо *ceil_pos(a, p, num)* = “більше”:

a -= 2^p2

Інакше:

a += 2^p2

Все якщо *{ a = 3 }*

b = num - a^p *{ b = -9 }*

n = 1

preans = 1

Поки |*add(a, p, b, n)*| >= eps:

ans += *add(a, p, b, n)*

{n =1 add = -0.00010973936899862827

n = 2 add = -2.4386526444139617e-06}

n += 1

Виведення ans*a = 2.977448559670782

Кінець.

Початок

Введення num = 1027, p = 10, eps = 10^(-3)

p2 = -1

a = 0

Поки *ceil_pos(a, p, num)* = “менше”:

p2 += 1

a = 2^p2

Все поки

Поки *ceil_pos(a, p, num)* != “правильно”:

p2 -= 1

Якщо *ceil_pos(a, p, num)* = “більше”:

a -= 2^p2

Інакше:

a += 2^p2

Все якщо *{ a = 3 }*

b = num - a^p

n = 1

preans = 1

Поки |*add(a, p, b, n)*| >= eps:

ans += *add(a, p, b, n)*

{n= 33 add = -0.0011222275406986707

n = 34 add = -0.001067033491355216

n= 35 add = -0.0010155232067703957}

n += 1

Виведення ans*a = 2.081253987007137

Кінець.

6. Висновки

Було досліджено подання операторів повторення дій та набуто практичних навичок їх використання під час складання циклічних програмних специфікацій.