Додаток 1

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 № 6 з дисципліни
«Алгоритми та структури даних-1.
Основи алгоритмізації»

«Дослідження рекурсивних алгоритмів»

Варіант 26

Виконав студент ІП-13 Паламарчук Олександр Олександрович
(шифр, прізвище, ім'я, по батькові)

Перевірила Вечерковська Анастасія Сергіївна
(прізвище, ім'я, по батькові)

Київ 2021

Лабораторна робота 6

Дослідження рекурсивних алгоритмів

Мета - дослідити особливості роботи рекурсивних алгоритмів та набути практичних навичок їх використання під час складання програмних специфікацій підпрограм.

Варіант 26

Задано натуральне ***n***. Обчислити

$$\sum_{k=m}^{n} \frac{(-1)^k}{k!}\left(\frac{a_k+2}{3}\right)^k$$

де

$$a_0 = 1, a_k = \sqrt{|4a_{k-1}+2|}$$

◆ Постановка задачі

Вхідним даним є натуральне число ***n***. Знайти суму на основі заданої формули використовуючи рекурсивний алггоритм, де $\boldsymbol{a_0 = 1}$, $\boldsymbol{a_k}$ - обчислюється за заданою формолою. Вивести отриманий результат.

◆ Побудова математичної моделі

Складемо таблицю змінних

Змінна	Тип	Призначення
Задане число ***n***	Натуральне	Початкове дане
Елемент обчислень ***k***	Натуральне	Проміжне значення
Поточний елемент суми ***a***	Дійсне	Проміжне значення
Попередній Елемент суми ***aPrevious***	Дійсне	Проміжне значення
Результат ***sum***	Дійсне	Кінцеве дане

Складемо таблицю операцій

Назва операції	Синтаксис	Призначення
Корінь квадратний з числа	sqrt(***a***)	Отримати корінь квадратний з числа ***a***
Абсолютна величина	abs(***a***)	Отримати абсолютну

		величину з числа ***a***
Піднесення до степеня	pow(***a***, ***b***)	Піднесення числа ***a*** до степення ***b***

Для вирішення задачі використаємо допоміжні алгоритми (підпрограми) виклик яких має вигляд sumFunction*(**c1, c2, c3, c4**), де **c1, c2, c3, c4*** - формальні параметри функції, та factorial(***c1***) де ***c1*** - формальний параметр фунцкії.

Рекурсивна гілка першої функції буде мати вигляд "**повернути** sumFunction(***c1, c2 + 1, c3, c4***)". Термінальна гілка першої функції буде мати вигляд "**повернути *sum***". При викликі першої функції ми перевіряємо умову ***c2*** == 0, у разі істини ми присвоюємо ***a*** значення 1, у разі хибності присвоюємо значення, яке обчислуємо за формулою $a_k = \sqrt{|4a_{k-1} + 2|}$. Далі перевіряємо умову входження у рекурсивну гілку, у якій збільшуємо значення аргументу ***c2*** на одиницю, у разі хибності повертаємо значення ***sum***.

Рекурсивна гілка другої функції буде мати вигляд "**повернути *c1*** * factorial(***c1*** - 1) ". Термінальна гілка другої функції буде мати вигляд "**повернути** 1". При викликі другої функції, за допомогою якої ми обчислюємо факторіал, ми перевіряємо умову ***c1*** == 0, у разі істини ми повертаємо 1. Далі ми перевіряємо умову ***c1*** == 1, у разі істини повертаємо 1, у разі хибності заходимо у рекурсивну гілку, у які зменшуємо значення аргументу на одиницю.

◆ Розв'язання

Програмні специфікації запишемо у псевдокоді, графічні схемі у вигляді блок-схеми, та у вигляді коду.

1. Основна програма.

   *Крок 1.*Визначимо основні дії.

   *Крок 2.*Ініціалізація змінної ***k***.

   *Крок 3.*Ініціалізація змінної ***aPrevious***.

   *Крок 4.*Ініціалізація змінної ***sum***.

   *Крок 5.*Перевірка ***n***.

   *Крок 6.*Виклик рекурсивної функції для обчислення суми ***sum***.

2. Підпрограма для функції sumFunction

   *Крок 1.*Визначимо основні дії.

*Крок 2.*Задамо умову для обчислення ***a***.

*Крок 3.*Обчислимо ***a.***

*Крок 4.*Задамо умову для повернення рекурсвної та термінальної гілок.

*Крок 5.*Обчислення суми.

*Крок 6.*Присвоєння **c1** значення ***a***

3. Підпрограма для функції factorial

*Крок 1.*Визначимо основні дії

*Крок 2.*Перевірка ***c1*** на рівність нулю.

*Крок 3.*Задамо умову для повернення рекурсвної та термінальної гілок.

◆ Псевдокод алгоритму основної програми

Крок 1

Початок

Ввід n

Ініціалізація змінної ***k***

Ініціалізація змінної ***aPrevious***

. Ініціалізація змінної ***sum***

Перевірка ***n***

Виклик рекурсивної функції для обчислення суми ***sum***

Вивід *sum*

Кінець

Крок 2

Початок

Ввід n

k = 0

Ініціалізація змінної ***aPrevious***

. Ініціалізація змінної ***sum***

Перевірка ***n***

Виклик рекурсивної функції для обчислення суми ***sum***

Вивід *sum*

Кінець

Крок 3

Початок

Ввід n

k = 0

aPrevious = 0

Ініціалізація змінної ***sum***

Перевірка ***n***

Виклик рекурсивної функції для обчислення суми ***sum***

Вивід *sum*

Кінець

Крок 4

Початок

Ввід n

k = 0

aPrevious = 0

. ***sum = 0***

Перевірка ***n***

Виклик рекурсивної функції для обчислення суми ***sum***

Вивід *sum*

Кінець

Крок 5

Початок

Ввід n

k = 0

aPrevious = 0

. ***sum = 0***

Якщо *n* > 0

То

Виклик рекурсивної функції для обчислення суми ***sum***

Інакше

Вивід “Некоректе число”

Все якщо

Вивід *sum*

Кінець

Крок 6

Початок

Ввід n

k = 0

aPrevious = 0

. ***sum = 0***

Якщо *n* > 0

То

sum = sumFunction***(n, k, aPrevious, sum)***

Інакше

Вивід “Некоректе число”

Все якщо

Вивід *sum*

Кінець

◆ Псевдокод алгоритму підпрограми sumFunction

Крок 1.

sumFunction(***c1, c2, c3, c4***)

Задамо умову для обчислення ***a***.

Обчислимо ***a.***

Задамо умову для повернення рекурсвної та термінальної гілок.

Обчислення суми.

Кінець sumFunction

Крок 2.

sumFunction(***c1, c2, c3, c4***)

Якщо *c2* == 0

То

a = 1

Інакше

Обчислимо ***a***

Все якщо

Задамо умову для повернення рекурсвної та термінальної гілок.

Обчислення суми.

Присвоєння ***c3*** значення ***a***

Кінець sumFunction

Крок 3.

sumFunction(***c1, c2, c3, c4***)

Якщо ***c2*** == 0

То

a = 1

Інакше

a = sqrt(abs(4 * ***c3*** + 2))

Все якщо

Задамо умову для повернення рекурсвної та термінальної гілок.

Обчислення суми.

Присвоєння ***c3*** значення ***a***

Кінець sumFunction

Крок 4.

sumFunction(***c1, c2, c3, c4***)

Якщо ***c2*** == 0

То

a = 1

Інакше

a = sqrt(abs(4 * ***c3*** + 2))

Все якщо

Якщо c1 >= ***c2***

То

Обчислення суми.

Присвоєння ***c3*** значення ***a***

Інакше

Повернути c4

Кінець sumFunction

Крок 5.

sumFunction(***c1, c2, c3, c4***)

Якщо ***c2*** == 0

То

a = 1

Інакше

a = sqrt(abs(4 * ***c3*** + 2))

Все якщо

Якщо ***c1*** >= ***c2***

То

c4 **=** ***c4*** **+ ((**pow(-1, ***c2***) / factorial(***c2***)) * (pow((***a*** + 2) / 3, **c2**)))

Присвоєння ***c3*** значення ***a***

Повернути sumFunction(***c1, c2 + 1, c3, c4***)

Інакше

Повернути ***c4***

Кінець sumFunction

Крок 6.

sumFunction(***c1, c2, c3, c4***)

Якщо ***c2*** == 0

То

a = 1

Інакше

a = sqrt(abs(4 * ***c3*** + 2))

Все якщо

Якщо ***c1*** >= ***c2***

То

c4 **=** ***c4*** **+ ((**pow(-1, ***c2***) / factorial(***c2***)) * (pow((***a*** + 2) / 3, **c2**)))

c3 = ***a***

Повернути sumFunction(*c1, c2 + 1, c3, c4*)

Інакше

Повернути ***c4***

Кінець sumFunction

◆ Псевдокод алгоритму підпрограми factorial

Крок 1

factorial(**c1**)

Перевірка **c1** на рівність нулю.

Задамо умову для повернення рекурсвної та термінальної гілок.

Кінець factorial

Крок 2

factorial(***c1***)

Якщо ***c1*** == 0

То

Повернути 1

Інакше

Все якщо

Задамо умову для повернення рекурсвної та термінальної гілок.

Кінець factorial

Крок 3

factorial(**c1**)

Якщо ***c1*** == 0

То

Повернути 1

Інакше

Все якщо

Якщо ***c1*** == 1

То

Повернути 1

Інакше

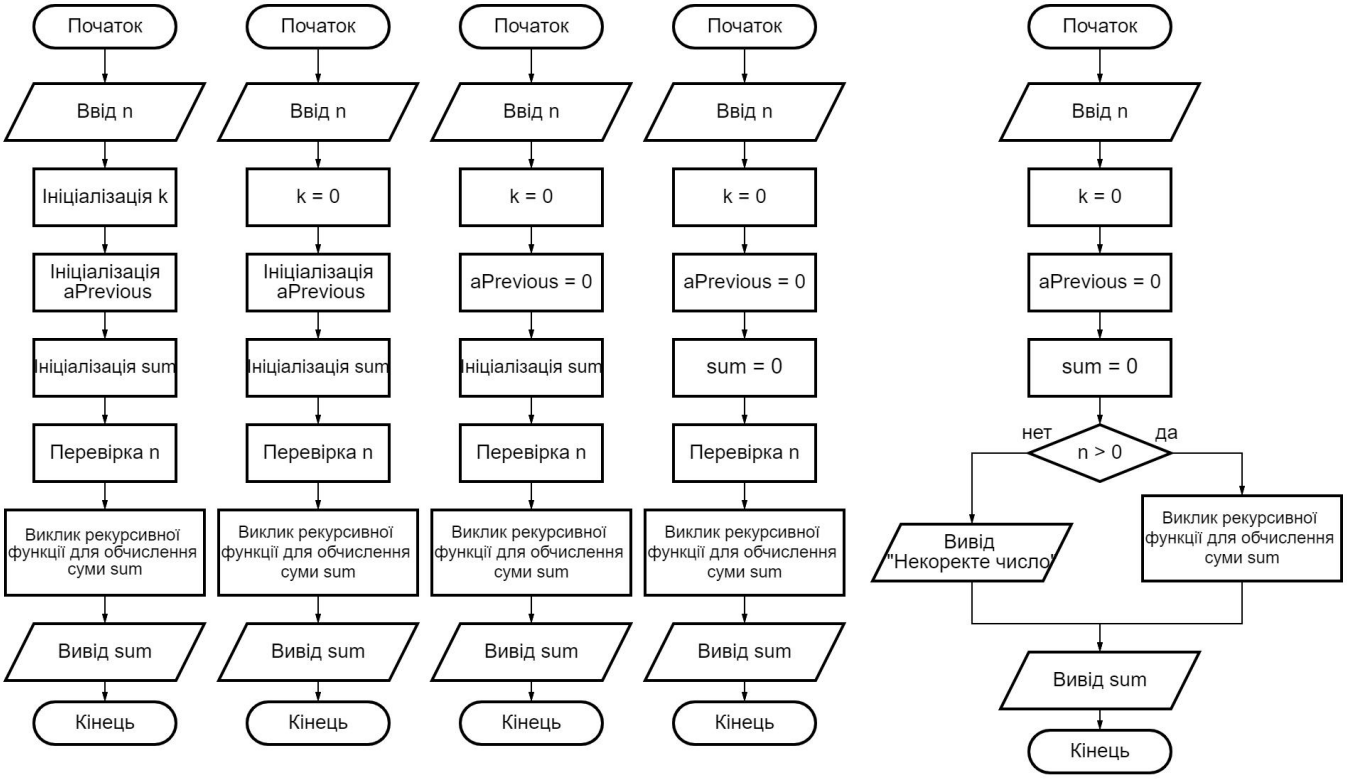
Повернути ***c1*** * factorial(***c1*** - 1)

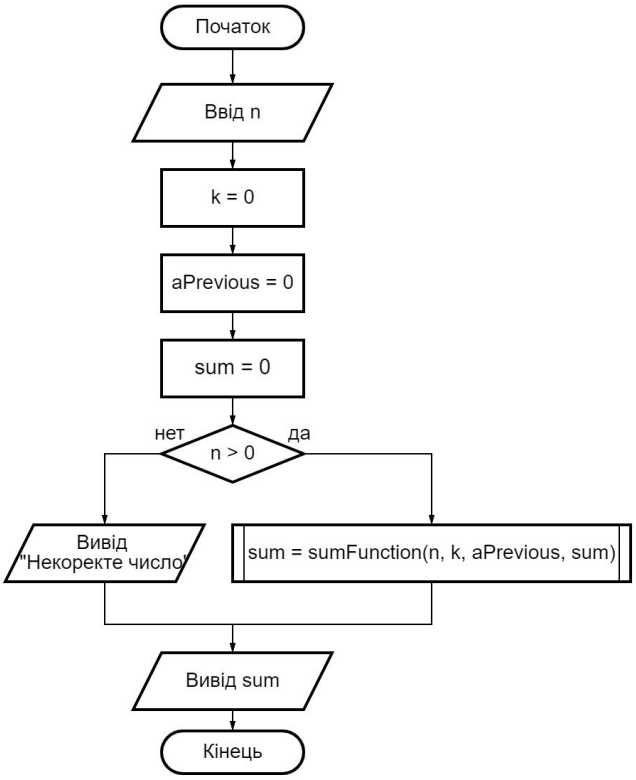
Все якщо

Повернути

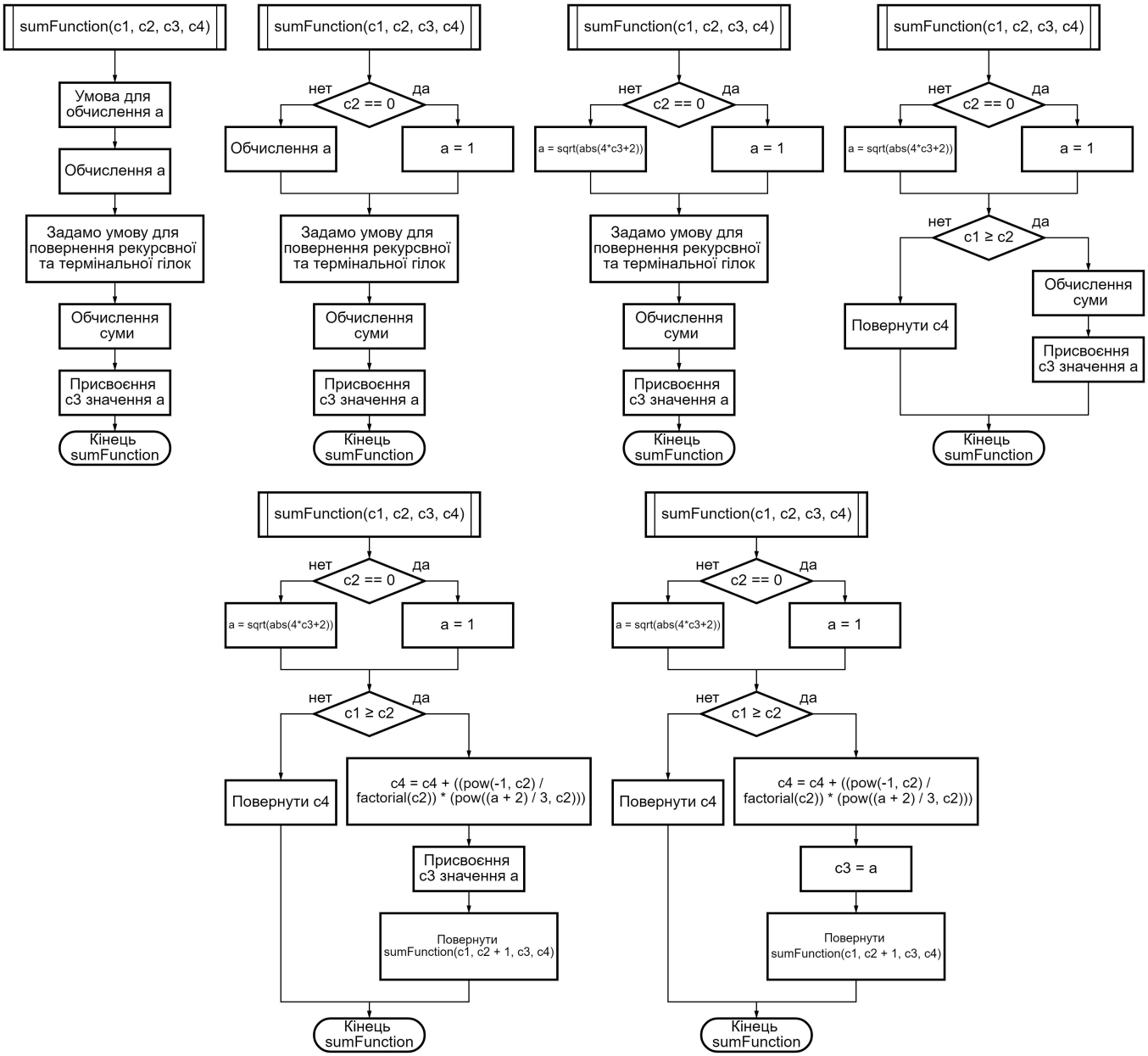
Кінець factorial

◆ Блок-схема алгоритму основної програми

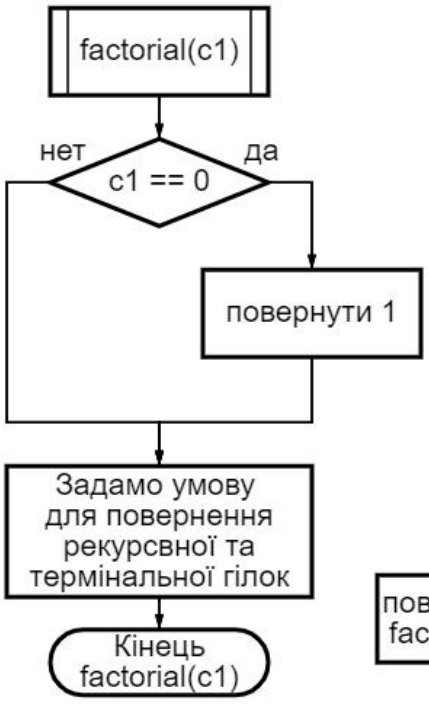


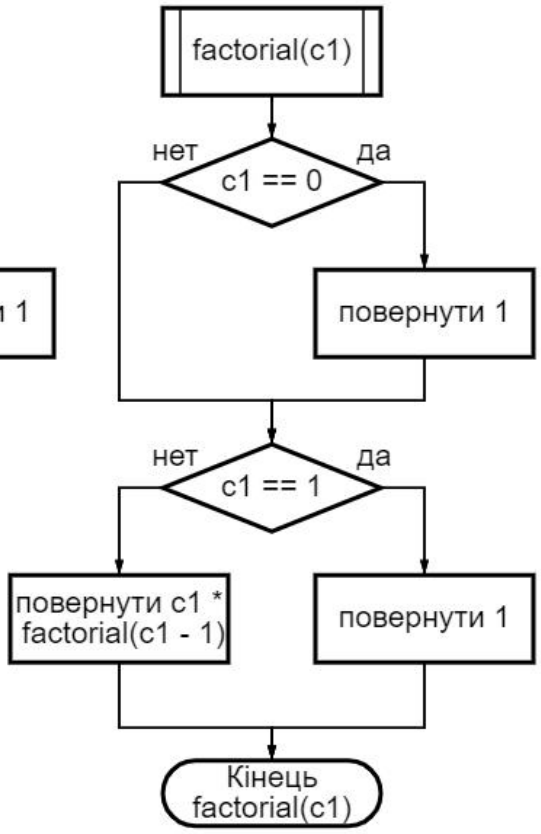


◆ Блок-схема алгоритму підпрограми sumFunction



◆ Блок-схема алгоритму підпрограми factorial





◆ Код програми

◆ Випробовування алгоритму

Блок	Дія
	Початок
1	Ввід 3

2	k = 0
3	aPrevious = 0
4	sum = 0
5	3 > 0 (true)
6	sum = sumFunction(3, 0, 0 ,0)
7	0 == 0 (true)
8	a = 1
9	3 >= 0 (true)
10	sum = 1
11	{ factorial(0)
12	0 == 0 (true)
13	повернути 1 }
14	aPrevious = 1
15	повернути sumFunction(3, 1, 1, 1)
16	sumFunction(3, 1, 1, 1)
17	1 == 0 (false)
18	a = 2.449489742783178
19	3 >= 1
20	sum = -0.48316324759439255
21	{ factorial(1)
22	0 == 1 (false)
23	1 == 1 (true)
24	повернути 1 }
25	aPrevious = 2.449489742783178
26	повернути sumFunction(3, 2, 2.449489742783178, -0.48316324759439255)
27	sumFunction(3, 2, 2.449489742783178, -0.48316324759439255)
28	2 == 0 (false)
29	a = 3.4348157113785174
30	3 >= 2
31	sum = 1.157793519997095
32	{ factorial(2)

33	0 == 2 (false)
34	1 == 2 (false)
35	повернути 2 * factorial(1)
36	factorial(1)
37	0 == 1 (false)
38	1 == 1 (true)
39	повернути 1
40	повернути 2 }
41	aPrevious = 3.4348157113785174
42	повернути sumFunction(3, 3, 3.4348157113785174, 1.157793519997095)
43	sumFunction(3, 3, 3.4348157113785174, 1.157793519997095)
44	3 == 0 (false)
45	a = 3.9672739816546665
46	3 >= 3
47	sum = -0.15384125056075937
48	factorial(3)
49	0 == 3 (false)
50	1 == 3 (false)
51	повернути 3 * factorial(2)
52	factorial(2)
53	0 == 2 (false)
54	1 == 2 (false)
55	повернути 2 * factorial(1)
56	factorial(1)
57	0 == 1 (false)
58	1 == 1 (true)
59	повернути 1
60	повернути 2
61	повернути 3}
62	aPrevious = 3.9672739816546665
63	повернути sumFunction(3, 4, 3.9672739816546665, -0.15384125056075937)

64	4 == 0 (false)
65	a = 4.227185343300985
66	3 >= 4 (false)
67	повернути -0.15384125056075937
68	вивід sum
	кінець

◆ Висновок

На лабораторній роботі було декомпозовано задачу на такі етапи: основна програма: визначення основних дій, ініціалізація змінної ***k,*** ініціалізація змінної ***aPrevious,*** ініціалізація змінної ***sum,*** перевірка ***n,*** виклик рекурсивної функції для обчислення суми ***sum,*** підпрограма для функції sumFunction: визначення основних дій, задання умови для обчислення ***a,*** обчислення ***a, з***адання умови для повернення рекурсвної та термінальної гілок, *о*бчислення суми, присвоєння **c1** значення ***a,*** підпрограма для функції factorial: визначення основних дій, перевірка ***c1*** на рівність нулю, задання умови для повернення рекурсвної та термінальної гілок. Було досліджено особливості роботи рекурсивних алгоритмів та набуто практичних навичок їх використання під час складання програмних специфікацій підпрограм.