**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ "ЛЬВІВСЬКА ПОЛІТЕХНІКА"**

Інститут **КНІТ**

Кафедра **ПЗ**

# ЗВІТ

До лабораторної роботи № 2

**На тему:** *“ПРОГРАМУВАННЯ ЦИКЛІЧНИХ ПРОЦЕСІВ В С”*

**З дисципліни:** *“Основи програмування”*

**Лектор:**

ст. викладач кафедри ПЗ

Муха Т.О.

**Виконав:**

студ. групи ПЗ-13

Топала Т.Д.

**Прийняв:**

доц. кафедри ПЗ

Дяконюк Л.М.

«\_\_\_» \_\_\_\_\_\_ 2021 р.

∑ = \_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

Львів – 2021

**Тема роботи:** ПРОГРАМУВАННЯ ЦИКЛІЧНИХ ПРОЦЕСІВ В С

**Мета роботи:** навчитися організовувати програми циклічної структури, які дозволяють

повторювати певну групу операторів задану кількість разів.

## Індивідуальне завдання

## 1. Розробити алгоритми розв’язання індивідуальних завдань і подати їх у вигляді блок-схем.

## 2. Скласти програми на мові С у відповідності із завданнями.

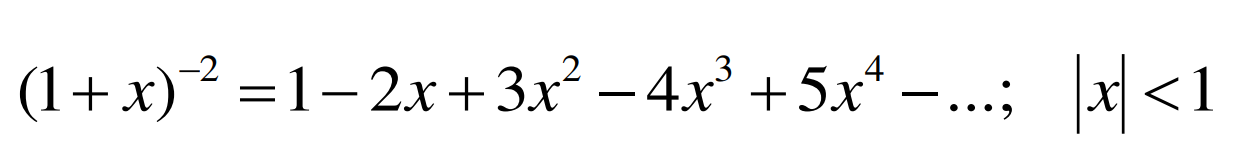
## 3. Виконати обчислення по програмам.

## 4. Підготувати та здати звіт про виконання лабораторної роботи.

## Завдання 1. Знайти і вивести всі цілі числа в діапазоні від 2 до n, для яких у двійковому представленні

## числа одиниці знаходяться тільки в парних розрядах.

## Завдання 2. З допомогою операторів циклу, протабулювати на відрізку від А до В з області визначення функцію, задану розкладом у ряд Тейлора. Для порівняння обчислити також у кожній точці табуляції значення функції задане формулою. Результати подати у виді таблиці з коментарями.



## Теоретичні відомості

#### 2.1. Поняття про цикли

Алгоритм називається *циклічним*, якщо певна послідовність однотипних дій (тіло циклу) виконується багато разів. Однократне виконання тіла циклу називається *ітерацією*. В залежності від постановки задачі розрізняють *арифметичні* цикли (тобто цикли з наперед відомою кількістю ітерацій) та *ітераційні* цикли (число ітерацій такого циклу наперед невідомо). В арифметичних циклах кількість повторень відома до його початку і задається за допомогою лічильника повторень. В ітераційних циклах кількість ітерацій задається неявно, наприклад досягненням якоїсь змінної заданої точності, або заданого значення.

Отже, цикли дозволяють багатократно виконувати сукупність однотипних операцій. Це означає, що програмний код за рахунок циклів виглядатиме компактнішим. Умовою виходу циклу або ж умовою продовження циклу є вираз, від істинності якого залежить, буде виконуватися тіло циклу чи ні. Цикл може мати лічильник (можливо, не один), що зберігає номер ітерації циклу. Значення лічильника водночас може бути і умовою виходу з циклу. У мові С використовуються оператори циклу **while**, **do…while** та **for**. Усі три оператори циклу в С можуть використовуватися для реалізації як арифметичних, так й ітераційних циклічних процесів. Цикл while є циклом *з передумовою*, а цикл do…while – *з післяумовою*. В циклах з передумовою *спочатку* перевіряється умова, а *тоді*, залежно від того, істинна вона чи хибна, виконується або не виконується тіло циклу. У циклах з післяумовою *спочатку* виконується тіло циклу, а *тоді* перевіряється умова, від істинності якого залежить виконання *наступної* ітерації циклу. Одне з типових застосувань циклу while – перевірка, чи не досягнуто кінець файлу. Цикл for, як правило, застосовується при заздалегідь відомій кількості ітерацій – наприклад, для опрацювання всіх елементів масиву.

**2.2. Конструкція while**

Оператор while – це оператор циклу з передумовою. Синтаксис:

#### while(<умова>) <оператор>;

Цикл виконується, поки умова є істинною. Якщо умова початково є хибною (вираз, що задає умову має нульове значення), цикл не виконається жодного разу. У ролі <оператор>, який і є тілом циклу, може бути будь-який допустимий оператор мови С. Єдине обмеження, яке накладає тут мова С полягає в тому, що, структурно, тіло циклу має мати вигляд одного єдиного оператора, що змушує нас використовувати операторні дужки тоді, коли нам потрібно задати тіло циклу за допомогою декількох операторів (аналогічно як було в операторі умови if).

**Приклад**.

Наступний цикл не виконається жодного разу, оскільки умова є хибною:

int k = 0; while(k>0)

{

printf("%d",k);

k--;

}

Цей цикл пройде 5 ітерацій:

int k = 5; while(k>0)

{

printf("%d",k);

k--;

}

***Слід не забувати змінювати змінну, яка використовується в умові циклу while, а інакше відбудеться зациклення!***

**Приклад**. З клавіатури здійснити ввід довільної кількості чисел та порахувати їхнє середнє арифметичне. Після вводу числа користувачеві задають питання, чи він бажає продовжити (користувач натискає клавіші ‘y’ для продовження або довільну іншу клавішу для завершення вводу даних).

#include<stdio.h> #include<conio.h> int main(){

char answer = 'y'; int n=0; double suma = 0; int k = 0; while (answer == 'y'){ printf("Enter a number\n"); scanf("%d",&n); printf("Go on?\n"); scanf("%c",&answer); printf("%c",answer);

suma = suma + n;

k++;

}

printf("Average: %lf ", suma/k);

\_getch(); return 0; }

Результат показаний на рис. 1.

Зображення, що містить стіл

Автоматично згенерований опис

Рис. 1 Результат виконання циклу для довільної кількості чисел, введених з клавіатури

**Приклад**. Порахуємо кількість цифр у введеному з клавіатури числі, за умови, що воно невід’ємне.

#include<stdio.h>

#include<conio.h> #include<math.h> int main(){

int k = 0,n=0; double basenum =10;

printf("Vvedit chyslo:"); scanf("%d",&n); if(n < 0)

printf("Chyslo vidyemne"); else {

while((pow(basenum,k))<=n) k++;

printf("K-st cyfr: %d",k);

}

\_getch(); return 0; }

Пояснимо роботу даного фрагмента коду. Нехай n = 235. Спочатку змінна k рівна 0, відповідно 10k = 100=1. Перевіряється, чи 1 менше рівне за введене число n. Оскільки умова справджується, то змінна k збільшується на 1 і стає рівною 1. На другій ітерації обчислюється вираз-умова (10k = 101=10)<=n. Оскільки умова істинна, то змінна k збільшується ще на 1 і стає рівною 2. На третій ітерації обчислюється вираз (102<=n). Оскільки 100<=235, то k = 3. На четвертій ітерації обчислюється вираз-умова 103<=235. Оскільки умова хибна, то змінна k залишається рівною 3 і робиться висновок, що число n – тризначне.

##### 2.3. Конструкція do…while

Цикл **do…while** відрізняється від циклу while тим, що умова його виконання перевіряється після виконання тіла циклу. Це означає, що тіло циклу виконається принаймні один раз.

Синтаксис цього оператоа:

#### do <тіло циклу> while(<умова>)

Інша відмінність оператора циклу do...while полягає в тому, що тіло циклу може складатися з довільної кількості операторів, тобто використання операторних дужок не є обов’язковим. **Приклад**.

int k = 0; do

{

printf("%d",k);

k--;

} while (k>0);

Умова є від початку хибною, проте цикл встигне виконатися один раз (на екран буде виведено число 0).

В наступному циклі робиться ввід символів з клавіатури, поки користувач не натисне символ

‘a’:

char ch;

do

scanf("%c",&ch); while (ch!='a');

##### 2.4. Конструкція for

Цикл **for** управляється змінною, що називається лічильником циклу. Величину зміни лічильника називають кроком. Синтаксис циклу є наступний:

**for(<початкова\_інструкція>; <умова>; <вираз>) <тіло циклу>;**

Спочатку виконується початкова інструкція та перевіряється умова. Якщо вона істинна, то виконуються інструкції з тіла циклу, а тоді обчислюється вираз та управління передається в початок циклу з тією різницею, що початкова інструкція вже не виконується.

**Приклад.**

#include<stdio.h> #include <conio.h> int main()

{

*/\* Змінній циклу і присвоюється початкове значення 0. Перевіряється умова, чи і менше 10. Якщо так, то на екран виводиться значення і (і=0). Змінна циклу і збільшується на 1. Перевіряється умова, чи і менше 10. Виводиться на екран поточне значення і (і=1). Цикл пройде 10 ітерацій, поки змінна і не стане рівною 10. В результаті на екран будуть виведені десять цифр від 0 до 9 \*/*

for(int i = 0; i != 10; i++) *// змінну циклу можна ініціалізувати прямо в початковій\_інструкції*  printf("%d",i);

\_getch(); return 0;

}

Лічильник циклу може визначатися всередині його заголовка. Область видимості лічильника сягає або кінця тіла циклу, або кінця функціїї. Значення лічильника можна змінювати всередині тіла циклу.

Цикл може використовувати декілька змінних:

for(int i = 0, j=10; i <= 10&&j!=5; i++,j--) printf("%d",i);

В результаті виконання цього циклу буде виведено на екран цифри від 0 до 4, оскільки на 6ій ітерації циклу змінна j стане рівною 5 і умова циклу перестане бути істинною. Наступний цикл виводить на екран всі натуральні парні числа від 1 до 100:

for(int i = 1; i <= 100; i++) if (!(i%2))

printf("%d",i);

Змінна циклу може бути не лише цілим числом, але й літерою. Наприклад:

for(char ch = 'a'; ch <= 'z'; ch++) printf("%c",ch);

В результаті виконання цього циклу на екран будуть виведені усі малі латинські літери.

Результатом виконання наступного циклу будуть виведені на екран усі дільники введеного з клавіатури числа n, а також їхня загальна кількість int n; printf("Enter n");

scanf("%d",&n); int k = 0; */\* число дільників зберігатиметься у цій змінній*

*\*/*

for(int i=1;i<=n;i++)

if(!(n%i)) */\* якщо n ділиться на i без остачі, то збільшуємо кількість дільників та виводимо i на екран \*/*

{ k++;

printf("Dil'nyk chysla %d #%d : %d",n,k,i);

}

У циклі **for** можуть бути відсутні ініціалізація лічильника і/або умова виходу і/або приріст. Відсутність умови виходу інтерпретується як істинний вираз, тобто, цикл стає нескінченним. Якщо ініціалізація не виконується в заголовку циклу, то її слід здійснити раніше. Відсутність всіх виразів одночасно (for(;;) {}) означає нескінченний цикл. Згідно синтаксису С, оператор **for** може бути пустим.

**2.5. Оператор break**

Оператор **break** дозволяє здійснити “достроковий” вихід із циклу, коли умова циклу є істинною. Він використовується тоді, коли продовжувати цикл недоцільно.

**Приклад**. Визначимо, чи введене з клавіатури число n є простим (число просте, якщо ділиться лише само на себе та на 1). int n; printf("Enter n"); scanf("%d",&n);

int proste = 1; */\* застосуємо змінну, що служитиме ознакою, є число простим чи ні. Спочатку вважатимемо, що число просте і у цю змінну запишемо 1 \*/* for(int i=2;i<n;i++) {

if(!(n%i)) */\* якщо знайшлося відмінне від 1 та від n число і, на яке*

*n ділиться без остачі, то ясно, що число n не є простим, а тому продовжувати цикл недоцільно \*/*

{ proste=0; */\* міняємо значення змінної-ознаки. Вона буде індиктором того, що вихід з циклу відбувся саме через оператор break, а не завершився через те, що умова циклу стала хибною\*/*

break;

}

}

if(proste) printf("Chyslo %d proste ",n); */\* якщо змінна-ознака лишилася незмінною, то число насправді просте \*/*

**2.6. Оператор continue**

Оператор **continue** дозволяє проігнорувати всі інструкції, записані у тілі циклу після нього, та перейти до наступної ітерації циклу.

**Приклад**. Виведемо тангенс кута angle, де angle перебігає множину всі парних значень від 0

*n* де n – непарне число. до 360 у градусах. Відомо, що функція тангенса не визначена для ,

2 Цикл виглядатиме так:

double pi = 3.1415926535897932384626433832795; for(int angle=0;angle <= 360; angle+=2) {

if(!(angle%90)&&((angle/90)%2)) */\* якщо кут ділиться на 90 без остачі, причому результат цього ділення є непарним числом, то переходимо до наступної ітерації циклу \*/*

continue;

printf("%d \t %lf",angle, tan(angle\*pi/180)); }

**Приклад**. Напишемо цикл для обчислення наступного виразу:

10 3 8 6    6 9 4 12 2 15    

Враховуючи, що це є сума добутків відповідних членів двох рядів, організуємо наступний цикл:

int n = 10; int m = 3; int i,j; int s = 0; */\* у змінній s зберігатимемо поточне значення суми \*/*  for(i=n,j=m;i>=2;i-=2,j+=3){ printf("%d\n", i); printf("%d\n", j);

s = s + i\*j;

}

printf("Sum: %d",s);

##### 2.7. Вкладені цикли

У мові С існують *вкладені* цикли. Тобто, один цикл (внутрішній) може знаходитися в тілі іншого циклу (зовнішнього). При цьому глибина вкладення (кількість вкладених циклів) не обмежується. Типове застосування вкладених циклів – для обробки матриць (двовимірних масивів). Цикл **for** може знаходитися у тілі циклу **while** (чи do…while), і навпаки.

**Приклад**. З клавіатури ввести довільну кількість чисел (ввід числа супроводжується натисканням клавіші ‘y’ для продовження або ж довільної іншої клавіші для завершення вводу даних, як розглядалося вище). Знайти та вивести число з максимальною кількістю дільників, а також вивести саме максимальне число дільників. Потрібний програмний код мовою С можна утворити, застосувавши розглянуті вище фрагменти кодів – один для вводу довільної кількості чисел з клавіатури, а другий – для знаходження кількості дільників числа.

char answer = 'y'; int n; int maxd = 0; int nummux = 0; */\* у змінній maxd зберігатимемо поточне значення максимальної кількості дільників введеного числа, а у змінній nummux – саме число, що володіє найбільшою кількістю дільників \*/* while (answer=='y'){ printf("Enter a number"); scanf("%d",&n); printf("Go on?"); scanf("%c",&answer);

int k = 0; */\* перед внутрішнім циклом слід не забути «скидати в нуль» лічильник дільників, оскільки для кожного нового розглядуваного числа підрахунок починається наново \*/*

for(int i=1;i<=n;i++)

if(!(n%i)) k++;

if(maxd<k) */\* якщо поточне число дільників виявилося більшим за maxd, тоді у змінну maxd записуємо поточне число дільників, а у змінну nummux – відповідне число n \*/*

{ maxd = k;

nummux = n;

}

}

printf("Najbil'she dilnykiv (%d) maye chyslo:%d",maxd, nummux);

Недоліком наведеного коду є те, що якщо два (і більше) числа мають однакову кількість дільників, то відображатиметься тим не менш лише одне число. Для усунення цього недоліку варто записувати всі пораховані значення в масив, а з масиву видобувати *всі* однакові найбільші значення.

Приклад результату виконання показано на рис. 2.

Зображення, що містить текст

Автоматично згенерований опис

Рис. 2 Результат виконання програми **Приклад**. Роздрукувати таблицю множення.

*/\* Цей цикл виводить в ряд цифри від 0 до 9, після чого пропускаються два рядки дл полегшення читання \*/* for(int i=1;i<10;i++)

printf ("\t%d",i); printf ("\n");

for(int i=1;i<10;i++) /\* у зовнішньому циклі “перебираємо цифри від 1

до 9” та виводимо кожне з них на початку рядка \*/

{printf ("%d\t",i);

for(int j = 1;j < 10; j++) /\* у внутрішньому циклі знову “перебираємо” усі цифри від 1 до 9 та результат множення j на поточне

значення i виводимо на екран \*/

{

printf ("%d\t",i\*j);

}

printf ("\n");

}

Для відокремлення чисел зручно використовувати символи табуляції.

Зображення, що містить текст, чорний, електроніка

Автоматично згенерований опис

Рис. 3 Результат роздруку таблиці множення

##### 2.8. Ітераційні цикли

Багато задач у математиці передбачає використання числових методів, в яких обчислення проводиться до досягнення заданої точності. Так, наприклад, обчислення суми нескінченого ряду Тейлора проводиться до тих пір поки наступний член ряду за абсолютним значенням стане меншим за значення точності , де - достатньо мале число. При розв’язуванні таких задач наперед не відома кількість ітерацій циклу. Крім того в більшості таких задач часто використовується *алгоритм накопичення добутку:*

*Spi* *Spi*1 \**p* Продемонструємо сказане на прикладі :

|  |
| --- |
| Обчислити значення функції, заданої розкладом в ряд:  *x*   1 *x x*2    *x*3 ... *xn* ... *e*  1! 2! 3! *n*!  з точністю =10-5, тобто |Si|<, де Si – елемент ряду. |

Оскільки обчислювати *xn* - на кожному кроці неефективно, використаємо, так звану *n*!

*рекурентну* формулу, тобто формулу знаходження наступного елемента за попереднім:

*S*1  *x* 1!

*S*2  *S*1 \* *p*  *x* \* *x*  *x*2 1! 2 2!

*S*2\* *p* *x*2 \* *x x* 3

*S*3

2! 3 3! ………………….

*Si*  *Si*1 \* *x*

*i*

Блок-схема алгоритму з детальними поясненнями наведена на рис.4.

## 1

**ні**

**так**

Початок

S

i

S

=

i

\*x/I

ввести

x,



|S

i

|



S

i

= S+S

i

Вивід

S

кінець

S

i

x

=

i=2

2

3

5

6

7

9

S

i

=

1+S

i

I= I+1

4

10

8

Блок 2 – ввід значення х і точності Е.

Блоки 3, 4 – присвоєння початкового значення змінним Si, S, I.

Блок 5 – початок циклу, який буде продовжуватись до того часу, поки

|Si|>.

Блоки 6, 7, 8 – тіло циклу, де обчислюється біжуче значення S, змінна i збільшується на 1.

Блок 9 – вивід результату

S.

Рис.4. Блок-схема обчислення суми елементів нескінченого ряду Тейлора.

Сама програма мовою С:

void main()

{

int i;

double x, s, si, eps, y;

printf("Vvedit x i eps:"); scanf("%lf, %lf",&x,&eps); si=x; i=2; s=1+si; while (fabs(si)>eps) {

si\*=x/i; s+=si; i++; }

y=exp(x); // точне значення

printf("x=%lf y=%lf s=%lf",x,y,s);

\_getch();

}

##### 2.9. Побітові операції

Оскільки числа в комп’ютері представляються у двійковій системі числення і комп’ютер виконує обчислення саме над двійковими числами, то в мові С передбачені побітові операції, які дозволяють перевіряти та встановлювати задані розряди чи множину розрядів у цілочисельних змінних. Часто побітові операції використовуються для швидкого множення на 2.

Оператор **побітового заперечення ~** інвертує всі розряди свого операнда (нулі стають одиницями, і навпаки). Основним призначенням цієї операції є інвертування логічних значень “істина” та “хибність”, а також перетворення додатних чисел у від’ємні і навпаки.

Порозрядний **оператор зсуву бітів вліво (<<)** переносить всі біти першого операнда на задану другим операндом кількість позицій вліво, при цьому позиції, що звільняються, заповнюються нулями. У двійковій системі числення зсув біта на 1 позицію вліво еквівалентний множенню на 2:

3<<4 = 3\*2\*2\*2\*2=3\*16=48

Аналогічно **порозрядний оператор зсуву бітів вправо (>>)** переносить усі біти на задану кількість позицій вправо, при цьому позиції, що звільняються справа, заповнюються нулями, а молодші біти зникають. Зсув біта на одну позицію вправо еквівалетне діленню на 2:

32>>4 = 32/(2\*2\*2\*2) = 32/16 = 2.

Порозрядна операція **& (“I”)** є бінарною та задається таблицею істинності:

# Біт1 Біт2 Біт1 & Біт2

0 0 0

## 0 1 0

1 0 0

1 1 1

Операція “І” використовується для фільтрування одиниць, які входять у двійкове представлення числа. У цьому випадку другий операнд називається маскою.

Порозрядна операція **| (“АБО”)** є бінарною та задається таблицею істинності:

##### Біт1 Біт2 Біт1 & Біт2

0 0 0

1. 1 1
2. 0 1

1 1 1

Операція “АБО” дозволяє вставляти в результат потрібні розряди.

Порозрядна операція **^ (“виключаюче АБО”)** є бінарною та задається таблицею істинності:

##### Біт1 Біт2 Біт1 & Біт2

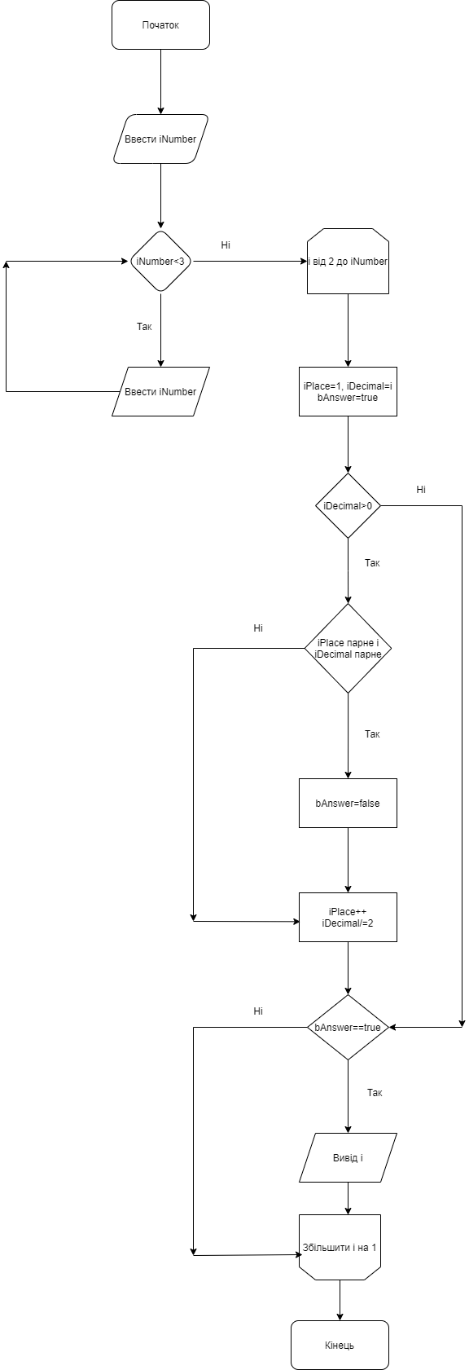
0 0 0

1. 1 1
2. 0 1

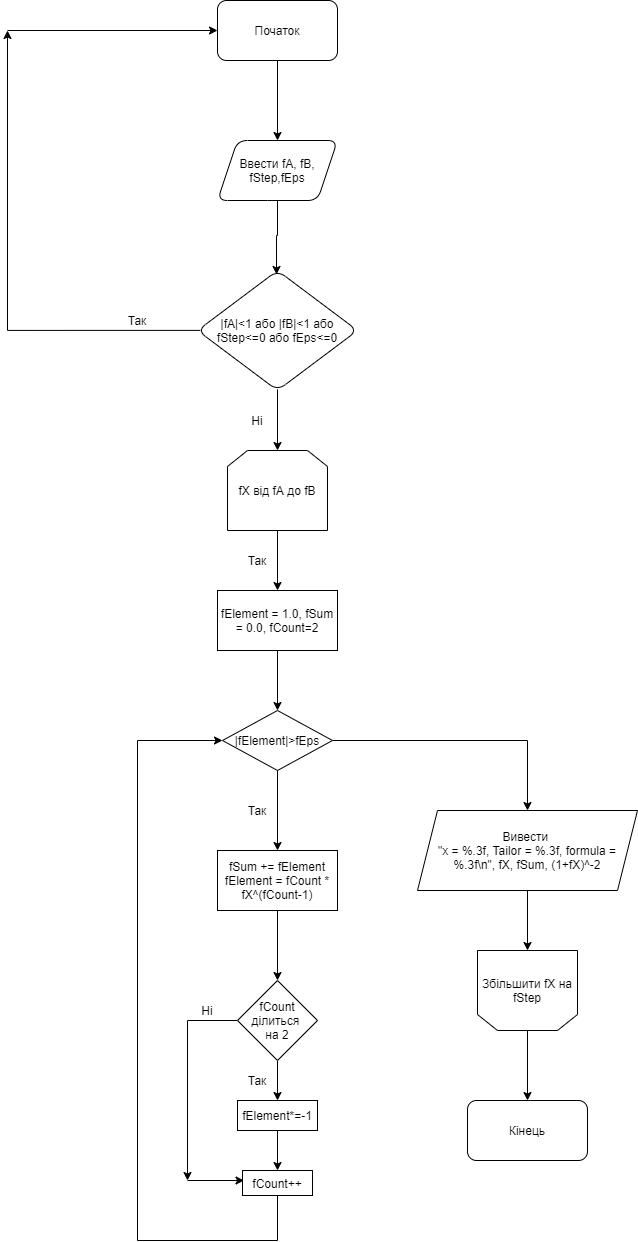
## 1 1 0

## Блок-схеми

Назва файлу: **Lab2Task1.png**



Назва файлу: **Lab2Task2.png**

****

## Код програми

Назва файлу: **Task1.c**

#include <stdio.h>

#include <stdbool.h>

int main(void)

{

int iNumber = 0;

scanf\_s("%i", &iNumber);

while (iNumber < 3)

{

printf("Number is less that 3! Try again... ");

scanf\_s("%i", &iNumber);

}

for (int i = 2; i <=iNumber; i++)

{

int iPlace = 1, iDecimal = i;

bool bAnswer = 1;

while (iDecimal)

{

if (iPlace % 2 && iDecimal % 2)

bAnswer = 0;

iPlace++;

iDecimal /= 2;

}

if (bAnswer)

printf("%i\n",i);

}

return 0;

}

Назва файлу: **Task2.c**

#include <stdio.h>

#include <math.h>

int main(void)

{

float fA, fB, fStep, fEps;

startInput:

printf("Enter a...");

scanf\_s("%f", &fA);

printf("Enter b...");

scanf\_s("%f", &fB);

printf("Enter step...");

scanf\_s("%f", &fStep);

printf("Enter eps...");

scanf\_s("%f", &fEps);

if (fabs(fA) >= 1 || fabs(fB) >= 1 || fA > fB || fStep <= 0 || fEps <= 0)

{

printf("Incorrect data!");

goto startInput;

}

for (float fX = fA; fX <= fB; fX += fStep)

{

float fElement = 1, fSum = 0;

int fCount = 2;

while (fabs(fElement) > fEps)

{

fSum += fElement;

fElement = fCount \* pow(fX, fCount - 1);

if (fCount % 2 == 0)

fElement \*= -1;

fCount++;

}

printf("x = %.3f, Tailor = %.3f, formula = %.3f\n", fX, fSum, pow(1 + fX, -2));

}

## }

## Протокол роботи

Завдання 1.

Протокол вводу:

36

Протокол виводу на екран:

*2*

*8*

*10*

*32*

*34*

На рис. 1 наведено результати виконання програми.

Зображення, що містить текст

Автоматично згенерований опис

Рис. 1. Результат виконання програми 1

Завдання 2.

Протокол вводу:

-0.5

0.5

0.1

0.0000001

Протокол виводу на екран:

x = -0.500, Tailor = 4.000, formula = 4.000

x = -0.400, Tailor = 2.778, formula = 2.778

x = -0.300, Tailor = 2.041, formula = 2.041

x = -0.200, Tailor = 1.562, formula = 1.563

x = -0.100, Tailor = 1.235, formula = 1.235

x = -0.000, Tailor = 1.000, formula = 1.000

x = 0.100, Tailor = 0.826, formula = 0.826

x = 0.200, Tailor = 0.694, formula = 0.694

x = 0.300, Tailor = 0.592, formula = 0.592

x = 0.400, Tailor = 0.510, formula = 0.510

x = 0.500, Tailor = 0.444, formula = 0.444

Зображення, що містить текст

Автоматично згенерований опис

Рис. 2. Результат виконання програми 2

## Висновки

На лабораторній роботі розглянуто роботу циклів в мові C. Для обох завдань створено блок-схеми алгоритмів їх вирішення та у відповідності з ними складено програми на C.