**Теоретичний матеріал**

[1. Циклічні алгоритми](#_Toc526173340)

[1.1 Цикли з передумовою](#_Toc526173341)

[1.2 Цикл while](#_Toc526173342)

[1.3 Цикл for](#_Toc526173343)

[1.4 Цикл do … while](#_Toc526173344)

[2. Рекурентні обчислення](#_Toc526173345)

[3. Обчислення суми нескінченого ряду](#_Toc526173346)

# 1. Циклічні алгоритми

Рішення багатьох практичних завдань досягається в результаті багаторазового повторення деяких дій. Для здійснення повторень в алгоритмічних мовах та мовах програмування застосовуються спеціальні команди повторення, які називаються операторами циклу.

У циклічних алгоритмах обчислення проводяться до тих пір, поки не виконається (або не порушиться) вказана в заголовку циклу умова. Залежно від місця перевірки умови, команди повторення можна розділити на дві групи: «цикл з передумовою» і «цикл з постумовою».

1.1 Цикли з передумовою

Цикл з передумовою буде виконуватися до тих пір, поки логічний вираз в заголовку циклу є істинною.

Для правильної організації роботи циклу з передумовою необхідно:

* до перевірки логічного виразу в заголовку циклу ініціювати параметри циклу (*задати їм початкове значення,* *наприклад, i: = iпоч*.);
* правильно скласти логічний вираз, що визначає умову входу в цикл (*наприклад, i <= iкін*.);
* передбачити зміну значення параметра циклу (*наприклад, i = i + h, де h - величина кроку зміни параметра циклу*).

До циклів з передумовою відносяться:

* цикл while;
* цикл for.

1.2 Цикл while

Блок схема виконання циклу ***while*** наведена на рис.1.

Підготовка циклу

Логічний вираз

Оператор(и) тіла циклу

+

**Підготовка циклу:** всім величинам, що змінюються по рекурентним формулами, присвоюються початкові значення (в тому числі – зміннимв заголовку циклу)

**Заголовок циклу**: перевірка умови продовження циклу

**Тіло циклу**:

* реалізація необхідних в завданні дій;
* отримання нових значень величин, які використовуються в заголовку циклу.

-

Рис.1 – Блок схема циклу while

Структура циклу ***while*** в мові С++:

Оператори\_підготовки\_циклу;

**while** (логічний вираз)

{

Оператори\_тіла\_циклу;

Оператори\_переходу\_до\_наступної\_ітерації\_циклу;

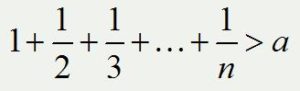
}

Послідовність операторів тіла циклу виконується до тих пір, поки логічний вираз повертає значення true. Як тільки вираз стає рівним false, виконання циклу while припиняється і управління передається наступному за циклом while оператору.

***Приклад 1.***

Дано дійсне позитивне число *a*. Знайти таке найменше *n*, при якому

.



**Рішення задачі:** На початку значення суми є меншим за значення *a*. При проходженні кожної ітерації значення суми поступово зростає. В якийсь момент (при якомусь значенні *n*) ця сума стане вищою за значення *a*. Цей момент (значення *n*) потрібно зафіксувати.

Для обчислення суми та кількості кроків використовуємо метод накопичення.

Фрагмент коду, що розв‘язує дану задачу:

float a;

int n;

float s;

//підготовка параметрів циклу та змінних для обчислення

cout<<”a=”;

cin>>a;

s = 0.0;

n = 0;

while (s <= a) //заголовок циклу: продовжуємо, поки сума <= значення a

{

s = s + 1.0/n; //обчислення наступного значення суми

n++; //перехід до наступного елементу ряду

}

1.3 Цикл for

Цикл ***for*** також відноситься до циклів з передумовою. Його особливість полягає в тому, що присвоювання початкового значення параметра циклу (*наприклад, i = iпоч.*), що змінюється по рекурентним формулам, перевірка умови продовження циклу (*наприклад, i≤iкін.*) і отримання нового значення параметру циклу (*i = i + крок*), виконується в заголовку циклу. Блок схема виконання циклу ***for*** наведена на рис.2.

Оператор(и) тіла циклу

пар =поч, кін, крок]

Рис.2 – Блок схема циклу for

В мові C++ цикл ***for*** може мати дуже широку реалізацію та застосування.

Загальна форма оператора циклу ***for***:

**for** (ініціалізація**;** логічний\_вираз**;** підготовка\_наступної\_ітерації)

{

Оператори\_тіла\_циклу;

}

***Зверніть увагу!***

Блоки в заголовку циклу ***for*** відокремлюються як і оператори мови – за допомогою **;**

Перший і третій блоки можуть містити декілька операторів, блок логічного виразу – тільки один вираз. Якщо перший чи третій блоки містять декілька операторів, то вони відокремлюються один від одного комою.

***Блок ініціалізації.*** Містить оператори присвоювання, в яких встановлюються початкові значення змінних циклу та початкові значення змінних, які використовуються в тілі циклу для вирішення задачі.

***Блок логічного виразу.*** Визначає можливість подальшого виконання циклу. Послідовність операторів тіла циклу виконується до тих пір, поки логічний вираз повертає значення true. Як тільки вираз стає рівним false, виконання циклу ***for*** припиняється і управління передається наступному за циклом ***for*** оператору.

***Блок підготовки наступної ітерації***. Зазвичай визначає, як будуть змінюватись значення змінних циклу після кожної ітерації. Оператори цього блоку виконуються **після** кожного виконання операторів тіла циклу!

Фрагмент коду, що розв‘язує задачу з прикладу 1:

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

n = 0;

//підготовка та зміна параметрів циклу виконується безпосередньо в циклі

for (s = 0.0; s <= a; s = s + 1.0/n)

{

n++;

}

***Зверніть увагу!***

Одне й те ж рішення може бути записано за допомогою циклу ***for*** декількома способами, які працюють абсолютно однаково!

Наприклад, фрагменти 1, 2 і 3 виконують одне й те ж!

**//Фрагмент 1**

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

n = 0;

for (s = 0.0; s <= a; s = s + 1.0/n)

n++;

**//Фрагмент 2**

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

s = 0.0;

n = 0;

for (; s <= a;)

{

n++;

s = s + 1.0/n;

}

**//Фрагмент 3**

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

for (s = 0.0, n = 0; s <= a; n++, s = s + 1.0/n);

Фрагмент 2 специфічний тим, що в заголовку циклу залишився тільки логічний вираз. Якщо ви стикаєтеся із такою ситуацією, то краще оформити такий цикл за допомогою оператору ***while***.

Фрагмент 3 специфічний тим, що в результаті перенесення операторів присвоєння, що відповідають за ініціалізацію, перехід до наступної ітерації та обчислення самої задачі, **тіло циклу стало пустим!** Таких ситуацій слід уникати, оскільки **це значно знижує розуміння коду**. Крім того, помилки в записі операторів в третьому блоці може призвести до обчислення зовсім інших результатів. Наступні фрагменти **дуже схожі, але призводять до різних результатів**!

**//Фрагмент 3**

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

for (s = 0.0, n = 0; s <= a; n++, s = s + 1.0/n);

**//Фрагмент 4**

float a;

int n;

float s;

cout<<”a=”;

cin>>a;

for (s = 0.0, n = 0; s <= a; s = s + 1.0/n, n++);

1.4 Цикл do … while

Цикл ***do … while*** відноситься до категорії циклів з постумовою. Його доцільно використовувати у випадках, коли ітерацію потрібно зробити хоча б 1 раз. На відміну від циклів for та while, у циклі ***do…while*** умова перевіряється при виході з циклу (а не при вході в цикл). Блок схема виконання циклу ***do…while*** наведена на рис.3.

Підготовка циклу

Оператор(и) тіла циклу

-

Всім величинам, що змінюються по рекурентним формулами, присвоюються початкове значення.

Реалізація необхідних в завданні дій.

Отримання нових значень величин, які використовуються в логічному виразі.

Перевірка умови продовження циклу.

+

Логічний вираз

Рис.3 – Блок схема циклу do…while

Загальна форма оператора циклу do…while в мові С++:

**do**

{

Оператори\_тіла\_циклу;

}

**while** (логічний\_вираз);

***Приклад 2.***

Наведений фрагмент дозволяє організувати перевірку введення додатного числа N. Введення виконується до тих пір, поки користувач не введе додатне число.

int N;

do {

cout<<”Enter positive N”;

cin>>N;

if(N <=0)

cout<<”N is not positive!”;

} while (N <=0);

# 2. Рекурентні обчислення

Рекурентні обчислення отримали широке розповсюдження в програмуванні, особливо при організації циклів. Будемо розуміти під рекурентним обчисленням ситуацію, коли наступне значення змінної розраховується з використанням її попереднього значення.

Для програмування рекурентних обчислень спочатку необхідно розробити їх математичну модель у вигляді рекурентної формули.

У загальному випадку ***рекурентним співвідношенням*** називається формула виду:

*an+1=F(an, an-1,...,an-k+1)*,

де *F* – деяка [функція](https://uk.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D1%96%D1%8F_(%D0%BC%D0%B0%D1%82%D0%B5%D0%BC%D0%B0%D1%82%D0%B8%D0%BA%D0%B0)) від *k* [аргументів](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D0%B3%D1%83%D0%BC%D0%B5%D0%BD%D1%82), яка дозволяє обчислити наступні члени [числової послідовності](https://uk.wikipedia.org/wiki/%D0%A7%D0%B8%D1%81%D0%BB%D0%BE%D0%B2%D0%B0_%D0%BF%D0%BE%D1%81%D0%BB%D1%96%D0%B4%D0%BE%D0%B2%D0%BD%D1%96%D1%81%D1%82%D1%8C) через значення попередніх членів.

Приклади рекурентних формул:

1. Рекурентне співвідношення [арифметичної прогресії](https://uk.wikipedia.org/wiki/%D0%90%D1%80%D0%B8%D1%84%D0%BC%D0%B5%D1%82%D0%B8%D1%87%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F):

*an+1*= *an* + d.

1. Рекурентне співвідношення [геометричної прогресії](https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BE%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D1%87%D0%BD%D0%B0_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B5%D1%81%D1%96%D1%8F):

*an+1*= *an* ·q.

1. [Послідовність Фібоначчі](https://uk.wikipedia.org/wiki/%D0%9F%D0%BE%D1%81%D0%BB%D1%96%D0%B4%D0%BE%D0%B2%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D0%A4%D1%96%D0%B1%D0%BE%D0%BD%D0%B0%D1%87%D1%87%D1%96):

*an+1*= *an* + *an-1*, a1=1, a2=1.

У найбільшій кількості випадків рекурентна формула має вигляд:

*(n)* або



де n – номер значення змінної, що розраховується, - вираз, що показує зв'язок між теперішнім та майбутнім значеннями змінної.



Для початку обчислень має бути задане початкове значення .



Основна задача при розробці рекурентної формули – це пошук виразу . Якщо відомий математичний вираз для розрахунку елемента послідовності , то для пошуку можна використати (в залежності від типу формули) наступний підхід:



або



Наприклад, розглянемо пошук рекурентної формули для розрахунку n!.

Як відомо, *n*!*=1\*2\*3\*…\*n, 0*!=1.



Таким чином, для розрахунку *n*! можна використовувати рекурентну формулу:



У складних випадках в тілі циклу може бути використано кілька рекурентних формул, які необхідно правильно узгодити між собою.

# 3. Обчислення суми нескінченого ряду

Однією з поширених задач, в якій використовуються рекурентні формули, є розрахунок значень функцій за допомогою їх розкладення у ступеневі ряди. Відомо, що будь-яку функцію можна наближено представити у вигляді деякого ряду. Підставивши значення аргументу, можна знайти часткову суму ряду і вважати її значенням функції від того самого аргументу з певною точністю наближення. Чим більша точність потрібна, тим більший відрізок ряду треба буде обчислювати.

***Зверніть увагу!***

Для обчислень в програмі можна використовувати будь-який з циклів. Головне правило – це враховувати в сумі тільки ті елементи, які необхідні для вирішення задачі!

***Приклад 3.***

Обчислити суму нескінченого ряду з точністю до члена ряду, меншого ε (0<ε<1).



***Рішення:***

Позначимо поточний член ряду як *Un = (-1)*

Для обчислення поточного члена ряду використаємо рекурентну формулу:

*Un =  \* Un-1.*

Коефіцієнт ** обчислюється таким чином:



Початкове значення елемента ряду:

.

Початкове значення суми:



Фрагмент програми для розрахунку суми із заданою точністю ε:

float u;

float x;

int n;

float S;

float eps;

do{

cout<<”x=”;

cin>>x;

if (x==0)

cout<<”Incorrect value x”<<endl;

}while (x==0);

do{

cout<<”eps=”;

cin>>eps;

if (eps<=0 || eps>=1)

cout<<”Incorrect value eps”<<endl;

}while (eps<=0 || eps>=1);

S = 0.0;

n = 0;

u = 1;

do{

S+=u;

n++;

u\*=-x\*x/(2\*n)/(2\*n-1);

}while (abs(u)>eps);

cout<<”S=”<<S<<endl;

cout<<”n=”<<n<<endl;

***Зверніть увагу!***

Для обчислень в програмі використовується цикл ***do…while***. Це призводить до того, що в сумі буде врахований мінімум один член ряду. Якщо проаналізувати саму послідовність та вимоги до точності, то можна побачити, що перший елемент завжди дорівнює 1 (він не залежить від значення *х*) і повинен бути врахованим в сумі. Для задач з іншими формулами для обчислень цикл ***do…while*** може не підійти.