ОРГАНІЗАЦІЯ ЦИКЛІВ. КЕРУЮЧІ ОПЕРАТОРИ В ЦИКЛАХ

Оператор циклу з параметром

Цикл `for` переважно використовується, коли наперед відома кількість повторень, або коли умова продовження виконання циклу записується коротким виразом.

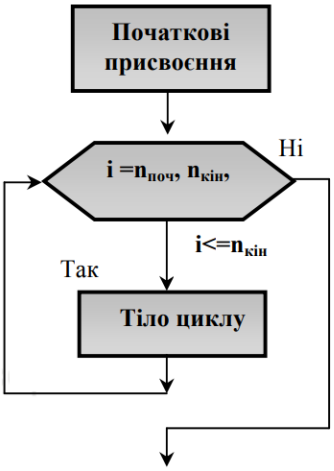
Оператор циклу for має такий ***формат:***

```
for (<ініціалізація>; <умова>; <модифікація>)
<тіло_циклу>;
```

За замовчуванням після оператора for виконується лише один оператор, тому якщо у циклі потрібно виконати групу команд, то їх слід записати в операторних дужках {}.

```
for (<ініціалізація>; <умова>; <модифікація>)
{
<оператор_1>;
<оператор_2>;
...
<оператор_N>; }
```

Для зображення циклу на блок-схемі використовується наступна сукупність блоків:



Оператор for складається з трьох основних блоків, записаних у круглих дужках і відокремлених один від одного крапкою з комою (;), та тіла циклу.

У блоці ***ініціалізації*** задаються початкові значення змінних (параметрів), які керують циклом. У блоці ***умови*** задається та перевіряється умова і, якщо вона виконується (`true` чи має ненульове значення), то виконується тіло циклу. Якщо ж умова не виконується (`false` чи дорівнює нулю), то відбувається вихід із циклу, а керування передається на перший оператор після циклу `for`.

Перевірка умови виконується на початку кожної ітерації циклу.

Для обчислення суми перших десяти чисел можна скористатись наступними конструкціями

```
int s = 0;
for (int i = 1; i <= 10; i++)
    s += i;
```

або

```
for (int s = 0, i = 10; i >= 1; i--)
    s += i;
```

або

```
for (int s = 0, i = 1; i <= 10; s += i++);
```

В операторі `for` блок ініціалізації може бути відсутнім, якщо початкове значення задати попередньо; блок умови також, якщо припускається, що умова завжди істинна (слід виконувати тіло циклу, поки не зустрінеться оператор `break`) а блок модифікації може бути пропущений, якщо зміну значення параметра виконувати у тілі циклу чи коли значення параметра змінювати непотрібно.

Коли певний блок відсутній, тоді вираз цього блока пропускається, але крапка з комою (;) обов'язково має залишитись. Крім того, можлива наявність порожнього оператора (оператор відсутній) у тілі циклу.

При використанні вкладених циклів потрібно слідкувати, щоб внутрішній цикл повністю вкладався у тіло зовнішнього циклу. Крім того, внутрішній цикл може містити власні вкладені цикли. Імена параметрів зовнішнього та внутрішнього циклів мають бути різними.

```
for (k = 1; k <= 10; k++)
  {
      for (i = 1; i <= 10; i++)
          {
             for (m = 1; m <= 10; m++)
                 {   }
           }
   }
```

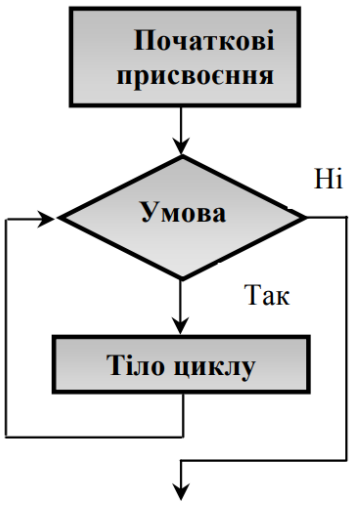
Оператор циклу з передумовою

Цикл з передумовою `while` використовується, якщо кількість повторень наперед невідома або ж немає явно вираженого кроку зміни параметра циклу.

Синтаксис циклу з передумовою:

```
while (<умова>)
  { <тіло_циклу> };
```

Для зображення циклу на блок-схемі використовується наступна сукупність блоків:



Умова цикла `while` перевіряється перед кожною ітерацією. Тіло циклу виконується доти, поки умова істинна (`true`, має ненульове значення), а вихід з цикла здійснюється, коли умова стає хибною (`false`, має нульове значення). Якщо умова є хибною при входженні у цикл, то послідовність операторів не буде виконуватися жодного разу, а керування передасться наступному оператору програми.

Для обчислення суми всіх непарних чисел у діапазоні від 10 до 100 можна використати наступний цикл:

```
int s = 0, i = 11;
while (i < 100)
{
    s += i;
    i += 2;
}
```

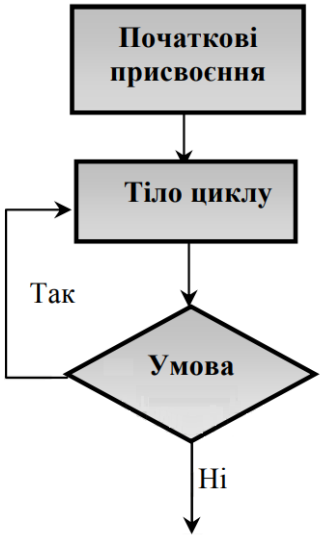
Оператор циклу з післяумовою

Відмінність циклу з післяумовою `do - while` від циклу з передумовою полягає в тому, що цикл з післяумовою *завжди виконується принаймні один раз незалежно від істинності чи хибності умови.*

Синтаксис циклу з післяумовою:

```
do {
 <тіло_циклу> }
while (<умова>);
```

Для зображення циклу на блок-схемі використовується наступна сукупність блоків:



Тіло циклу виконується доти, поки умова є істинна (ненульова). Якщо тіло циклу складається з одного оператора, то операторні дужки {} не обов'язкові.

Обчислити програмно суму всіх непарних чисел у діапазоні від 10 до 100 використовуючи цикл з післяумовою можна так:

```
int s = 0, i = 11;
do {
    s += i; i += 2;
   }
while (i < 100);
```

У тілі циклів з передумовою та післяумовою слід передбачати зміну параметрів, які використовуються в умові, інакше вихід із циклу ніколи не виконається і відбуватиметься зациклювання.

Оператори переривання виконання циклів

Для завчасного виходу з циклу використовують оператори `break` (вихід з конструкції), goto (безумовний перехід) чи return (вихід з поточної функції).

Оператор `break` перериває виконання оператора, в якому він розміщений, а керування передається на наступний оператор:

```
 int k = 1;
for (int i = 0; i < m - 1; i++)
{
    for (int j = i + 1; j < m; j++)
        if (i == m - j) break;
        if (j == m) k++;
}
```

Всередині вкладених операторів `do-while, for, while` чи `switch` оператор `break` завершує лише той оператор, якому він належить, тому break неможна використовувати для виходу з декількох вкладених циклів. Навіть подвійне

послідовне використання двох операторів `break` не забезпечить вихід із вкладених циклів.

```
for (i = 0; i < 100; i++)
    for (j = 0; j < 100; j++)
     {
    ...
        if (j – i) < 0) { break; break; }
    ...
     }
```

Після виконання умови (j-i)<0 відбудеться вихід лише з внутрішнього циклу по змінній j, а виконання зовнішнього циклу по змінній і продовжиться, незважаючи на те, що оператор `break` записано двічі.

Оператор `break` є оператором переходу і оператори записані після нього виконуватись не будуть, якщо тільки їм не буде передано керування за допомогою інших операторів переходів.

Для *переходу до наступної ітерації циклу* призначений оператор continue, який переходить до наступної ітерації відповідного циклу і використовується лише всередині операторів циклів `for, while, do-while.`

```
for (int i = 3; i < 7; i++)
    if (i == 5) continue; // Пропустити елемент, рівний 5
    else
        cout << i; // Результат: 3  4  6  7
```

Крім того, передавати керування за межі вкладеної структури, можна використовуючи оператори `return` та `goto`.