Лекція 13

Циклічні процеси

Цикли - це фундаментальний і невід'ємний елемент будь-якої програми. Вони вкрай важливі з кількох причин:

1. Автоматизація повторюваних завдань: Багато завдань в програмуванні потребують повторюваного виконання. Наприклад, обробка всіх елементів масиву, робота з файлами або обчислення значень у циклі. Без циклів важко було б ефективно обробляти такі ситуації.
2. Робота з масивами та колекціями: Цикли надають можливість легко переглядати, змінювати або аналізувати кожен елемент в масиві або колекції. Це дозволяє ефективно працювати з великими обсягами даних.
3. Управління потоком програми: Цикли дозволяють вам контролювати, яка частина програми виконується скільки разів. Це важливо для створення умов та обробки даних у великих проектах.
4. Мінімізація дублювання коду: Завдяки циклам можна писати менше коду, оскільки один і той же блок коду може використовуватися для обробки багатьох даних чи ситуацій.
5. Можливість створення багаторазових та модульних рішень: Цикли, в поєднанні з функціями та об'єктами, дозволяють створювати загальноприйняті та повторно використовувані шаблони.
6. Ефективне використання ресурсів: Важливо використовувати цикли ефективно, щоб не перевантажувати систему зайвими операціями.

В загальному, цикли важливі, оскільки дозволяють програмі взаємодіяти з даними та виконувати завдання в автоматичному режимі. Вони є невід'ємною частиною будь-якої програми, незалежно від її складності.

В С ++ визначені три різних циклу:

- цикл з передумовою:

while (вираз-умова) тіло_циклу

- цикл з післяумовою:

do тіло_циклу
while (вираз-умова);

- цикл з параметром:

for (ініціалізація_циклу; вираз-умова; вираз_корекції) тіло_ циклу.

Тіло циклу – це окремий оператор, який завжди завершується крапкою з комою, або складовою оператор, або блок. Тільки опис або визначення не може бути тілом циклу. Оператори циклу задають багаторазове виконання операторів тіла циклу.

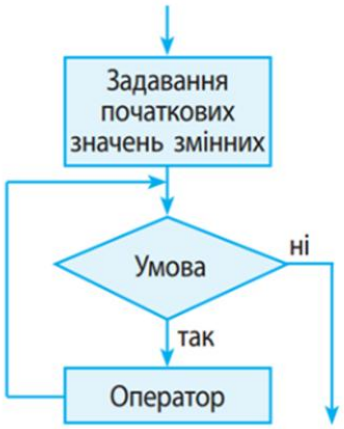
Вираз-умова – у всіх операторах скалярний вираз (найчастіше логічний або арифметичний) визначає умову продовження повторення обробки, якщо вона э істиною (відмінна від нуля). припинення виконання циклу відбувається у випадках: не істиного (нульового) значення виразу-умови; виконання в тілі циклу оператора передачі управління за межі тіла циклу.

Розглянемо більш докладно різні типи циклів.

Цикл while

Слово **while** перекладається як «поки». Тобто, поки істина якась умова, повторювати цикл.

Схематично цей цикл можна зобразити так, як показано на малюнку:



Як бачимо – це цикл з передумовою. Спочатку перевіряємо істинність деякої умови, а потім, якщо вона істина, виконуємо оператори (один або кілька), що становлять тіло циклу.

Такий цикл не виконується жодного разу, якщо умова спочатку помилкова.

Умова – це будь-який вираз, що має результат логічного або арифметичного типу, тобто умова для операторів циклу записується так само, як для оператора розгалуження **if**.

Формально оператор **while** можна записати так:

```
while (умова) // заголовок
{
  оператори тіла циклу
}
```

Тіло циклу практично завжди необхідно оформляти як блок, так як оператор **while** рідко містить в своєму тілі тільки один оператор.

Алгоритм роботи оператора простий: обчислюється значення умови в заголовку оператора. Якщо воно істинне, то виконуються оператори тіла циклу, а потім управління знову передається на заголовок. Якщо умова помилкова, то оператор закінчує роботу. Після цього буде виконуватися оператор, наступний за оператором циклу.

Приклад. Знайти суму квадратів перших n натуральних чисел.

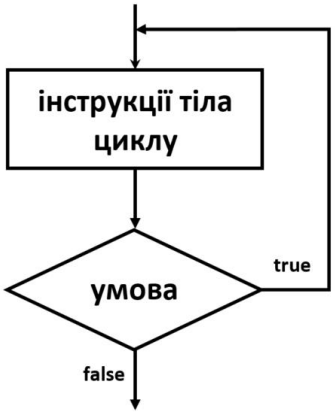
Можливий текст програми:

```
#include <iostream>
using namespace std;
int main() {
  int i, n, s;
  cout << "n=";
  cin >> n;
  s = 0; //очищення пам’яті
  i = 1;
```

```
while(i <= n)
{
  s += i * i;
  i++;
}
cout << "s=" << s << endl;
return 0; }
```

Цикл do

Цикл **do** – це цикл с післяумовою. Його схема виглядає так:



Формальний запис:

do{
  Операторы тела цикла
}**while**(*Условие*);

Цикл **do** завжди виконується хоча б один раз. І далеко не у всіх завданнях це добре. Для нашої щойно розглянутої задачі цей цикл може дати побічний ефект: якщо *n* помилково задати негативним або рівним нулю, то *s* дорівнюватиме 1, що явно не так. А ось використовувати цикл **do** для перевірки вхідних даних дуже зручно! Що ми і зробимо, тим самим позбудемося побічного ефекту.

Отже, те ж завдання: знайти суму квадратів перших n натуральних чисел.

Можливий текст програми з використанням циклу **do**:

```
#include <iostream>
using namespace std;
int main(){
  int i, n, s;
  do{
    cout << "n=";
    cin >> n;
  }while(n <= 0);
  s = 0;
  i = 1;
  do{
    s += i * i;
    i++;
```

```
  }while(i <= n);
  cout << "s=" << s << endl;
  return 0;
}
```

Цикл for

Це найпопулярніший оператор циклу для всіх С-подібних мов. Популярність пояснюється надзвичайною гнучкістю і міццю цього оператора. Його ще називають циклом з лічильником.

Формальний запис оператора **for** наступний:

for(початкова_дія; *умова додаткова_дія*)
{
Оператори тіла циклу
}

Початкові дії – це будь-які оператори присвоювання, записані через кому. Тут, як правило, робиться ініціалізація різних змінних, необхідних для роботи в циклі **for**, наприклад, встановлення початкового значення лічильника.

Умова – логічний або арифметичний вираз, записується так само, як і для інших операторів циклу і оператора if. Найчастіше це умова завершення циклу.

Додаткові дії – будь-які записані через кому оператори присвоювання і (або) оператори-вирази (збільшення або зменшення). Найчастіше це зміна значення лічильника.

Алгоритм роботи оператора **for**:

Обчислюються оператори, що складають початкові дії.

Обчислюється умова. Якщо вона помилкова, то оператор закінчує свою роботу. Якщо умова істинна, то виконуються оператори тіла циклу, потім – оператори, що складають групу додаткових дій, і потім знову робиться перевірка істинності умови. І так до тих пір, поки вона залишається істиною.

Кількість операторів в початкових і в додаткових діях може бути будь-якою. Будь-яка частина в заголовку (або навіть всі частини) можуть бути відсутні:

for(; ;) { *оператори тіла циклу* **}**

Але крапки з комою обов'язково повинні бути!

Знову повернемося до нашого прикладу, але реалізуємо його за допомогою оператора **for**: знайти суму квадратів перших *n* натуральних чисел.

Можливий текст програми:

```
#include <iostream>
using namespace std;
int main() {
  int i, n, s;
  do{
    cout << "n=";
    cin >> n;
```

```
  }while(n <= 0);

  for(s = 0, i = 1; i <= n; i++)
    s += i * i;
  cout << "s=" << s << endl;
  return 0;
}
```

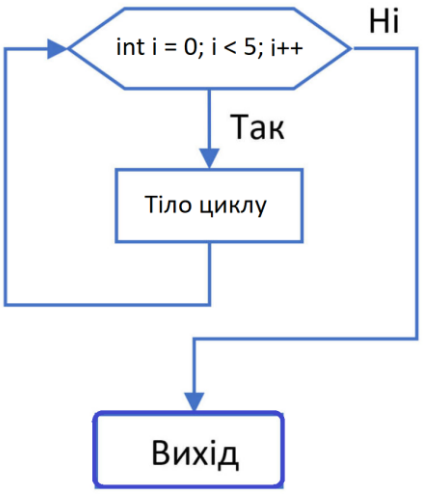


Рис. Приклад блок-схеми роботи програми

Часто цикли використовують для знаходження суми ряду. Для обчислення деякої функції використовується її розкладання в нескінченний ряд. Наприклад, функцію e^x можна представити у вигляді ряду Маклорена:

$$e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots + \frac{x^n}{n!} + \ldots$$

Ряди можна підсумувати, враховуючи кількість доданків (наприклад, знайти суму перших n доданків) або, якщо ряд збігається, то обчислювати суму із заданою точністю.

Правило. Кажуть, що сума ряду підрахована із заданою точністю ε, тобто виконується нерівність $|S_n - S_{n+1}| \leqslant \varepsilon$, якщо $|P_{n+1}| \leqslant \varepsilon$, де

S_n – сума перших n доданків,

S_{n+1} – сума перших n + 1 доданків,

P_n – n + 1 доданок.

Виходячи з цього правила, нескладно зрозуміти, що для обчислення суми ряду із заданою точністю необхідно продовжувати процес підсумовування до тих пір, поки чергові складові більше по модулю шуканої точності. Якщо ж черговий доданок менше (або дорівнює) шуканої точності, то процес підсумовування можна припинити.

Приклад 1. Перша група – ряди, в яких кожний доданок, крім, може бути, одного або декількох перших, повністю виражається через попередній доданок. Зазвичай це ряди, в яких для кожного доданка потрібно обчислювати факторіали і зводити якийсь аргумент в ступеня вищих порядків.

Знайти суму ряду із заданою точністю ε: $S=1-x+\frac{x^2}{2!}-\frac{x^3}{3!}+\frac{x^4}{4!}-\dots$

Введемо позначення: $P_0=1 \quad P_1=-x \quad P_2=\frac{x^2}{2!} \quad P_3=-\frac{x^3}{3!}$ … Тоді

$P_1=-P_0\cdot\frac{x}{1} \quad P_2=-P_1\cdot\frac{x}{2} \quad P_3=-P_2\cdot\frac{x}{3} \quad P_4=-P_3\cdot\frac{x}{4} \quad \dots \quad P_i=-P_{i-1}\cdot\frac{x}{i}$

```
#include <iostream>
using namespace std;
#include <cmath>
int main()
{ double x, p, s, i, eps = 1.0e-4;
  cout << "x=";
  cin >> x;
  for(i = 1, s = 1, p = 1; fabs(p) > eps; i++)
  { p = - p * x / i;
    s += p; }
  cout << "s=" << s << endl;
  return 0;}
```

Друга група – це ряди, в яких не можна повністю виразити чергове доданок через попереднє, але можна складові частини цього доданка висловити через відповідні частини попереднього доданка.

Знайти суму ряду із заданою точністю ε: $S=1-x+\frac{x^2}{2\cdot 2!}-\frac{x^3}{3\cdot 3!}+\frac{x^4}{4\cdot 4!}-\dots$

Для цього ряду $P_i=\frac{a_i}{i}$, де $a_i=-a_{i-1}\cdot\frac{x}{i}, \quad i=i+1.$

```
#include <iostream>
#include <cmath>
int main()
{ double a, x, p, s, i, eps = 1.0e-4;
  cout << "x=";
  cin >> x;
  for(i = 1, a = 1, s = 1, p = 1; fabs(p) > eps; i++)
  {  a = - a * x / i;
     p = a / i;
     s += p; }
  cout << "s=" << s << endl;
  return 0;}
```

Прклад 2. Третя група – це ряди, в яких кожний доданок залежить тільки від його номера.

Знайти суму перших *n* доданків: $S=\frac{3}{2^3}-\frac{8}{3^3}+\frac{15}{4^3}-\frac{24}{5^3}+\dots$ Для цього ряду:

$P_i=z\cdot\frac{i^2-1}{i^3}.$

```
#include <iostream>
using namespace std;
int main()
{ double i, p, s = 0, z = 1;
```

```
int n = 10;
for(i = 2; i <= n + 1; i++)
{  p = z * (i * i - 1) / (i * i * i);
   s += p;
   z = -z; }
cout << "s=" << s << endl;
return 0;}
```

Застосування циклічних процесів для різних задач.

1. Алгоритм Евкліда знаходження НСД (найбільшого спільного дільника)

Дано два цілих невід'ємних числа *a* і *b*. Потрібно знайти їх найбільший спільний дільник, тобто найбільше число, яке є дільником одночасно і *a*, і *b*. Англійською мовою "найбільший спільний дільник" пишеться "greatest common divisor", і поширеним його позначенням є {gcd}. Алгоритм полягає у наступному: більше число ділиться на менше і находиться залишок, далі менше ділиться на залишок від ділення і знову знаходиться залишок. Процедура продовжується поки залишок не буде рівний 0.

$$\gcd(a,b) = \begin{cases} a, & \text{if b=0} \\ \gcd(b, a \bmod b), & \text{otherwise} \end{cases}$$

```
int a, b;
if (b>a) swap (a, b); //нехай а більше число
        while (b) {
                a %= b;
                swap (a, b);
        }
        return a;
```

Функція **swap** міняє значення місцями.

2. Сума цифр цілого числа.

```
#include <iostream>
using namespace std;

int main()
{  int n;
   int sum=0;

   cout << "please, enter n = ";   cin >> n;

   while (n!=0)
   {  sum += n%10; // сума залишків відділення
      n /= 10;  }
   cout << "sum = " << sum << endl;
   return 0;}
```

3. Обчислення n чисел Фібоначчі

Числа Фібоначчі це числова послідовність де $P_1 = 1$, $P_2 = 2$, а кожне наступне число дорівнює сумі двох попередніх $P_i = P_{i-1} + P_{i-2}$

```
#include <iostream.h>
#include <conio.h>
int main () {
int a=1, b=1, c=1,n;
cout << "please, enter n = ";   cin >> n;
while (b<=n)
{ cout <<«c= » << c <<endl;
c=a+b;a=b;b=c;}
getch ();
return 0;}
```

4. Циклічне виконання програми

Якщо ми хочемо, щоб після виконання програма не завершувалась, а продовжувала свою роботу, наприклад виконувала обчислення для інших значень параметрів, то для обчислень можна організувати цикл з післяумовою, який завершується при натисканні, наприклад, клавіші Esc. Код клавіші Esc=27.

```
do{
тіло циклу
   cout << "Press Esc for Exit or any rey for continue "<< endl;
} while(getch()!=27);
```