Оператори циклу з параметрами після умови та передумови.

В мові С++ існують три оператори циклу **while**, **do**, **for**.

Цикл **for**

– інструкція **for** використовується для організації циклів з фіксованим, відомим під час розробки програми, числом повторень;

– кількість повторень циклу визначається початковим значенням змінної-лічильника і умовою завершення циклу;

– змінна-лічильник повинна бути цілого (int) типу і може бути оголошена безпосередньо в інструкції циклу.

Приклад. Підсумувати всі парні числа від парного числа a до парного числа b (a<b).

```
int sum=0, a, b ,i;
cout<< "Vvedi a, a=";
cin>>a;
cout<< "Vvedi b, b=";
cin>> b;
   for (i=a;i<b;i+=2)
          sum+=i;
   cout<< " Summa ravna sum" << sum;
cout<< "\n" ;
```

Оператор циклу з передумовою **while** виконується, якщо умова перевіряється до початку циклу, і має вигляд

while(вираз-умова**)** оператор**;** ,

де **оператор** – тіло циклу, що може бути представлено простим або складеним оператором, який взагалі не обчислюється, якщо **вираз-умова** має нульове значення (неправда), а керування передається наступному за циклом **while** оператору. Якщо **вираз-умова** відмінний від нуля (істинно), тоді обчислюється **оператор**, і керування передається до початку циклу. У результаті тіло циклу **while оператор** виконується доки вираз прийме значення нуль (неправда). Коли вираз прийме значення нуль, керування буде передане наступному за циклом оператору. Таким чином, число повторень **while** визначається ходом виконання програми.

Наприклад:

```
// Обчислення 10!
int i=1, p=1;
while(i<=10)
  {p*=i;
```

```
    i++;
  }
```

Для завершення циклу **while** в тілі циклу обов'язково повинні бути інструкції, виконання яких впливає, умова завершення циклу.

Оператор циклу з післяумовою **do while** застосовуються у випадках, коли тіло циклу виконується хоча б один раз і має таку форму запису:

do оператор
while(вираз-умова); .

У процесі виконання оператора **do while** спочатку здійснюється вхід до тіла циклу і виконується **оператор**, який може простим або складеним, далі перевіряється вираз і, якщо він правдивий (істинно) – цикл повторюється, а коли вираз помилковий (неправда) –здійснюється вихід з циклу.

Наприклад:

```
int n=1,p1,p2;
do {
     p1=1./n;
     n++;
     p2=p1+1./n;
  }
while(1./n<0.001);
```

Оператори циклу **do while**, як правило, використовується для організації наближених обчислень, в задачах пошуку і обробки даних, що вводяться з клавіатури або з файлу.

Іноді необхідно повторювати одну і ту ж дію кілька разів підряд. Для цього використовують цикли. У цьому уроці ми навчимося програмувати цикли на C++, після чого порахуємо суму усіх чисел від 1 до 1000.

Цикл for

Якщо ми знаємо точну кількість дій (ітерацій) циклу, то можемо використати цикл for. Синтаксис його виглядає приблизно так:

```
for (дія до початку циклу;
    умова продовження циклу;
    дії у кінці кожної ітерації циклу){
       інструкція циклу;
       інструкція циклу 2;
       інструкція циклу N;
}
```

Ітерацією циклу називається один прохід цього циклу

Існує окремий випадок цього запису, який ми сьогодні і розберемо :

```
for (лічильник = значення; лічильник < значення; крок циклу){
   тіло циклу;
}
```

Лічильник циклу — це змінна, в якій зберігається кількість проходів цього циклу.

Опис синтаксису

Спочатку привласнюється первинне значення лічильнику, після чого ставиться крапка з комою.

Потім задається кінцеве кінцеве значення лічильника циклу. Після того, як значення лічильника досягне вказаної межі, цикл завершиться. Знову ставимо крапку з комою.

Задаємо крок циклу.

Крок циклу — це значення, на яке збільшуватиметься або зменшуватиметься лічильник циклу при кожному проході.

Приклад коду

Напишемо програму, яка рахуватиме суму усіх чисел від 1 до 1000.

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

int main()
{
   int i; // лічильник циклу
   int sum = 0; // сума чисел від 1 до 1000.
   setlocale(0, "");
   for (i = 1; i <= 1000; i++) // задаємо початкове значення 1, кінцеве 1000 і задаємо крок циклу - 1.
   {
      sum = sum + i;
   }
   cout << "Сума чисел від 1 до 1000 = " << sum << endl;
   return 0;
}
```

Якщо ми скомпілюємо цей код і запустимо програму, то вона покаже нам відповідь: 500500. Це і є сума усіх цілих чисел від 1 до 1000. Якщо рахувати це вручну, знадобиться дуже багато часу і сил. Цикл виконав усю рутинну роботу за нас.

Зверніть увагу, що кінцеве значення лічильника я задав нестрогою нерівністю (<= — менше або рівно), оскільки, якби я поставив знак менше, то цикл провів би 999 ітерацій, тобто на одну менше, ніж вимагається. Це досить важливий момент, оскільки тут новачки часто припускаються помилки, особливо при роботі з масивами (про них буде розказано в наступному уроці). Значення кроку циклу я задав рівне одиниці. i++ — це теж саме, що i i = i + 1.

У тілі циклу, при кожному проході програма збільшує значення змінної sum на одиницю. Ще один дуже важливий момент — на початку програми я присвоїв змінній sum значення нуля. Якби я цього не зробив, програма вилетіла ви в дефолт. При оголошенні змінної без її ініціалізації що ця змінна зберігатиме «сміття».

Природно до сміття ми нічого додати не можемо. Деякі компілятори, такі як gcc, ініціалізував змінну нулем при її оголошенні.

Цикл while

Коли ми не знаємо, скільки ітерацій повинен провести цикл, нам знадобиться цикл while або do...while. Синтаксис циклу while в C++ виглядає таким чином.

```
while (Умова){
   Тіло циклу;
}
```

Цей цикл виконуватиметься, поки умова, вказана в круглих дужках є істиною. Вирішимо те ж завдання за допомогою циклу while. Хоча тут ми точно знаємо, скільки ітерацій повинен виконати цикл, дуже часто бувають ситуації, коли це значення невідоме.

Нижче приведений початковий код програми, що рахує суму усіх цілих чисел від 1 до 1000.

```
setlocale(0, "");
   int i = 0; // ініціалізували лічильник циклу.
   int sum = 0; // ініціалізували лічильник суми.
   while (i < 1000)
   {
      i++;
      sum += i;
   }
   cout << "Сума чисел від 1 до 1000 = " << sum << endl;
```

Після компіляції програма видасть результат, аналогічний результату роботи попередньої програми. Але пояснимо декілька важливих моментів. Я задав строгу нерівність в умові циклу і ініціалізував лічильник і нулем, оскільки в циклі while відбувається на одну ітерацію більше, тому він виконуватиметься, до тих пір, поки значення лічильника перестає задовольняти умові, але ця ітерація все одно виконається. Якби ми поставили нестрогу нерівність, то цикл би закінчився, коли змінна і дорівнювала б 1001 і виконалося б на одну ітерацію більше.

Тепер давайте розглянемо по порядку початковий код нашої програми. Спочатку ми ініціалі ували лічильник циклу і змінну, що зберігає суму чисел.

В даному випадку ми обов'язково повинні присвоїти лічильнику циклу яке-небудь значення, оскільки в попередній програмі ми це значення привласнювали усередині циклу for, тут же, якщо ми не ініціалізували лічильник циклу, то в нього потрапить «сміття» і компілятор у кращому разі видасть нам помилку, а в гіршому, якщо програма збереться — сегфолт практично неминучий.

Потім ми описуємо умову циклу — «доки змінна i менше 1000 — виконуй цикл». При кожній ітерації циклу значення змінної-лічильника i збільшується на одиницю усередині циклу.

Коли виконається 1000 ітерацій циклу, лічильник стане рівним 999 і наступна ітерація вже не виконається, оскільки 1000 не менше 1000. Вираження sum += i є укороченим записом sum = sum + i.

Після закінчення виконання циклу, виводимо сполучення з відповіддю.
Цикл do while

Цикл do while дуже схожий на цикл while. Єдина їх відмінність в тому, що при виконанні циклу do while один прохід циклу буде виконаний незалежно від умови. Рішення задачі на пошук суми чисел від 1 до 1000, із застосуванням циклу do while.

```
setlocale(0, "");
   int i = 0; // ініціалізували лічильник циклу.
   int sum = 0; // ініціалізували лічильник суми.
   do {// виконуємо цикл.
      i++;
      sum += i;
   } while (i < 1000); // поки виконується умова.
   cout << "Сума чисел від 1 до 1000 = " << sum << endl;
```

Принципової відмінності немає, але якщо присвоїти змінній i значення, більше, ніж 1000, то цикл все одно виконає хоч би один прохід.
Попрактикуйте, поекспериментуйте над власними прикладами завдань. Цикли — дуже важлива річ, тому їм варто приділити більше уваги. Коли зрозумієте, як працюють цикли — можете сміливо переходити до вивчення наступного уроку.