Операторы цикла. Цикл с параметром. Операторы цикла итерационного типа.

- ✔ Операторы цикла.
- ✔ Цикл с параметром.
- ✔ Вложенные циклы.
- ✔ Операторы цикла итерационного типа: с предусловием, с постусловием.

С другой стороны, мы не можем игнорировать эффективность

Циклы

Существует два типа циклов: типа «пока» и типа «n-раз».

Первый тип «пока» предназначен для повторения некоторых действий до тех пор, пока выполняется некоторое условие.

Второй тип «n-раз» предназначен для повторения некоторых действий заранее известное количество раз.

Цикл типа «n-раз» (оператор for)

Оператор for содержит три параметра.

- Первый называется инициализацией,
- Второй — условием повторения,
- Третий — итерацией.

```
for (инициализация; условие; итерация)
    {
        //тело цикла, т. е. действия
повторяемые циклично
    }
```

Цикл типа «n-раз» (оператор for)

В первом параметре обычно выбирают какую-то переменную, с помощью которой будет подсчитываться количество повторений цикла. Её называют счетчиком. Счётчику задают некоторое начальное значение (указывают, начиная с какого значения он будет изменяться).

Во втором параметре указывают некоторое ограничение на счётчик (указывают, до какого значения он будет изменяться).

Цикл типа «n-раз» (оператор for)

В третьем параметре указывают выражение, изменяющее счётчик после каждого шага цикла. Обычно это инкремент или декремент, но можно использовать любое выражение, где счётчику будет присваиваться некоторое новое значение.

Цикл типа «n-раз» (оператор for)

Перед первым шагом цикла счётчику присваивается начальное значение (выполняется инициализация). Это происходит лишь однажды.

Представленная программа выводит на экран числа от 1 до 100:

```
for (int i = 1; i <= 100; i++)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

Перед каждым шагом цикла (но после инициализации) проверяется условие повторения, если оно истинно, то в очередной раз выполняется тело цикла. При этом, тело цикла может не выполниться ни разу, если условие будет ложным в момент первой же проверки.

```
for (int i = 1; i <= 100; i++)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

После завершения каждого шага цикла и перед началом следующего (и, значит, перед проверкой условия повторения) выполняется итерация.

```
for (int i = 1; i <= 100; i++)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

Представленная программа выводит на экран числа от 10 до −10:

```
for (int s = 10; s > -11; s--)
   {
      cout << s << " ";
   }
```

Представленная программа выводит на экран нечётные числа от 1 до 33:

```
for (int i = 1; i <= 33; i = i + 2)
   {
      cout << i << " ";
   }
```

Цикл типа «n-раз» (оператор for)

Представленная программа вычислит сумму элементов фрагмента последовательности 2, 4, 6, 8,… 98, 100.

Итак:

```
int sum = 0; // Сюда будем накапливать результат
for (int j = 2; j <= 100; j=j+2) {
  sum = sum + j;
}
cout << sum;
```

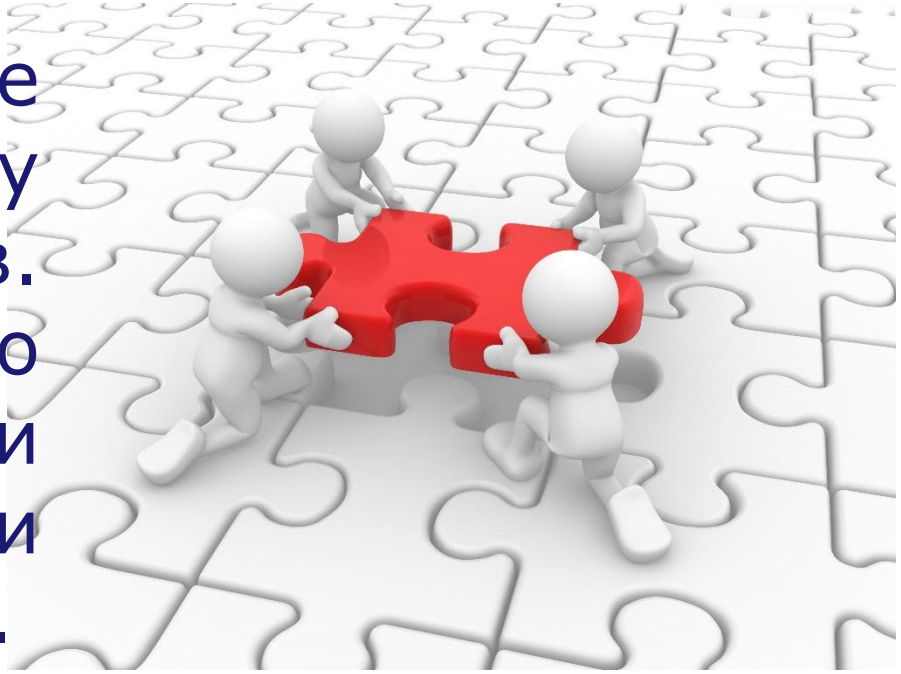
Цикл типа «n-раз» (оператор for)

Представленная программа будет возводить число из переменной a в натуральную степень из переменной n:

```
double a = 2;
int n = 10;
double res = 1; // Сюда будем накапливать результат
for (int i = 1; i <= n; i++) {
  res = res * a;
}
cout << res;
```

Цикл типа «n-раз» (оператор for)

В одном цикле можно задавать сразу несколько счётчиков. При этом несколько выражений в итерации и в инициализации разделяются запятыми.



Условие повторения можно задавать только одно, но оно может быть выражением, содержащим сразу несколько счётчиков.

Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран 10 первых элементов последовательности $2a_{n-1}-2$, где $a_1=3$:

```
for (int a=3, i=1; i<=10; a=2*a-2, i++)
    {
       cout << a << " ";
    }
```

Цикл типа «n-раз» (оператор for)

Представленная программа выведет на экран такую последовательность «0 -1 -4 -9 -16 -25»:

```
for (int a=0, b=0; a-b<=10; a++, b--)
    {
       cout << a*b << " ";
    }
```

Досрочное завершение цикла (оператор break)

Как цикл типа «пока» так и цикл типа «n-раз» можно завершить досрочно, если внутри тела цикла вызвать оператор **break**.

При этом произойдёт моментальный выход из цикла, не будет закончен даже текущий шаг (т. е. если после break присутствовали какие-то ещё операторы, то они не выполнятся).

Досрочное завершение цикла (оператор break)

В результате работы следующего примера на экран будут выведены только числа «1 2 3 4 Конец»:

```
for (int a=1; a<=10; a++) {
  if(a == 5) {
    break;
  }
  cout << a << " ";
}
cout << "Конец";
```

Досрочное завершение цикла (оператор break)

Когда программа будет выполнять цикл в пятый раз (войдёт в цикл с счётчиком равным 5), сразу же будет проверено и окажется истинным условие при котором выполнится оператор break.

Оставшаяся часть тела цикла (вывод на экран) уже производится не будет: программа сразу перейдёт к выполнению операций указанных после цикла и далее.

Досрочное завершение цикла (оператор break)

С помощью оператор break можно прервать заведомо бесконечный цикл. Пример (на экран выведется «100 50 25 12 6 3 1 0 » и после этого цикл остановится):

```
int s = 100;
while (true) {
  cout << s << " ";
  s = s / 2;
  if(s == 0) {
    break;
  }
}
```

Досрочное завершение цикла (оператор break)

Оператор break имеет смысл вызывать только при наступлении какого-то условия, иначе цикл будет завершен досрочно на первом же своём шаге.

```
int a;
for (a=25; a>0; a--)
 {
   break;
   cout << a << " ";
 }
cout << "a=" << a;
```

Досрочное завершение цикла (оператор break)

В представленном выше примере вывода в цикле на экран не произойдёт ни разу, а когда переменная a выведется на экран после цикла, то окажется, что её значение ни разу не менялось, т. е. выведено будет «a=25» (и ничего больше).

Обратите внимание также на то, что переменная была объявлена до начала цикла. Когда переменная объявляется в параметрах цикла, то она оказывается недоступной за его пределами, а в данном случае требовалось иное — узнать какое значение будет у счётчика после завершения цикла.

Цикл типа «пока» (оператор while)

Оператор **while** повторяет указанные действия до тех пор, пока его параметр имеет истинное значение.

Например, такой цикл выполнится 4 раза, а на экран будет выведено «1 2 3 4 »:

```
int i = 1;
while (i < 5)
  {
    i++;
    cout << i << " ";
  }
```

Цикл типа «пока» (оператор while)

Такой цикл не выполнится ни разу и на экран ничего не выведется:

```
int i = 1;
  while (i < 0)
    {
        i++;
        cout << i << " ";
    }
```

Цикл типа «пока» (оператор while)

Такой цикл будет выполняться бесконечно, а на экран выведется «1 2 3 4 5 6 7 …»:

```
int i = 1;
while (true)
  {
    i++;
    cout << i << " ";
  }
```

Цикл типа «пока» (оператор while)

Условие, определяющее будет ли цикл повторятся снова, проверяется перед каждым шагом цикла, в том числе перед самым первым.

Таким образом происходит *предпроверка* условия.

Цикл типа «пока» (оператор do…while)

Бывает цикл типа «пока» с постпроверкой условия. Для его записи используется конструкция из операторов do…while.

Такой цикл выполнится 4 раза, а на экран будет выведено «2 3 4 5 »:

```
int i = 1;
do {
    i++;
    cout << i << " ";
} while (i < 5);
```

Цикл типа «пока» (оператор do…while)

Такой цикл выполнится 1 раз, а на экран будет выведено «2 »:

```
int i = 1;
do {
    i++;
    cout << i << " ";
} while (i < 0);
```

Цикл типа «пока» (оператор do…while)

Тело цикла **do…while** выполняется по крайней мере один раз. Этот оператор удобно использовать, когда некоторое действие в программе нужно выполнить по крайней мере единожды, но при некоторых условиях придётся повторять его многократно.

Задачи

- Создайте программу, выводящую на экран первые 55 элементов последовательности 1 3 5 7 9 11 13 15 17 ….
- Создайте программу, выводящую на экран все неотрицательные элементы последовательности 90 85 80 75 70 65 60 ….
- Создайте программу, выводящую на экран первые 20 элементов последовательности 2 4 8 16 32 64 128 ….