# Ветвления и циклы

1) Теория
2) Задачи для самостоятельного решения.
   Для решения задач необходимо зарегистрироваться на сайте https://codeforces.com/

## Логический тип данных и логические операции

Переменная логического типа может принимать только два значения: *true* (истина) или *false* (ложь). Логический тип данных в языке C++ обозначается словом *bool*.

Например, сохраним в переменную ***b*** результат вычисления выражения ***x*** < 3:

```
bool b = x < 3;
```

Переменная ***b*** примет значение *true* (истина), если ***x*** **<** **3**, и *false* (ложь) — в противном случае (если ***x*** **⩾ 3**).

Есть шесть операций сравнения:

- < меньше
- > больше
- <= меньше или равно
- >= больше или равно
- == равно
- != не равно

Приоритет операций сравнения меньше, чем у арифметических операций. Например, если написать ***x*** **+ 5 <** ***x*** **∗ 2**, то сначала будет посчитан результат выражений ***x*** **+ 5** и ***x*** **∗ 2** и только затем произойдёт сравнение.

Операции <, >, <=, >= имеют одинаковый приоритет. Операции == и != тоже имеют одинаковый приоритет, но более низкий, чем <, >, <= и >=.

Существуют ещё две логические операции:

- **&&** «логическое И» используется в ситуации, когда должны быть верны оба выражения
- || «логическое ИЛИ» используется в ситуации, когда должно быть верно одно из выражений

Приоритет этих логических операций самый низкий, они выполняются после арифметических операций и операций сравнения. При этом сначала выполняются операции отрицания, потом операции «логического И», и только затем операции «логического ИЛИ». Изменить приоритет выполнения операций можно с помощью круглых скобок.

Проверим, является ли число ***x*** номером месяца:

```
bool c = (x >= 1 && x <= 12);
```

Двойные неравенства в C++ в подобных случаях использовать **нельзя!** Если записать выражение **1 <=** ***x*** **<= 12**, то сначала выполнится первое сравнение, а затем его результат *true* или *false*, приведённый к числовому значению 1 или 0 соответственно, будет сравниваться с числом 12, в результате всегда получится значение *true*.

**Пример 1.**

Пусть на дереве было ***n*** белочек и ***x*** орешков, которые они делят между собой поровну (остаток скидывается с дерева). Необходимо проверить, верно ли, что каждой белочке достанется ровно ***y*** орешков.

Проверим также, что задача корректная, а именно количество белок — положительное число.

```
bool ok = n > 0 && x / n == y;
```

Важно то, что если $n = 0$, то первая часть выражения вернёт ложь, поэтому вторая часть выражения вычисляться не будет, а значит не будет деления на 0. Если поменять части логического выражения местами, то программа выдаст ошибку при $n = 0$.

Логический тип данных можно преобразовать в числовой. В этом случае *true* будет преобразован в 1, а *false* в 0.

**Пример 2.**

Пусть космонавтом может быть человек, который удовлетворяет двум характеристикам из трёх:

- рост ***h*** **<= 170** см;
- вес ***w*** **<= 65** кг;
- ***IQ*** **>= 150**.

Проверим, можем ли мы взять человека с данным ростом, весом и IQ в космонавты:

```
int n = (h <= 170) + (w <= 65) + (IQ >= 150);
bool goToCosmos = n >= 2;
```

## Условный оператор if

**Пример 3.**

По данному числу ***x*** определить его абсолютную величину (модуль). Программа должна напечатать значение переменной ***x***, если ***x*** **> 0**, или же величину **−*x*** в противном случае.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int x;
   cin >> x;
   if (x < 0){
      x = -x;
   }
   cout << x;
   return 0;
}
```

В этой программе используется условный оператор *if* (если).

После слова *if* указывается проверяемое логическое выражение ***x*** **> 0** в круглых скобках. После этого идёт блок (последовательность) инструкций, который будет выполнен, если значение логического выражения истинно. Блок выделяется фигурными скобками.

Данную задачу можно решить другим способом с помощью инструкции *else* (иначе):

```
#include <iostream>
using namespace std;
int main()
{
   int x;
   cin >> x;
```

```
    if (x >= 0){
        cout << x;
    } else{
        cout << -x;
    }
    return 0;
}
```

Блок внутри инструкции *else* будет выполнен только в том случае, если логическое выражение внутри *if* вернуло *false* (ложь). Инструкция *else* всегда относится к какой-либо инструкции *if*.

**Пример 4. Задача про високосный год**

Дано натуральное число. Требуется определить, является ли год с данным номером високосным. Если год является високосным, то выведите YES, иначе выведите NO. Напомним, что в соответствии с григорианским календарём, год является високосным, если его номер делится на 4, но при этом не делится на 100, или если он кратен 400.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
    int year;
    cin >> year;
    if (year % 4 == 0){
        if (year % 100 == 0){
            if (year % 400 == 0){
                cout << "YES" << endl;
            }
            else {
                cout << "NO" << endl;
            }
        } else {
            cout << "YES" << endl;
        }
    } else {
        cout << "NO" << endl;
    }
    return 0;
}
```

Программа получилась довольно громоздкой. Можно написать решение короче:

```
#include <iostream>
using namespace std;
int main()
{
    int year;
    cin >> year;
    if ((year % 4 == 0 && year % 100 != 0) || year % 400 == 0 ){
        cout << "YES" << endl;
    } else {
        cout << "NO" << endl;
    }
    return 0;
```

```
}
```

**Пример 5.**

Даны два числа, выведите минимальное из них.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int a, b;
   cin >> a >> b;
   if (b < a){
      a = b;
   }
   cout << a;
   return 0;
}
```

**Пример 6.**

Даны три числа, выведите минимальное из них.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int a, b, c;
   cin >> a >> b >> c;
   if (b < a){
      a = b;
   }
   if (c < a){
      a = c;
   }
   cout << a;
   return 0;
}
```

**Пример 7.**

Пусть дано число ***x***. Необходимо вывести на экран:

- "One", если ***x*** = 1;
- "Two", если ***x*** = 2;
- "Three", если ***x*** = 3;
- "Other" в любом другом случае.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int x;
```

```
    cin >> x;
    if (x == 1){
        cout << "One" << endl;
    }
    if (x == 2){
        cout << "Two" << endl;
    }
    if (x == 3){
        cout << "Three" << endl;
    }
    if (x < 1 || x > 3){
        cout << "Other" << endl;
    }
    return 0;
}
```

Если понадобится добавить отдельный вывод "Four" для x = 4, то в программу придётся добавить ещё одно условие, а также исправить условие для вывода "Other". Такой подход может привести к ошибкам. Поэтому лучше запись программу иначе:

```
#include <iostream>
using namespace std;
int main()
{
    int x;
    cin >> x;
    if (x == 1){
        cout << "One" << endl;
    } else if (x == 2){
        cout << "Two" << endl;
    } else if (x == 3){
        cout << "Three" << endl;
    } else {
        cout << "Other" << endl;
    }
    return 0;
}
```

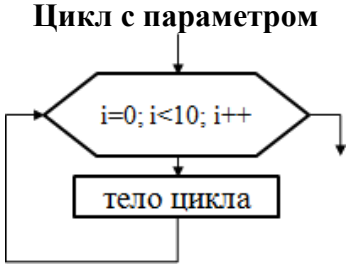
При таком способе можно легко модифицировать программу при добавлении других чисел.

## Циклы

В программировании часто возникает задача повторения одного и того же действия несколько раз. Для этого обычно используются циклы.

### Типы циклов

**Цикл с параметром**

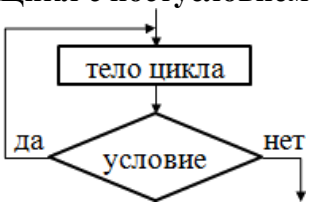



```
for (счетчик = значение; счетчик < значение; шаг цикла)
{
    тело цикла;
}
```

**Цикл с предусловием**


```
while (условие)
{
   тело цикла;
}
```

**Цикл с постусловием**




```
do{
   тело цикла;
 }while (условие)
```

## Цикл while (пока)

**Пример 8.**

Вывести на экран все числа от 1 до 100 и их сумму.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int i, s;
   i = 1;
   s = 0;
   while (i <= 100){
      cout << i << " ";
      s = s + i;
      i = i + 1;
   }
   cout << s << endl;
   return 0;
}
```

В данном примере действия внутри цикла *while* выполняются до тех пор, пока логическое выражение ***i*** <= 100 истинно.

**Важно** при написании цикла всегда изменять переменную, которая используется в условии продолжения цикла, так как в противном случае цикл будет выполняться бесконечно долго.

**Пример 9.**

Найти наибольшую степень двойки, которая не превосходит 1000.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
   int ans = 1;
   while (ans * 2 < 1000){
      ans = ans * 2;
   }
   cout << ans;
```

```
    return 0;
}
```

Если написать в условии выхода из цикла ***ans***<1000, то получим первую степень двойки, большую или равную 1000.

**Пример 10.**

Дана последовательность чисел, необходимо найти самое большое число в последовательности. Признаком завершения последовательности является число 0.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
    int x, ma = 0;
    cin >> x;
    while (x != 0){
        if (x > ma){
            ma = x;
        }
        cin >> x;
    }
    cout << ma << endl;
    return 0;
}
```

**Пример 11.**

Дано число x. Необходимо посчитать количество и сумму его цифр.

**Решение**

```
#include <iostream>
using namespace std;
int main()
{
    int x, k = 0, s = 0;
    cin >> x;
    while (x > 0){
        k = k + 1;
        s = s + x % 10;
        x = x / 10;
    }
    cout << k << " " << s << endl;
    return 0;
}
```

Программа даст неправильный ответ для числа 0 (в его записи присутствует одна цифра). Этот случай нужно рассмотреть отдельно.

**Пример 12.**

Дана последовательность чисел. Необходимо найти самое большое число в последовательности. Признаком завершения последовательности является число 0.

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

```
int main()
{
   int mx, n;
   cin >> n;
   mx = n;
   while (n > 0){
      if (n > mx){
         mx = n;
      }
      cin >> n;
   }
   cout << mx << endl;
   return 0;
}
```

**Пример 13.**

Дана последовательность чисел. Необходимо найти максимум в последовательности и количество элементов, значение которых равно максимуму. Признаком завершения последовательности является число 0.

```
#include <iostream>
using namespace std;
int main()
{
   int mx, n, k = 0;
   cin >> n;
   mx = n;
   while (n > 0){
      if (n > mx){
         mx = n;
         k = 1;
      } else if (n == mx){
         k = k + 1;
      }
      cin >> n;
   }
   cout << mx << " " << k << endl;
   return 0;
}
```

**Бесконечный цикл**

Если в реализации программы допустить ошибку, то цикл может работать бесконечно долго.

**Пример 14.** Требуется вывести все числа от 1 до n.

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

```
    return 0;
}
```

Данный код будет бесконечно долго выводить единицы, т.к. в программе не изменяется значение счётчика ***i*** внутри цикла.

Верное решение:

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

## Цикл for

Некоторые операции в языке C++ имеют сокращённую запись. Например, увеличить ***x*** на единицу можно несколькими способами:

```
x = x + 1;
x += 1;
++x;
x++;
```

**Пример 15. Задача про числа от** 1 **до** 100

Выведите на экран все числа от 1 до 100.

| **Решение с циклом while** | **Решение с помощью цикла for** |
|---|---|
| `while (i <= 100){`<br>`  cout << i << " ";`<br>`  i++;`<br>`}` | `for (i = 1; i <= 100; ++i){`<br>`  cout << i << " ";`<br>`}` |

Цикл *for* содержит три инструкции управления, разделённые точкой с запятой:

1. Инициализация переменной.
2. Условие продолжения цикла.
3. Инструкция, выполняемая после завершения всех операций внутри цикла (обычно изменение счётчика).

**Пример 16.**

Выведите на экран таблицу умножения для чисел от 1 до 10.

**Решение**

Решим данную задачу с помощью вложенного цикла *for*.

```
for (i = 1; i <= 10; ++i){
    for (int j = 1; j <= 10; ++j){
```

```
        cout << i * j << " ";
    }
    cout << endl;
}
```

Иногда может быть важно не выводить пробел после последнего числа в строке. Изменим код, чтобы учесть это требование:

```
for (i = 1; i <= 10; ++i){
    for (int j = 1; j <= 9; ++j){
        cout << i * j << " ";
    }
    cout << i * 10 << endl;
}
```

Изменим программу так, чтобы не выводить перевод строки после вывода таблицы:

```
for (i = 1; i <= 10; ++i){
    for (int j = 1; j <= 10; ++j){
        cout << i * j << " ";
    }
    if (i != 10){
        cout << endl;
    }
}
```

**Пример 17.**

Дана последовательность из ***n*** натуральных чисел. Найдите сумму чётных чисел в последовательности. Значения всех чисел не превышают $2^{31-1}$.

**Решение**

Для хранения элементов последовательности хватит типа *int*, для хранения ответа необходимо использовать больший тип данных — *long long*.

```
#include <iostream>
using namespace std;
int main() {
    long long sum = 0;
    int now, n;
    cin >> n;
    for (int i = 0; i < n; ++i){
        cin >> now;
        if  (now % 2 == 0){
            sum += now;
        }
    }
    cout << sum << endl;
    return 0;
}
```

**Пример 18.**

Даны два натуральных числа — A и B. Необходимо вывести все нечётные числа на отрезке [A;B]. Гарантируется, что A — нечётное и A<B.

**Решение**

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

```
int main() {
    int a, b;
    cin >> a >> b;
    for (int i = a; i <= b; i += 2){
        cout << i << " ";
    }
    return 0;
}
```

# Задачи для самостоятельного решения

## https://codeforces.com/gym/471007

Для решения задач необходимо выпонить регистрацию на сайте.

В процессе решения можно задавать на сайте вопросы.

*Постарайтесь решить как можно больше задач.*