ШАГ
компьютерная
АКАДЕМИЯ

ОСНОВЫ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ C++

Урок № 3

Циклы

Содержание

Материалы урока прикреплены к данному PDF-файлу. Для доступа к материалам, урок необходимо открыть в программе Adobe Acrobat Reader.

1. Понятие цикла

Очень часто, и в жизни и при написании программы, существует необходимость повторения какого-либо действия несколько раз. Например, представим алгоритм, реализующий мытьё тарелок.

```
0. Взять тарелку из раковины.
1. Намылить тарелку средством для мытья посуды.
2. Потереть тарелку мочалкой.
3. Смыть мыльную пену с тарелки.
4. Вытереть тарелку.
5. Поставить тарелку на полку.
6. Конец программы.
```

В данном, на первый взгляд толковом, алгоритме есть одна маленькая неувязочка — если тарелок будет больше одной, то вымытой все равно окажется только одна. Это связано с тем, что программа выполняет все действия линейным образом — сверху вниз по порядку. Следовательно, нам необходимо придумать каким способом заставить программу вернутся до нужного момента и повторить набор конкретных действий, и при этом определить нужное количество повторов. Правильный алгоритм будет выглядеть так.

```
0. Взять тарелку из раковины.
1. Намылить тарелку средством для мытья посуды.
2. Потереть тарелку мочалкой.
3. Смыть мыльную пену с тарелки.
4. Вытереть тарелку.
```

```
5. Поставить тарелку на полку.
6. Если есть еще грязные тарелки вернуться к пункту 0.
7. Конец программы.
```

Обратим внимание на то, что для того, что бы определить, повторять ли действия сначала используется условие «Если есть еще грязные тарелки». Если это условие истинно — действия повторяются, если ложно, выполняется следующий, 7-ой пункт алгоритма.

А вот представим себе теперь другую ситуацию. Мы находимся в другом городе и едем на общественном транспорте. Наша остановка будет только шестой по счету. Каковы же наши действия?

1. Войти в автобус и оплатить за проезд
2. Сесть на свободное место
3. Автобус продолжил свое движение
4. Через некоторое время он делает заданную остановку по его маршруту.
5. Мы же условно прибавляем себе единицу в уме
6. Проверяем — Это шестая остановка? Если «да» — выходим, а если «нет» — переходим к третьему пункту нашего алгоритма.

Как видим, многие действия из нашей жизни цикличны: смена времени года, ход часов, распорядок нашего рабочего дня, расписание поездов, игры. А ведь это ведь еще не весь список!

Итак, мы пришли к тому, что нам необходима некая конструкция, которая заключает в себе набор действий

для повторения. При этом количество повторений должно зависеть от какого-то условия, содержащегося в этой же конструкции.

Невольно, мы только что дали определение так называемого ЦИКЛА. Повторим еще раз!!!

Цикл — *специальный оператор языка программирования, с помощью которого то или иное действие можно выполнить нужное количество раз, в зависимости от некоего условия.*

Примечание: *Кстати — другое название цикла — конструкция повторения. А, каждое повторение действия — ШАГ ЦИКЛА или ИТЕРАЦИЯ.*

В языке C++ существует несколько реализаций такой формы, как цикл:

- Цикл с предусловием (**while**);
- Цикл с постусловием (**do…while**);
- Цикл с параметрами (**for**).

В этом уроке речь пойдет о двух таких реализациях — **while** и **do while**.

2. Цикл с предусловием

Все циклы состоят из двух частей:

- заголовок;
- тело цикла.

Заголовок отвечает за настройку цикла, то есть в каком случае цикл снова повторится.

Тело же цикла отвечает за сами действия, которые должны повторно выполниться. Вот, к примеру, представим себе первоклассника, который безумно любит мороженое. Ему мама выделила некоторую сумму денег. Естественно он побежал его покупать и вдруг вспомнил, что умножать и делить он не умеет. Как он будет решать свою проблему? Сначала он проверит, хватит ли ему денег на покупку первого мороженного. Если да, он это сделает и снова посмотрит на свой остаток денег.

Как видим в этом примере можно выделить заголовок цикла — пока денег достаточно, и само тело цикла — покупка мороженного. Рассмотрим эту задачу на блок-схеме:

Входные данные:

- Сумма денег первоклассника
- Стоимость одного мороженного

Выходные данные:

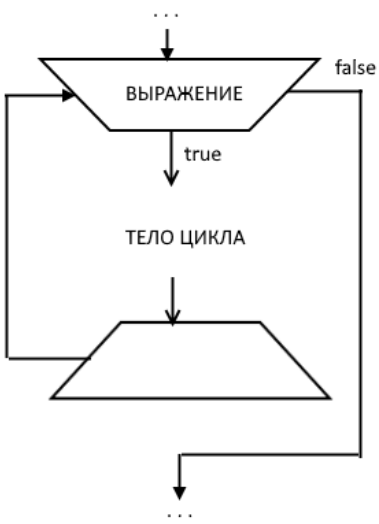
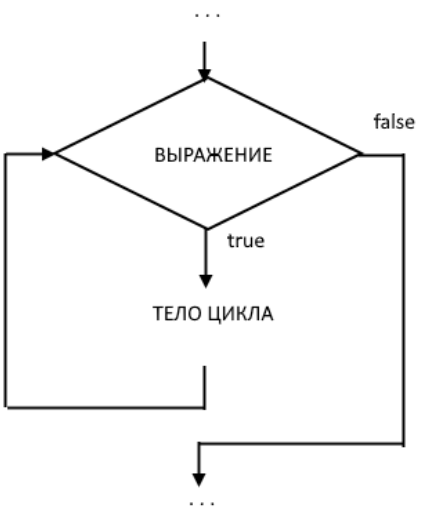
- Сдача после каждой покупки мороженого.

Особенность этой задачи состоит в том, что первоклассник перед покупкой проверяет достаточно ли у него

денег, и ни в коем случае наоборот: покупка, а после проверка. Ведь денег у него может и не хватить. Поэтому при решении этой задачи лучше всего использовать цикл с предусловием. В самом названии уже кроется ответ на принцип его работы: сначала идет проверка условия и только если он о истинно выполняется тело цикла.

При описании алгоритма работы цикла с предусловием на блок-схеме, можно использовать несколько способов (таблица 1).

Таблица 1.

Классический (соответствие стандартно принятому обозначению) — помещение цикла в парные фигуры трапеций	Обычный (упрощение конструкции в учебных целях) — использование блока условия (ромб)
	

А теперь, наконец-то построим решение задачи с помощью блок-схемы в двух вариантах (рис. 1).

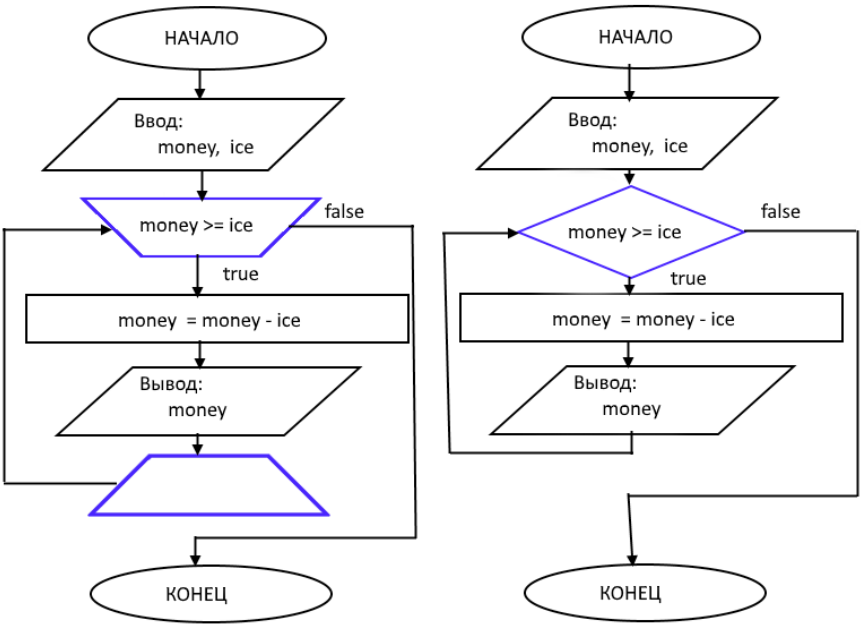


Рисунок 1

Давайте, проследим всю пошаговую цепочку действий от начала и до конца. Для более наглядной демонстрации примера примем, что входные данные деньги и стоимость мороженого равны 13 и 5 соответственно.

1. Блок. Начало.
2. Ввод данных: **money = 13**, **ice =5**.
3. Проверяем достаточно ли денег?
   money >= ice (13 >= 5).
   Да, достаточно. Поэтому можем опуститься в тело цикла.
4. Покупаем мороженое. Следовательно, деньги уменьшаются на стоимость мороженого
   money = money – ice (money = 13 – 5).

PS: *напоминаем, что оператор присвоения всегда сначала выполняет вычисления в правой части, а после результат присваивает левой. Поэтому первым делом вычитаем из денег стоимость мороженого, а далее обновляем данные для переменной money.*

5. Опускаемся еще ниже. Выводим оставшиеся деньги (**money = 7**).
6. Опять возвращаемся на проверку условия. Достаточно ли теперь денег для покупки мороженого?
   money >= ice (7 >= 5)
   Да, достаточно.
7. Покупаем еще одно мороженное:
   money = money – ice (money = 7 – 5).
8. Выводим остаток денег **(money = 2)**.
9. И снова проверка условия:
   money >= ice (2 >= 5).
   И вот сейчас первокласснику уже не хватает денег. Выполнение цикла заканчивается. И мы переходи в блок Конец.

После полного рассмотрения примера пришло время изучить конструкцию цикла с предусловием на языке C++.

2. Цикл while

Общий синтаксис и порядок выполнения цикла **while**

```
while(утверждение)
{
    действие для повторения;
}
```

1. Как уже говорилось, прежде всего осуществляется проверка утверждения.
2. Если утверждение в круглых скобках истинно, выполнятся действие, находящееся внутри фигурных скобок.
3. Если утверждение в круглых скобках ложно, программа перейдет на следующую строчку за закрывающейся фигурной скобкой цикла.
4. Если утверждение в круглых скобках было истинно и действие выполнилось, снова следует проверка утверждения.

Рисунок 2

Как видите, проверка утверждения повторяется при каждом выполнении цикла. Как только оно перестает быть верным, цикл завершается. Обратите внимания, что если утверждение ложно с самого начала, действие внутри цикла не будет выполнено ни разу.

Рассмотрим пример

Предположим, что некоему человеку необходимо написать очерк о 7 чудесах света. Перед тем как это сделать ему необходимо отправиться и посмотреть на каждое из чудес. И, только затем писать о последних.

```
#include <iostream>
using namespace std;
int main()
{
    // объявление управляющей переменной
    int counter=0;
    while(counter<7)// проверка утверждения
    {
        counter++;// изменение управляющей переменной
                  // действие для повторения
                  // вы увидели ... чудо света
        cout<<"You seen"<<counter<<
              "miracle of world!!!\n";
    }
    cout<<"Now, you can start your work.\n";
    return 0;
}
```

Теперь подробно разберемся как работает наш пример.

1. Объявляем переменную изначально равную 0.
2. Далее в условии цикла мы проверяем значение нашей переменной. Поскольку, именно от этого значения

зависит, будет цикл выполняться или нет, то такая переменная называется управляющей переменной цикла.

3. Значение переменной увеличиваем на единицу.

Примечание: *Данное действие является обязательным, так как если не изменять значение переменной управляющей циклом, результат проверки утверждения тоже никогда не изменится. Это может привести к очень распространенной ошибке под названием — вечный цикл. Если утверждение цикла — верно, а управляющая переменная всегда имеет одинаковое значение, следовательно — утверждение верно всегда. Представьте, грязные тарелки никогда не заканчиваются — их число всегда постоянно. Насколько хватит посудомойки?! Ненадолго, правда? Вот и программа не выдержит такого натиска и через некоторое время после запуска вечного цикла — выдаст ошибку на этапе выполнения. В нашем примере переменная counter объявлена как int. Это указывает не только на хранение целочисленного значения, а и на выделяемый в памяти размер (int -> 4 или 8 байт). И как можем догадаться, что ошибка во время выполнения произойдет ровно в тот момент, когда значение переменной превысит выделенный под нее размер в памяти. Во избежание таких ошибок нужно внимательно следить за тем, чтобы внутри тела цикла происходило изменение управляющей переменной.*

4. Далее, выводим на экран текущее значение нашей переменной в виде сообщения о номере просмотренного чуда света.
5. Опять возвращаемся к условию и проверяем значение управляющей переменной.

Цикл будет продолжать свою работу до тех пор, пока значение переменной не станет равно 7. В этом случае произойдет вывод на экран строки «*You seen 7 miracle of world!!!*», затем программа вернется к проверке условия. 7 < 7 — является ложью. Программа больше в цикл не войдет и перейдет к строке «*Now, you can start your work.*».

В процессе выполнения программы на экране мы увидим следующую картину:

Рисунок 3

Сейчас мы с Вами познакомились с одной из разновидностей цикла в языке C++. Надеемся, было не сложно. В следующем разделе урока мы узнаем о цикле альтернативном конструкции **while**.

3. Цикл с постусловием

Представим себе задачу, в которой необходимо смоделировать работу Датчика пожароопасности. Все свое время датчик должен сканировать температуру воздуха в помещении. И как только температура превысила пороговое значение (60°), сообщить о пожаре (рис. 4).

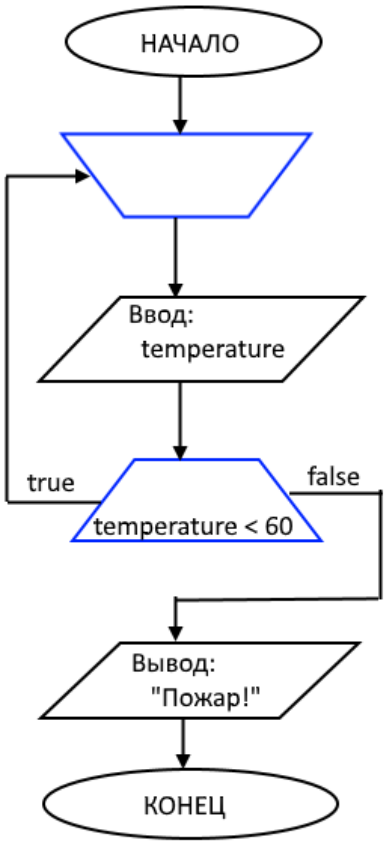


Рисунок 4

Что же тогда будет условием продолжения сканирования воздуха в помещении? Каждый раз датчик должен брать текущую температуру и проверять ее на допустимый предел. То есть первым делом, нужно сначала запросить температуру, а после только ее сравнивать. И никак не наоборот. Поэтому в этом случае лучше всего использовать цикл с постусловием: выполнение действий, а после проверка условия.

Представим другую ситуацию, нам необходимо определить результат деления одного числа на другое. Казалось бы все очень просто и скорее всего это как раз линейная задача. Да не тут-то было. Есть очень маленький и практически незаметный нюанс. А на ноль то делить нельзя! Благодаря конструкции цикла можно с легкостью решить такую проблему. Для этого достаточно у пользователя запрашивать ввод второго числа до тех пор, пока он не введет любое ненулевое значение. Давайте теперь рассмотрим блок-схему этого примера (рис. 5).

Выполнение цикла данного алгоритма возможно по двум сценариям:

1. Пользователь введет первое число, а далее введет второе ненулевое число. Условие не сработает на повтор действия ввода (т.к. В будет не равно нулю).
2. Во втором случае, когда все-таки пользователь введет число В равное нулю, алгоритм повторно предложит ввести только второе число. И так до тех пор, пока не получить нужный результат. Таким образом, мы сможем делать различные проверки на ввод корректных данных и обезопасить систему от ненужных ошибок.

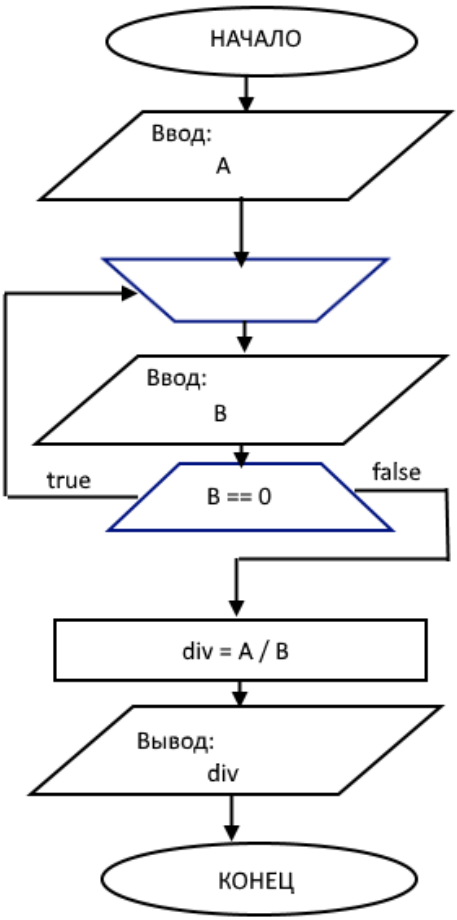


Рисунок 5

Давайте теперь приступим к изучению правильного написанию цикла с постусловием на языке C++.

4. Конструкция do while

Общий синтаксис и принцип работы **do while**:

```
do
{
    действие;
}
while(условие);
```

Цикл **do while** похож на цикл **while**. Разница состоит в том, что в **while** проверка условия производится сразу же при входе в цикл, и, лишь затем, если условие истинно — выполняется действие. В **do while** в любом случае сначала выполняется действие и только потом идет проверка условия. Если условие истинно, выполнение действия продолжается, а если нет, то выполнение передается следующему за **while** оператору. Другими словами, в отличие от **while** внутри **do while** действие хотя бы один раз выполняется. Давайте рассмотрим это на схеме:

Рисунок 6

Применение do while на практике

Предположим, нам необходимо написать программу, в которой пользователю предоставляется право выбора какого-то действия несколько раз подряд. Реализуем данную задачу сначала с помощью **while**, а затем с помощью **do while**.

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

int main()
{
    int answer,A,B,RES;
    // запрос на выбор операции
    cout<<"\nSelect operation:\n";
    cout<<"\n 1 - if you want to see SUM.\n";
    cout<<"\n 2 - if you want to see DIFFERENCE.\n";
    cout<<"\n 3 - if you want to exit.\n";
    cin>>answer;
    while(answer!=3){ // проверка условия
        switch(answer){
            case 1: // если пользователь выбрал
                    // сложение
                cout<<"Enter first digit:\n";
                cin>>A;
                cout<<"Enter second digit:\n";
                cin>>B;
                RES=A+B;
                cout<<"\nAnswer: "<<RES<<"\n";
                break; // остановка switch
            case 2: // если пользователь выбрал
                    // вычитание
                cout<<"Enter first digit:\n";
                cin>>A;
                cout<<"Enter second digit:\n";
                cin>>B;
```

```
                RES=A-B;
                cout<<"\nAnswer: "<<RES<<"\n";
                break; // остановка switch
            case 3: // если пользователь выбрал выход
                cout<<"\nEXIT!!!\n";
                break;
            default: // если выбранное действие
                     // некорректно
                cout<<"\nError!!! This operator isn't"
                    " correct\n";
        }
        // запрос на выбор операции
        cout<<"\nSelect operation:\n";
        cout<<"\n 1 - if you want to see SUM.\n";
        cout<<"\n 2 - if you want to see DIFFERENCE.\n";
        cout<<"\n 3 - if you want to exit.\n";
        cin>>answer;
    }
    cout<<"\nBye. \n";
    return 0;
}
```

В данном примере пользователю предлагается выбрать действие. Затем, после ввода, программа проверяет: если это действие — выход из программы — программа завершается, если нет, то производится вход в цикл, анализ действия и выполнение математической операции. Затем программа, снова спросит у пользователя, что он хочет сделать.

Данный код является не оптимальным решением. Как видите фрагмент

```
// запрос на выбор операции
cout<<"\nSelect operation:\n";
cout<<"\n 1 - if you want to see SUM.\n";
```

```
cout<<"\n 2 - if you want to see DIFFERENCE.\n";
cout<<"\n 3 - if you want to exit.\n";
cin>>answer;
```

повторяется несколько раз. В этом случае следует использовать **do while**. Данная конструкция приведет код к надлежащему виду.

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

int main()
{
    int answer,A,B,RES;

    do{ // вход в цикл
        // запрос на выбор операции
        cout<<"\nSelect operation:\n";
        cout<<"\n 1 - if you want to see SUM.\n";
        cout<<"\n 2 - if you want to see DIFFERENCE.\n";
        cout<<"\n 3 - if you want to exit.\n";
        cin>>answer;
        // анализ действия
        switch(answer){
        case 1: // если пользователь выбрал сложение
           cout<<"Enter first digit:\n";
           cin>>A;
           cout<<"Enter second digit:\n";
           cin>>B;
           RES=A+B;
           cout<<"\nAnswer: "<<RES<<"\n";
           break; // остановка switch
        case 2: // если пользователь выбрал вычитание
           cout<<"Enter first digit:\n";
           cin>>A;
           cout<<"Enter second digit:\n";
```

```
            cin>>B;
            RES=A-B;
            cout<<"\nAnswer: "<<RES<<"\n";
            break; // остановка switch
         case 3: // если пользователь выбрал выход
            cout<<"\nEXIT!!!\n";
            break;
         default: // если выбранное действие
                  // некорректно
            cout<<"\nError!!! This operator isn't "
                  "correct\n";
         }
      }
      while(answer!=3);
      cout<<"\nBye. \n";
      return 0;
}
```

Исходя из вышесказанного, вы должны понимать, что обе описанные в сегодняшнем уроке конструкции полезны. Вам необходимо лишь научиться выбирать ту или иную, в зависимости от задачи.

Теперь, когда с циклами мы немного знакомы, вы можете перейти к следующему разделу данного урока. Мы приготовили для вас еще несколько примеров по сегодняшней теме.

5. Примеры к уроку

ПРИМЕР 1

Постановка задачи

Написать программу, которая находит сумму всех целых чисел от 1 до 5 включительно. Название проекта **Summ**.

Код реализации

```
#include <iostream>
using namespace std;
int main(){
    int BEGIN=1;  // начало диапазона суммируемых
                  // значений
    int END=5;    // конец диапазона суммируемых
                  // значений
    int SUMM=0;   // переменная для накопления суммы
    int i=BEGIN;  // управляющая переменная цикла
                  // проверка условия
    while(i<=END){// (сравнение управляющей переменной
                  // с окончанием диапазона)
        SUMM+=i;  // накапливание суммы
        i++;      // изменение управляющей переменной
    }
    // показ результата
    cout<<"Result "<<SUMM<<"\n\n";
    return 0;
}
```

Комментарий к коду

В качестве комментария к коду, мы решили представить таблицу, которая досконально описывает каждую итерацию цикла:

ВХОДНЫЕ ДАННЫЕ			
BEGIN=1		END=5	
РАБОТА ЦИКЛА			
номер итерации	i	условие	SUMM
1	1	1<=5 - true	0+1=1
2	2	2<=5 - true	1+2=3
3	3	3<=5 - true	3+3=6
4	4	4<=5 - true	6+4=10
5	5	5<=5 - true	10+5=15
6	6	6<=5 - false	X
SUMM=15			

Рисунок 7

При изучении таблицы не трудно заметить, что управляющая переменная, так же выполняет роль переменной последовательно перебирающей значения для суммирования.

> **Примечание:** *Распространенным заблуждением является то, что управляющая переменная может изменяться только на единицу — это не так. Главное, чтобы переменная изменялась любым логичным образом.*

ПРИМЕР 2

Постановка задачи

Написать программу, выводящую на экран линию из 5 звёздочек.

Код реализации.

```
#include <iostream>
using namespace std;
int main(){
    int COUNT=5; // количество звездочек (длина линии)
    int i=0;     // управляющая переменная цикла
```

```
    while(i<=COUNT){ // проверка условия
        cout<<"*"; // вывод звездочки
        i++; // изменение управляющей переменной
    }
    cout<<"\n\n";
    return 0;
}
```

Комментарии к коду

1. Управляющая переменная на момент проверки условия равна количеству уже нарисованных звездочек. Так происходит потому, что переменная **i** увеличивается на единицу после каждого вывода *.
2. Цикл остановиться тогда, когда **i = 5**, что будет соответствовать количеству нарисованных *.
   Теперь, вам необходимо перейти к домашнему заданию!

6. Домашнее задание

1. Разработать программу, которая выводит на экран горизонтальную линию из символов. Число символов, какой использовать символ, и какая будет линия — вертикальная, или горизонтальная — указывает пользователь.
2. Написать программу, которая находит сумму всех целых нечетных чисел в диапазоне, указанном пользователе.
3. Дано натуральное число **n**. Написать программу, которая вычисляет факториал неотрицательных целых чисел **n** (т.е. число целое и больше **0**). Формула вычисления факториала приведена ниже.

```
n! = 1*2*3*....*n, (формула вычисления факториала
                    числа n)
0! = 1 (факториал 0 равен 1
       (по определению факториала))
```

4. У швеи имеется ткань длиной **L**. Ей необходимо пошить подушки длиной **P**. Условно договоримся, что ширина ткани совпадает с шириной подушки. Определить сколько подушек сможет пошить швея, если умножать и делить она не умеет.
5. На складе имеется определенное количество ящиков с яблоками (**N>=0** — запрашивается у пользователя). Необходимо освободить склад. Машины по очереди подъезжают и забирают определенное количество ящиков. Определить сколько машин подъехало к складу.

Урок № 3
Циклы

© Татьяна Лапшун
© Компьютерная Академия «Шаг», www.itstep.org

Все права на охраняемые авторским правом фото-, аудио- и видеопроизведения, фрагменты которых использованы в материале, принадлежат их законным владельцам. Фрагменты произведений используются в иллюстративных целях в объёме, оправданном поставленной задачей, в рамках учебного процесса и в учебных целях, в соответствии со ст. 1274 ч. 4 ГК РФ и ст. 21 и 23 Закона Украины «Про авторське право і суміжні права». Объём и способ цитируемых произведений соответствует принятым нормам, не наносит ущерба нормальному использованию объектов авторского права и не ущемляет законные интересы автора и правообладателей. Цитируемые фрагменты произведений на момент использования не могут быть заменены альтернативными, не охраняемыми авторским правом аналогами, и как таковые соответствуют критериям добросовестного использования и честного использования.

Все права защищены. Полное или частичное копирование материалов запрещено. Согласование использования произведений или их фрагментов производится с авторами и правообладателями. Согласованное использование материалов возможно только при указании источника.

Ответственность за несанкционированное копирование и коммерческое использование материалов определяется действующим законодательством Украины.