



ОСНОВЫ
ПРОГРАММИРОВАНИЯ
НА ЯЗЫКЕ C++

Урок № 4

Цикл for

Содержание

| | |
|--|-----------|
| Конструкция for | 3 |
| Некоторые особенности синтаксиса for | 6 |
| Условие | 8 |
| Оператор break | 9 |
| Оператор continue..... | 13 |
| Практические примеры..... | 15 |
| Пример 1 | 15 |
| Пример 2 | 16 |
| Пример 3 | 18 |
| Пример 4 | 19 |
| Домашнее задание..... | 24 |

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

Конструкция for

В прошлом уроке мы разобрали понятие цикла и две его конструкции — **while** и **do while**. Теперь же пришла очередь к рассмотрению еще одной конструкции цикла — оператор **for**. Суть его работы тоже заключается в повторении действий, но с заданным количеством раз. Оператор **for** содержит дополнительную переменную в виде счетчика. Именно для нее и настраивается сам цикл.

Рассмотрим пример, когда учитель физкультуры попросил ученика присесть 5 раз. Данная ситуация разделена на две части: учитель делает подсчет приседаний, а ученик выполняет действия. Таким образом, в тот момент, когда произносятся 1, 2, 3 ... , как раз и срабатывает счетчик, а сам цикл повторения действий выполняет ученик. То есть мы можем отследить начальный момент, конечный момент и изменение счетчика.

А теперь приступим к изучению самой конструкции. Общий синтаксис и принцип работы конструкции **for**:

```
for (инициализация переменной; проверка условия;  
    изменение переменной)  
{  
    действие;  
}
```

Принцип выполнения цикла:

1. Инициализация переменной.
2. Проверка условия.

3. Выполнение действия, если условие истинно.
4. Если условие ложно, выполнение следующего за циклом оператора.
5. Если условие было истинно – изменение управляющей переменной.
6. Проверка условия. Далее снова пункт 3 или 4.

Давайте теперь напишем программу про нашего ученика на уроке физкультуры.

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

int main() {
    for (int count = 1; count <= 5; count++)
    {
        cout << "The student squatted " << count <<
            " time\n";
    }

    return 0;
}
```

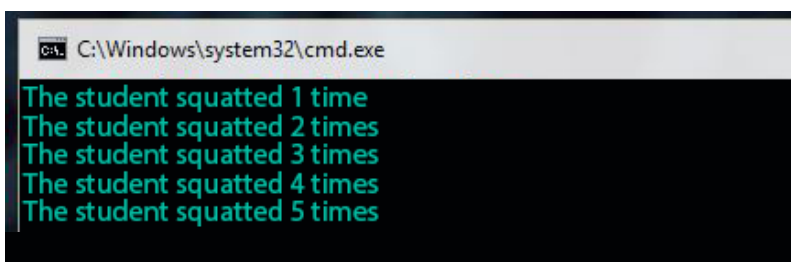


Рисунок 1

Сейчас более детально рассмотрим последовательность выполнения всех шагов этой программы (рис. 2).

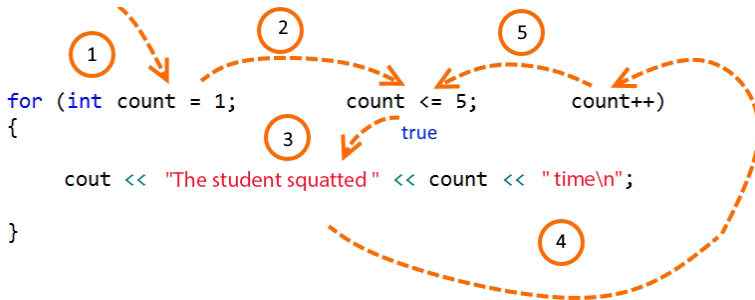


Рисунок 2

Комментарий к примеру

1. Внутри цикла объявляется переменная **count** равная **1**. Это и будет управляющая переменная.
2. Затем, осуществляется проверка значения этой переменной с помощью условия **count<=5**;
3. Если условие истинно (а так будет, пока **count** не достигнет значения **6**) выполняется показ значения **count** на экран (**cout << "The student sat down " << count << " time\n";**) и изменение управляющей переменной **count** на **1** (**count ++**). Затем, снова проверяется условие.
4. Если условие ложно (то есть значение **count** стало равно **6**), то программа переходит на следующую строчку за закрывающейся фигурной скобкой цикла.

Примечание: Обратите внимания, что первый шаг — СОЗДАНИЕ И ИНИЦИАЛИЗАЦИЯ ПЕРЕМЕННОЙ — всегда выполняется только один раз.

Чаще всего в отличие от обычной жизни в программировании отчет идет не с единицы, а с нуля. Поэтому существует еще один способ реализации данного примера.

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

int main() {
    for (int count = 0; count < 5; count++)
    {
        cout << "The student squatted " << count <<
            " time\n";
    }
    return 0;
}
```

Некоторые особенности синтаксиса for

Несмотря на простоту работы оператора, он обладает некоторыми особенностями форм записи.

Инициализация управляющей переменной

1. Инициализация и создание переменной производится в цикле.

```
// вывод целых чисел от 1 до 10 с шагом изменения 1
for (int x = 1; x <= 10; x++)
{
    cout << x << " ";
}
cout << endl;
```

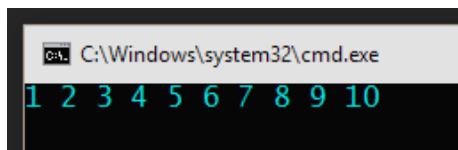


Рисунок 3

2. Создание переменной производится до цикла, а инициализация в цикле.

```
// вывод чисел от 0 до 10 с шагом изменения 2
int x;
for (x = 0; x <= 10; x += 2)
{
    cout << x << " ";
}
cout << endl;
```

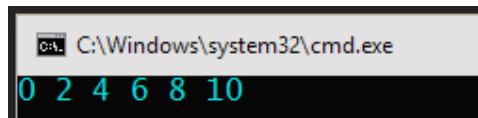


Рисунок 4

3. Инициализация и создание переменной производятся до цикла.

```
// вывод чисел от 0 до 1 с шагом изменения 0.2
float x = 0;
for (; x <= 1; x = x + 0.2)
{
    cout << x << " ";
}
cout << endl;
```

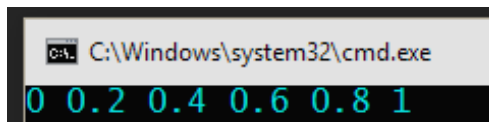


Рисунок 5

Все три примера являются абсолютно функционирующими и равновесными.

Изменение управляющей переменной

Изменение управляющей переменной можно перенести внутрь тела цикла, как это происходит в **while** и **do while**.

```
// вывод чисел от 1 до 5 с шагом изменения 1
for (int x = 1; x <= 5; )
{
    cout << x << " ";
    x++;
}
cout << endl;
```

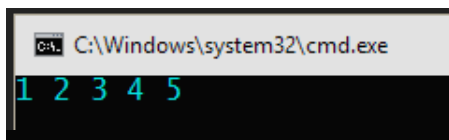


Рисунок 6

Условие

Условие конструкции также можно пропустить, однако в этом случае оно будет считаться по умолчанию истинным. Таким образом, мы получаем постоянно истинное условие и, как следствие — ВЕЧНЫЙ ЦИКЛ.

```
// вечный цикл
for (int x = 1; ; x++)
{
    cout << x << " ";
}
cout << endl;
```

Примечание. Как пропустить условие и избежать вечного цикла — читайте следующий раздел урока.

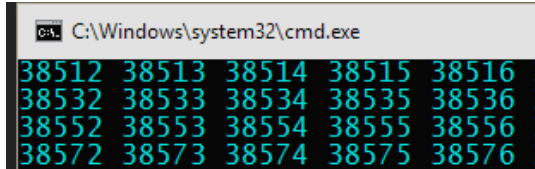


Рисунок 7

Исходя из вышеописанного, мы можем сделать следующий вывод: Ни одна из частей цикла **for** не является обязательной.

Как видите, работа **for** проста и аналогична работе **while**. Что выбрать?! Это зависит от поставленной задачи и от вашего решения.

Оператор break

Нередко при работе с циклами, возникает необходимость искусственно прервать выполнение цикла. Для этого используется, уже знакомый вам (по изучению **switch**), оператор **break**. Этот оператор должен находиться в теле цикла, в том месте где необходимо сделать остановку. Например, именно с помощью этого оператора, мы можем решить проблему вечного цикла, в ситуации, когда условие в цикле **for** не указывается.

Рассмотрим пример:

```
#include <iostream>
using namespace std;
int main()
{
    for (int x = 1;; x++)
    {
        if (x == 4) break; // если x стал равен 4 -
                          // остановить цикл
    }
}
```

```

        cout << x << " ";
    }
    cout << "Bye!\n";
    return 0;
}

```

Снимок экрана выполнения программы:

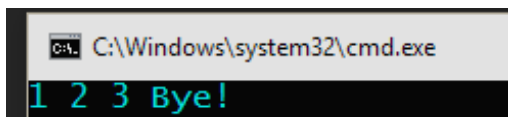


Рисунок 8

Комментарии к примеру

1. Согласно правилу, условие цикла всегда истинно, так как его просто нет.
2. При значениях 1, 2 и 3 переменной **x** условие оператора **if** выполняться не будет. **break**, естественно не сработает, так как находится в теле **if**. Между тем, на экран последовательно будут выводиться числа 1, 2, 3.
3. Когда **x** станет равно 4, программа попадет в тело **if** и выполнится **break**. Цикл сразу же будет остановлен, а выполнение программы перейдет на следующую строчку за закрывающейся фигурной скобкой оператора **for**.
4. На экране появится надпись *Bye!*
5. Цифра 4 на экране никогда не появится, так как, если сработал **break**, все что находится в цикле ниже него уже не выполнится.

Примечание: *break* может быть использован либо в цикле, либо в операторе *switch*. Любое другое

размещение приводит к ошибке на этапе компиляции.

Кроме этого оператор **break** может использоваться не только для прерывания бесконечного цикла. Допустим, мы загадали целое число в пределах от 1 до 10. Наша программа будет предлагать угадать это число за пять попыток. В этой задаче цикл должен выполняться 5 раз (**for (int n = 1; n <= 5; n++){}).** Правда пользователь может отгадать раньше, чем на пятой попытке. Поэтому программа должна каждый раз проверять отгадали ли ее число и в случае ввода правильного ответа прервать цикл.

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

int main(){
    // объявляем переменную с загаданным числом
    int magicNum = 2;
    cout << "=====\n\n";
    cout << " My magic number between 1 and 10\n";
    cout << "=====\n\n";
    int user = 0;
    for (int n = 1; n <= 5; n++)
    {
        cout << "Your number is -> ";
        cin >> user;
        // проверяем, угадал ли пользователь наше
        // число если да, выводим поздравление
        // и прерываем цикл
        if (user == magicNum)
        {
            cout << "Congrats!!!\n";
            break;
        }
    }
}
```

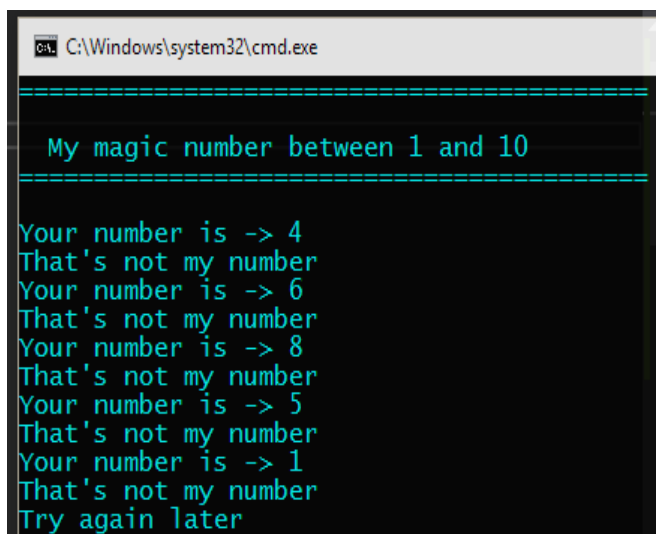
```

else
{
    // иначе пользователь еще не отгадал
    cout << "That's not my number\n";
}
// если счетчик достиг 5, выводим сообщение
// попробовать сыграть снова
if (n == 5)
{
    cout << "Try again later\n";
}
}
return 0;
}

```

Варианты выполнения программы

Пользователь НЕ отгадал число за пять попыток



```

C:\Windows\system32\cmd.exe

=====
My magic number between 1 and 10
=====

Your number is -> 4
That's not my number
Your number is -> 6
That's not my number
Your number is -> 8
That's not my number
Your number is -> 5
That's not my number
Your number is -> 1
That's not my number
Try again later

```

Рисунок 9

Пользователь отгадал число

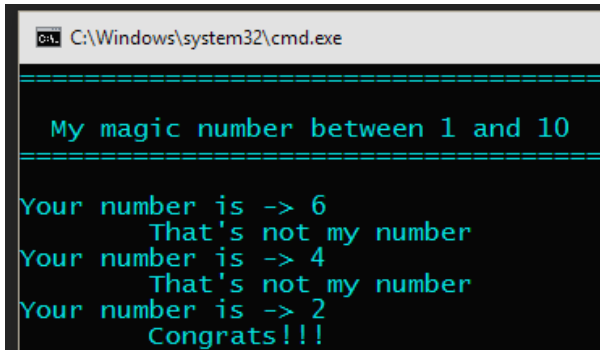


Рисунок 10

Оператор continue

Оператор **continue** используется для прерывания текущей итерации цикла и осуществления перехода на следующий шаг. В ряде случаев, такие действия являются необходимыми. Если выполняется оператор **continue**, то зависимости от вида цикла происходит следующее: Циклы **while** и **do while** останавливают выполнение шага и переходят к проверке условия.

Цикл **for** также останавливает выполнение шага. Но, сначала переходит к изменению управляющей переменной, а потом уже к проверке условия.

Рассмотрим пример: показать на экран все нечетные целые числа, в диапазоне от нуля до 25 включительно.

```

#include <iostream>
using namespace std;
int main()
{
    for (int i = 0; i < 26; i++)
    {
  
```

```

    if (i % 2 == 0) // если число делится на два
                    // без остатка
    {
        continue; // остановить итерацию цикла
                  // и перейти к i++
    }
    cout << i << " ";
}
cout << endl;
return 0;
}

```

Снимок экрана выполнения программы:

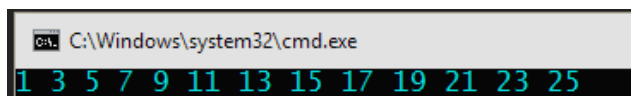


Рисунок 11

Комментарии к примеру

1. Цикл начинает свое движение с нуля и проходит итерации до 25 включительно.
2. Внутри цикла предусмотрено условие: если число **i** — четное, нужно остановить текущий шаг цикла (**continue**;) и перейти к конструкции **i++**.
3. То, что располагается ниже сработавшего оператора **continue** на текущем шаге уже не выполнится.
4. Если условие **if** не выполняется, значит число **i** нечетное, **if** будет проигнорирован, а число — отображено на экран.

Теперь, когда мы познакомились с теоретическими материалами урока, давайте перейдем к следующему разделу, где будет рассмотрено несколько практических задач.

Практические примеры

Пример 1

Постановка задачи

Часы бьют каждый час, столько раз, сколько времени. Написать программу, которая подсчитает, сколько раз пробьют часы за 12 часов.

Код реализации

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

int main() {
    int sum = 0;
    for (int bom = 1; bom <= 12; bom++)
    {
        sum += bom; // накопление суммы ударов
    }
    // Часы проббили 78 раз.
    cout << " Hours have punched " << sum
         << " times.\n\n";
    return 0;
}
```

Снимок экрана выполнения программы:

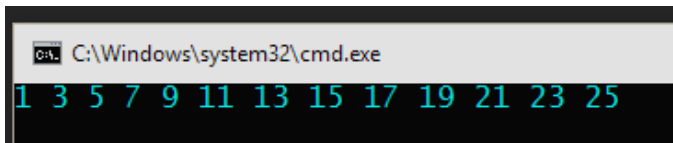


Рисунок 12

Комментарии к коду

1. Изначально объявляется переменная **sum** равная нулю.
2. Цикл формируется из трех конструкций **int bom=1;** — начальная инициализация, **bom<=12;** — условие, **bom++** — изменение управляющей переменной.
3. Внутри тела цикла накапливается сумма ударов путем прибавления управляющей переменной к значению общей суммы.
4. Когда **i** достигнет значения **13**, цикл остановится и на экран покажется результат.

Пример 2

Постановка задачи

Пользователь с клавиатуры последовательно вводит целые числа. Как только пользователь ввел **0**, необходимо показать на экран сумму всех введенных чисел.

Код реализации

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

int main() {
    int digit, sum = 0;

    for (;;) { // реализация бесконечного цикла
        cout << "Enter digit:";
        cin >> digit; // ввод числа

        if (digit == 0) // если введен 0
```



```

        break; // остановить цикл
        sum += digit; // накопление суммы
    }

    // показ результата
    cout << "Sum of digits" << sum << "\n\n";
    return 0;
}

```

Снимок экрана выполнения программы:

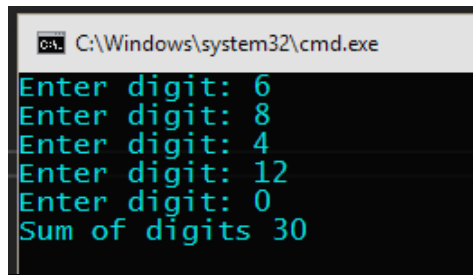


Рисунок 13

Комментарии к коду

1. В программе реализован условно бесконечный цикл. То есть остановка цикла происходит искусственным путем (**break**).
2. На каждой итерации пользователь вводит число.
3. Осуществляется проверка, если это число — **0**, значит пора остановить цикл, если не **0**, необходимо прибавить число к общей сумме.
4. После того, как отработает **break** и цикл прекратит работу, на экран покажется сумма всех введенных с клавиатуры чисел.

Пример 3

Постановка задачи

Написать программу, которая показывает все числа, которым кратно число, введённое с клавиатуры.

Код реализации

```
#include <iostream>
using namespace std;
int main() {
    int digit;
    cout << "Enter digit:";
    cin >> digit;

    // цикл перебирает числа от 2 до введенного числа
    for (int i = 2; i < digit; i++) {
        // если число не делится на текущее значение i
        // без остатка остановить данный шаг и перейти
        // к следующему

        if (digit%i != 0) continue;
        // показать i на экран
        cout<<i<<" ";
    }
    cout << endl << endl;
    return 0;
}
```

Снимок экрана выполнения программы:

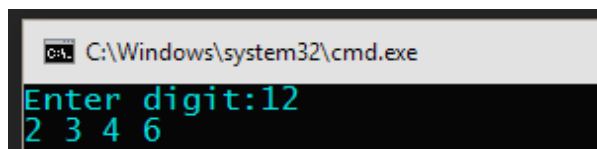


Рисунок 14

Комментарии к коду

1. Пользователь вводит число для анализа.
2. Цикл последовательно перебирает все числа от 2 до исходного. Осуществляется проверка: если искомое число на текущее без остатка не делится, необходимо прервать данный шаг цикла и перейти к части `i++`. (`continue`).
3. Если искомое число на текущее без остатка делится, то на экран показывается текущее число.

Пример 4

Постановка задачи

За сутки на заводе изготавливается 4 детали. Суммарное время изготовления одной детали не должно превышать 120 минут. При этом каждая деталь проходит по 2 станка. Разработать программу, которая должна подсчитать количество качественно изготовленных деталей за сутки без нарушения технологии производства.

Код реализации

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

int main() {
    // количество качественных деталей
    int amount = 0;

    // цикл для перебора 4 деталей
    for (int d = 1; d <= 4; d++)
    {
```

```

// суммарное время изготовления детали
int alltime = 0;
// переменная для хранения времени на станке
int time = 0;

// запрашиваем время на первом станке
cout << "How much time the first machine "
      "spends for " << d << " part\n";
cin >> time;
alltime += time;

// проверяем не превышен ли лимит времени
if (alltime > 120)
{
    cout << "That part is defective. "
          "The time limit exceeded\n";
    continue;
}

// запрашиваем время на втором станке
cout << "How much time the second machine "
      "spends for " << d << " part\n";
cin >> time;
alltime += time;

// проверяем не превышен ли лимит времени
if (alltime > 120)
{
    cout << "That part is defective. "
          "The time limit exceeded\n";
    continue;
}
// если время не было превышено на обоих
// станках увеличиваем количество правильно
// изготовленной детали
amount++;
}

```

```

    cout << "\nThe factory manufactured " << amount <<
        " parts\n\n";
    return 0;
}

```

Снимок экрана выполнения программы (рис. 15).

```

C:\Windows\system32\cmd.exe
How much time the first machine spends for 1 part
50
How much time the second machine spends for 1 part
60
How much time the first machine spends for 2 part
140
That part is defective. The time limit exceeded
How much time the first machine spends for 3 part
30
How much time the second machine spends for 3 part
50
How much time the first machine spends for 4 part
30
How much time the second machine spends for 4 part
110
That part is defective. The time limit exceeded
The factory manufactured 2 parts

```

Рисунок 15

Комментарии к коду

1. Объявляем переменную **amount** со значением ноль. Именно она будет накапливать значения только тех деталей, которые были изготовлены без нарушения технологии.
2. Настраиваем цикл **for** для переменной **d**, которая будет изменяться от **1** до **4** (перебор всех деталей фабрики за сутки — **for (int d = 1; d <= 4; d++)**).

3. Изначально переменная **d** хранит номер первой детали.
4. Проверяем, этот номер меньше или равен четырём? Если да, переходим к пункту 3, а если нет — к пункту 10.
5. Описываем тело цикла, которое будет выполняться для каждой из 4 деталей. Объявляем две переменные с начальным значением ноль для времени изготовления текущей детали:
 - **alltime** — накопление времени изготовления текущей детали;
 - **time** — время для изготовления детали на станке.
6. Запрашиваем время изготовления текущей детали на первом станке (**cin >> time;**). Используя сокращённую форму присвоения (**+=**), прибавляем введённое время в общее (**alltime += time;**).
7. Делаем проверку — не превышен ли лимит времени (**if(alltime > 120)**) ?
8. Если да, изготавливать эту деталь больше не имеет смысла т.к. нарушена технология производства. Выводим соответствующее сообщение (**cout << "That part is defective. The time limit exceeded\n";**). Прерываем выполнения текущей итерации (**continue;**) и переходим снова в настройку цикла **for** с новым значением переменной **d** (к пункту 2 -> **d++**)
9. Если нет, движемся ниже по алгоритму.
10. Запрашиваем время изготовления текущей детали на втором станке(**cin >> time;**). Используя сокращённую

форму присвоения (**+=**), снова прибавляем введенное время в общее (**alltime += time;**).

11. Делаем проверку — не превышен ли сейчас лимит времени (**if(alltime > 120))**)?
12. Если да, изготавливать также эту деталь больше нет смысла т.к. нарушена технология производства. Выводим соответствующее сообщение (**cout << "That part is defective. The time limit exceeded\n";**). Прерываем выполнения текущей итерации (**continue;**) и переходим снова в настройку цикла **for** с новым значением переменной **d** (к пункту **2 -> d++**)
13. Если нет, движемся ниже по алгоритму.
14. Если пункты 5 и 7 не сработали, значит, что деталь суммарно изготавливалась меньше 120 минут. По-этому переменная **amount** увеличится на единицу (**amount++;**)
15. Переходим у пункту 2 для изготовления следующей по очереди детали (**d++**).
16. Когда цикл увеличил переменную **d** до значения **5**, условие продолжения цикла стало ложным. Поэтому выполнение алгоритма продолжается за всем телом цикла, то есть срабатывает оператор вывода сообщения с количеством правильно изготовленных деталей (**cout << "\nThe factory manufactured "<< amount << "\n parts\n\n";**).

Домашнее задание

1. Пользователь вводит с клавиатуры число больше нуля, необходимо вывести все его цифры, начиная с конца.

Примечание. Например, пользователь ввел число 12345. На экране должно появиться число наоборот — 54321.

2. Пользователь вводит с клавиатуры число, необходимо показать на экран сумму его цифр.

Примечание. Например, пользователь ввел число 12345. На экране должно появиться сообщение о том, что сумма цифр числа 15.

3. В первый день улитка проползла 15 см. Каждый следующий день она проползала на 2 см дальше. Определить какое общее расстояние проползет улитка через N дней.

Примечание. Например, пользователь ввел число 4. Следовательно, улитка ползла 4 дня, поэтому суммарный путь составит 72 см.

4. Для принятия решения студент Д. подбрасывал монетку 9 раз. Если в результате количество выпавших монеток стороной «орел» было четным числом, принимал решение в положительную сторону, иначе в отрицательную. Напишите программу, которая 9 раз запрашивает число 1 или 0 (орел/решка) и выдает соответствующий результат решения проблемы студента Д.

5. Вывести на дисплей календарь на выбранный месяц с учетом указанного номера дня недели для начала месяца.

Подсказка. Программу условно разбить на две части. Первый цикл будет выводить нужное количество пустых клеток. Второй же цикл начнет выводить календарь с первого дня по последний день в заданном месяце. Переход на новую строку считать кратный семи с указанным смещением номера дня недели.

Бонусное задание: определить количество выходных в заданном месяце.

6. Пример выполнения программы:

```

C:\Windows\system32\cmd.exe
Input a month number (1-12) -> 8
Input a number day of week (1-7) when that month started-> 7

                        August

    Mo      Tu      We      Th      Fr      Sa      Su
    2       3       4       5       6       7       8
    9      10      11      12      13      14      15
   16      17      18      19      20      21      22
   23      24      25      26      27      28      29
   30      31

That month weekends is 9
  
```

Рисунок 16



Урок № 4

Цикл for

© Татьяна Лапшун

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

Все права на охраняемые авторским правом фото-, аудио- и видео-произведения, 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.