Урок 4. Цикл for

При написании программ очень часто встречается необходимость многократного повторения различных действий. Для этого можно повторять один и тот же оператор много раз, но число повторов часто составляет десятки, сотни, тысячи. В языке Си есть несколько видов операторов цикла, которые резко сокращают объём кода.

На этом уроке мы рассмотрим самый употребительный цикл – цикл **for**.

Задание 1.1

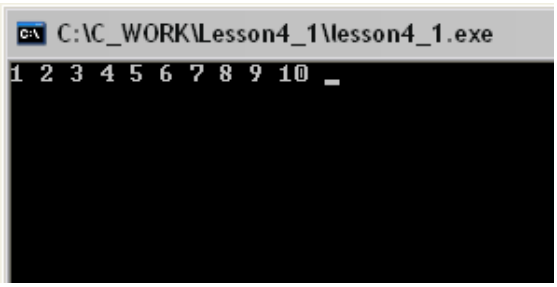
Вывести на экран целые числа от 1 до 10.

Решение:

```
#include <stdio.h>

void main()
{ int i;
  for (i = 1; i <= 10; i++)
  {
    printf("%d ", i);
  }
  getchar();
}
```

После набора текста программы откомпилируйте её (F5) и запустите (CTRL+R). Появится последовательность целых чисел от 1 до 10:



Пояснения к тексту программы

Новым в этой программе является **цикл for**.

```
for (i = 1; i <= 10; i++)
{
  printf("%d ", i);
}
```

Он состоит из трёх частей:

1. Ключевого слова `for`
2. Трёх выражений в круглых скобках, разделённых точкой с запятой.
3. Тела цикла (операторы между фигурными скобками `{ }`).

Разберём подробнее его работу. Для этого разобьём цикл на отдельные шаги:

Шаг	Выражение	Что делать
1	`i = 1`	Выполнить *выражение 1* (поместить число 1 в переменную `i`. `i` называется счётчиком цикла)
2	`i <= 10`	Проверить условие в *выражении 2*. Если выполняется – перейти к пункту 3, иначе – выйти из цикла Если значение переменной `i` превысит 10, то цикл завершится.

3	`{`
	`  printf("%d ", i);`
	`}`
		Выполнить *тело цикла*
		Вывести переменную `i` (счётчик цикла) на экран

4	`i++`	Выполнить *выражение 3* Увеличить значение переменной `i` на единицу (`i++` - сокращение от `i=i+1`)
5		Перейти к шагу 2

В общем виде цикл `for` выглядит таким образом:

```
for (выражение1; выражение2; выражение3)
{
  /* Тело цикла */
}
```

Задание 1.2

Вывести на экран целые числа от 10 до 1 (т.е. в обратном порядке – 10 9 ... 2 1).

Решение

```
#include <stdio.h>

void main()
{ int i;
  for (i = 10; i >= 1; i--)
  {
    printf("%d ",i);
  }
  getchar();
}
```

Здесь отсчёт идёт в обратном порядке – от 10 до 1. Поэтому здесь условие – `i>=1`, а выражение 3 (итерация) – `i--` (оператор декремента). Оператор декремента уменьшает значение переменной `i` на единицу (`i--` - сокращение от `i=i+1`).

В языке Си есть ряд арифметических операторов, являющихся сокращениями. Они очень часто используются при программировании на Си, поэтому приводим их здесь.

Оператор	Развёрнутый вариант
`i++;`	`i = i + 1;`
`i--;`	`i = i - 1;`
`i += a;`	`i = i + a;`
`i -= a;`	`i = i - a;`
`i *= a;`	`i = i * a;`
`i /= a;`	`i = i / a;`
`i %= a;`	`i = i % a;`

Задание 2.1

Вывести на экран чётные числа от 2 до 50.

Решение

```
#include <stdio.h>

void main()
{ int i;
  for (i = 2; i <= 50; i += 2)
  {
    printf("%d ",i);
  }
  getchar();
}
```

В этом задании применён цикл с шагом 2. Для этого в качестве выражения 3 использован оператор `i+=2`, который является сокращением от `i=i+2`. Он увеличивает значение переменной `i` на 2. И в теле цикла печатаются только чётные числа.

Задание 2.2

Вывести на экран чётные числа от 50 до 2

Решение

```
#include <stdio.h>

void main()
{ int i;
  for (i = 50; i >= 2; i-=2)
  {
    printf("%d ",i);
  }
  getchar();
}
```

В качестве выражения 3 использован оператор `i-=2`, который является сокращением от `i=i-2`. Он уменьшает значение переменной `i` на 2. И в теле цикла печатаются чётные числа в обратном порядке.

Задание 3.1

Вывести на экран возрастающую геометрическую прогрессию: 1, 2, 4, 8, 16, … 1024

Решение

```
#include <stdio.h>

void main()
{ int i;
  for (i = 2; i <= 1024; i *= 2)
  {
    printf("%d ",i);
  }
  getchar();
}
```

В качестве выражения 3 использован оператор `i*=2`, который является сокращением от `i=i*2`. Он умножает значение переменной `i` на 2.

Задание 3.2

Вывести на экран убывающую геометрическую прогрессию: 1024, …, 16, 8, 4, 2, 1.

Решение

```
#include <stdio.h>

void main()
{ int i;
  for (i = 1024; i >= 2; i /= 2)
  {
    printf("%d ",i);
  }
  getchar();
}
```

Оператор `i/=2` является сокращением от `i=i/2`. Он делит значение переменной `i` на 2.

Задание 4.

Найти сумму целых чисел от 1 до 100, используя цикл `for`. Не используйте формулу для суммы арифметической прогрессии: это задание на цикл.

Решение

```
#include <stdio.h>

void main()
{ int i, s;

  s = 0;
  for (i = 1; i <= 100; i++)
  {
      s += i;
  }
  printf("Ответ: %d\n", s);
  getchar();
}
```

Задание 5.*

Напишите программу, выводящую таблицу синусов и косинусов для углов от 0 до 90 градусов. Шаг – 5 градусов.

Решение

```
#include <stdio.h>
#include <math.h>

void main()
{ double i, r;
  const double PI = 3.141592653589793;
  printf("x     sin(x)     cos(x)\n");
  for (i = 0.0; i <= 90.0; i += 5.0)
  { r = i * PI / 180;
    printf("%5.0lf%10.7lf%10.7lf\n", i, sin(r),cos(r));
  }
  getchar();
}
```

Пояснения к тексту программы

Разберём некоторые строки программы

```
#include <math.h>
```

Подключение стандартной математической библиотеки. В ней есть функции для вычисления квадратного корня, синуса, косинуса и других математических функций.

```
const double PI = 3.141592653589793;
```

Объявление **константы** – числа π. В отличие от переменных, значения констант изменить нельзя: они задаются в исходном тексте программы. Константы объявляются так же, как и переменные, но перед типом пишется ключевое слово `const`.

```
r = i * PI / 180;
```

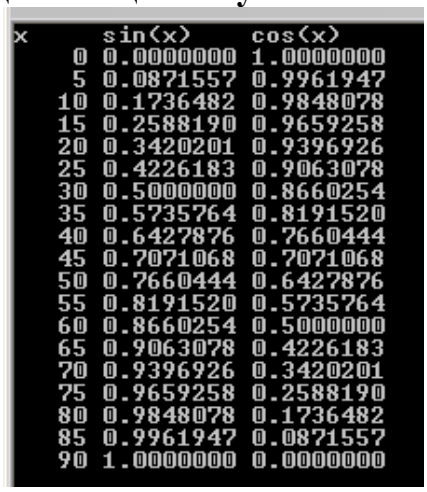
Перевод градусов в радианы. Тригонометрические функции математической библиотеки работают с углами в радианах.

```
printf("%5.0lf%10.7lf%10.7lf\n", i, sin(r), cos(r));
```

В этой строке программы происходит вывод строки таблицы синусов и косинусов на экран. Строка в кавычках задаёт количество, тип и формат выводимых чисел. Разберём каждый спецификатор этой строки

`%5.0lf`	Число типа double, 5 знакомест на экране, нет знаков после запятой
`%10.7lf`	Число типа double, 10 знакомест на экране, 7 знаков после запятой
`%10.7lf`	То же
`\n`	Переход на новую строку

Примерный вид таблицы синусов и косинусов на экране



```
x      sin(x)     cos(x)
    0 0.0000000 1.0000000
    5 0.0871557 0.9961947
   10 0.1736482 0.9848078
   15 0.2588190 0.9659258
   20 0.3420201 0.9396926
   25 0.4226183 0.9063078
   30 0.5000000 0.8660254
   35 0.5735764 0.8191520
   40 0.6427876 0.7660444
   45 0.7071068 0.7071068
   50 0.7660444 0.6427876
   55 0.8191520 0.5735764
   60 0.8660254 0.5000000
   65 0.9063078 0.4226183
   70 0.9396926 0.3420201
   75 0.9659258 0.2588190
   80 0.9848078 0.1736482
   85 0.9961947 0.0871557
   90 1.0000000 0.0000000
```

Нетиповое применение цикла for

До этого момента мы рассматривали традиционные способы применения цикла for – для реализации арифметической прогрессии. Но в языке Си в качестве трёх выражений в круглых скобках после слова for может стоять любое выражение. Это редко используется, но всё же приведём примеры для лучшего понимания цикла for. Тем более, что ни в QBasic, ни тем более в Turbo Pascal Вы такого не увидите.

Задание 6*

Напишите программу, которая печатает два столбца цифр:

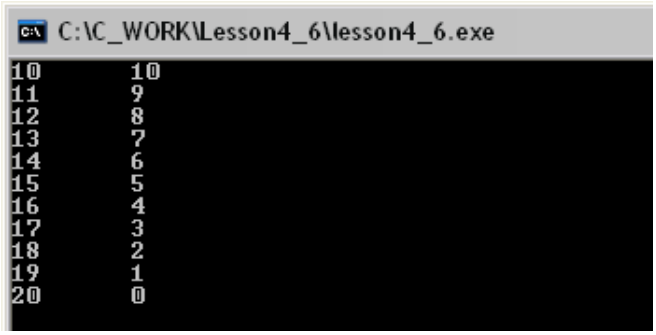
```
10   10
11   9
12   8
…
19   1
20   0
```

Решение:

```
#include <stdio.h>

void main()
{
  int i, j;

  for (i = 10, j = 10; i <= 20; i++, j--)
  {
    printf("%d\t%d\n", i, j);
  }
  getchar();
}
```



Пояснения к тексту программы

```
for (i = 10, j = 10; i <= 20; i++, j--)
{
  printf("%d\t%d\n", i, j);
}
```

В этом цикле не один счётчик, а два. В *выражении 1* они инициализируются; два оператора присваивания разделяются запятой. В *выражении 2* контролируется лишь первый счётчик – `i`. В *выражении 3* счётчик `i` увеличивает своё значение на единицу, а счётчик `j` – уменьшает на единицу; операторы инкремента и декремента также разделены запятой

В строке, передаваемой функцией printf, определяется формат вывода чисел. `%d` – вывести целое число, `\t` – символ табуляции, `\n` – символ перехода на новую строку.

Задание 7**

Имеется генератор случайных чисел, который выдаёт одно число за другим. Выводите эти числа на экран до тех пор, пока они не делятся на 11. Для решения задачи используйте цикл `for`.

Указание: в библиотеке `stdlib.h` есть функция `int rand()`, которая возвращает случайное число.

Если Вы хотите, чтобы последовательность случайных чисел была каждый раз разной, то для сброса генератора случайных чисел используйте функцию `srand`. На вход можно подать выход функции `time` из библиотеки `time.h` (см. справку C Library Reference от компилятора Open Watcom)

Решение

```
#include <stdio.h>  // Библиотека ввода-вывода
#include <stdlib.h> // Библиотека общего назначения
#include <time.h>   // Библиотека для работы
                    // со временем
void main()
{
  int i, t;
  // Сброс генератора случайных чисел
  t = time(NULL); // Получить текущее время
  srand(t);       // Сброс генератора
  /* Перебор чисел от генератора до появления
     числа, делящегося на 11 */
  for (i = rand(); i % 11 != 0; i = rand())
    printf("%d\n", i);

  printf("%d\n", i); // Вывод числа, остановившего
                     // цикл (делится на 11)
  getchar();         // Ожидание нажатия Enter
}
```

Пояснения к тексту программы

```
#include <stdio.h>  // Библиотека ввода-вывода
#include <stdlib.h> // Библиотека общего назначения
#include <time.h>   // Библиотека для работы
                    // со временем
```

Здесь подключаются все необходимые для работы программы библиотеки. Стандартная библиотека ввода-вывода `stdio.h` нам уже встречалась. `stdlib.h` – библиотека общего назначения (набор разных, но часто использующихся функций); `time.h` – библиотека для работы с датой и временем.

```
// Сброс генератора случайных чисел
t = time(NULL); // Получить текущее время
srand(t);       // Сброс генератора
```

В этих строках происходит инициализация генератора случайных чисел. Это нужно для того, чтобы он каждый раз выдавал разную последовательность случайных чисел.

Функция `time` из библиотеки `time.h` возвращает текущее время в секундах, прошедших с 00:00:00 1 января 1970 г.(по Гринвичу). На входе у неё – адрес области памяти, в которую записывается текущее время в виде набора из дня, месяца, года и т.д. Нам это не нужно, поэтому на входе – NULL, т.е. нет адреса (точнее, нулевой адрес).

Функция `srand` из библиотеки `stdlib.h` сбрасывает (инициализирует) генератор случайных чисел, используя при этом свой аргумент – число типа `int`. Функция не возвращает никаких значений.

```
for (i = rand(); i % 11 != 0; i = rand())
  printf("%d\n", i);
```

Тело цикла состоит из одного оператора (вызова функции `printf`), поэтому оно не заключено в фигурные скобки. Обратите внимание на выражения в скобках. Разберём их:

`i = rand()`	Получить случайное число и инициализировать им счётчик `i` (т.е. поместить его туда)
`i % 11 != 0`	Если значение `i` не делится на 11 (ненулевой остаток от деления), то продолжать цикл
`i = rand()`	Поместить в счётчик `i` новое случайное число

Задание 8**

Напишите программу для определения суммы целых чисел от 1 до 100. Используйте цикл `for`. Новое условие по сравнению с *заданием 4* – не пишите операторов в теле цикла, оно должно быть пустым. Существует по меньшей мере четыре способа решить задачу.

Указание: выясните, что такое префиксная и постфиксная форма операторов инкремента и декремента (есть в учебниках по Си, см. рекомендуемую литературу).

Решение:

```
/* Программа находит сумму
    целых чисел от 1 до 100 */
#include <stdio.h>

void main()
{ // Объявление переменных
  int i, sum;
  // Вычисление суммы
  for (i = 1, sum = 0; i <= 100; sum += i++);
  // Вывод результата на экран
  printf("sum = %d\n", sum);
  getchar();
}
```

Пояснения к тексту программы

```
for (i = 1, sum = 0; i <= 100; sum += i++);
```

В *выражении 1* происходит инициализация двух переменных – счётчика цикла i и переменной sum для хранения суммы первых 100 натуральных чисел. Условие в *выражение 2* ограничивает суммирование первыми 100 числами. Но вот на *выражении 3* остановимся подробнее.

```
sum += i++
```

Здесь применён **оператор инкремента в постфиксной форме** (выполняется после вычисления выражения). Запишем развёрнутый аналог этой строки:

Кратко	**Развёрнуто**	**Очень развёрнуто**
`sum += i++;`	`sum += i;` `i++;`	`sum = sum + i;` `i = i + 1;`

Можно использовать и другой вариант цикла:

```
for (i = 0, sum = 0; i < 100; sum += ++i);
```

В первых двух выражениях мало что изменилось, разве что в счётчик помещается значение 0, а условие `i<=100` заменено на `i<100`. Эти изменения связаны с изменениями в *выражении 3*.

```
sum += ++i
```

Здесь применён **оператор инкремента в префиксной форме** (выполняется до вычисления выражения). Запишем развёрнутый аналог этой строки:

Кратко	**Развёрнуто**	**Очень развёрнуто**
`sum += ++i;`	`i++;` `sum += i;`	`i = i + 1;` `sum = sum + i;`

Точка с запятой сразу после for и круглых скобок при нём означает то, что тело цикла - пустое. На практике это не очень часто используется – если увидите такое в реальной программе, то проверьте, не ошибка ли это!

Оператор декремента тоже существует как в **префиксной**, так и в **постфиксной** форме. Это даёт ещё три варианта цикла (хотя в последнем варианте есть одна лишняя итерация):

```
for (i = 100, sum = 0; i >= 1; sum += i--);
for (i = 101, sum = 0; i >= 2; sum += --i);
for (i = 101, sum = 0; i >= 1; sum += --i);
```

Внимание! Хотя язык Си предоставляет очень много способов для компактной записи выражений, ни в коем случае не пользуйтесь ими в ущерб понятности и читаемости кода программы! Лучше набрать лишнюю строку-другую, но не запутывать программу. Именно из-за возможности написания запутанных арифметических выражений язык Си прозвали «Write Only» (т.е. написать можно, прочесть и понять нельзя ☺) и считают подходящим для написания программ-анекдотов.

Но и отказываться от сокращений не нужно: если их правильно и умеренно использовать, то программа будет наоборот понятнее и читабельнее. Например, в типовом цикле `for` вместо `i=i+1` пишут `i++`. Даже префиксная/постфиксная форма операторов инкремента/декремента бывают полезными и облегчают понимание.

Упражнения

1. Написать программу, выводящую нечётные числа от 1 до 100
2. Вычислить факториал числа n; n!=1*2*3*…*n (n вводить с клавиатуры)
3. Написать программу вывода квадратов целых чисел от 1 до N
4. Написать программу вывода квадратных корней целых чисел от 1 до N (используйте функцию `sqrt` из библиотеки `math.h` для вычисления квадратного корня)
5. Написать программу получения в порядке убывания всех делителей данного числа
6. Вывести все члены геометрической прогрессии: 3, 9, 27, 81, … 59049
7. На первую клетку шахматной доски помещают одно зерно, на вторую – два, на третью – четыре, на четвёртую – восемь, на пятую – шестнадцать и т.д. (всего 64 клетки). Сколько всего зёрен будет на шахматной доске? **Указание**: используйте переменные типа `double` – переменные типа `int` не рассчитаны на хранения столь больших чисел и переполнятся. Результат будет приблизительным: значащими будут только 15-17 цифр. Если захотите увидеть число в стандартном виде, то используйте спецификатор `%e` вместо `%lf`. (Ответ - 18 446 744 073 709 551 615)

Упражнения повышенной сложности

8. Написать программу вычисления факториала числа (см. упр. 2) и использованием постфиксной формы оператора инкремента. Тело цикла должно быть пустым (см. задание 8 в качестве примера)
9. Перепишите задание 6 и использованием префиксной/постфиксной форм операторов инкремента/декремента
10. Напишите программу, которая печатает два столбца цифр ($1024=2^{10}$) и которая содержит один оператор в теле единственного цикла `for`:

```
1024 1
512  2
256  4
...
2    512
1    1024
```

Резюме

Новым в этом уроке является цикл **цикл for**.

```
for (i = 1; i <= 10; i++)
{
  printf("%d ", i);
}
```

Он состоит из трёх частей:

1. Ключевого слова `for`
2. Трёх выражений в круглых скобках, разделённых точкой с запятой.
3. Тела цикла (операторы между фигурными скобками `{ }`).

Шаг	Выражение	Что делать
1	`i = 1`	Выполнить *выражение 1* (поместить число 1 в переменную `i`. `i` называется счётчиком цикла)
2	`i <= 10`	Проверить условие в *выражении 2*. Если выполняется – перейти к пункту 3, иначе – выйти из цикла Если значение переменной `i` превысит 10, то цикл завершится.

3	`{`
	`  printf("%d ", i);`
	`}`
		Выполнить *тело цикла*
		Вывести переменную `i` (счётчик цикла) на экран

4	`i++`	Выполнить *выражение 3* Увеличить значение переменной `i` на единицу (`i++` - сокращение от `i=i+1`)
5		Перейти к шагу 2

В общем виде цикл `for` выглядит таким образом:

```
for (выражение1; выражение2; выражение3)
{
  /* Тело цикла */
}
```

Арифметические операторы-сокращения

Оператор	Развёрнутый вариант
`i++;` **(оператор инкремента)**	`i = i + 1;`
`i--;` **(оператор декремента)**	`i = i - 1;`
`i += a;`	`i = i + a;`
`i -= a;`	`i = i - a;`
`i *= a;`	`i = i * a;`
`i /= a;`	`i = i / a;`
`i %= a;`	`i = i % a;`

Префиксная и постфиксная формы операторов инкремента и декремента

Постфиксная форма оператора инкремента

Кратко	Развёрнуто	Очень развёрнуто
`sum += i++;`	`sum += i;` `i++;`	`sum = sum + i;` `i = i + 1;`

Префиксная форма оператора инкремента

Кратко	Развёрнуто	Очень развёрнуто
`sum += ++i;`	`i++;` `sum += i;`	`i = i + 1;` `sum = sum + i;`

Постфиксная форма оператора декремента

Кратко	Развёрнуто	Очень развёрнуто
`sum += i--;`	`sum += i;` `i--;`	`sum = sum + i;` `i = i - 1;`

Префиксная форма оператора декремента

Кратко	Развёрнуто	Очень развёрнуто
`sum += --i;`	`i--;` `sum += i;`	`i = i - 1;` `sum = sum + i;`