

**Лекция 5.**  
**Цикл for.**  
**Break и continue**

# Цикл for

- Очень удобный вариант цикла для прохода по диапазону чисел
- **Синтаксис:**
- **for** (инициализация; лог. выражение; модификация)  
инструкция
- Инструкция как обычно – это одна команда или блок кода

# Цикл for

- **for** (инициализация; лог. выражение; модификация) инструкция
- **Инициализация** – установка начальных значений счетчиков. Выполняется 1 раз при входе в цикл
- **Логическое выражение** – условие продолжения цикла. Проверяется перед началом каждой итерации (аналогично другим циклам)
- **Модификация** – изменение значений счетчиков. Выполняется после каждой итерации

# Пример цикла for

- ```
for (int i = 0; i <= 100; i++)  
{  
    // код  
}
```
- **Инициализация** – создается целый счетчик *i* с начальным значением 0
- Условие продолжения цикла – если  $i \leq 100$
- После каждой итерации выполняется **модификация** – счетчик увеличивается на 1
- После этого снова проверяется условие  $i \leq 100$ , потом выполняется следующая итерация и т.д. пока условие не нарушится

# Эквивалентность с while

- Цикл `for` эквивалентен следующему коду:
- инициализация;  
`while` (логическое выражение)  
{  
    инструкция;  
    модификация;  
}

# Задача

- Распечатать числа от 0 до 100 при помощи цикла `for`
- Распечатать числа от 7 до 121 при помощи цикла `for`
- Распечатать числа от 100 до 30 при помощи цикла `for`

# Необязательность частей `for`

- В объявлении цикла можно указать инициализацию, условие или модификацию пустыми
- Но все равно нужно поставить две точки с запятой внутри круглых скобок цикла `for`

# Примеры циклов без некоторых частей

- **Бесконечный цикл**

```
for (;;)
{
    // код
}
```

- **Цикл без инициализации**

```
for (; i < 100; i++)
{
    // код
}
```



# Цикл с несколькими счетчиками

- ```
for (int i = 0, j = 5; i < 100; i += 3, j--)  
{  
    // код  
}
```

# Вкладывание конструкций друг в друга

- Циклы и конструкции `if` могут вкладываться друг в друга любым образом
- Например, данный цикл выводит все четные числа от 1 до 100 включительно
- ```
for (int i = 1; i <= 100; i++)  
{  
    if (i % 2 == 0)  
    {  
        Console.WriteLine(i);  
    }  
}
```

# Объявление переменной внутри цикла

- ```
for (int i = 0; i < 100; i++)  
{  
    // код  
}
```
- Переменная `i` видна только внутри цикла
- Попытка обращения к ней снаружи приводит к ошибке компиляции

# Счетчик из существующей переменной

- В качестве счетчика можно использовать и существующую переменную
- Тогда она будет доступна и после цикла
- ```
int i;  
for (i = 0; i < 100; i++)  
{  
    // код  
}
```

# Задачи

- Распечатать числа, кратные четверке от 1 до 100, причем в обратном порядке (то есть, начиная с 100). Использовать цикл `for`
- Распечатать квадраты чисел от 1 до  $n$  при помощи цикла `for`, где  $n$  – задается в коде
- Например, при  $n = 4$ : 1, 4, 9, 16
- Решить задачу про среднее арифметическое чисел от  $x$  до  $y$  при помощи цикла `for`

# Оператор break

- Часто бывает нужно досрочно прекратить цикл
- Например, у нас такая задача: найти в наборе чисел первое отрицательное число
- Когда мы находим первое такое число, то нам уже нет смысла продолжать цикл дальше, хотелось бы выйти из него
- Это хорошо скажется на производительности, не нужно продолжать ненужные итерации
- Для этого используется оператор **break**

# Оператор break

- Оператор `break` немедленно завершает исполнение самого внутреннего цикла, внутри которого он находится
- ```
for (int i = 0; i < 100; i++)  
{  
    if (i >= 10)  
    {  
        break;  
    }  
    Console.WriteLine(i);  
}
```

# Оператор break

- Оператор `break` немедленно завершает исполнение самого внутреннего цикла, внутри которого он находится
- `break` можно применять для циклов любого типа

- ```
for (int i = 0; i < 100; i++)  
{  
    for (int j = 0; j < 100; j++)  
    {  
        if (j >= 10)  
        {  
            break;  
        }  
        Console.WriteLine(j);  
    }  
    // после break управление перейдет сюда  
}
```



# Задача

- Реализовать цикл `for`, который печатает числа от 1 до 100, и завершается если текущее число больше либо равно 30

# Оператор continue

- Часто бывает нужным завершить текущую итерацию цикла и перейти к следующей
- Например, когда в цикле есть итерации с особыми случаями
- См. следующий слайд

# Оператор continue

- Пример: хотим вывести все числа от 1 до 100, кроме числа 10
- ```
for (int i = 1; i <= 100; i++)  
{  
    if (i == 10)  
    {  
        continue;  
    }  
    Console.WriteLine(i);  
}
```

# Оператор continue

- Как и `break`, оператор `continue` можно применять в циклах любого вида, и он относится к ближайшему циклу, внутри которого находится
- Когда применяется `continue`, всегда проверяется условие выхода из цикла, даже для цикла `do-while`
- А для цикла `for` еще выполняется блок модификации счетчика

# Оператор continue для while

- ```
int i = 1;
while (i <= 100)
{
    if (i == 10)
    {
        ++i; // продублировали модификацию
        continue;
    }
    ++i;
}
```
- Для цикла `while` и `do-while` при `continue` не выполняется модификация счетчика, поэтому надо вставлять этот код самым перед `continue`

# Задача

- Распечатать числа от 0 до 100, кроме числа 5, кроме всех чисел, кратных 3 и кроме чисел из диапазона от 60 до 80 включительно
- \* Сделать версию с одним `if`'ом
- \* Переписать при помощи `while`

# Как выйти из двух циклов

- Как выйти сразу из двух циклов?
- Нужно завести булеву переменную, которой будем присваивать `true` перед `break` из внутреннего цикла
- А потом проверим значение переменной во внешнем цикле и, если надо, сделаем `break` и там

# Как выйти из двух циклов

- ```
for (int i = 0; i < 100; ++i)
{
    bool needBreak = false;
    for (int j = 0; j < 100; ++j)
    {
        if (i == j)
        {
            needBreak = true;
            break;
        }
    }
    if (needBreak)
    {
        break;
    }
}
```

Завели булеву переменную, присвоили ей false

Перед break присвоили ей true

После выхода из внутреннего цикла, проверяем надо ли прервать и внешний цикл, и делаем это, если нужно

Как реализовать continue внешнего цикла из внутреннего цикла?



# Область видимости переменных

- Переменные, объявленные внутри функции, называются **локальными переменными**
- У переменных есть **области видимости** – часть кода внутри функции, откуда можно обратиться к этой переменной
- Область видимости переменной зависит от вложенности блока, в котором она объявлена

# Область видимости переменных

- Пусть у нас есть функция, в которой есть различные вложенные блоки: ветвления и циклы
- ```
int x = 10; // не объявлена внутри блока
if (x == 5)
{
    int a = 4; // объявлена внутри блока if
    for (int i = 0; i < 10; i++)
    {
        // переменная i объявлена внутри блока for
    }
}
else
{
    int a = 5; // объявлена внутри блока else
}
```

# Область видимости переменных

- Переменная будет видна в том блоке, в котором она объявлена, и в блоках, которые вложены в этот блок
- Например, переменная `x` видна везде в этом коде

- `int x = 10; // не объявлена внутри блока`  
`if (x == 5)`  
`{`  
 `int a = 4; // объявлена внутри блока if`  
 `for (int i = 0; i < 10; i++)`  
 `{`  
 `// переменная i объявлена внутри блока for`  
 `}`  
`}`  
`else`  
`{`  
 `int a = 5; // объявлена внутри блока else`  
`}`

# Область видимости переменных

- Переменная `i` видна только внутри блока цикла `for`
- Одна переменная `a` видна внутри блока `if` и цикле `for`
- Вторая переменная `a` видна только внутри блока `else`
- `int x = 10; // не объявлена внутри блока`

```
if (x == 5)
```

```
{
```

```
    int a = 4; // объявлена внутри блока if
```

```
    for (int i = 0; i < 10; i++)
```

```
    {
```

```
        // переменная i объявлена внутри блока for
```

```
    }
```

```
}
```

```
else
```

```
{
```

```
    int a = 5; // объявлена внутри блока else
```

```
}
```

# Область видимости переменных

- В одной области видимости не должно быть двух переменных с одинаковыми именами
- Например, во всем этом коде нельзя объявить еще одну переменную с именем `x`

# Область видимости переменных

- Если области видимости не пересекаются, то можно иметь переменные с одинаковыми именами, как в случае с переменными `a` – блоки, в которых они объявлены, не вложены друг в друга
- ```
int x = 10; // не объявлена внутри блока
if (x == 5)
{
    int a = 4; // объявлена внутри блока if
    for (int i = 0; i < 10; i++)
    {
        // переменная i объявлена внутри блока for
    }
}
else
{
    int a = 5; // объявлена внутри блока else
}
```

# Область видимости переменных - итог

- Переменная видна с момента объявления до конца блока, где она была объявлена
- В одной области видимости не должно быть двух переменных с одинаковым именем
- В непересекающихся областях видимости могут быть переменные с одним именем

# Область видимости - рекомендации

- Область видимости следует делать как можно меньшей, тогда код проще читать, понимать и отлаживать
- Все переменные объявляем в месте первого присваивания. Если области видимости недостаточно, то объявляем выше, пока не найдём нужное место
- ```
int max;  
if (a > b) {  
    max = a;  
} else {  
    max = b;  
}
```



# Область видимости - рекомендации

- Но не следует делать меньшую область видимости если это портит производительность

# Задача на дом «Break»

- В программе должна быть некоторая загаданная фиксированная строка
- Далее программа предлагает пользователю ввести строку, пользователь вводит
- Если введена та загаданная строка, то программа должна завершаться
- Иначе пользователю дается следующая попытка для ввода и т.д., пока не введет правильно. При этом каждый раз пользователю должно выдаваться приглашение для ввода
- В этой задаче используйте бесконечный цикл и **break**

# Задача на дом «Простые числа»

- Прочитать с консоли целое число
- Найти и распечатать все простые числа, не превышающие введенное число

# Задача на курс «Алгоритм Евклида»

- Для нахождения наибольшего общего делителя двух чисел удобно использовать алгоритм Евклида:

$$\text{НОД}(a, b) = \begin{cases} b, & \text{если } a \% b = 0 \\ \text{НОД}(b, a \% b) & \text{иначе,} \end{cases}$$

- Реализовать вычисление НОД алгоритмом Евклида
- Использовать цикл

# Задача на курс «Таблица умножения»

- Вывести в консоль таблицу умножения чисел от 1 до 10 при помощи циклов `for`
- Но программа должна работать верно и если попросят таблицу от 1 до другого числа
- Примерно так, только без границ клеток, консоль этого не позволяет
- Добейтесь чтобы числа были выровнены по столбцам (добейте числа нужным количеством пробелов)
- \* Добавьте «шапку» таблицы, отделите ее символами, например, | и --

|    | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   | 10  | 11  | 12  |
|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|
| 1  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   | 10  | 11  | 12  |
| 2  | 2  | 4  | 6  | 8  | 10 | 12 | 14 | 16 | 18  | 20  | 22  | 24  |
| 3  | 3  | 6  | 9  | 12 | 15 | 18 | 21 | 24 | 27  | 30  | 33  | 36  |
| 4  | 4  | 8  | 12 | 16 | 20 | 24 | 28 | 32 | 36  | 40  | 44  | 48  |
| 5  | 5  | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45  | 50  | 55  | 60  |
| 6  | 6  | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54  | 60  | 66  | 72  |
| 7  | 7  | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63  | 70  | 77  | 84  |
| 8  | 8  | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72  | 80  | 88  | 96  |
| 9  | 9  | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81  | 90  | 99  | 108 |
| 10 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90  | 100 | 110 | 120 |
| 11 | 11 | 22 | 33 | 44 | 55 | 66 | 77 | 88 | 99  | 110 | 121 | 132 |
| 12 | 12 | 24 | 36 | 48 | 60 | 72 | 84 | 96 | 108 | 120 | 132 | 144 |

# Задача на курс «Вклад»

- Напишите программу, которая рассчитывает сумму банковского вклада с заданной ставкой % годовых на заданное число месяцев
- Вклад с капитализацией – считаем, что после истечения каждого месяца к сумме вклада прибавляется начисленный за месяц процент от суммы вклада на начало месяца
- Также распечатать прибыль