Карагандинский технический университет имени Абылкаса Сагинова
Кафедра «Кибербезопасность и искусственный интеллект»

СЛАЙД-ЛЕКЦИЯ

Тема: «Операторы языка C#»

Дисциплина: Программирование на C#

Специальность: ВТиПО

Автор: Ст.преп. Сницарь Лилия Ринатовна

ПЛАН ЛЕКЦИИ

1. Арифметические операторы.
2. Операторы отношений и логические операторы.
3. Оператор присваивания.
4. Поразрядные операторы.
5. Операторы сдвига.
6. Тернарный оператор.
7. Приоритет операторов.

1. Арифметические операторы

В C# имеется четыре общих класса операторов:

1. арифметические,
2. поразрядные,
3. логические и
4. операторы отношений.

Помимо этого есть оператор *присвоения* и оператор ***?***.

В C# определены также операторы для обработки специальных ситуаций, но их мы рассмотрим после изучения средств, к которым они применяются.

1. Арифметические операторы

В C# определены следующие арифметические операторы

Оператор	Действие
+	Сложение
-	Вычитание, унарный минус
*	Умножение
/	Деление
%	Деление по модулю
--	Декремент
++	Инкремент

1. Арифметические операторы

Пример:

```
int num1, num2;
float f1, f2;

num1 = 13 / 3;
num2 = 13 % 3;

f1 = 13.0f / 3.0f;
f2 = 13.0f % 3.0f;

Console.WriteLine("Результат и остаток от деления 13 на 3: {0} __ {1}", num1, num2);
Console.WriteLine("Результат деления 13.0 на 3.0: {0:#.###} {1}", f1, f2);

Console.ReadLine();
```

```
Результат и остаток от деления 13 на 3: 4 __ 1
Результат деления 13.0 на 3.0: 4,333 1
```

1. Арифметические операторы

Инкремент и декремент

Операторы инкремента (++) и декремента (--) увеличивают и уменьшают значение операнда на единицу, соответственно.

Итак, оператор инкремента выполняет **сложение** операнда с числом **1**, а оператор декремента **вычитает 1** из своего операнда.

Это значит, что инструкция:

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

аналогична такой инструкции:

```
x++;.
```

1. Арифметические операторы

Операторы инкремента и декремента могут стоять как перед своим операндом, так и после него.

Например, инструкцию

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

можно переписать в виде префиксной формы:

```
++x;
```

или в виде постфиксной формы:

```
x++;.
```

1. Арифметические операторы

Пример:

```
short d = 1;

for (byte i = 0; i < 10; i++)
    Console.Write(i + d++ + "\t");

Console.WriteLine();
d = 1;

for (byte i = 0; i < 10; i++)
    Console.Write(i + ++d + "\t");

Console.ReadLine();
```

```
1       3       5       7       9       11      13      15      17      19
2       4       6       8       10      12      14      16      18      20
```

2. Операторы отношений и логические операторы

В обозначениях оператор отношения и логический оператор термин отношения означает взаимосвязь, которая может существовать между двумя значениями, а термин логический – взаимосвязь между логическими значениями «ИСТИНА» и «ЛОЖЬ».

И поскольку операторы отношения дают истинные или ложные результаты, то они нередко применяются вместе с логическими операторами. Именно по этой причине они и рассматриваются совместно.

2. Операторы отношений и логические операторы

Операторы отношения

Оператор	Значение
==	Равно
!=	Не равно
>	Больше
<	Меньше
>=	Больше или равно
<=	Меньше или равно

2. Операторы отношений и логические операторы

Логические операторы, в порядке убывания приоритета:

Оператор	Значение
!	**НЕ** (логическое отрицание)
&	логическое **И**
^	логическое **исключающее ИЛИ**
|	логическое **ИЛИ**
&&	**Условный оператор логического И** (сокращенное И)
||	У**словный оператор логического ИЛИ** (сокращенное ИЛИ)

2. Операторы отношений и логические операторы

В C# на равенство или неравенство можно сравнивать (соответственно, с помощью операторов == и !=) все объекты.

Но такие операторы сравнения, как <, >, <= или >=, можно применять только к типам, которые поддерживают отношения упорядочения.

Это значит, что все операторы отношений можно применять ко всем числовым типам.

Однако значения типа bool можно сравнивать только на равенство или неравенство, поскольку значения true и false не упорядочиваются. Например, в C# сравнение true > false не имеет смысла.

2. Операторы отношений и логические операторы

Что касается логических операторов, то их операнды должны иметь тип bool, и результат логической операции всегда будет иметь тип bool. Логические операторы &, |, ^ и ! выполняют базовые логические операции И, ИЛИ, исключающее ИЛИ и НЕ в соответствии со следующей таблицей истинности.

x	y	x & y	x | y	x ^ y	!x
false	false	false	false	false	true
true	false	false	true	true	false
false	true	false	true	true	true
true	true	true	true	false	false

2. Операторы отношений и логические операторы

Пример:

```
int i, j;
bool b1, b2;
i = 10;
j = 11;
if (i < j) Console.WriteLine("i < j");
if (i <= j) Console.WriteLine("i <= j");
if (i != j) Console.WriteLine("i != j");
if (i == j) Console.WriteLine("Это не будет выполнено.");
if (i >= j) Console.WriteLine("Это не будет выполнено.");
if (i > j) Console.WriteLine("Это не будет выполнено.");
b1 = true;
b2 = false;
if (b1 & b2) Console.WriteLine("Это не будет выполнено.");
if (!(b1 & b2)) Console.WriteLine("!(b1 & b2) -- ИСТИНА");
if (b1 | b2) Console.WriteLine("b1 | b2 -- ИСТИНА");
if (b1 ^ b2) Console.WriteLine("b1 ^ b2 -- ИСТИНА");
```

```
i < j
i <= j
i != j
!(b1 & b2) -- ИСТИНА
b1 | b2 -- ИСТИНА
b1 ^ b2 -- ИСТИНА
```

2. Операторы отношений и логические операторы

Так, C# поддерживает набор логических операторов, на базе которых можно построить любую другую логическую операцию, например операцию *импликации*.

Импликация – это логическая операция, результат которой будет ложным только в случае, когда левый операнд имеет значение ИСТИНА, а правый – ЛОЖЬ. (Операция импликации отражает идею о том, что истина не может подразумевать ложь.)

2. Операторы отношений и логические операторы

Таблица истинности для оператора импликации:

x	y	Импликация x и y
true	true	true
true	false	false
false	false	true
false	true	true

Операцию импликации можно создать, используя комбинацию операторов ! и |.

```
!x | y
```

2. Операторы отношений и логические операторы

Сокращенные логические операторы

C# поддерживает специальные ***сокращенные*** (short-circuit) версии логических операторов И и ИЛИ, которые можно использовать для создания более эффективного кода.

Если в операции И (&&) один операнд имеет значение ЛОЖЬ, результат будет ложным независимо от того, какое значение имеет второй операнд. А если в операции ИЛИ (||) один операнд имеет значение ИСТИНА, результат будет истинным независимо от того, какое значение имеет второй операнд.

Таким образом, в этих двух случаях вычислять второй операнд не имеет смысла. Если не вычисляется один из операндов, тем самым экономится время и создается более эффективный код.

2. Операторы отношений и логические операторы

Сокращенный оператор И обозначается символом &&, а сокращенный оператор ИЛИ – символом || (их обычные версии обозначаются одинарными символами **&** и **|**, соответственно).

Единственное различие между обычной и сокращенной версиями этих операторов состоит в том, что при использовании обычной операции всегда вычисляются оба операнда, в случае же сокращенной версии второй операнд вычисляется только при необходимости.

2. Операторы отношений и логические операторы

```
bool t = true;
int i = 0;
// Использование обычного оператора |
if (t | (++i < 5))
    Console.WriteLine("i равно {0}", i); // оба операнда вычисляются: t = true,
                                         // (++i < 5) = true, i = 1
i = 0;
// Использование укороченного оператора ||
// Здесь второй операнд (++i < 5) не вычисляется,
// так как первый операнд t = true уже делает результат всего выражения true
if (t || (++i < 5))
    Console.WriteLine("i равно {0}", i); // t = true и второй операнд
                                         // не вычисляется, i остается 0
```

2. Операторы отношений и логические операторы

Сокращенный оператор И (&&) также называется ***условным И***, а сокращенный ИЛИ (||) – ***условным ИЛИ***.

3. Оператор присваивания

Оператор присваивания обозначается одинарным знаком равенства (=). Его роль в языке C# во многом такая же, как и в других языках программирования. Общая форма записи оператора присваивания имеет следующий вид:

переменная = выражение;

Здесь тип элемента переменная должен быть совместим с типом элемента выражение.

Оператор присваивания интересен тем, что позволяет создавать целую цепочку присвоений.

3. Оператор присваивания

Рассмотрим, например, следующий фрагмент кода.

```
int x, y, z;
x = y = z = 100; // Устанавливаем переменные x, y
                 // и z равными 100.
```

3. Оператор присваивания

В C# предусмотрены специальные составные операторы присваивания, которые упрощают программирование определенных инструкций присваивания.

Рассмотрим следующую инструкцию:

```
x = x + 10;
```

Используя составной оператор присваивания, ее можно переписать в таком виде:

```
x += 10;
```

3. Оператор присваивания

Пара операторов += служит указанием компилятору присвоить переменной x сумму текущего значения переменной x и числа 10.

Инструкция

```
x = x - 100;
```

аналогична такой:

```
x -= 100;
```

Обе эти инструкции присваивают переменной x ее прежнее значение, уменьшенное на 100.

3. Оператор присваивания

Составные версии операторов присваивания существуют для всех бинарных операторов (т.е. для всех операторов, которые работают с двумя операндами).

Общая форма их записи такова:

переменная op = выражение;

Здесь элемент ***op*** означает конкретный арифметический или логический оператор, объединяемый с оператором присваивания.

3. Оператор присваивания

Возможны следующие варианты объединения операторов.

+=	**-=**	***=**	**/=**
%=	**&=**	**|=**	**^=**

Поскольку составные операторы присваивания выглядят короче своих несоставных эквивалентов, то составные версии часто называют ***укороченными операторами присваивания***.

Составные операторы присваивания обладают двумя заметными достоинствами.

Во-первых, они компактнее своих эквивалентов. Во-вторых, их наличие приводит к созданию более эффективного кода (поскольку операнд в этом случае вычисляется только один раз).

4. Поразрядные операторы

Поразрядные операторы действуют непосредственно на разряды своих операндов.

Они определены только для целочисленных операндов и не могут быть использованы для операндов типа `bool`, `float` или `double`.

Поразрядные операторы предназначены для тестирования, установки или сдвига битов (разрядов), из которых состоит целочисленное значение.

Поразрядные операторы очень часто используются для решения широкого круга задач программирования системного уровня, например, при опросе информации о состоянии устройства или ее формировании.

4. Поразрядные операторы

Поразрядные операторы

Оператор	Значение
&	Поразрядное И (AND)
|	Поразрядное ИЛИ (OR)
^	Поразрядное исключающее ИЛИ (XOR)
>>	Сдвиг вправо
<<	Сдвиг влево
~	Дополнение до 1 (унарный оператор НЕ)

4. Поразрядные операторы

Поразрядные операторы И, ИЛИ, исключающее ИЛИ и НЕ обозначаются символами &, |, ^ и ~, соответственно.

Они выполняют те же операции, что и их логические эквиваленты, описанные выше. Различие состоит лишь в том, что поразрядные операции работают на побитовой основе.

4. Поразрядные операторы

x	y	x & y	x | y	x ^ y	~x
0	0	0	0	0	1
1	0	0	1	1	0
0	1	0	1	1	1
1	1	1	1	0	0

4. Поразрядные операторы

С точки зрения наиболее распространенного применения поразрядную операцию И можно рассматривать как способ подавления отдельных двоичных разрядов.

Это означает, что если какой-нибудь бит в любом из операндов равен 0, то соответствующий бит результата будет сброшен в 0.

```
  1101 0011
  1010 1010
& ———————
  1000 0010
```

4. Поразрядные операторы

```
ushort num;
ushort i;
for (i = 1; i <= 10; i++)
{
    num = i;
    Console.WriteLine("num: " + num);
    num = (ushort)(num & 0xFFFE); // num & 1111 1111 1111 1110
    Console.WriteLine("num после сброса младшего бита: " + num + "\n");
}
```

```
num: 1
num после сброса младшего бита: 0

num: 2
num после сброса младшего бита: 2

num: 3
num после сброса младшего бита: 2

num: 4
num после сброса младшего бита: 4

num: 5
num после сброса младшего бита: 4

num: 6
num после сброса младшего бита: 6

num: 7
num после сброса младшего бита: 6

num: 8
num после сброса младшего бита: 8

num: 9
num после сброса младшего бита: 8

num: 10
num после сброса младшего бита: 10
```

4. Поразрядные операторы

Значение `0xFFFE`, используемое в этой программе, в двоичном коде представляется числом 1111 1111 1111 1110.

Таким образом, операция `num & 0xFFFE` оставляет все биты неизменными за исключением младшего, который устанавливается в нуль.

Поэтому любое четное число, пройдя через эту операцию, остается четным, а любое нечетное «выходит» из него уже четным (за счет уменьшения на единицу).

4. Поразрядные операторы

Оператор И также используется для определения значения разряда. Например, следующая программа определяет, является ли заданное число нечетным.

```
num = 10;
if((num & 1) == 1)
   Console.WriteLine("Этот текст не будет отображен.");

num = 11; if((num & 1) == 1)
   Console.WriteLine(num + " – нечетное число.");
```

4. Поразрядные операторы

Возможности поразрядного тестирования, которые предоставляет поразрядный оператор &, можно использовать для создания программы, которая отображает значение типа byte в двоичном формате.

Рассмотрим один из возможных вариантов решения этой задачи.

```
int t;
byte val;
val = 123; //0111 1011
for(t=128; t> 0; t = t/2)//1000 0000, 0100 0000, 0010 0000...
{
    if((val & t) != 0) Console.Write ("1 ");
    if((val & t) == 0) Console.Write ("0 ") ;
}
```

4. Поразрядные операторы

В цикле for с помощью поразрядного оператора **И** последовательно тестируется каждый бит переменной val.

Если оказывается, что этот бит установлен, отображается цифра 1, в противном случае – цифра 0.

4. Поразрядные операторы

Поразрядный оператор ИЛИ, в противоположность поразрядному И, удобно использовать для установки нужных битов в единицу.

При выполнении операции ИЛИ наличие в операнде бита, равного 1, означает, что в результате соответствующий бит также будет равен единице.

Пример:

```
  1101 0011
  1010 1010
| ———————————
  1111 1011
```

4. Поразрядные операторы

Поразрядное исключающее ИЛИ (XOR) устанавливает в единицу бит результата только в том случае, если соответствующие биты операндов отличаются один от другого, т.е. не равны.

Пример:

```
  0111  1111
  1011 1001
^ ____________
  1100  0110
```

4. Поразрядные операторы

Оператор XOR обладает одним интересным свойством, которое позволяет использовать его для кодирования сообщений.

Если выполнить операцию XOR между значением X и значением Y, а затем снова выполнить операцию XOR между результатом первой операции и тем же значением Y, получим исходное значение X.

Это значит, что после выполнения двух операций:

```
R1 = X ^ Y;
R2 = R1 ^ Y;
```

значение R2 совпадет со значением x.

5. Операторы сдвига

В C# определены следующие операторы поразрядного сдвига:

```
<< сдвиг влево;
>> сдвиг вправо.
```

Общий формат записи этих операторов такой:

значение << число_битов;
значение >> число_битов.

Здесь *значение* – это объект операции сдвига, а элемент *число_битов* указывает, на сколько разрядов должно быть сдвинуто значение.

5. Операторы сдвига

При сдвиге влево на один разряд все биты, составляющее значение, сдвигаются влево на одну позицию, а в младший разряд записывается нуль.

При сдвиге вправо все биты сдвигаются, соответственно, вправо. Если сдвигу вправо подвергается значение без знака, в старший разряд записывается нуль.

Если же сдвигу вправо подвергается значение со знаком, значение знакового разряда сохраняется.

Отрицательные целые числа представляются установкой старшего разряда числа равным единице.

Таким образом, если сдвигаемое значение отрицательно, при каждом сдвиге вправо в старший разряд записывается единица, а если положительно – нуль.

5. Операторы сдвига

При сдвиге как вправо, так и влево крайние биты теряются. Следовательно, при этом выполняется нециклический сдвиг, и содержимое потерянного бита узнать невозможно.

Рассмотрим пример. Значение, которое будет сдвигаться, устанавливается сначала равным единице, т.е. только младший разряд этого значения "на старте" равен 1, все остальные равны 0.

После выполнения каждого из восьми сдвигов влево программа отображает младшие восемь разрядов нашего "подопытного" значения. Затем описанный процесс повторяется, но в зеркальном отображении.

На этот раз перед началом сдвига в переменную val заносится не 1, а 128, что в двоичном коде представляется как 1000 0000. И, конечно же, теперь сдвиг выполняется не влево, а вправо.

5. Операторы сдвига

```
int val = 1;
int t, i;

for (i = 0; i < 8; i++)
{
    for (t = 128; t > 0; t = t / 2)
    {
        if ((val & t) != 0) Console.Write("1 ");
        if ((val & t) == 0) Console.Write("0 ");
    }
    Console.WriteLine();
    val = val << 1; // Сдвиг влево.
}
Console.WriteLine();

val = 128;
for (i = 0; i < 8; i++)
{
    for (t = 128; t > 0; t /= 2)
    {
        if ((val & t) != 0) Console.Write("1 ");
        if ((val & t) == 0) Console.Write("0 ");
    }
    Console.WriteLine();
    val = val >> 1; // Сдвиг вправо.
    }
```

```
0 0 0 0 0 0 0 1
0 0 0 0 0 0 1 0
0 0 0 0 0 1 0 0
0 0 0 0 1 0 0 0
0 0 0 1 0 0 0 0
0 0 1 0 0 0 0 0
0 1 0 0 0 0 0 0
1 0 0 0 0 0 0 0

1 0 0 0 0 0 0 0
0 1 0 0 0 0 0 0
0 0 1 0 0 0 0 0
0 0 0 1 0 0 0 0
0 0 0 0 1 0 0 0
0 0 0 0 0 1 0 0
0 0 0 0 0 0 1 0
0 0 0 0 0 0 0 1
```

5. Операторы сдвига

Пример:

```
4<<1
```

сдвигает число 4 (которое в двоичном представлении 100) на один разряд влево, то есть в итоге получается 1000 или число 8 в десятичном представлении.

```
16>>1
```

сдвигает число 16 (которое в двоичном представлении 10000) на один разряд вправо, то есть в итоге получается 1000 или число 8 в десятичном представлении.

6. Тернарный оператор

Одним из самых замечательных операторов C# является **тернарный оператор ?**.

Оператор `?` часто используется для замены определенных типов конструкций `if-then-else`. Оператор `?` называется тернарным, поскольку он ***работает с тремя операторами***.

Его общий формат записи имеет такой вид:

```
Выражение1 ? Выражение2 : Выражение3;
```

Здесь *Выражение1* должно иметь тип `bool`. Типы элементов *Выражение2* и *Выражение3* должны быть одинаковы. Обратите внимание на использование и расположение двоеточия.

6. Тернарный оператор

Значение *?-выражения* определяется следующим образом.

Вычисляется *Выражение1*.

Если оно оказывается **истинным**, вычисляется *Выражение2*, и результат его вычисления становится **значением** всего ***?-выражения***.

Если результат вычисления элемента *Выражение1* оказывается **ложным**, значением всего *?-выражения* становится результат вычисления элемента *Выражение3*.

6. Тернарный оператор

Рассмотрим пример, в котором переменной `abs_val` присваивается абсолютное значение переменной `val`.

```
abs_val = val < 0 ? -val : val; // Получаем абсолютное
                                // значение val.
```

Здесь переменной `abs_val` присваивается значение переменной `val`, если оно больше или равно нулю.

Если же значение переменной `val` отрицательно, переменной `abs_val` присваивается результат применения к ней операции "унарный минус", который будет представлять собой положительное значение.

6. Тернарный оператор

В этой программе выполняется деление числа 100 на разные числа, но попытка деления на нуль реализована не будет

```
int result, i;
for (i = -5; i < 6; i++)
{
    result = i != 0 ? 100 / i : 0;
    if (i != 0)
        Console.WriteLine("100 / " + i + " равно " + result);
}
```

```
100 / -5 равно -20
100 / -4 равно -25
100 / -3 равно -33
100 / -2 равно -50
100 / -1 равно -100
100 / 1 равно 100
100 / 2 равно 50
100 / 3 равно 33
100 / 4 равно 25
100 / 5 равно 20
```

7. Приоритет операторов

`( ) [ ] . ++`(постфиксный) `--`(постфиксный) **`checked new sizeof typeof unchecked`**
`! ~` ***Операторы приведения типа*** `+`(унарный) `-`(унарный) `++`(префиксный) `--`(префиксный)
`* / %`
`+ -`
`<< >>`
`< <= > >= is as`
`== !=`
`&`
`^`
`|`
`&&`
`||`
`? :`
`= op= (x+=)`

Контрольные вопросы

1. Какие две общие категории встроенных типов данных имеются в языке C#?
2. Что представляет собой переменная?
3. Что такое константа?
4. Перечислите особенности констант.
5. Что такое область видимости переменной?

Список литературы

1. Вагнер Б. C# Эффективное программирование М.: ЛОРИ, 2013. - 320 с.

2. Ишкова Э. А. Самоучитель C#. Начало программирования М.: Наука и техника, 2013. - 496 с.

3. Подбельский В. В. Язык C#. Базовый курс М.: Финансы и статистика, Инфра-М, 2011. - 384 с.

4. Троелсен Э. Язык программирования C# 2010 и платформа .NET4, М.: Москва: Огни, 2016. - 238 с.

ЛЕКЦИЯ ОКОНЧЕНА,
СПАСИБО ЗА ВНИМАНИЕ!