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

СЛАЙД-ЛЕКЦИЯ

Тема: «Типы данных, литералы и переменные»

Дисциплина: Программирование на C#

Специальность: ВТиПО

Автор: Ст.преп. Сницарь Лилия Ринатовна

ПЛАН ЛЕКЦИИ

1. Типы данных.
2. Литералы.
3. Переменные.
4. Преобразования типов.

1. Типы данных

Как и во многих языках программирования, в C# есть своя система типов данных, которая используется для создания переменных.

Тип данных определяет внутреннее представление данных, множество значений, которые может принимать объект, а также допустимые действия, которые можно применять над объектом.

1. Типы данных

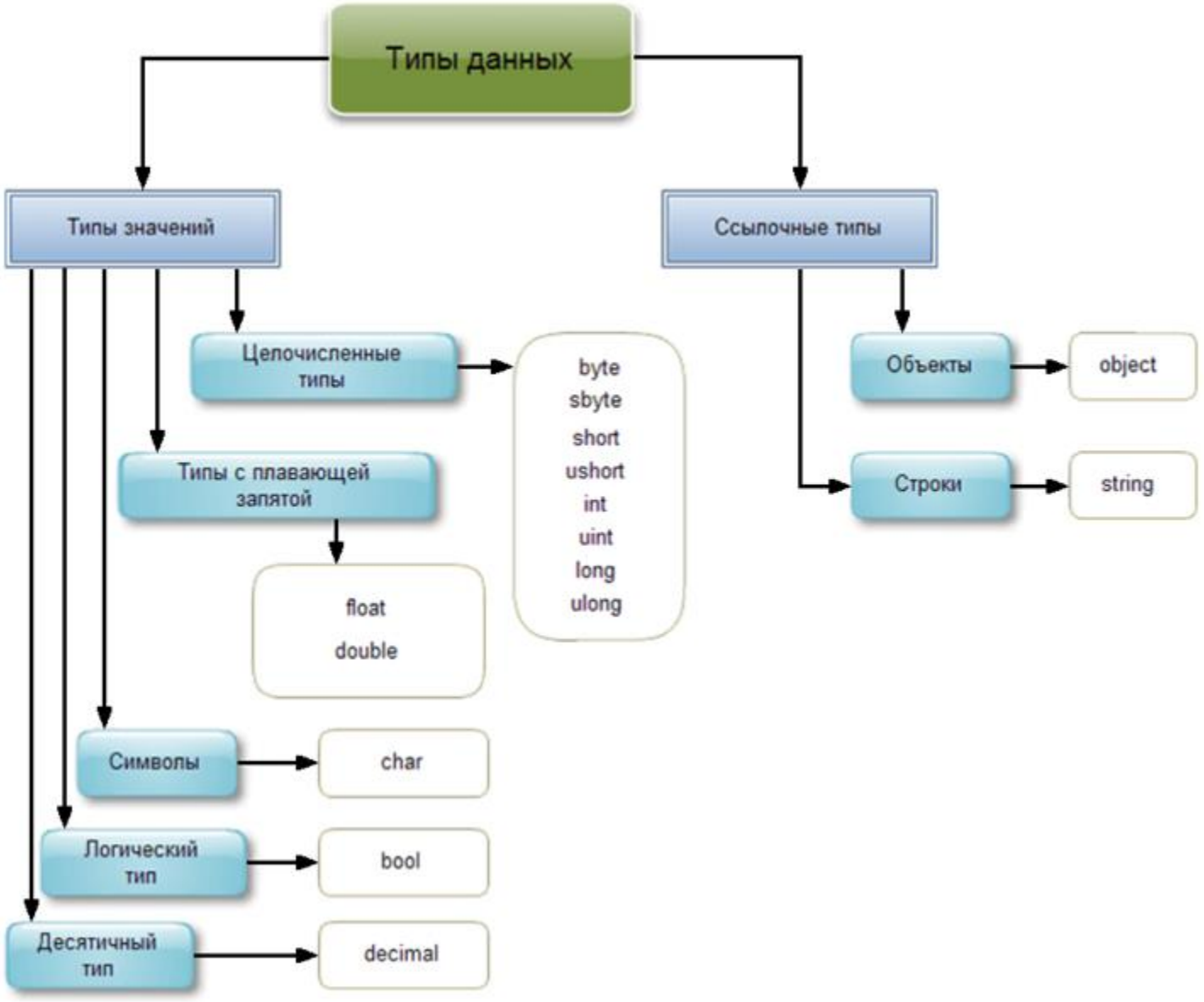
В C# имеются две общие категории встроенных типов данных: ***типы значений*** и ***ссылочные типы***.

Они отличаются по содержимому переменной.

Концептуально разница между ними состоит в том, что тип значения (value type) хранит данные непосредственно, в то время как ссылочный тип (reference type) хранит ссылку на значение.

Эти типы сохраняются в разных местах памяти: типы значений сохраняются в области, известной как *стек*, а ссылочные типы – в области, называемой *управляемой кучей*.

1. Типы данных



1. Типы данных

Таблица 1. Типы значений в C#

Тип в C#	Системный тип	Диапазон	CLS	Размер в битах
byte	System. Byte	0...255	да	8
sbyte	System.SByte	-128...127	нет	8
short	System.Int16	-32768…32767	да	16
ushort	System.UInt16	0…65535	нет	16
int	System.Int32	-2147483648…2147483647	да	32
uint	System.UInt32	0…4294967295	нет	32
long	System.Int64	-9223372036854775808… 9223372036854775807	да	64
ulong	System.UInt64	0…18446744073709551615	нет	64

1. Типы данных

Таблица 2. Другие типы данных в C#

Тип в C#	Системный тип	Диапазон	CLS	Размер в битах
bool	System. Boolean	true, false	да	булево значение
char	System.Char	U+0000...U+FFFF	да	16
float	System.Single	$-3.4*10^{38}…3.4*10^{38}$	нет	32
double	System.Double	$\pm 5.0*10^{-324}…\pm 1.7*10^{308}$	да	64
decimal	System.Decimal	$\pm 1.0*10^{-28}... \pm 7.9228*10^{28}$	да	128
string	System.String	Ограничен объемом доступной памяти	да	Ряд символов в кодировке Unicode
object	System.Object	Используется для хранения любого типа в памяти	да	Базовый класс для всех типов в .NET

1. Типы данных

При присвоении значений надо иметь в виду следующую тонкость: все вещественные литералы (дробные числа) рассматриваются как значения типа double.

И чтобы указать, что дробное число представляет тип float или тип decimal, необходимо к литералу добавлять суффикс: **F/f** – для float и **M/m** – для decimal.

```
float a = 3.14F;
float b = 30.6f;

decimal c = 1005.8M;
decimal d = 334.8m;
```

1. Типы данных

Подобным образом все целочисленные литералы рассматриваются как значения типа int.

Чтобы явным образом указать, что целочисленный литерал представляет значение типа uint, надо использовать суффикс U/u, для типа long – суффикс **L/l**, а для типа ulong – суффикс **UL/ul**:

```
uint a = 10U;

long b = 20L;
ulong c = 30UL;
```

1. Типы данных

char: хранит одиночный символ в кодировке Unicode и занимает 2 байта.

Этому типу соответствуют символьные литералы:

```
char a = 'A'; //символ
char b = '\x5A'; //шестнадцатеричная управляющая последовательность
char c = '\u0420'; //unicode
```

1. Типы данных

string: хранит набор символов Unicode.

Этому типу соответствуют строковые литералы.

```
string hello = "Hello";
string word = "world";
```

1. Типы данных

object: может хранить значение любого типа данных и занимает 4 байта на 32-разрядной платформе и 8 байт на 64-разрядной платформе.

```
object a = 22;
object b = 3.14;
object c = "hello code";
```

2. Литералы

В C# ***литералами*** называются фиксированные значения, представленные в понятной форме. Например, число 100 – это литерал. Литералы также называют константами.

C#-литералы могут иметь любой тип значений. Способ их представления зависит от их типа.

Символьные константы заключаются между двумя одинарными кавычками. Например, как 'a' так и ' % ' – символьные константы.

Целочисленные литералы задаются как числа без дробной части. Например, 10 и -100 – это целочисленные константы.

Константы с плавающей точкой должны обязательно иметь десятичную точку, а за ней – дробную часть числа. Примером константы с плавающей точкой может служить число 11.123.

2. Литералы

Управляющие последовательности символов

Обозначение	Описание	Обозначение	Описание
\a	Звуковой сигнал (alert)	**\v**	Вертикальная табуляция
\b	Возврат на одну позицию (backspace)	**\0**	Пустой символ
\f	Перевод страницы (переход на новую страницу)	**\'**	Одинарная кавычка (апостроф)
\n	Новая строка (перевод строки)	**\"**	Двойная кавычка
\r	Возврат каретки	****	Обратный слэш
\t	Горизонтальная табуляция		

2. Литералы

Язык C# поддерживает еще один тип литерала: **строковый**.

Строка – это набор символов, заключенных в двойные кавычки. Например, фрагмент кода:

```
"This is text"
```

представляет собой строку.

Что выведет следующий код?

```
Console.WriteLine("Первая строка\nВторая строка\nТретья строка");
Console.WriteLine("Один\tДва\tТри");
Console.WriteLine("Четыре\tПять\tШесть");
// Вставляем кавычки.
Console.WriteLine("\"Зачем?\", спросил он.");
```

2. Литералы

Помимо формы только что описанного строкового литерала можно также определить **буквальный (verbatim) строковый литерал**.

Такой литерал начинается с символа @, после которого следует строка, заключенная в кавычки.

Содержимое строки в кавычках воспринимается без изменений и может быть расширено до двух и более строк.

Это означает, что в буквальный строковый литерал можно включить символы новой строки, табуляции и прочие, не прибегая к управляющим последовательностям.

Единственное исключение составляют двойные кавычки ("), для указания которых необходимо использовать двойные кавычки с обратным слешем ("\").

2. Литералы

Пример. Что выведет данный код?

```
            Console.WriteLine(@"Это буквальный
строковый литерал,
который занимает несколько строк.
");
            Console.WriteLine(@"А теперь воспользуемся табуляцией:
1 2 3 4
5 6 7 8
");
            Console.WriteLine(@"Отзыв программиста: ""Мне нравится C#.""");


            Console.ReadLine();
```

3. Переменные

Для хранения данных в программе применяются **переменные**. **Переменная** представляет собой именованную область памяти, в которой хранится значение определенного типа. Переменная имеет тип, имя и значение. Тип определяет, какого рода информацию может хранить переменная.

Синтаксис объявления переменных в C# выглядит следующим образом:

```
тип имя_переменной;
```

При создании переменной создается экземпляр соответствующего типа. Таким образом, возможности переменной определяются ее типом.

3. Переменные

В качестве имени переменной может выступать любое произвольное название, которое удовлетворяет следующим требованиям:

- имя может содержать любые цифры, буквы и символ подчеркивания, при этом первый символ в имени должен быть буквой или символом подчеркивания;
- в имени не должно быть знаков пунктуации и пробелов;
- имя не может быть ключевым словом языка C#. Таких слов не так много, и при работе в Visual Studio среда разработки подсвечивает ключевые слова синим цветом.

3. Переменные

Помимо типов переменные различаются и другими качествами. Например, переменные, которые мы использовали в примерах программ до сих пор, называются **локальными**, поскольку они объявляются внутри метода.

Инициализация переменной

Переменная до использования должна получить значение. Это можно сделать с помощью инструкции присваивания. Можно также присвоить переменной начальное значение одновременно с ее объявлением.

Общий формат инициализации переменной имеет такой вид:

```
тип имя_переменной = значение;
```

3. Переменные

```
int count =10; //Присваиваем переменной count начальное значение 10
char ch = 'X1'; //Инициализируем ch буквой X
float f = 1.2F //Переменная f инициализируется числом 1.2
```

При объявлении двух или более переменных одного типа с помощью списка (с разделением элементов списка запятыми) одной или нескольким из этих переменных можно присвоить начальные значения. Например, в инструкции:

```
int a, b = 8, c = 19, d; // Переменные b и c инициализируются числами
```

3. Переменные

```
string name;
string Name;
```

Динамическая инициализация

В C# допускается также ***динамическая инициализация*** переменных с помощью любого выражения, действительного на момент объявления переменной:

```
int i1 = 3, i2 = 4;

// Инициализируем динамически переменную result
double result = Math.Sqrt(i1*i1 + i2*i2);
```

3. Переменные

Константы

Константы – это переменные, значение которых нельзя изменить во время выполнения программы.

Константа объявляется с помощью служебного слова const, после которого следует тип константы:

```
const int j = 100;
```

3. Переменные

Особенности констант:

- Константы должны инициализироваться при объявлении, присвоенные им значения никогда не могут быть изменены.
- Значение константы вычисляется во время компиляции. Поэтому инициализировать константу значением, взятым из другой переменной, нельзя.
- Константы всегда являются неявно статическими. Но при этом не нужно указывать модификатор `static`.

3. Переменные

Неявно типизированные переменные

Как правило, при объявлении переменной сначала указывается тип, например int или bool, а затем имя переменной.

Но начиная с версии C# 3.0, компилятору предоставляется возможность самому определить тип локальной переменной, исходя из значения, которым она инициализируется.

Такая переменная называется **неявно типизированной**.

3. Переменные

Неявно типизированная переменная объявляется с помощью ключевого слова **var** и должна быть непременно инициализирована.

Для определения типа этой переменной компилятору служит тип ее инициализатора, т.е. значения, которым она инициализируется:

```
var i = 12; //переменная i инициализируется целочисленным литералом
var d = 12.3;   //переменная d инициализируется литералом с плавающей точкой, имеющему тип double
var f = 0.34F;  //переменная f теперь имеет тип float
```

3. Переменные

Единственное отличие неявно типизированной переменной от обычной, явно типизированной переменной – в способе определения ее типа.

Как только этот тип будет определен, он закрепляется за переменной до конца ее существования.

3. Переменные

Область видимости и время существования переменных

Область видимости, или ***контекст переменной*** – это часть кода, в пределах которого доступна данная переменная. В общем случае такая область определяется описанными ниже правилами:

Поле, также известное как переменная-член класса, находится в области видимости до тех пор, пока в этой области находится содержащий поле класс.

Локальная переменная находится в области видимости до тех пор, пока закрывающая фигурная скобка не укажет конец блока операторов или метода, в котором она объявлена.

Локальная переменная, объявленная в операторах цикла for, while или подобных им, видима в пределах тела цикла.

3. Переменные

Чтобы лучше понять суть вложенных областей видимости, рассмотрим следующую программу:

```
int x; // Переменная x известна всему коду в пределах
       // метода Main().
x = 10;
if (x == 10)
{ // Начало новой области видимости,
    int y = 20; // Переменная y известна только
                // этому блоку.
                // Здесь известны обе переменные x и y.
    Console.WriteLine("x и y: " + x + " " + y);
    x = y * 2;
}
// y = 100; // Ошибка! Переменная y здесь неизвестна.
// Переменная x здесь известна.
Console.WriteLine("Значение x равно " + x);
```

4. Преобразования типов

В программировании переменной одного типа часто присваивается значение переменной другого типа.

Например, как показано в следующем фрагменте программы, мы могли бы присвоить переменной типа float значение типа int.

```
int i;
float f;
i = 10;
f = i; //float-переменной присваивается int-значение
```

4. Преобразования типов

В C# не все типы совместимы и действует строгий контроль типов, не все преобразования типов разрешены в неявном виде.

Например, типы bool и int несовместимы. Тем не менее с помощью операции приведения типов все-таки возможно выполнить преобразование между несовместимыми типами.

Приведение типов – это выполнение преобразования типов в явном виде.

4. Преобразования типов

Автоматическое преобразование типов

При присвоении значения одного типа данных переменной другого типа будет выполнено автоматическое преобразование типов, если:

- эти два типа совместимы;
- тип приемника больше (т.е. имеет больший диапазон представления чисел), чем тип источника.

4. Преобразования типов

При соблюдении этих двух условий выполняется преобразование с расширением, или расширяющее преобразование.

Например, тип int – достаточно «большой» тип, чтобы сохранить любое допустимое значение типа byte, а поскольку как int, так и byte – целочисленные типы, здесь может быть применено автоматические преобразование.

Для расширяющих преобразований числовые типы, включая целочисленные и с плавающей точкой, совместимы один с другим.

Например, следующая программа совершенно легальна, поскольку преобразование типов из long в double является расширяющим, которое выполняется автоматически.

4. Преобразования типов

```
long L;
double D;

L = 100123285L;
D = L;

Console.WriteLine("L и D: " + L + " " + D);
```

Несмотря на возможность автоматического преобразования типов из long в double, обратное преобразование типов (из double в long) автоматически не выполняется, поскольку это преобразование не является расширяющим.

4. Преобразования типов

Приведение несовместимых типов

Несмотря на большую пользу автоматического преобразования типов оно не в состоянии удовлетворить все нужды программирования, поскольку реализуется только при расширяющем преобразовании между совместимыми типами. Во всех остальных случаях приходится применять приведение к типу.

Приведение к типу – это явно заданная инструкция компилятору преобразовать один тип в другой.

Инструкция приведения записывается в следующей общей форме:

```
(тип_приемника)выражение
```

4. Преобразования типов

Здесь элемент *тип_приемника* определяет тип для преобразования заданного выражения.

Пример:

```
int i1 = 455, i2 = 84500;
decimal dec = 7.98845m;

// Приводим два числа типа int
// к типу short
Console.WriteLine((short)i1);
Console.WriteLine((short)i2);

// Приводим число типа decimal
// к типу int
Console.WriteLine((int)dec);

Console.ReadLine();
```

```
455
18964
7
```

4. Преобразования типов

Перехват сужающих преобразований данных

В предыдущем примере приведение переменной i2 к типу short не является приемлемым, т.к. возникает потеря данных. Для создания приложений, в которых потеря данных должна быть недопустимой, в C# предлагаются такие ключевые слова, как *checked* и *unchecked*, которые позволяют гарантировать, что потеря данных не окажется незамеченной.

По умолчанию, в случае, когда не предпринимается никаких соответствующих исправительных мер, условия переполнения (overflow) и потери значимости (underflow) происходят без выдачи ошибки.

Обрабатывать условия переполнения и потери значимости в приложении можно двумя способами. Это можно делать вручную, полагаясь на свои знания и навыки в области программирования.

4. Преобразования типов

В случае возникновения условия переполнения во время выполнения будет генерироваться исключение System.OverflowException.

Рассмотрим пример, в котором будем передавать в консоль значение исключения:

```
byte var1 = 250;
byte var2 = 150;
try
{
    byte sum = checked((byte)(var1 + var2));
    Console.WriteLine("Сумма: {0}", sum);
}
catch (OverflowException ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}
```

```
Arithmetic operation resulted in an overflow.
```

4. Преобразования типов

Роль класса System.Convert

В завершении темы преобразования типов данных стоит отметить, что в пространстве имен System имеется класс Convert, который тоже может применяться для расширения и сужения данных:

```
byte sum = Convert.ToByte(var1 + var2);
```

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!