Тема: *Типы данных языка C#*

Преподаватель
Мельникова Татьяна Федоровна

Классификация типов


Данные, с которыми работает программа, хранятся в оперативной памяти. Компилятору необходимо точно знать, сколько места они занимают, как именно объявлены и какие действия с ними можно выполнять.

Все это задается при описании данных с помощью типа. Тип данных однозначно определяет:

- внутреннее представление данных, а следовательно и множество их возможных значений;
- допустимые действия над данными (операции и функции).

Классификация типов


Например, целые и вещественные числа, даже если они занимают одинаковый объем памяти, имеют совершенно разные диапазоны возможных значений.

Каждое выражение в программе имеет определенный тип.

Компилятор использует информацию о типе при проверке допустимости описанных в программе действий.

Разделение памяти



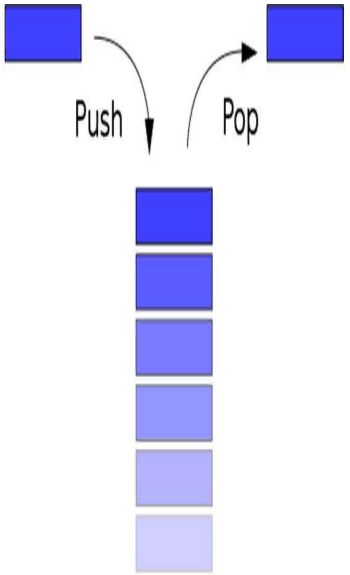
Память, в которой хранятся данные во время выполнения программы, делится на три области:

Статическая (static), стек (stack) и динамическая область, или хип (heap).

Статическая память выделяется еще до начала работы программы, на стадии компиляции и сборки. Статические переменные имеют фиксированный адрес, известный до запуска программы и не изменяющийся в процессе ее работы. Статические переменные создаются и инициализируются до входа в функцию Main, с которой начинается выполнение программы.

Разделение памяти

Область видимости статической переменной ограничена одним файлом, внутри которого она определена.



Стек используется для хранения величин, память под которые выделяет компилятор.

Локальные, или стековые, переменные - это переменные, описанные внутри функции.

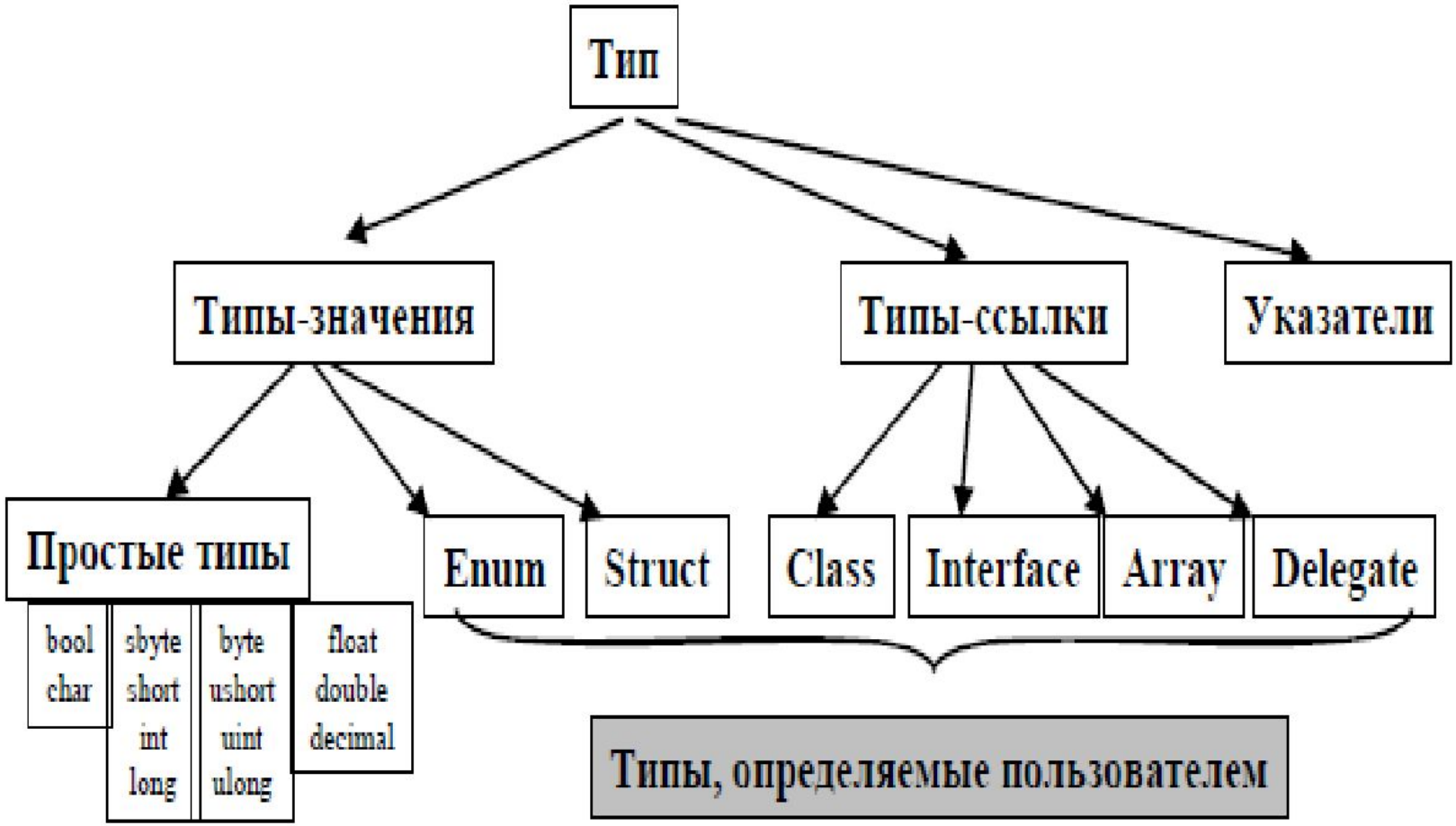
Память выделяется в момент входа в функцию или блок и освобождается в момент выхода из функции или блока.

Динамическая память, или куча

Основным местом для хранения данных в C# является хип. Динамическая память состоит из захваченных и свободных сегментов, каждому из которых предшествует описатель сегмента. При выполнении запроса на захват памяти исполняющая система производит поиск свободного сегмента достаточного размера и захватывает в нем отрезок требуемой длины. При освобождении сегмента памяти он помечается как свободный, при необходимости несколько подряд идущих свободных сегментов объединяются.

Куча схож со стеком, но если стек представляется в виде последовательности коробок, складируемых друг на друге, в случае с кучей эти самые коробки аккуратно разложены и мы можем получить к ним доступ в любое время.

Классификация типов



Встроенные типы

Встроенные типы не требуют предварительного определения. Для каждого типа существует ключевое слово, которое используется при описании переменных, констант и т. д.

Встроенные типы однозначно соответствуют стандартным классам библиотеки .NET, определенным в пространстве имен System.

Существует несколько вариантов представления целых и вещественных величин. Программист выбирает тип каждой величины, используемой в программе, с учетом необходимого ему диапазона и точности представления данных.

Встроенные типы

Имя типа	Ключевое слово	Системный тип .NET	Диапазон значений	Описание	Размер, байт
Логический тип	bool	Boolean	true, false		
Целые типы	sbyte	SByte	–128 .. 127	Со знаком	1
	byte	Byte	0 ..255	Без знака	1
	short	Int16	–32768 .. 32767	Со знаком	2
	ushort	UInt16	0 .. 65535	Без знака	2
	int	Int32	$-2\ 10^9$ .. $2\ 10^9$	Со знаком	4
	uint	UInt32	0 .. $4\ 10^9$	Без знака	4
	long	Int64	$-9\ 10^{18}$.. $9\ 10^{18}$	Со знаком	8
	ulong	UInt64	0 .. $18\ 10^{18}$	Без знака	8

Встроенные типы

Название	Ключевое слово	Тип .NET	Диапазон значений	Описание	Размер, байт
Символьный тип	char	Char	U+0000 .. U+ffff	Unicode-символ	2
Вещественные	float	Single	$1.5\ 10^{-45}$.. $3.4\ 10^{38}$	7 цифр	4
	double	Double	$5.0 10^{-324}$ $1.7\ 10^{308}$	15–16 цифр	8
Финансовый тип	decimal	Decimal	$1.0\ 10^{-28}$.. $7.9\ 10^{28}$	28–29 цифр	16
Строковый тип	string	String	Длина ограничена объемом доступной памяти	Строка из Unicode-символов	
Тип object	object	Object	Можно хранить все, что угодно	Всеобщий предок	

Тип данных

Любой встроенный тип C# соответствует стандартному классу библиотеки .NET, определенному в пространстве имен System. Везде, где используется имя встроенного типа, его можно заменить именем класса библиотеки.

Типы также могут быть заданы как:

- Типы значений, то есть хранящие значения. К ним относятся простые числовые типы, перечисления и структуры, а также версии этих типов, допускающие значения NULL.
- Ссылочные типы, то есть хранящие ссылки на фактические данные. К ним относятся классы, интерфейсы, массивы и делегаты.

Переменные и операции

Переменная — это именованная область памяти, предназначенная для хранения данных определенного типа.

Все переменные, используемые в программе, должны быть описаны явным образом. При описании для каждой переменной задаются ее **тип и имя.**

```
int a;   float x;
```

Имя переменной служит для обращения к области памяти, где хранится значение переменной. Тип переменной выбирается, исходя из диапазона и требуемой точности представления данных. При объявлении можно присвоить переменной некоторое начальное значение, то есть инициализировать ее, например:

```
int a, b = 1; float x = 0.1, y = 0.1f; int c = b * a + 25;
```

Схема неявного преобразования числовых типов



Преобразование типов бывает неявным и явным.

Расширяющие преобразования от типа с меньшей разрядностью к типу с большей разрядностью компилятор поводит неявно.

Это могут быть следующие цепочки преобразований:

byte -> short -> int -> long -> decimal
int -> double
short -> float -> double
char -> int

Схема неявного преобразования числовых типов

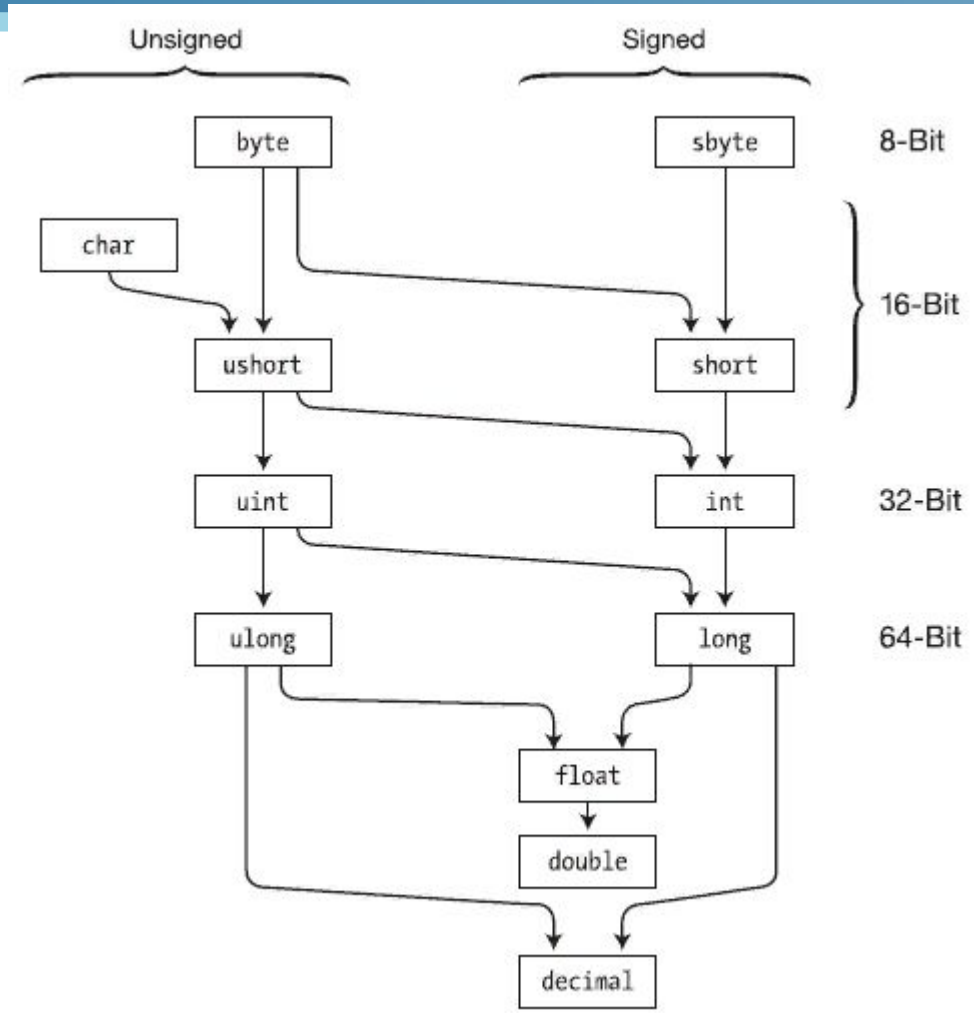


Схема неявного преобразования числовых типов


В остальных случаях следует использовать явные преобразования типов.
Для явного преобразования требуется применять оператор приведения в форме
(<целевой тип>)<выражение>.
При выполнении явного преобразования ответственность за его корректность возлагается на программиста. Суть операции преобразования типов состоит в том, что перед значением указывается в скобках тип, к которому надо привести данное значение:

```
int k = 100;
byte i;                  // тип byte «меньше» типа int
i = (byte) (k+10);     // требуется явное преобразование
```

Также следует отметить, что несмотря на то, что и double, и decimal могут хранить дробные данные, а decimal имеет большую разрядность, чем double, но все равно значение double нужно явно приводить к типу decimal:

```
double a = 4.0;
decimal b = (decimal)a;
```

Приведение

Приведение явно вызывает оператор преобразования из одного типа в другой. Если ни один такой оператор не определен, приведение завершается неудачей.

```
static void TestCasting()
{    int i = 10;
     float f = 0;
     f = i;                    //неявные преобразования
     i = (int)f;          //явные преобразования
   double x = 1234.7;
        int a;
        a = (int)x;      // преобразования  double в  int
        System.Console.WriteLine(a);
}                                   Результат   1234
```

Использование системных типов



Название встроенного типа представляет собой сокращенное обозначение системного типа.

Например, следующие переменные будут эквивалентны по типу:

```
int a = 4;
System.Int32 b = 4;
```

Символьные литералы

Символьные литералы представляют одиночные символы, заключаются в одинарные кавычки.

Символьные литералы бывают нескольких видов:

- обычные символы: '2' , 'A' , 'T'
- в виде шестнадцатеричных кодов, также заключенный в одинарные кавычки:

```
Console.WriteLine('\0x78');    // представляет символ "x"
Console.WriteLine('\0x5A');    // представляет символ "Z "
```

-с применением кодов из таблицы символов Unicode. Для этого в одинарных кавычках указываются символы '\u', после которых идет шестнадцатеричный код Unicode.

Например, код - '\u0411' //представляет кириллический символ 'Б':

```
Console.WriteLine('\u0420');    // P
Console.WriteLine('\u0421');    // C
```

Ключевое слово var

В язык C# было добавлено ключевое слово **var,** которое позволяет создавать переменные без явного указания типа данных.

Тип данных такой переменной определяет компилятор по контексту инициализации.

```
var number = 5;            // number будет типа int
var text ="some text"; // text будет типа string
var number2 = 0.5;      // number2 будет типа double
```

Ключевое слово var

var сохраняет принцип строгой типизации в Си-шарп.

Это означает, что после того, как для переменной уже был определен тип, в нее нельзя записать данные другого типа:

```
static void Main(string[] args)
{
  var number = 5;
  number = "some text";     // !!! ошибка, number определен как int
}
```

Ключевое слово var следует использовать в первую очередь с LINQ выражениями (при работе с базами данных)

```
var query = from s in bdContext.Students select s;
```

Ключевое слово var

Ключевое слово **var** имеет ограничения по его использованию - **var** не может быть в качестве:

- поля класса
- аргумента функции
- возвращаемого типа функции
- переменной, которой присваивается null

Нововведение var является достаточно противоречивым среди разработчиков на C#, некоторые используют его где только возможно, другие его избегают (код становится плохо читаемым).

Ключевое слово var

можно использовать и модель неявной типизации:

```
var hello = "Hell to World";
var c = 20;
  Console.WriteLine(c.GetType().ToString());
  Console.WriteLine(hello.GetType().ToString());
```

Для неявной типизации вместо названия типа данных используется ключевое слово var. Затем уже при компиляции компилятор сам выводит тип данных исходя из присвоенного значения. В примере выше использовалось выражение Console.WriteLine(c.GetType().ToString()); , которое позволяет нам узнать выведенный тип переменной **c** (int или System.Int32).

Ссылочные типы

Также существуют ссылочные типы. Из базовых типов к ссылочным относятся **object и string.**

Тип **object** является базовым для всех остальных типов данных.

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

Представлен системным типом ***System.Object,*** который является базовым для всех других типов и классов .NET.

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

Класс Object

Приведение к объектному типу (**boxing**) — процесс приведения экземпляра значимого типа к ссылочному типу.

Ссылочным типом в данном случае может быть либо класс object, либо интерфейс. Восстановление значения из объектного образа (**unboxing**) — обратный процесс, приведение object назад к изначальному значимому типу.

```
int x = 9;
object obj = x; // Boxing
int y = (int)obj; // Unboxing
```

Восстановление значения требует явного приведения, при этом проверяется, что указанный значимый тип соответствует реальному типу объекта, и если это не так выбрасывается InvalidCastException

Список членов типа object

Практическое значение типа объект в том, что помимо методов и свойств, которые определяет программист, также появляется доступ к множеству общедоступных и защищенных методов-членов, которые определены в классе Object.

Эти методы можно использовать во всех определяемых классах

- public extern Type **GetType**();
- public virtual bool **Equals** (object obj);
- public static bool **Equals** (object objA, object objB);
- public static bool **ReferenceEquals** (object objA, object objB);
- public virtual int **GetHashCode**();
- public virtual string **ToString**();
- protected override void **Finalize**();
- protected extern object **MemberwiseClone**();

Методы System.Object
ToString()

Метод ToString() возвращает символьную строку, содержащую описание того объекта, для которого он вызывается.

Метод ToString() автоматически вызывается при выводе содержимого объекта с помощью метода WriteLine().

Этот метод переопределяется во многих классах, что позволяет приспосабливать описание к конкретным типам объектов, создаваемых в этих классах.

Применяйте этот метод, когда нужно получить представление о содержимом объекта — возможно, в целях отладки.

Методы System.Object
GetType()

Этот метод возвращает экземпляр класса, унаследованный от **System.Type.**

Этот объект может предоставить большой объем информации о классе, членом которого является ваш объект, включая базовый тип, методы, свойства и т.п. System.Type также представляет собой стартовую точку технологии рефлексии .NET.

Методы System.Object
Clone()

Этот метод создает копию объекта и возвращает ссылку на эту копию (а в случае типа значения — ссылку на упаковку).

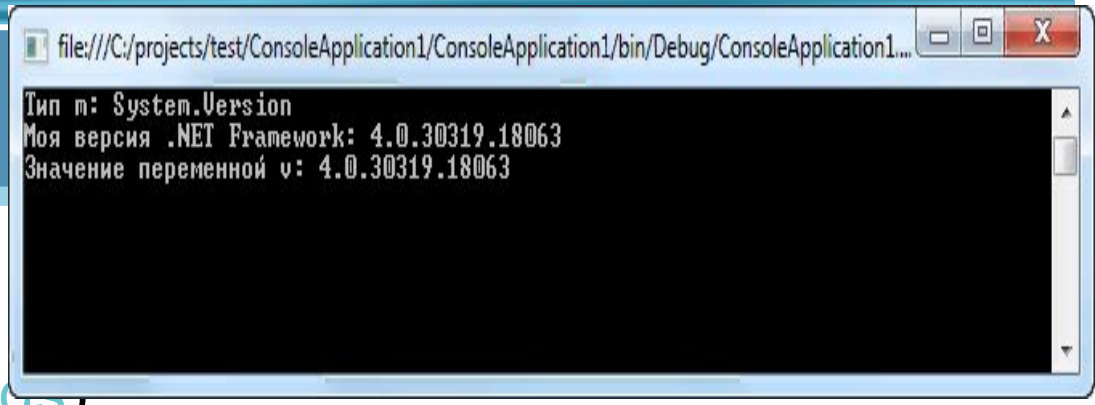
При этом выполняется неглубокое копирование, т.е. копируются все типы значений в классе.

Если же класс включает в себя члены ссылочных типов, то копируются только ссылки, а не объекты, на которые они указывают.

Этот метод является защищенным, а потому не может вызываться для копирования внешних объектов.

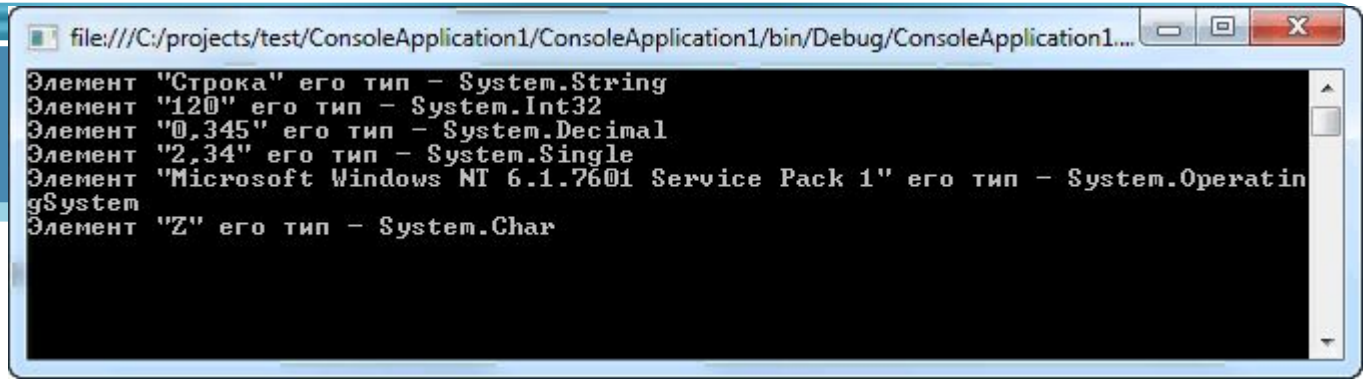
К тому же он не виртуальный, а потому переопределять его реализацию нельзя.

Пример




```
class Program
{ static void Main(string[] args)
 { var m = Environment.Version;
 Console.WriteLine("Тип m: "+m.GetType());
string s = m.ToString();
Console.WriteLine("Моя версия .NET Framework: " + s);
    Version v = (Version)m.Clone();
    Console.WriteLine("Значение переменной v: "+v); Consol
}
```

Пример




```
class Program
{ static void Main(string[] args)
 { var myOS = Environment.OSVersion;
 object[] myArr = { "Строка", 120, 0.345m, 2.34f, myOS, 'Z' };
    foreach (object obj in myArr)
Console.WriteLine("Элемент \"{0}\" его тип - {1}", obj,
    obj.GetType());
 Console.ReadLine(); } }
```

Спасибо за внимание!