# Глава 3. Примитивни типове и променливи

## В тази тема...

В настоящата тема ще разгледаме примитивните типове и променливи в C# – какво представляват и как се работи с тях. Първо ще се спрем на типовете данни – целочислени типове, реални типове с плаваща запетая, булев тип, символен тип, стринг и обектен тип. Ще продължим с промен­ливите, какви са техните характеристики, как се декларират, как им се присвоява стойност и какво е инициализация на променлива. Ще се запознаем и с типовете данни в C# – стойностни и референтни. Накрая ще разгледаме различните видове литерали и тяхното приложение.

## Какво е променлива?

Една типична програма използва различни стойности, които се променят по време на нейното изпълнение. Например създаваме програма, която извършва някакви пресмятания върху стойности, които потребителят въвежда. Стойностите, въведени от даден потре­бител, ще бъдат очевидно различни от тези, въведени от друг потребител. Това означава, че когато създава програмата, **програмистът не знае всички възможни стойности**, които ще бъдат въвеждани като вход, а това налага да се обработват всички различни стойности, въвеждани от различните потребители.

Когато потребителят въведе нова **стойност**, която ще участва в процеса на пресмятане, **можем да я съхраним** (временно) в оперативната памет на нашия компютър. Стойностите в тази част на паметта се променят посто­янно и това е довело до наименованието им – променливи.

## Типове данни

**Типовете данни** представляват множества (диапазони) от стойности, които имат еднакви характеристики. Например типът byte задава множе­ството от цели числа в диапазона [0….255].

### Характеристики

Типовете данни се характеризират с:

* Име – например int;
* Размер (колко памет заемат) – например 4 байта;
* Стойност по подразбиране (**default value**) – например 0.

### Видове

Базовите типове данни в C# се разделят на следните видове:

* Целочислени типове – sbyte, byte, short, ushort, int, uint, long, ulong;
* Реални типове с плаваща запетая – float, double;
* Реални типове с десетична точност – decimal;
* Булев тип – bool;
* Символен тип – char;
* Символен низ (стринг) – string;
* Обектен тип – object.

Тези типове данни се наричат **примитивни (built-in types)**, тъй като са вградени в езика C# на най-ниско ниво. В таблицата по-долу можем да видим изброените по-горе типове данни, техния обхват и стойностите им по подразбиране:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данни** | **Стойност по подразбиране** | **Минимална стойност** | **Максимална стойност** |
| **sbyte** | 0 | -128 | 127 |
| **byte** | 0 | 0 | 255 |
| **short** | 0 | -32768 | 32767 |
| **ushort** | 0 | 0 | 65535 |
| **int** | 0 | -2147483648 | 2147483647 |
| **uint** | 0u | 0 | 4294967295 |
| **long** | 0L | -9223372036854775808 | 9223372036854775807 |
| **ulong** | 0u | 0 | 18446744073709551615 |
| **float** | 0.0f | ±1.5×10-45 | ±3.4×1038 |
| **double** | 0.0d | ±5.0×10-324 | ±1.7×10308 |
| **decimal** | 0.0m | ±1.0×10-28 | ±7.9×1028 |
| **boolean** | false | Възможните стойности са две – **true** или **false** | |
| **char** | '\u0000' | '\u0000' | ‘\uffff’ |
| **object** | null | - | - |
| **string** | null | - | - |

### Съответствие на типовете в C# и в .NET Framework

Примитивните типове данни в C# имат директно съответствие с типове от общата система от типове (CTS) от .NET Framework. Например типът int в C# съответства на типа System.Int32 от CTS и на типа Integer в езика VB.NET, a типът long в C# съответства на типа System.Int64 от CTS и на типа Long в езика VB.NET. Благодарение на общата система на типовете (CTS) в .NET Framework има съвместимост между различните езици за програмиране (като например C#, Managed C++, VB.NET и F#). По същата причина типовете int, Int32 и System.Int32 в C# са всъщност различни псевдоними за един и същ тип данни – 32 битово цяло число със знак.

### Целочислени типове

Целочислените типове отразяват целите числа и биват sbyte, byte, short, ushort, int, uint, long и ulong. Нека ги разгледаме един по един.

Типът sbyte е 8-битов целочислен тип **със знак** (**signed integer**). Това означава, че броят на възможните му стойности е 28, т.е. 256 възможни стойности общо, като те могат да бъдат както положителни, така и отрицателни. Минимал­ната стойност, която може да се съхранява в sbyte, е SByte.MinValue = -128 (-27), а максимал­ната е SByte.MaxValue = 127 (27-1). Стойността по подразбиране е числото 0.

Типът byte е 8-битов **беззнаков** (**unsigned**) целочислен тип. Той също има 256 различни целочислени стойности (28), но те могат да бъдат само неотрица­телни. Стойността по подразбиране на типа byte е числото 0. Минимал­ната му стойност е Byte.MinValue = 0, а максималната е Byte.MaxValue = 255 (28-1).

Целочисленият тип short е 16-битов тип със знак. Минималната стойност, която може да заема, е Int16.MinValue = -32768 (-215), а максималната – Int16.MaxValue = 32767 (215-1). Стойността по подразбиране е числото 0.

Типът ushort е 16-битов беззнаков тип. Минималната стойност, която може да заема, е UInt16.MinValue = 0, а максималната – UInt16. MaxValue = 65535 (216-1). Стой­ността по подраз­биране е числото 0.

Следващият целочислен тип, който ще разгледаме, е int. Той е 32- битов знаков тип. Както виждаме, с нарастването на битовете нарастват и възможните стойности, които даден тип може да заема. Стой­ността по подразбиране е числото 0. Минималната стойност, която може да заема, е Int32.MinValue = -2 147 483 648 (-231), а максималната e Int32.MaxValue = 2 147 483 647 (231-1).

Типът int е най-често използваният тип в програмирането. Обикновено програмистите използват int, когато работят с цели числа, защото този тип е естествен за 32-битовите микропроцесори и е достатъчно "голям" за повечето изчис­ления, които се извършват в ежедневието.

Типът uint е 32-битов беззнаков тип. Стойността по подразбиране е числото **0u** или **0U** (двата записа са еквивалентни). Символът 'u' указва, че числото е от тип uint (иначе се подразбира int). Минималната стойност, която може да заема, е UInt32.MinValue = 0, а максималната му стойност е UInt32.MaxValue = 4 294 967 295 (232-1).

Типът long е 64-битов знаков тип със стойност по подразбиране 0l или 0L (двете са еквивалентни, но за предпочитане е да използвате 'L', тъй като 'l' лесно се бърка с цифрата единица '1'). Символът 'L' указва, че числото е от тип **long** (иначе се подразбира int). Минималната стойност, която може да заема типът long, е Int64.MinValue = -9 223 372 036 854 775 808 (-263), а максималната му стойност е Int64.MaxValue = 9 223 372 036 854 775 807 (263-1).

Най-големият целочислен тип е типът ulong. Той е 64-битов беззнаков тип със стойност по подразбиране числото **0u** или **0U** (двата записа са екви­валентни). Символът 'u' указва, че числото е от тип **ulong** (иначе се подразбира long). Минималната стойност, която може да бъде записана в типа ulong, е UInt64.MinValue = 0, а макси­малната – UInt64.MaxValue = 18 446 744 073 709 551 615 (264-1).

#### Целочислени типове – пример

Нека разгледаме един пример, в който декларираме няколко променливи от познатите ни целочислени типове, инициализираме ги и отпечатваме стойностите им на конзолата:

|  |
| --- |
| // Declare some variables  byte centuries = 20;  ushort years = 2000;  uint days = 730480;  ulong hours = 17531520;  // Print the result on the console  Console.WriteLine(centuries + " centuries are " + years +  " years, or " + days + " days, or " + hours + " hours.");  // Console output:  // 20 centuries are 2000 years, or 730480 days, or 17531520  // hours.  ulong maxIntValue = UInt64.MaxValue;  Console.WriteLine(maxIntValue); // 18446744073709551615 |

Какво представлява деклариране и инициа­лизация на променлива, можем да прочетем в детайли по-долу в секциите "[Декла­риране на променливи](#_Деклариране_на_променливи)" и "[Инициа­лизация на променливи](#_Инициализация_на_променливи)", но това става ясно и от примерите.

В разгледания по-горе пример демонстрираме използването на целочис­лените типове. За малки числа използваме типа byte, а за много големи – ulong. Използваме беззнакови типове, тъй като всички използвани стой­ности са **положителни числа**.

### Реални типове с плаваща запетая

Реалните типове в C# представляват реалните числа, които познаваме от математиката. Те се представят чрез **плаваща запетая** (**floating-point**) според стандарта IEEE 754 и биват float и double. Нека разгледаме тези два типа данни в детайли, за да разберем по какво си приличат и по какво се различават.

#### Реален тип float

Първият тип, който ще разгледаме, е 32-битовият реален тип с плаваща запетая float. Той се нарича още **реален тип с единична точност** (single precision real number). Стойността му по подразбиране е 0.0f или 0.0F (двете са еквивалентни). Символът 'f' накрая указва изрично, че числото е от тип float (защото по подразбиране всички реални числа се приемат за double). Повече за този специален суфикс можем да прочетем в секцията "[Реални литерали](#_Реални_литерали)". Разглежданият тип има точност до 7 десетични знака (оста­налите се губят). Например, числото 0.123456789 ако бъде записано в типа float ще бъде закръглено до 0.1234567. Диапазонът на стой­ностите, които могат да бъдат записани в типа float (със закръгляне до точност 7 значещи десетични цифри) е от ±1.5 × 10-45 до ±3.4 × 1038.

Най-малката реална стойност на типа float е Single.MinValue = -3.40282e+038f, а най-голямата е Single.MaxValue = 3.40282e+038f. Най-близкото до 0 положително число от тип float е Single.Epsilon = 1.401298E-45.

#### Специални стойности на реалните типове

Реалните типове данни имат и няколко специални стойности, които не са реални числа, а представляват математически абстракции:

* Минус безкрайност -∞ (Single.NegativeInfinity). Получава се например като разделим -1.0f на 0.0f.
* Плюс безкрайност +∞ (Single.PositiveInfinity). Получава се например като разделим 1.0f на 0.0f.
* Неопределеност (Single.NaN) – означава, че е извършена невалидна операция върху реални числа. Получава се например като разделим 0.0f на 0.0f, както и при коренуване на отрицателно число.

#### Реален тип double

Вторият реален тип с плаваща запетая в езика C# е типът double. Той се нарича още **реално число с двойна точност** (double precision real number) и представлява 64-битов тип със стойност по подразбиране 0.0d или 0.0D (символът 'd' не е задължителен, тъй като по подразбиране всички реални числа в C# са от тип double). Разглежданият тип има точност от 15 до 16 десетични цифри. Диапазонът на стойностите, които могат да бъдат записани в double (със закръгляне до точност 15-16 значещи десетични цифри) е от ±5.0 × 10-324 до ±1.7 × 10308.

Най-малката реална стойност на типа double е константата Double. MinValue = -1.79769e+308, а най-голямата – Double.MaxValue = 1.79769e+308. Най-близкото до 0 положително число от тип double е Double.Epsilon = 4.94066e-324. Както и при типа float, променливите от тип double могат да получават специалните стойности Double. PositiveInfinity, Double.NegativeInfinity и Double.NaN.

#### Реални типове – пример

Ето един пример, в който декларираме променливи от тип реално число, присвояваме им стойности и ги отпечатваме:

|  |
| --- |
| float floatPI = 3.14f;  Console.WriteLine(floatPI); // 3.14  double doublePI = 3.14;  Console.WriteLine(doublePI); // 3.14  double nan = Double.NaN;  Console.WriteLine(nan); // NaN  double infinity = Double.PositiveInfinity;  Console.WriteLine(infinity); // Infinity |

#### Точност на реалните типове

Реалните числа в математиката в даден диапазон са неизброимо много (за разлика от целите числа), тъй като между всеки две реални числа a и b съществуват безброй много други реални числа c, за които a < c < b. Това налага необходимостта реалните числа да се съхраняват в паметта на компютъра с **определена точност**.

Тъй като математиката и най-вече физиката работят с изключително големи числа (положителни и отрицателни) и изключително малки числа (много близки до нула), е необходимо реалните типове в изчислителната техника да могат да ги съхраняват и обработват по подходящ начин. Например според физиката масата на електрона е приблизително 9.109389\*10-31 килограма, а в един мол вещество има около 6.02\*1023 атома. И двете посочени величини могат да бъдат записани безпроблемно в типовете float и double. Поради това удобство в съвременната изчислителна техника често се използва **пред­ставянето с плаваща запетая** – за да се даде възможност за работа с максимален брой значещи цифри при много големи числа (например положителни и отрицателни числа със стотици цифри) и при числа много близки до нулата (например положи­телни и отрица­телни числа със стотици нули след десетичната запетая преди първата значеща цифра).

#### Точност на реални типове – пример

Разгледаните реални типове в C# float и double се различават освен с порядъка на възможните стойности, които могат да заемат, и по точност (броят десетични цифри, които запазват). Първият тип има точност 7 знака, вторият – 15-16 знака.

Нека разгледаме един пример, в който декларираме няколко променливи от познатите ни реални типове, инициализираме ги и отпечатваме стой­ностите им на конзолата. Целта на примера е да онагледим разликата в точността им:

|  |
| --- |
| // Declare some variables  float floatPI = 3.141592653589793238f;  double doublePI = 3.141592653589793238;  // Print the results on the console  Console.WriteLine("Float PI is: " + floatPI);  Console.WriteLine("Double PI is: " + doublePI);  // Console output:  // Float PI is: 3.141593  // Double PI is: 3.14159265358979 |

Виждаме, че числото **π**, което декларирахме от тип float, е закръглено на 7-мия знак, а при тип double – на15-тия знак. Изводът, който можем да си направим, е, че реалният тип double запазва доста по-голяма точност от float и ако ни е необходима голяма точност след десетичния знак, ще използваме него.

#### За представянето на реалните типове

Реалните числа с плаваща запетая в C# се състоят от три компонента (съгласно стандарта IEEE 754): **знак** (1 или -1), **мантиса** и **порядък** (**експонента**), като стойността им се изчислява по сложна формула. По-подробна информация за представянето на реалните числа сме предви­дили в темата "[Бройни системи](#_Глава_8._Бройни_1)", където разглеждаме в дълбочина пред­ставянето на числата и другите типове данни в изчислителната техника.

#### Грешки при пресмятания с реални типове

При пресмятания с реални типове данни с плаваща запетая е възможно да наблюдаваме странно поведение, тъй като при представянето на дадено реално число много често се губи точност. Причината за това е невъзмож­ността някои реални числа да се представят точно сума от отрицателни степени на числото 2. Примери за числа, които нямат точно представяне в типовете float и double, са 0.1, 1/3, 2/7 и други. Следва примерен C# код, който демонстрира грешките при пресмятания с числа с пла­ваща запетая в C#:

|  |
| --- |
| float f = 0.1f;  Console.WriteLine(f); // 0.1 (correct due to rounding)  double d = 0.1f;  Console.WriteLine(d); // 0.100000001490116 (incorrect)  float ff = 1.0f / 3;  Console.WriteLine(ff); // 0.3333333 (correct due to rounding)  double dd = ff;  Console.WriteLine(dd); // 0.333333343267441 (incorrect) |

Причината за неочаквания резултат в първия пример е фактът, че числото 0.1 (т.е. 1/10) няма точно представяне във формата за реални числа с плаваща запетая IEEE 754 и се записва в него с приближение. При непо­средствено отпечатване резултатът **изглежда коректен заради закръглява­нето**, което се извършва скрито при преобразуването на числото към стринг. При преминаване от float към double грешката, получена заради приближеното представяне на числото в IEEE 754 формат, става вече явна и не може да бъде компенсирана от скритото закръгляване при отпечатва­нето, и съответно след осмата значеща цифра се появяват грешки.

При втория случай числото 1/3 няма точно представяне и се закръглява до число, много близко до 0.3333333. Кое е това число се вижда отчетливо, когато то се запише в типа double, който запазва много повече значещи цифри.

И двата примера показват, че аритметиката с числа с плаваща запетая може да прави грешки и по тази причина **не е подходяща за прецизни финансови пресмятания**. За щастие C# поддържа аритметика с десетична точност, при която числа като 0.1 се представят в паметта без закръгляне.

|  |  |
| --- | --- |
|  | Не всички реални числа имат точно представяне в типо­вете **float** и **double**. Например числото 0.1 се представя закръглено в типа **float** като 0.099999994. |

### Реални типове с десетична точност

В C# се поддържа т. нар. **десетична аритметика с плаваща запетая (decimal floating-point arithmetic)**, при която числата се представят в десетична, а не в двоична бройна система и така не се губи точност при записване на десетично число в съответния тип с плаваща запетая.

Типът данни за реални числа с десетична точност в C# е 128-битовият тип decimal. Той има точност от 28 до 29 десетични знака. Минималната му стойност е **-7.9×1028**, а максимал­ната е **+7.9×1028**. Стойността му по подразбиране е 0.0m или 0.0М. Символът 'm' накрая указва изрично, че числото е от тип decimal (защото по подразбиране всички реални числа са от тип double). Най-близките до 0 числа, които могат да бъдат записани в decimal са ±1.0 × 10-28. Видно е, че decimal не може да съхранява много големи положителни и отрица­телни числа (например със стотици цифри), нито стойности много близки до 0. За сметка на това този тип почти не прави грешки при финансови пресмятания, защото **представя числата като сума от степени на числото 10**, при което загубите от закръгляния са много по-малки, отколкото когато се използва двоично представяне. Реалните числа от тип decimal са изключително удобни за пресмятания с пари – изчисляване на приходи, задължения, данъци, лихви и т.н.

Следва пример, в който декларираме променлива от тип decimal и й присвояваме стойност:

|  |
| --- |
| decimal decimalPI = 3.14159265358979323846m;  Console.WriteLine(decimalPI); // 3.14159265358979323846 |

Числото decimalPI, което декларирахме от тип **decimal**, не е закръглено дори и с един знак, тъй като го зададохме с точност 21 знака, което се побира в типа **decimal** без закръгляне.

Много голямата точност и липсата на аномалии при пресмятанията (каквито има при float и double) прави типа **decimal** много подходящ за финансови изчисления, където точността е критична.

|  |  |
| --- | --- |
|  | Въпреки по-малкия си обхват, типът **decimal** запазва точ­ност за всички десетични числа, които може да побере! Това го прави много подходящ за прецизни сметки, най-често финансови изчисления. |

Основната разлика между реалните числа с плаваща запетая и реалните числа с десетична точност е в **точността на пресмятанията** и в степента, до която те закръглят съхраняваните стойности. Типът double позволява работа с много големи стойности и стойности много близки до нулата, но за сметка на точността и неприятни грешки от закръгляне. Типът decimal има по-малък обхват, но гарантира голяма точност при пресмятанията и липса на аномалии с десетичните числа.

|  |  |
| --- | --- |
|  | Ако извършвате пресмятания с пари, използвайте типа **decimal**, а не **float** или **double**. В противен случай може да се натъкнете на неприятни аномалии при пресмятанията и грешки в изчисленията! |

Тъй като всички изчисления с данни от тип decimal се извършват софтуерно (а не директно на ниско ниво в микропроцесора), изчисленията с този тип са от порядъка на **десетки пъти по-бавни**, отколкото същите изчисления с double, така че ползвайте този тип внимателно. Можете да проверите това като извършите например 10 милиона умножения с decimal и double и сравните времето за изпълнение.

### Булев тип

Булевият тип се декларира с ключовата дума bool. Той има две стойности, които може да приема – true и false. Стойността по подразбиране е false. Използва се най-често за съхраняване на резултата от изчислява­нето на логически изрази.

#### Булев тип – пример

Нека разгледаме един пример, в който декларираме няколко променливи от вече познатите ни типове, инициализираме ги, извършваме сравнения върху тях и отпечатваме резултатите на конзолата:

|  |
| --- |
| // Declare some variables  int a = 1;  int b = 2;  // Which one is greater?  bool greaterAB = (a > b);  // Is 'a' equal to 1?  bool equalA1 = (a == 1);  // Print the results on the console  if (greaterAB)  {  Console.WriteLine("A > B");  }  else  {  Console.WriteLine("A <= B");  }  Console.WriteLine("greaterAB = " + greaterAB);  Console.WriteLine("equalA1 = " + equalA1);  // Console output:  // A <= B  // greaterAB = False  // equalA1 = True |

В примера декларираме две променливи от тип int, сравняваме ги и присвояваме резултата на променливата от булев тип greaterAB. Анало­гично извършваме и за променливата equalA1. Ако променливатаgreaterAB е true, на конзолата се отпечатва А>B, в противен случай се отпечатва A<=B.

### Символен тип

Символният тип представя единичен символ (16-битов номер на знак от Unicode таблицата). Той се декларира с ключо­вата дума char в езика C#. **Unicode таблицата** е технологичен стандарт, който съпоставя цяло число или поредица от няколко цели числа на всеки знак от човешките писмености по света (всички езици и техните азбуки). Повече за Unicode таблицата можем да прочетем в темата "[Символни низове](#_Глава_13._Символни)". Минималната стойност, която може да заема типът char, е 0, а максималната – 65535. Стойностите от тип char представляват букви или други символи и се ограждат в апострофи.

#### Символен тип – пример

Нека разгледаме един пример, в който декларираме една променлива от тип char, инициализираме я със стойност 'a', след това с 'b' и 'A' и отпечатваме Unicode стойностите на тези букви на конзолата:

|  |
| --- |
| // Declare a variable  char symbol = 'a';  // Print the results on the console  Console.WriteLine("The code of '" + symbol + "' is: " + (int)symbol);  symbol = 'b';  Console.WriteLine("The code of '" + symbol + "' is: " + (int)symbol);  symbol = 'A';  Console.WriteLine("The code of '" + symbol + "' is: " + (int)symbol);  // Console output:  // The code of 'a' is: 97  // The code of 'b' is: 98  // The code of 'A' is: 65 |

### Символни низове (стрингове)

Символните низове представляват **поредица от символи**. Декларират се с ключо­вата дума string в C#. Стойността им по подразбиране е null. Стринговете се ограждат в двойни кавички. Върху тях могат да се извършват различни текстообработващи операции: конкатениране (до­лепване един до друг), разделяне по даден разделител, търсене, знако­заместване и други. Повече информация за текстообработката можем да прочетем в темата "[Символни низове](#_Глава_13._Символни)", в която детайлно е обяснено какво е символен низ, за какво служи и как да го използваме.

#### Символни низове – пример

Нека разгледаме един пример, в който декларираме няколко променливи от тип символен низ, инициализираме ги и отпечатваме стойностите им на конзолата:

|  |
| --- |
| // Declare some variables  string firstName = "Ivan";  string lastName = "Ivanov";  string fullName = firstName + " " + lastName;  // Print the results on the console  Console.WriteLine("Hello, " + firstName + "!");  Console.WriteLine("Your full name is " + fullName + ".");  // Console output:  // Hello, Ivan!  // Your full name is Ivan Ivanov. |

### Обектен тип

Обектният тип е специален тип, който се явява **родител** на всички други типове в .NET Framework. Декларира се с клю­човата дума оbject и може да приема стойности от всеки друг тип. Той представлява референтен тип, т.е. указател (адрес) към област от паметта, която съхранява неговата стойност.

#### Използване на обекти – пример

Нека разгледаме един пример, в който декларираме няколко променливи от обектен тип, инициализираме ги и отпечатваме стойностите им на конзолата:

|  |
| --- |
| // Declare some variables  object container1 = 5;  object container2 = "Five";  // Print the results on the console  Console.WriteLine("The value of container1 is: " + container1);  Console.WriteLine("The value of container2 is: " + container2);  // Console output:  // The value of container1 is: 5  // The value of container2 is: Five |

Както се вижда от примера, в променлива от тип object можем да запишем стойност от **всеки друг тип**. Това прави обектния тип универ­сален контейнер за данни.

### Нулеви типове (Nullable Types)

**Нулевите типове** (**nullable types**) представляват специфични обвивки (**wrappers**) около стойностните типове (като int, double и bool), които позволяват в тях да бъде записвана null стойност. Това дава възможност в типове, които по принцип не допускат липса на стойност (т.е. стойност null), все пак да могат да бъдат използвани като **референтни типове** и да приемат както нормални стойности, така и специалната стойност null.

Обвиването на даден тип като нулев става по два начина:

|  |
| --- |
| Nullable<int> i1 = null;  int? i2 = i1; |

Двете декларации са еквивалентни. По-лесният начин е да се добави въпросителен знак (?) след типа, например int?, a по-трудният е да се използва Nullable<…> синтаксиса.

Нулевите типове са референтни типове, т.е. представляват референция към обект в динамичната памет, който съдържа стойността им. Те могат да имат или нямат стойност и могат да бъдат използвани както нормалните примитивни типове, но с някои особености, които ще илюстрираме в следващия пример:

|  |
| --- |
| int i = 5;  int? ni = i;  Console.WriteLine(ni); // 5  // i = ni; // this will fail to compile  Console.WriteLine(ni.HasValue); // True  i = ni.Value;  Console.WriteLine(i); // 5  ni = null;  Console.WriteLine(ni.HasValue); // False  //i = ni.Value; // System.InvalidOperationException  i = ni.GetValueOrDefault();  Console.WriteLine(i); // 0 |

От примера е видно, че на променлива от нулев тип (int?) може да се присвои директно стойност от ненулев тип (int), но обратното не е директно възможно. За целта може да се използва свойството на нулевите типове Value, което връща стойността записана в нулевия тип или предизвиква грешка (InvalidOperationException) по време на изпълне­ние на програмата, ако стойност липсва. За да проверим дали промен­лива от нулев тип има стойност, можем да използваме булевото свойство HasValue. Ако искаме да вземем стойността на променлива от нулев тип или стойността за типа по подразбиране (най-често 0) в случай на null, можем да използваме метода GetValueOrDefault().

Нулевите типове се използват за съхраняване на информация, която **не е задължителна**. Например, ако искаме да запазим данните за един студент, като името и фамилията му са задължителни, а възрастта му не е задължителна, можем да използваме int? за възрастта:

|  |
| --- |
| string firstName = "Svetlin";  string lastName = "Nakov";  int? age = null; |

## Променливи

След като разгледахме основните типове данни в C#, нека видим как и за какво можем да ги използваме. За да работим с данни, **трябва да използваме променливи**. Вече се сблъскахме с променливите в примерите, но сега нека ги разгледаме по-подробно.

**Променливата** е контейнер на информация, който може да променя стойността си. Тя осигурява възможност за:

* запазване на информация;
* извличане на запазената информация;
* модифициране на запазената информация.

Програмирането на C# е свързано с постоянно използване на променливи, в които се съхраняват и обработват данни.

### Характеристики на променливите

Променливите се характеризират с:

* име (идентификатор), например age;
* тип (на запазената в тях информация), например int;
* стойност (запазената информация), например 25.

Една променлива представлява именувана област от паметта, в която е записана стойност от даден тип, **достъпна в програмата по своето име**. Променливите могат да се пазят непосредствено в работната памет на програмата (в стека) или в динамичната памет, в която се съхраняват по-големи обекти (например символни низове и масиви). Примитивните типове данни (числа, char, bool) се наричат **стойностни типове**, защото пазят непосредствено своята стойност в стека на програмата. **Референт­ните типове данни** (например стрингове, обекти и масиви) пазят като стойност адрес от динамичната памет, където е записана стойността им. Те могат да се заделят и освобождават динамично, т.е. размерът им не е предварително фиксиран, както при стойностните типове. Повече информация за стойностите и референтните типове данни сме предвидили в секцията "[Стойностни и референтни типове](#_Стойностни_и_референтни)".

#### Именуване на променлива – правила

Когато искаме компилаторът да задели област в паметта за някаква информация, използвана в програмата ни, **трябва да й зададем име**. То служи като идентификатор и позволява да се реферира нужната ни област от паметта.

Името на променливите може да бъде всякакво по наш избор, но трябва да следва определени правила:

* Имената на променливите се образуват от буквите a-z, A-Z, цифрите 0-9, както и символа '\_'.
* Имената на променливите не може да започват с цифра.
* Имената на променливите не могат да съвпадат със **служебна дума** (**keyword**) от езика C#.

В следващата таблица са дадени всички служебни думи в C#. Някои от тях вече са ни известни, а с други ще се запознаем в следващите глави от книгата:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| [abstract](http://msdn.microsoft.com/en-us/library/sf985hc5%28VS.100%29.aspx) | [as](http://msdn.microsoft.com/en-us/library/cscsdfbt%28VS.100%29.aspx) | [base](http://msdn.microsoft.com/en-us/library/hfw7t1ce%28VS.100%29.aspx) | [bool](http://msdn.microsoft.com/en-us/library/c8f5xwh7%28VS.100%29.aspx) | [break](http://msdn.microsoft.com/en-us/library/adbctzc4%28VS.100%29.aspx) |
| [byte](http://msdn.microsoft.com/en-us/library/5bdb6693%28VS.100%29.aspx) | [case](http://msdn.microsoft.com/en-us/library/06tc147t%28VS.100%29.aspx) | [catch](http://msdn.microsoft.com/en-us/library/0yd65esw%28VS.100%29.aspx) | [char](http://msdn.microsoft.com/en-us/library/x9h8tsay%28VS.100%29.aspx) | [checked](http://msdn.microsoft.com/en-us/library/74b4xzyw%28VS.100%29.aspx) |
| [class](http://msdn.microsoft.com/en-us/library/0b0thckt%28VS.100%29.aspx) | [const](http://msdn.microsoft.com/en-us/library/e6w8fe1b%28VS.100%29.aspx) | [continue](http://msdn.microsoft.com/en-us/library/923ahwt1%28VS.100%29.aspx) | [decimal](http://msdn.microsoft.com/en-us/library/364x0z75%28VS.100%29.aspx) | [default](http://msdn.microsoft.com/en-us/library/25tdedf5%28VS.100%29.aspx) |
| [delegate](http://msdn.microsoft.com/en-us/library/900fyy8e%28VS.100%29.aspx) | [do](http://msdn.microsoft.com/en-us/library/370s1zax%28VS.100%29.aspx) | [double](http://msdn.microsoft.com/en-us/library/678hzkk9%28VS.100%29.aspx) | [else](http://msdn.microsoft.com/en-us/library/5011f09h%28VS.100%29.aspx) | [enum](http://msdn.microsoft.com/en-us/library/sbbt4032%28VS.100%29.aspx) |
| [event](http://msdn.microsoft.com/en-us/library/8627sbea%28VS.100%29.aspx) | [explicit](http://msdn.microsoft.com/en-us/library/xhbhezf4%28VS.100%29.aspx) | [extern](http://msdn.microsoft.com/en-us/library/e59b22c5%28VS.100%29.aspx) | [false](http://msdn.microsoft.com/en-us/library/67bxt5ee%28VS.100%29.aspx) | [finally](http://msdn.microsoft.com/en-us/library/zwc8s4fz%28VS.100%29.aspx) |
| [fixed](http://msdn.microsoft.com/en-us/library/f58wzh21%28VS.100%29.aspx) | [float](http://msdn.microsoft.com/en-us/library/b1e65aza%28VS.100%29.aspx) | [for](http://msdn.microsoft.com/en-us/library/ch45axte%28VS.100%29.aspx) | [foreach](http://msdn.microsoft.com/en-us/library/ttw7t8t6%28VS.100%29.aspx) | [goto](http://msdn.microsoft.com/en-us/library/13940fs2%28VS.100%29.aspx) |
| [if](http://msdn.microsoft.com/en-us/library/5011f09h%28VS.100%29.aspx) | [implicit](http://msdn.microsoft.com/en-us/library/z5z9kes2%28VS.100%29.aspx) | [in](http://msdn.microsoft.com/en-us/library/ttw7t8t6%28VS.100%29.aspx) | [in (generic)](http://msdn.microsoft.com/en-us/library/dd469484%28VS.100%29.aspx) | [int](http://msdn.microsoft.com/en-us/library/5kzh1b5w%28VS.100%29.aspx) |
| [interface](http://msdn.microsoft.com/en-us/library/87d83y5b%28VS.100%29.aspx) | [internal](http://msdn.microsoft.com/en-us/library/7c5ka91b%28VS.100%29.aspx) | [is](http://msdn.microsoft.com/en-us/library/scekt9xw%28VS.100%29.aspx) | [lock](http://msdn.microsoft.com/en-us/library/c5kehkcz%28VS.100%29.aspx) | [long](http://msdn.microsoft.com/en-us/library/ctetwysk%28VS.100%29.aspx) |
| [namespace](http://msdn.microsoft.com/en-us/library/z2kcy19k%28VS.100%29.aspx) | [new](http://msdn.microsoft.com/en-us/library/51y09td4%28VS.100%29.aspx) | [null](http://msdn.microsoft.com/en-us/library/edakx9da%28VS.100%29.aspx) | [object](http://msdn.microsoft.com/en-us/library/9kkx3h3c%28VS.100%29.aspx) | [operator](http://msdn.microsoft.com/en-us/library/s53ehcz3%28VS.100%29.aspx) |
| [out](http://msdn.microsoft.com/en-us/library/t3c3bfhx%28VS.100%29.aspx) | [out (generic)](http://msdn.microsoft.com/en-us/library/dd469487%28VS.100%29.aspx) | [override](http://msdn.microsoft.com/en-us/library/ebca9ah3%28VS.100%29.aspx) | [params](http://msdn.microsoft.com/en-us/library/w5zay9db%28VS.100%29.aspx) | [private](http://msdn.microsoft.com/en-us/library/st6sy9xe%28VS.100%29.aspx) |
| [protected](http://msdn.microsoft.com/en-us/library/bcd5672a%28VS.100%29.aspx) | [public](http://msdn.microsoft.com/en-us/library/yzh058ae%28VS.100%29.aspx) | [readonly](http://msdn.microsoft.com/en-us/library/acdd6hb7%28VS.100%29.aspx) | [ref](http://msdn.microsoft.com/en-us/library/14akc2c7%28VS.100%29.aspx) | [return](http://msdn.microsoft.com/en-us/library/1h3swy84%28VS.100%29.aspx) |
| [sbyte](http://msdn.microsoft.com/en-us/library/d86he86x%28VS.100%29.aspx) | [sealed](http://msdn.microsoft.com/en-us/library/88c54tsw%28VS.100%29.aspx) | [short](http://msdn.microsoft.com/en-us/library/ybs77ex4%28VS.100%29.aspx) | [sizeof](http://msdn.microsoft.com/en-us/library/eahchzkf%28VS.100%29.aspx) | [stackalloc](http://msdn.microsoft.com/en-us/library/cx9s2sy4%28VS.100%29.aspx) |
| [static](http://msdn.microsoft.com/en-us/library/98f28cdx%28VS.100%29.aspx) | [string](http://msdn.microsoft.com/en-us/library/362314fe%28VS.100%29.aspx) | [struct](http://msdn.microsoft.com/en-us/library/ah19swz4%28VS.100%29.aspx) | [switch](http://msdn.microsoft.com/en-us/library/06tc147t%28VS.100%29.aspx) | [this](http://msdn.microsoft.com/en-us/library/dk1507sz%28VS.100%29.aspx) |
| [throw](http://msdn.microsoft.com/en-us/library/1ah5wsex%28VS.100%29.aspx) | [true](http://msdn.microsoft.com/en-us/library/eahhcxk2%28VS.100%29.aspx) | [try](http://msdn.microsoft.com/en-us/library/0yd65esw%28VS.100%29.aspx) | [typeof](http://msdn.microsoft.com/en-us/library/58918ffs%28VS.100%29.aspx) | [uint](http://msdn.microsoft.com/en-us/library/x0sksh43%28VS.100%29.aspx) |
| [ulong](http://msdn.microsoft.com/en-us/library/t98873t4%28VS.100%29.aspx) | [unchecked](http://msdn.microsoft.com/en-us/library/a569z7k8%28VS.100%29.aspx) | [unsafe](http://msdn.microsoft.com/en-us/library/chfa2zb8%28VS.100%29.aspx) | [ushort](http://msdn.microsoft.com/en-us/library/cbf1574z%28VS.100%29.aspx) | [using](http://msdn.microsoft.com/en-us/library/zhdeatwt%28VS.100%29.aspx) |
| [virtual](http://msdn.microsoft.com/en-us/library/9fkccyh4%28VS.100%29.aspx) | [void](http://msdn.microsoft.com/en-us/library/yah0tteb%28VS.100%29.aspx) | [volatile](http://msdn.microsoft.com/en-us/library/x13ttww7%28VS.100%29.aspx) | [while](http://msdn.microsoft.com/en-us/library/2aeyhxcd%28VS.100%29.aspx) |  |

#### Именуване на променливи – примери

Позволени имена:

* name
* first\_Name
* \_name1

Непозволени имена (ще доведат до грешка при компилация):

* 1 (цифра)
* if (служебна дума)
* 1name (започва с цифра)

#### Именуване на променливи – препоръки

Ще дадем някои препоръки за именуване, тъй като не всички позволени от компилатора имена са подходящи за нашите променливи.

* Имената трябва да са описателни и да обясняват за какво служи дадената променлива. Например за име на човек подхо­дящо име е personName, а неподходящо име е a37.
* Трябва да се използват само латински букви. Въпреки че кирили­цата е позволена от компилатора, не е добра практика тя да бъде използвана в имената на променливите и останалите идентифика­тори от програ­мата.
* В C# e прието променливите да започват винаги с малка буква и да съдържат малки букви, като всяка следваща дума в тях започва с главна буква. Например правилно име е firstName, a не firstname или first\_name. Използването на символа \_ в имената на промен­ливите се счита за лош стил на именуване.
* Името на променливите трябва да не е нито много дълго, нито много късо – просто трябва да е ясно за какво служи променливата в контекста, в който се използва.
* Трябва да се внимава за главни и малки букви, тъй като C# прави разлика между тях. Например age и Age са различни променливи.

Ето няколко примера за добре именувани променливи:

* firstName
* age
* startIndex
* lastNegativeNumberIndex

Ето няколко примера за лошо именувани променливи (макар и имената да са коректни от гледна точка на компилатора на C#):

* \_firstName (започва с \_)
* last\_name (съдържа \_)
* AGE (изписана е с главни букви)
* Start\_Index (започва с главна буква и съдържа \_)
* lastNegativeNumber\_Index (съдържа \_)

Променливите трябва да имат име, което обяснява накратко за какво служат. Когато една променлива е именувана с неподходящо име, това силно затруднява четенето на програмата и нейната следваща промя­на (след време, когато сме забравили как работи тя). Повече за правилното именуване на променливите ще научите в главата "[Качествен програмен код](#_Глава_21._Качествен)".

|  |  |
| --- | --- |
|  | Стремете се винаги да именувате променливите с кратки, но достатъчно ясни имена. Следвайте правилото, че от името на променливата трябва да става ясно за какво се използва, т.е. името трябва да отговаря на въпроса "каква стойност съхранява тази променлива". Ако това не е изпълнено, потърсете по-добро име. |

### Деклариране на променливи

Когато декларираме променлива, извършваме следните действия:

* задаваме нейния тип (например int);
* задаваме нейното име (идентификатор, например age);
* можем да зададем начална стойност (например 25), но това не е задължително.

Синтаксисът за деклариране на променливи в C# е следният:

|  |
| --- |
| <тип данни> <идентификатор> [= <инициализация>] |

Ето един пример за деклариране на променливи:

|  |
| --- |
| string name;  int age; |

### Присвояване на стойност

Присвояването на стойност на променлива представлява задаване на стойност, която да бъде записана в нея. Тази операция се извършва чрез **оператора за присвояване '**=**'**. От лявата страна на оператора се изписва име на променлива, а от дясната страна – новата й стойност.

Ето един пример за присвояване на стойност на променливи:

|  |
| --- |
| name = "Svetlin Nakov";  age = 25; |

### Инициализация на променливи

Терминът **инициализация** в програмирането означава задаване на начална стойност. Задавайки стойност на променливите в момента на тяхното деклариране, ние всъщност ги инициали­зираме.

Всеки тип данни в C# има **стойност по подразбиране** (инициализация по подразбиране), която се използва, когато за дадена променлива не бъде изрично зададена стойност. Можем да си припомним стойностите по под­разбиране за типовете, с които се запознахме, от следващата таблица:

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип данни** | **Стойност по подразбиране** | **Тип данни** | **Стойност по подразбиране** |
| sbyte | 0 | float | 0.0f |
| byte | 0 | double | 0.0d |
| short | 0 | decimal | 0.0m |
| ushort | 0 | bool | false |
| int | 0 | char | '\u0000' |
| uint | 0u | string | null |
| long | 0L | object | null |
| ulong | 0u |  |  |

Нека обобщим как декларираме променливи, как ги инициализираме и как им присвояваме стойности в следващия пример:

|  |
| --- |
| // Declare and initialize some variables  byte centuries = 20;  ushort years = 2000;  decimal decimalPI = 3.141592653589793238m;  bool isEmpty = true;  char symbol = 'a';  string firstName = "Ivan";  symbol = (char)5;  char secondSymbol;  // Here we use an already initialized variable and reassign it  secondSymbol = symbol; |

## Стойностни и референтни типове

Типовете данни в C# са 2 вида: **стойностни** и **референтни**.

**Стойностните типове (value types)** се съхраняват в стека за изпъл­нение на програ­мата и съдържат директно стойността си. Стойностни са примитивните числови типове, символният тип и булевият тип: sbyte, byte, short, ushort, int, long, ulong, float, double, decimal, char, bool. Те се освобождават при излизане от обхват, т.е. когато блокът с код, в който са дефинирани, завърши изпълнението си. Например една променлива, декларирана в метода Main() на програмата, се пази в стека докато програмата завърши изпълнението на този метод, т.е. докато не завърши.

**Референтните типове (reference types)** съдържат в стека за изпъл­нение на програмата референция (адрес) към **динамичната памет** (**heap**), където се съхранява тяхната стойност. Референцията представ­лява **указател** (адрес на клетка от паметта), сочещ реалното местополо­жение на стой­ността в динамичната памет. Пример за стойност на адрес в стека за изпълнение е 0x00AD4934. Референцията има тип и може да съдържа като стойност само обекти от своя тип, т.е. тя представлява строго типизиран указател. Всички референтни типове могат да получават стойност null. Това е специална служебна стойност, която означава, че липсва стойност.

Референтните типове заделят динамична памет при създаването си и се освобождават по някое време от системата за **почистване на паметта (garbage collector)**, когато тя установи, че вече не се използват от програмата. Не е известно точно в кой момент дадена референтна променлива ще бъде освободена от garbage collector, тъй като това зависи от натоварването на паметта и от други фактори. Тъй като заделянето и освобождаването на памет е бавна операция, може да се каже, че референтните типове са по-бавни от стойностните.

Тъй като референтните типове данни се заделят и освобождават динамично по време на изпълнение на програмата, техният размер може да не е предварително известен. Например в променлива от тип string могат да бъдат записвани текстови данни с различна дължина. Реално текстовата стойност на типа string се записва в **динамичната памет** и може да заема различен обем (брой байтове), а в променливата от тип string се записва **неговият адрес**.

Референтни типове са всички класове, масивите и интерфейсите, напри­мер типовете: object, string, byte[]. С класовете, обектите, символните низове, масивите и интерфейсите ще се запознаем в следващите глави на книгата. Засега е достатъчно да знаете, че всички типове, които не са стойностни, са референтни и се разполагат в дина­мичната памет.

### Стойностни и референтни типове в паметта

Нека илюстрираме с един пример как се представят в паметта стойност­ните и референтните типове. Нека е изпълнен следния програмен код:

|  |
| --- |
| int i = 42;  char ch = 'A';  bool result = true;  object obj = 42;  string str = "Hello";  byte[] bytes = { 1, 2, 3 }; |

В този момент променливите са разположени в паметта по следния начин:



Ако сега изпълним следния код, който променя стойностите на промен­ливите, ще видим какво се случва с паметта при промяна на стойностни и референтни типове:

|  |
| --- |
| i = 0;  ch = 'B';  result = false;  obj = null;  str = "Bye";  bytes[1] = 0; |

След тези промени променливите и техните стойности са разположени в паметта по следния начин:



Както можете да забележите от фигурата, при промяна на стойностен тип (i=0) се променя директно **стойността му в стека**. При промяна на референтен тип нещата са по-различни: променя се директно стойността му в динамичната памет (bytes[1]=0). Променливата, която държи рефе­рен­цията, **остава непроменена** (0x00190d11). При запис­ване на стойност null в референтен тип съответната референция се разкача от стойността си и променливата остава без стойност (obj=null).

При присвояване на нова стойност на обект (променлива от референтен тип) новият обект се заделя в динамичната стойност, а старият обект остава свободен (нерефериран). Референцията се пренасочва към новия обект (str="Bye"), а старите обекти ("**Hello**"), понеже не се използват, ще бъдат почистени по някое време от системата за почистване на паметта (garbage collector).

## Литерали

Примитивните типове, с които се запознахме вече, са специални типове данни, вградени в езика C#. Техните стойности, зададени в сорс кода на програмата, се наричат **литерали**. С един пример ще ни стане по-ясно:

|  |
| --- |
| bool result = true;  char capitalC = 'C';  byte b = 100;  short s = 20000;  int i = 300000; |

В примера литерали са true, 'C', 100, 20000 и 300000. Те представляват стойности на променливи, зададени непосредствено в сорс кода на програмата.

### Видове литерали

В езика C# съществуват няколко вида литерали:

* булеви
* целочислени
* реални
* символни
* низови
* обектният литерал null

### Булеви литерали

Булевите литерали са:

* true
* false

Когато присвояваме стойност на променлива от тип bool, можем да използваме единствено някоя от тези две стойности или израз от булев тип (който се изчислява до true или false).

#### Булеви литерали – пример

Ето пример за декларация на променлива от тип bool и присвояване на стойност, което представлява булевият литерал true:

|  |
| --- |
| bool result = true; |

### Целочислени литерали

Целочислените литерали представляват поредица от цифри, знак (+, -), окончания и представки. С помощта на представки можем да представим целите числа в сорс кода на програмата в десетичен или шестнадесетичен формат. Повече информация за различните бройни системи можем да получим в темата "[Бройни системи](#_Глава_8._Бройни_1)". В целочисле­ните литерали могат да участват и следните представки и окончания:

* "0x" и "0X" като представки означават шестнадесетична стойност, например 0xA8F1;
* 'l' и 'L' като окончания означават данни от тип **long**, например 357L.
* 'u' и 'U' като окончания означават данни от тип **uint** или **ulong**, например 112u.

По подразбиране (ако не бъде използванo никакво окончание) целочис­лените литерали са от тип int.

#### Целочислени литерали – примери

Ето няколко примера за използване на целочислени литерали:

|  |
| --- |
| // The following variables are initialized with the same value  int numberInDec = 16;  int numberInHex = 0x10;  // This will cause an error, because the value 234L is not int  int longInt = 234L; |

### Реални литерали

Реалните литерали, представляват поредица от цифри, знак (+, -), окончания и символа за десетична запетая. Използваме ги за стойности от тип float, double и decimal. Реалните литерали могат да бъдат предста­вени и в експоненциален формат. При тях се използват още следните означения:

* 'f' и 'F' като окончания означават данни от тип float;
* 'd' и 'D' като окончания означават данни от тип double;
* 'm' и '**M**' като окончания означават данни от тип decimal;
* 'e' означава експонента, например "e-5" означава цялата част да се умножи по 10-5.

По подразбиране (ако липсва окончание) реалните числа са от тип double.

#### Реални литерали – примери

Ето няколко примера за използване на реални литерали:

|  |
| --- |
| // The following is the correct way of assigning a value:  float realNumber = 12.5f;  // This is the same value in exponential format:  realNumber = 1.25e+1f;  // The following causes an error, because 12.5 is double  float realNumber = 12.5; |

### Символни литерали

Символните литерали представляват единичен символ, ограден в апос­трофи (единични кавички). Използваме ги за задаване на стойности от тип char. Стойността на символните литерали може да бъде:

* символ, например 'A';
* код на символ, например '\u0065';
* escaping последователност;

#### Екранирани (Escaping) последователности

Понякога се налага да работим със символи, които не са изписани на клавиатурата, или със символи, които имат специално значение, като например символът "нов ред". Те **не могат да се изпишат директно в сорс кода** на програмата и за да ги ползваме, са ни необходими **специални техники**, които ще разгледаме сега.

**Escaping последователностите** са литерали, които представляват пос­ледо­вателност от специални символи, които задават символ, който по някаква причина не може да се изпише директно в сорс кода. Такъв е например символът за нов ред. Те ни дават заобиколен начин (**escaping**) да напишем някакъв символ на екрана и затова се наричат още **контролиращи комбинации от символи (escaping sequences)**.

Примери за символи, които не могат да се изпишат директно в сорс кода, има много: двойна кавичка, табулация, нов ред, наклонена черта и други. Ето някои от най-често използваните escaping последователности:

* \' – единична кавичка
* \" – двойна кавичка
* \\– лява наклонена черта
* \n – нов ред
* \t– отместване (табулация)
* \uXXXX– символ, зададен с Unicode номера си, например \u03A7.

Символът \ (лява наклонена черта) се нарича още **екраниращ символ** (**escaping character**), защото той позволява да се изпишат на екрана символи, които имат специално значение или действие и не могат да се изпишат директно в сорс кода.

#### Escaping последователности – примери

Ето няколко примера за символни литерали:

|  |
| --- |
| // An ordinary symbol  char symbol = 'a';  Console.WriteLine(symbol);  // Unicode symbol code in a hexadecimal format  symbol = '\u003A';  Console.WriteLine(symbol);  // Assigning the single quote symbol (escaped as \')  symbol = '\'';  Console.WriteLine(symbol);  // Assigning the backslash symbol(escaped as \\)  symbol = '\\';  Console.WriteLine(symbol);  // Console output:  // a  // :  // '  // \ |

### Литерали за символен низ

Литералите за символен низ се използват за данни от тип string. Те представляват последователност от символи, заградена в двойни кавички.

За символните низове важат всички [правила за escaping](#_Escaping_последователности), които вече разгледахме за литералите от тип char.

Символните низове могат да се изписват предхождани от символа @, който задава **цитиран низ**. В цитираните низове правилата за escaping не важат, т.е. символът \ означава \ и не е екраниращ символ. В цитираните низове кавичката " може да се екранира с двойна "", а всички останали символи се възприемат буквално, дори новият ред. Цитираните низове се използват често пъти при задаване на имена на пътища във файловата система.

#### Литерали за символен низ – примери

Ето няколко примера за използване на литерали от тип символен низ:

|  |
| --- |
| string quotation = "\"Hello, Jude\", he said.";  Console.WriteLine(quotation);  string path = "C:\\Windows\\Notepad.exe";  Console.WriteLine(path);  string verbatim = @"The \ is not escaped as \\.  I am at a new line.";  Console.WriteLine(verbatim);  // Console output:  // "Hello, Jude", he said.  // C:\Windows\Notepad.exe  // The \ is not escaped as \\.  // I am at a new line. |

Повече за символните низове ще намерим в темата "[Символни низове](#_Глава_13._Символни)".

## Упражнения

1. Декларирайте няколко променливи, като изберете за всяка една най-подходящия от типовете sbyte, byte, short, ushort, int, uint, long и ulong, за да им присвоите следните стойности: 52130, -115, 4825932, 97, -10000, 20000; 224; 970700000; 112; -44; -1000000; 1990; 123456789123456789.
2. Кои от следните стойности може да се присвоят на променливи от тип float, double и decimal: 34.567839023; 12.345; 8923.1234857; 3456.091124875956542151256683467?
3. Напишете програма, която изчислява вярно променливи с плаваща запетая с точност до 0.000001.
4. Инициализирайте променлива от тип int със стойност 256 в шестна­десетичен формат (256 е 100 в бройна система с основа 16).
5. Декларирайте променлива от тип char и присвоете като стойност символа, който има Unicode код 72 (използвайте калкулатора на Windows за да намерите шестнайсетичното представяне на 72).
6. Декларирайте променлива isMale от тип boolи присвоете стойност на последната в зависимост от вашия пол.
7. Декларирайте две променливи от тип string със стойности "Hello" и "World". Декларирайте променлива от тип object. Присвоете на тази променлива стойността, която се получава от конкатенацията на двете стрингови променливи (добавете интервал, ако е необходимо). Отпе­чатайте променливата от тип object.
8. Декларирайте две променливи от тип string и им присвоете стойности "Hello" и "World". Декларирайте променлива от тип object и и присвоете стойността на конкатенацията на двете променливи от тип string (не изпускайте интервала по средата). Декларирайте трета променлива от тип string и я инициализирайте със стойността на променливата от тип object ( трябва да използвате type casting).
9. Декларирайте две променливи от тип string и им присвоете стойност "The "use" of quotations causes difficulties." (без първите и последни кавички). В едната променлива използвайте quoted string, а в другата не го използвайте.
10. Напишете програма, която принтира фигура във формата на сърце със знака "o".
11. Напишете програма, която принтира на конзолата равнобедрен триъгълник, като страните му са очертани от символа "©".
12. Фирма, занимаваща се с маркетинг, иска да пази запис с данни на нейните служители. Всеки запис трябва да има следната характе­ристика – първо име, фамилия, възраст, пол (‘м’ или ‘ж’) и уникален номер на служителя (27560000 до 27569999). Декларирайте необходи­мите променливи нужни, за да се запази информацията за един служи­тел, като използвате подходящи типове данни и описателни имена.
13. Декларирайте две променливи от тип int. Задайте им стойности съответно 5 и 10. Разменете стойностите им и ги отпечатайте.

## Решения и упътвания

1. Погледнете размерността на числените типове.
2. Имайте предвид броя символи след десетичния знак. Направете справка в таблицата с размерите на типовете float, double и decimal.
3. Използвайте типа данни decimal.
4. Вижте секцията за [целочислени литерали](#_Целочислени_литерали). За да преобразувате лесно числата в различна бройна система, използвайте вградения в Windows калкулатор. За шестнайсетично представяне на литерал използвайте префикса 0x.
5. Вижте секцията за [целочислени литерали](#_Целочислени_литерали).
6. Вижте секцията за [булеви променливи](#_Булев_тип).
7. Вижте секциите за [символни низове](#_Символни_низове_(стрингове)) и за [обектен тип](#_Обектен_тип) данни.
8. Вижте секциите за [символни низове](#_Символни_низове_(стрингове)) и за [обектен тип](#_Обектен_тип) данни.
9. Погледнете частта за [символни литерали](#_Символни_литерали). Необходимо е да използ­вате символа за escaping (наклонена черта "\").
10. Използвайте Console.WriteLine(…) като използвате само символа ‘о’ и интервали.
11. Използвайте Console.WriteLine(…) като използвате само знака © и интервали. Използвайте Windows Character Map, за да намерите Unicode кода на знака "©".
12. За имената използвайте тип string, за пола използвайте тип char (имаме само един символ м/ж), а за уникалния номер и възрастта използвайте подходящ целочислен тип.
13. Използвайте трета временна променлива за размяната на променливи. За целочислените променливи е възможно и друго решение, което не използва трета променлива. Например, ако имаме 2 променливи a и b:

|  |
| --- |
| int a = 2;  int b = 3;  a = a + b;  b = a - b;  a = a - b; |