CSHARP BASICS

Съдържание

[1. Типове данни 5](#_Toc394009778)

[Видове 5](#_Toc394009779)

[Извличане на минимална и максимална стойност на типове данни 5](#_Toc394009780)

[2. Променливи 5](#_Toc394009781)

[Именуване на променливи 5](#_Toc394009782)

[Деклариране, присвояване на стойност и инициализация на променливи 6](#_Toc394009783)

[Деклариране 6](#_Toc394009784)

[Присвояване на стойност 6](#_Toc394009785)

[Инициализация 6](#_Toc394009786)

[3. Литерали 6](#_Toc394009787)

[Булеви 7](#_Toc394009788)

[Целочислени 7](#_Toc394009789)

[Реални 7](#_Toc394009790)

[Символни 7](#_Toc394009791)

[Екранирани (Escaping) последователности 7](#_Toc394009792)

[Литерали за символен низ 8](#_Toc394009793)

[4. Оператори: 8](#_Toc394009794)

[Категории оператори 8](#_Toc394009795)

[Аритметични оператори 9](#_Toc394009796)

[Логически оператори 9](#_Toc394009797)

[Побитови оператори 10](#_Toc394009798)

[Оператори за сравнение 10](#_Toc394009799)

[Оператори за присвояване 11](#_Toc394009800)

[Оператор за съединяване на низове 11](#_Toc394009801)

[Други оператори 11](#_Toc394009802)

[Приоритет на операторите в C# 12](#_Toc394009803)

[Преобразуване на типовете 12](#_Toc394009804)

[Неявно (implicit) преобразуване на типове 12](#_Toc394009805)

[Изрично (explicit) преобразуване на типове 13](#_Toc394009806)

[5. Вход и изход от конзолата 14](#_Toc394009807)

[Печатане на конзолата 14](#_Toc394009808)

[Console.Write(…) 14](#_Toc394009809)

[Console.WriteLine(…) 14](#_Toc394009810)

[Форматиран изход 14](#_Toc394009811)

[Indexкомпонента 14](#_Toc394009812)

[Alignmentкомпонента 14](#_Toc394009813)

[Стандартно дефинирани формати за числа 15](#_Toc394009814)

[Потребителскиформати за числа 15](#_Toc394009815)

[Стандартно дефинирани формати за дати 16](#_Toc394009816)

[Custom формати за дати 16](#_Toc394009817)

[Компоненти за енумерации 16](#_Toc394009818)

[Форматиращи низове и локализация 17](#_Toc394009819)

[Вход от конзолата 17](#_Toc394009820)

[Console.ReadLine() 17](#_Toc394009821)

[Console.Read() 17](#_Toc394009822)

[Console.ReadKey() 18](#_Toc394009823)

[Условно парсване на числа 18](#_Toc394009824)

[6. Условни конструкции 18](#_Toc394009825)

[If 18](#_Toc394009826)

[If - else 19](#_Toc394009827)

[Switch – case 19](#_Toc394009828)

[7. Цикли 19](#_Toc394009829)

[Оператор break 19](#_Toc394009830)

[Оператор continue 20](#_Toc394009831)

[While 20](#_Toc394009832)

[Do-while 20](#_Toc394009833)

[For 20](#_Toc394009834)

[For-цикъл с няколко променливи 21](#_Toc394009835)

[Foreach 21](#_Toc394009836)

[8. Масиви 21](#_Toc394009837)

[Едномерни масиви 21](#_Toc394009838)

[Деклариране на масив 21](#_Toc394009839)

[Създаване (заделяне) на масив 21](#_Toc394009840)

[Инициализация на масив 21](#_Toc394009841)

[Граници на масив 21](#_Toc394009842)

[Достъп до елементите на масив 21](#_Toc394009843)

[Четене на масив от конзолата 22](#_Toc394009844)

[Отпечатване на масив на конзолата 22](#_Toc394009845)

[Многомерни масиви 22](#_Toc394009846)

[Деклариране на многомерен масив 22](#_Toc394009847)

[Заделяне на многомерен масив 22](#_Toc394009848)

[Инициализация на двумерен масив 22](#_Toc394009849)

[Достъп до елементите на многомерен масив 22](#_Toc394009850)

[Извличане дължината на многомерен масив 23](#_Toc394009851)

[Четене на матрица от конзолата 23](#_Toc394009852)

[Отпечатване на матрица на конзолата 23](#_Toc394009853)

[9. Бройни системи 23](#_Toc394009854)

[Десетични числа 24](#_Toc394009855)

[Двоични числа 24](#_Toc394009856)

[Шестнайсетични числа 25](#_Toc394009857)

[Преминаване от една бройна система в друга 25](#_Toc394009858)

[От двоична към десетична бройна система 25](#_Toc394009859)

[От десетична към двоична бройна система 25](#_Toc394009860)

[От десетична към шестнайсетична бройна система 26](#_Toc394009861)

[От шестнайсетична към десетична бройна система 26](#_Toc394009862)

[Действия с двоични числа 26](#_Toc394009863)

[Побитово "и" 26](#_Toc394009864)

[Побитово "или" 27](#_Toc394009865)

[Побитово "изключващо и" 27](#_Toc394009866)

[Побитово отрицание 27](#_Toc394009867)

[10. Методи 27](#_Toc394009868)

[11. Листи 27](#_Toc394009869)

[12. Речници 27](#_Toc394009870)

[13. Полезно за изпита 27](#_Toc394009871)

# Типове данни

## 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**.



## Извличане на минимална и максимална стойност на типове данни



# Променливи

## Именуване на променливи

* 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.
* 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.
* Имената на променливите не могат да съвпадат със **служебна дума (keyword)** от езика C#.



## Деклариране, присвояване на стойност и инициализация на променливи

### Деклариране



### Присвояване на стойност



### Инициализация



# Литерали

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

## Булеви

* true
* false

## Целочислени

* **"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 използвана никакво окончание) целочис­лените литерали са от тип **int**.

## Реални

* **'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.

## Символни

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

### Екранирани (Escaping) последователности

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



## Литерали за символен низ

За символните низове важат всички правила за escaping, които вече разгледахме за литералите от тип char.



# Оператори:

## Категории оператори



### Аритметични оператори

* Аритметичните оператори **+** , **-** , **\*** са същите като в математика.
* **/** (деление) - има различно действие върху цели и реални числа
* 7 / 3 = 2
* 5.0 / 2 = 2.5
* 5.0 / 2.0 = 2.5
* **%** - остатъкът от целочислено делене на цели числа
* 7 % 3 = 1
* **++** (increment) - добавя единица към стойността на променливата
* **--** (decrement) - изважда единица към стойността на променливата

**Важно:** Когато използваме операторите ++ и -- като префикс (поставяме ги непосредствено преди променливата), първо се пресмята новата стойност, а после се връща резултата, докато при използването на операторите като постфикс (поставяме оператора непосредствено след променливата) първо се връща оригиналната стойност на операнда, а после се добавя или изважда единица към нея.

### Логически оператори

Логическите оператори приемат булеви стойности и връщат булев резултат (**true** или **false**).

* **&&** (И) - връща истина, само тогава, когато и двете променливи съдържат истина
* **||** (ИЛИ) - връща истина, когато поне един от операндите е истина
* **^** (изключващо ИЛИ )- сменя стойността на аргумента
* **!** (отрицание) - връща резултат true, когато само един от двата операнда има стойност true



### Побитови оператори

"И" (&), побитово "ИЛИ" (|), побитово отрицание (~) и изключващо "ИЛИ" (^).

* **&** (И) -
* **|** (ИЛИ) -
* **~** (отрицание) -
* **^** (изключващо ИЛИ) –
* **<<** - побитово изместване в ляво
* **>>** - побитово изместване в дясно





### Оператори за сравнение

* по-голямо (>)
* по-малко (<)
* по-голямо или равно (>=)
* по-малко или равно (<=)
* равенство (==)
* различие (!=)

Всички оператори за сравнение в C# са двуаргументни (приемат два операнда), а върнатият от тях резултат е булев (**true** или **false**).

### Оператори за присвояване

* оператори за присвояване: =



* комбинирани оператори за присвояване: += , -= , \*= , /= , %= , &= , |= , ^= , <<= , >>=



### Оператор за съединяване на низове

* **+** - слепва два или повече низа и връща резултата като нов низ. Ако поне един от аргументите в израза е от тип string, и има други операнди, които не са от тип string, то те автоматично ще бъдат преобразувани към тип string



### Други оператори

* **?:** - използва булевата стойност от един израз, за да определи кой от други два израза да бъде пресметнат и върнат като резултат



* операторът за достъп **"."** (точка) - използва се за достъп до член променливите или методите на даден клас или обект
* квадратни скоби **[]** - използват се за достъп до елементите на масив по индекс и затова се нарича още индексатор
* скоби **()** – използват се за предефиниране приоритета на изпълнение на изразите и операторите
* операторът за преобразуване на типове **(type)** – използва се за преобразуване на променлива от един тип в друг
* операторът **as** - също се използва за преобразуване на типове, но при невалидност на преобразуването връща null, а не изключение
* операторът **new** - използва се за създаването и инициализирането на нови обекти
* операторът **is** - използва се за проверка дали даден обект е съвместим с даден тип
* операторът **??** - подобен е на условния оператор „**?:**“. Разликата е, че той се поставя между два операнда и връща левия операнд само ако той няма стойност null, в противен случай връща десния



## Приоритет на операторите в C#



## Преобразуване на типовете

### Неявно (implicit) преобразуване на типове

Неявното (скритото) преобразуване на типове е възможно единствено, когато няма възможност от загуба на данни при преобразуването, т.е. когато конвертираме от тип с по-малък обхват към тип с по-голям обхват (примерно от int към long). За да направим неявно преобразуване не е нужно да използваме какъвто и да е оператор и затова такова преобра­зуване се нарича още скрито (implicit).



Ето някои от възможните неявни (implicit) преобразувания на примитивни типове в C#:

* **sbyte** → short, int, long, float, double, decimal;
* **byte** → short, ushort, int, uint, long, ulong, float, double, decimal;
* **short** → int, long, float, double, decimal;
* **ushort** → int, uint, long, ulong, float, double, decimal;
* **char** → ushort, int, uint, long, ulong, float, double, decimal (въпреки, че char е символен тип, в някои случаи той може да се разглежда като число и има поведение на числов тип, дори може да участва в числови изрази);
* **uint** → long, ulong, float, double, decimal;
* **int** → long, float, double, decimal;
* **long** → float, double, decimal;
* **ulong** → float, double, decimal;
* **float** → double.

### Изрично (explicit) преобразуване на типове

Изричното преобразуване на типове (explicit typecasting) се използва винаги, когато има вероятност за загуба на данни. Когато конвертираме тип с плаваща запетая към цело­числен тип, винаги има загуба на данни, идваща от премахването на дробната част и е задължително използването на изрично преобразуване (например double към long). За да направим такова конвертиране е нужно изрично да използваме оператора за преобразуване на данни (type). Възможно е да има загуба на данни също, когато конвертираме от тип с по-голям обхват към тип с по-малък (double към float или long към int).



Възможно е между всяка двойка измежду следните типове данни:

* sbyte, byte, short, ushort, char, int, uint, long, ulong, float, double, decimal

# Вход и изход от конзолата

## Печатане на конзолата

### Console.Write(…)



### Console.WriteLine(…)



## Форматиран изход

### Indexкомпонента



### Alignmentкомпонента

Alignment компонентата е незадължителна и указва подравняване на стринга. Тя е цяло положително или отрицателно число, като положител­ните стойности означават подравняване от дясно, а отрицателните – от ляво. Стойността на числото обозначава броя на позициите, в които да се подравни числото. Ако стрингът, който искаме да изобразим има дължина по-голяма или равна на стойността на числото, тогава това число се пренебрегва. Ако е по-малка обаче, незаетите позиции се допълват с интервали.



### Стандартно дефинирани формати за числа

* **"C" или "c"** - Обозначава валута и резултатът ще се изведе заедно със знака на валутата за текущата "култура" (например българската). Прецизността указва броя на знаците след десетичната запетая.
* **"D" или "d"** - Цяло число. Прецизността указва минималния брой знаци за изобразяването на стринга, като при нужда се извършва допълване с нули отпред.
* **"E" или "e"** - Експоненциален запис. Прецизността указва броя на знаците след десетичната запетая.
* **"F" или "f"** - Цяло или дробно число. Прецизността указва броя на знаците след десетичната запетая.
* **"N" или "n"** - Еквивалентно на "F", но изобразява и съответния разделител за хилядите, милионите и т.н. (например в английския език често числото "1000" се изписва като "1,000" - със запетая между числото 1 и нулите).
* **"P" или "p"** - Ще умножи числото по 100 и ще изобрази отпред символа за процент. Прецизността указва броя на знаците след десетичната запетая.
* **"X" или "x"** - Изписва числото в шестнадесетична бройна система. Работи само с цели числа. Прецизността указва мини­малния брой знаци за изобразяването на стринга, като недостигащите се допълват с нули отпред.



### Потребителскиформати за числа

* **0** - Обозначава цифра. Ако на тази позиция в резултата липсва цифра, се изписва цифрата 0.
* **#** - Обозначава цифра. Не отпечатва нищо, ако на тази позиция в резултата липсва цифра.
* **.** - Десетичен разделител за съответната "култура".
* **,** - Разделител за хилядите в съответната "култура".
* **%** - Умножава резултата по 100 и отпечатва символ за процент.
* **E0 или Е+0 или Е-0** - Обозначава експоненциален запис. Броят на нулите указва броя на знаците на експонентата. Знакът "+" обозначава, че искаме винаги да изпишем и знакът на числото, докато минус означава да се изпише знакът, само ако стойността е отрицателна.



### Стандартно дефинирани формати за дати

* **d** - 23/10/2009г.
* **D** - 23 Октомври 2009 г.
* **t** - 15:30 (час)
* **T** - 15:30:22 ч. (час)
* **Y или y** - Октомври 2009 г. (само месец и година)

### Custom формати за дати

* **d** - Ден – от 0 до 31
* **dd** - Ден – от 00 до 31
* **M** - Месец – от 0 до 12
* **MM** - Месец – от 00 до 12
* **yy** - Последните две цифри на годината (от 00 до 99)
* **yyyy** - Година, изписана с 4 цифри (например 2011)
* **hh** - Час – от 00 до 11
* **HH** - Час – от 00 до 23
* **m** - Минути – от 0 до 59
* **mm** - Минути – от 00 до 59
* **s** - Секунди – от 0 до 59
* **ss** - Секунди – от 00 до 59



### Компоненти за енумерации

* **G или g** - Представя енумерацията като стринг.
* **D или d** - Представя енумерацията като число.
* **X или x** - Представя енумерацията като число в шестнадесетичната бройна система и с осем цифри.



### Форматиращи низове и локализация



## Вход от конзолата

### Console.ReadLine()

При извикването му програмата преустановява работата си и чака за вход от конзолата. Потребителят въвежда някакъв стринг в конзолата и натиска клавишът [Enter]. В този момент конзолата разбира, че потребителят е свършил с въвеждането и прочита стринга. Методът Console.ReadLine()връща като резултат въведения от потребителя стринг.



### Console.Read()

Методът Read() работи по малко по-различен начин от ReadLine(). Като за начало той прочита само един символ, а не цял ред. Другата основна разлика е че методът не връща директно прочетения символ, а само неговия код. Ако желаем да използваме резултата като символ, трябва да го преобразуваме към символ или да използваме метода Convert. ToChar() върху него. Има и една важна особеност: символът се прочита чак когато се натисне клавишът [Enter]. Тогава целият стринг написан на конзолата се прехвърля в буфера на стандарт­ния входен поток и методът Read() прочита първия символ от него. При последващи извиквания на метода, ако буферът не е празен (т.е. има вече въведени, но все още непрочетени символи), то изпълнението на програмата няма да спре и да чака, а директно ще прочете следващия символ от буфера и така докато буферът не се изпразни. Едва тогава програмата ще чака наново за потребителски вход, ако отново се извика Read().

### Console.ReadKey()

Методът Console.ReadKey() изчаква натискане на клавиш на конзолата и прочита неговият символен еквивалент, без да е необходимо да се натиска [Enter]. Резултатът от извикването на ReadKey() е информация за натиснатия клавиш (или по-точно клавишна комбинация), във вид на обект от тип ConsoleKeyInfo. Полученият обект съдържа символа, който се въвежда чрез натиснатата комбинация от клавиши (свойство KeyChar), заедно с информация за клавишите [Shift], [Ctrl] и [Alt] (свойство Modifiers). Например, ако натиснем [Shift+A], ще прочетем главна буква 'А', а в свойството Modifiers ще присъства флага Shift.



## Условно парсване на числа



# Условни конструкции

## If



## If - else



## Switch – case



# Цикли

## Оператор break

Използва се за преждевременно излизане от цикъл, преди той да е завършил изпълнението си по естествения си начин. При срещане на оператора break цикълът се прекратява и изпълнението на програмата продължава от следващия ред веднага след тялото на цикъла.



## Оператор continue

Спира текущата итерация на най-вътрешния цикъл, без да излиза от него.



## While



## Do-while



## For



## For-цикъл с няколко променливи



## Foreach



# Масиви

## Едномерни масиви

### Деклариране на масив



### Създаване (заделяне) на масив



### Инициализация на масив



### Граници на масив

Масивите по подразбиране са нулево-базирани, т.е. номерацията на елемен­тите започва от 0. Първият елемент има индекс 0, вторият 1 и т.н. Ако един масив има N елемента, то последният елемент се намира на индекс N-1.

### Достъп до елементите на масив



### Четене на масив от конзолата



### Отпечатване на масив на конзолата



## Многомерни масиви

### Деклариране на многомерен масив



### Заделяне на многомерен масив



### Инициализация на двумерен масив



### Достъп до елементите на многомерен масив



### Извличане дължината на многомерен масив



### Четене на матрица от конзолата



### Отпечатване на матрица на конзолата



# Бройни системи



## Десетични числа

Числата представени в десетична бройна система (decimal numeral system), се задават в първичен вид, т.е. вид удобен за възприемане от човека. Тази бройна система има за основа числото 10. Числата записани в нея са подредени по степените на числото 10. Младшият разряд (първият отдясно на ляво) на десетичните числа се използва за представяне на единиците (100=1), следващият за десетиците (101=10), следващият за стотиците (102=100) и т.н.



## Двоични числа

За тяхното представяне, се използват само цифрите 0 и 1. Както и при десетичните числа, гледано от дясно наляво, всяко двоично число изразява степените на числото 2 в съответната последователност. На младшата позиция в двоично число съответства нулевата степен (20=1), на втората позиция съответства първа степен (21=2), на третата позиция съответства втора степен (22=4) и т.н. Ако числото е 8-битово, степените достигат до седма (27=128). Ако числото е 16-битово, степените достигат до петнадесета (215=32768). Чрез 8 двоични цифри (0 или 1) могат да се представят общо 256 числа, защото 28=256. Чрез 16 двоични цифри могат да се представят общо 65536 числа, защото 216=65536.





## Шестнайсетични числа

При шестнайсетичните числа (hexadecimal numbers) имаме за основа на бройната система числото 16, което налага да бъдат използвани 16 знака (цифри) за представянето на всички възможни стойности от 0 до 15 включително. Както вече беше показано в една от таблиците в предходните точки, за представянето на шестнайсе­тичните числа се използват числата от 0 до 9 и латинските букви от A до F. Всяка от тях има съответната стойност:

A=10, B=11, C=12, D=13, E=14, F=15

Като примери за шестнайсетични числа могат да бъдат посочени съответно, D2, 1F2F1, D1E и др.

## Преминаване от една бройна система в друга

### От двоична към десетична бройна система

Всяка една двоична цифра се умножава по 2 на степен, позицията, на която се намира (в двоичното число). Накрая се извършва събиране на числата, получени за всяка от двоичните цифри, за да се получи десетичната равностойност на двоичното число.



### От десетична към двоична бройна система

При преминаване от десетична в двоична бройна система, се извършва преобразуване на десетичното число в двоично. За целите на преобразу­ването се извършва делене на две с остатък. Така се получават частно и остатък, който се отделя.



### От десетична към шестнайсетична бройна система

Преминаването от десетична към шестнайсетична бройна система става като се дели десетичното число на 16 и се вземат остатъците в обратен ред.



### От шестнайсетична към десетична бройна система

Преминаването към десетична система става като се умножи по 160 стойността на най-дясната цифра, по 161 следващата вляво, по 162 следващата вляво и т.н. и накрая се съберат.



## Действия с двоични числа

### Побитово "и"

Резултатът е 1, когато и двата бита са 1-ци.



### Побитово "или"

Резултатът е 1, когато поне единия бит е 1-ца.



### Побитово "изключващо и"

Резултатът е 1, когато само единия бит е 1-ца.



### Побитово отрицание

Обръща стойностите на битовете.



# Методи

# Листи

# Речници

# Полезно за изпита

























