**Оператори за сравнение**

В C# има няколко оператора за сравнение, които се използват за сравняване на двойки цели числа, числа с плаваща запетая, символи, низове и други типове данни:

|  |  |
| --- | --- |
| **Оператор** | **Действие** |
| == | равно |
| != | различно |
| **>** | по-голямо |
| **>=** | по-голямо или равно |
| **<** | по-малко |
| **<=** | по-малко или равно |

Операторите за сравнение могат да сравняват произволни изрази, например две числа, два числови израза или число и променлива. Резултатът от сравнението е булева стойност (**true** или **false**).

Нека погледнем един пример, в който използваме сравнения:

|  |
| --- |
| int weight = 700;Console.WriteLine(weight >= 500); // True char gender = **'m'**;Console.WriteLine(gender <= **'f'**); // False double colorWaveLength = 1.630;Console.WriteLine(colorWaveLength > 1.621); // True int a = 5;int b = 7;bool condition = (b > a) && (a + b < a \* b);Console.WriteLine(condition); // True Console.WriteLine(**'B'** == **'A'** + 1); // True |

В примерния програмен код извършваме сравнение между числа и между символи. При сравнението на числа те се сравняват по големина, а при сравнението на символи се сравнява тяхната лексикограф­ска подредба (сравняват се Unicode номерата на съответните символи).

Както се вижда от примера, типът **char** има поведение на число и може да бъде събиран, изваждан и сравняван свободно с числа, но тази възмож­ност трябва да се ползва внимателно, защото може да доведе до труден за четене и разбиране код.

Стартирайки примера ще получим следния резултат:

|  |
| --- |
| **True****False****True****True****True** |

В C# има няколко различни типа данни, които могат да бъдат сравнявани:

-     числа (**int**, **long**, **float**, **double**, **ushort**, **decimal**, …)

-     символи (**char**)

-     булеви стойности (**bool**)

-     референции към обекти, познати още като обектни указатели (**string**, **object**, масиви и други)

Всяко едно сравнение може да засегне две числа, две **bool** стойности, или две референции към обекти. Позволено е да се сравняват изрази от различни типове, например цяло число с число с плаваща запетая, но не всяка двойки типове данни могат директно да се сравняват. Например не можем да сравняваме стринг с число.

**Сравнение на цели числа и символи**

Когато се сравняват числа и символи, се извършва сравнение директно между техните двоични представяния в паметта, т. е. сравняват се техните стойности. Например, ако сравняваме две числа от тип **int**, ще бъдат сравнени стойностите на съответните поредици от 4 байта, които ги съставят. Ето един пример за сравнение на символи и числа:

|  |
| --- |
| Console.WriteLine(**"char 'a' == 'a'? "** + (**'a'** == **'a'**)); // TrueConsole.WriteLine(**"char 'a' == 'b'? "** + (**'a'** == **'b'**)); // FalseConsole.WriteLine(**"5 != 6? "** + (5 != 6)); // TrueConsole.WriteLine(**"5.0 == 5L? "** + (5.0 == 5L)); // TrueConsole.WriteLine(**"true == false? "** + (true == false)); // False |

Резултатът от примера изглежда по следния начин:

|  |
| --- |
| **char 'a' == 'a'? True****char 'a' == 'b'? False****5 != 6? True****5.0 == 5L? True****true == false? False** |

**Сравнение на референции към обекти**

В .NET Framework съществуват референтни типове данни, които не съдър­жат директно стойността си (както числовите типове), а съдържат адрес от динамичната памет, където е записана стойността им. Такива типове са стринговете, масивите и класовете. Те имат поведение на указател към някакви стойности и могат да имат стойност **null**, т.е. липса на стойност. При сравняването на променливи от референтен тип се сравняват адре­сите, които те пазят, т.е. проверява се дали сочат на едно и също място в паметта, т.е. към един и същ обект.

Два указателя към обекти (референции) могат да сочат към един и същи обект или към различни обекти или някой от тях може да не сочи никъде (да има стойност **null**). В следващия пример създаваме две променливи, които сочат към една и съща стойност (обект) в динамичната памет:

|  |
| --- |
| string str = "beer";string anotherStr = str; |

След изпълнението на този код, двете променливи **str** и **anotherStr** ще сочат към един и същи обект (**string** със стойност "beer"), който се намира на някакъв адрес в динамичната памет (managed heap).

Променливите от тип референция към обект могат да бъдат проверени, дали сочат към един и същ обект, посредством оператора за сравнение **==**. За повечето референтни типове този оператор не сравнява съдържанието на обектите, а само дали се намират на едно и също място в паметта, т. е. дали са един и същ обект. За променливи от тип обект, не са приложими сравненията по големина (**<**, **>**, **<=** и **>=**).

Следващият пример илюстрира сравнението на референции към обекти:

|  |
| --- |
| string str = **"beer"**;string anotherStr = str;string thirdStr = **"be"** + **'e'** + **'r'**;Console.WriteLine(**"str = {0}"**, str);Console.WriteLine(**"anotherStr = {0}"**, anotherStr);Console.WriteLine(**"thirdStr = {0}"**, thirdStr);Console.WriteLine(str == anotherStr); // True - same objectConsole.WriteLine(str == thirdStr); // True - equal objectsConsole.WriteLine((object)str == (object)anotherStr); // TrueConsole.WriteLine((object)str == (object)thirdStr); // False |

Ако изпълним примера, ще получим следния резултат:

|  |
| --- |
| str = beeranotherStr = beerthirdStr = beerTrueTrueTrueFalse |

Понеже стринговете, използвани в примера (инстанциите на класа **System.String**, дефинирани чрез ключовата дума **string** в C#), са от референтен тип, техните стойности се заделят като обекти в динамичната памет. Двата обекта, които се създават **str** и **thirdStr** имат равни стойности, но са различни обекти, разположени на различни адреси в паметта. Променливата **anotherStr** също е от референтен тип и приема адреса (референцията) на **str**, т.е. сочи към вече съществуващия обект **str**. Така при сравнението на променливите **str** и **anotherStr** се оказва, че те са един и същ обект и съответно са равни. При сравнението на **str** с **thirdStr** резултатът също е равенство, тъй като операторът **==** сравнява стринговете по стойност, а не по адрес (едно много полезно изключение от правилото за сравнение по адрес). Ако обаче преобразуваме трите променливи към обекти и тогава ги сравним, ще получим сравнение на адресите, където стоят стойностите им в паметта и резултатът ще е различен. Това показва, че операторът **==** има специално поведение, когато се сравняват стрингове, но за останалите референтни типове (например масиви или класове) той работи като ги сравнява по адрес.

Повече за класа **String** и за сравняването на символните низове ще научите в главата "[Символни низове](http://www.introprogramming.info/intro-csharp-book/read-online/glava13-simvolni-nizove)".

**Логически оператори**

Да си припомним логическите оператори в C#, тъй като те често се ползват при съставянето на логически (булеви) изрази. Това са операторите: **&&**, **||**, **!** и **^**.

**Логически оператори && и ||**

Логическите оператори **&&** (логическо И) и **||** (логическо ИЛИ) се използват само върху булеви изрази (стойности от тип **bool**). За да бъде резултатът от сравняването на два израза с оператор **&&** **true** (истина), то и двата операнда трябва да имат стойност **true**. Например:

|  |
| --- |
| bool result = (2 < 3) && (3 < 4); |

Този израз е "истина", защото и двата операнда: (2 < 3) и (3 < 4) са "истина". Логическият оператор **&&** се нарича още и съкратен оператор, тъй като той не губи време за допълнителни изчисления. Той изчислява лявата част на израза (първи операнд) и ако резултатът е **false**, не губи време за изчисляването на втория операнд, тъй като е невъзможно крайният резултат да е "истина", ако първият операнд не е "истина". По тази причина той се нарича още **съкратен логически оператор "и"**.

Аналогично операторът || връща дали поне единият операнд от двата има стойност "истина". Пример:

|  |
| --- |
| bool result = (2 < 3) || (1 == 2); |

Този израз е "истина", защото първият му операнд е "истина". Както и при **&&** оператора, изчислението се извършва съкратено – ако първият операнд е **true**, вторият изобщо не се изчислява, тъй като резултатът е вече известен. Той се нарича още **съкратен логически оператор "или"**.

**Логически оператори & и |**

Операторите за сравнение **&** и **|** са подобни, съответно на **&&** и **||**. Разликата се състои във факта, че се изчисляват и двата операнда един след друг, независимо от това, че крайния резултат е предварително ясен. Затова и тези оператори за сравнение се наричат още **несъкратени логически оператори** и се ползват много рядко.

Например, когато се сравняват два операнда с **&** и първият операнд се сведе до "лъжа", въпреки това се продължава с изчисляването на втория операнд. Резултатът е ясно, че ще бъде сведен до "лъжа". По същия начин, когато се сравняват два операнда с **|** и първия операнд се сведе до "истина", независимо от това се продължава с изчисляването на втория операнд и резултатът въпреки всичко се свежда до "истина".

Не трябва да бъркате булевите оператори **&** и **|** с побитовите оператори **&** и **|**. Макар и да се изписват по еднакъв начин, те приемат различни аргументи (съответно булеви изрази или целочислени изрази) и връщат различен резултат (**bool** или цяло число) и действията им не са съвсем идентични.

**Логически оператори ^ и !**

Операторът **^**, известен още като **изключващо ИЛИ (XOR)**, се прилага само върху булеви стойности. Той се причислява към несъкратените оператори, поради факта, че изчислява и двата операнда един след друг. Резултатът от прилагането на оператора е "истина", когато само и точно един от операндите е истина, но не и двата едновременно. В противен случай резултатът е "лъжа". Ето един пример:

|  |
| --- |
| Console.WriteLine("Изключващо ИЛИ: " + ((2 < 3) ^ (4 > 3))); |

Резултатът е следният:

|  |
| --- |
| Изключващо ИЛИ: False |

Предходният израз е сведен до лъжа, защото и двата операнда: (2 < 3) и (4 > 3) са истина.

Операторът **!** връща като резултат противоположната стойност на булевия израз, към който е приложен. Пример:

|  |
| --- |
| bool value = !(7 == 5); // ТrueConsole.WriteLine(value); |

Горният израз може да бъде прочетен, като "обратното на истинността на израза "**7 == 5**". Резултатът от примера е **True** (обратното на **False**).

**Условни конструкции if и if-else**

След като си припомнихме как можем да сравняваме изрази, нека преминем към условните конструкции, които ни позволяват да имплемен­тираме програмна логика.

**Условните конструкции** **if** и **if-else** представляват тип условен контрол, чрез който програмата може да се държи различно, в зависимост от някакво условие, което се проверява по време на изпълнение на конструкцията.

**Условна конструкция if**

Основният формат на условната конструкция **if** е следният:

|  |
| --- |
| if (булев израз) { тяло на условната конструкция;} |

Форматът включва: **if**-клауза, булев израз и тяло на условната конструкция.

Булевият израз може да бъде променлива от булев тип или булев логически израз. Булевият израз не може да бъде цяло число (за разлика от други езици за програмиране като C и C++).

Тялото на конструкцията е онази част, заключена между големите къдрави скоби: **{}**. То може да се състои от един или няколко операции (statements). Когато се състои от няколко операции, говорим за съставен блоков оператор, т.е. поредица от команди, следващи една след друга, заградени във фигурни скоби.

Изразът в скобите след ключовата дума **if** трябва да бива изчислен до булева стойност **true** или **false**. Ако изразът бъде изчислен до стойност **true**, тогава се изпълнява тялото на условната конструкция. Ако резултатът от изчислението на булевия израз е **false**, то операторите в тялото няма да бъдат изпълнени.

**Условна конструкция if – пример**

Да разгледаме един пример за използване на условна конструкция **if**:

|  |
| --- |
| static void Main(){ Console.WriteLine(**"Enter two numbers."**); Console.Write(**"Enter first number: "**); int firstNumber = int.Parse(Console.ReadLine()); Console.Write(**"Enter second number: "**); int secondNumber = int.Parse(Console.ReadLine()); int biggerNumber = firstNumber; if (secondNumber > firstNumber) { biggerNumber = secondNumber; } Console.WriteLine(**"The bigger number is: {0}"**, biggerNumber);} |

Ако стартираме примера и въведем числата 4 и 5, ще получим следния резултат:

|  |
| --- |
| Enter two numbers.Enter first number: 4Enter second number: 5The bigger number is: 5 |

**Конструкцията if и къдравите скоби**

При наличието на само един оператор в тялото на **if**-конструкцията, къдравите скоби, обозначаващи тялото на условния оператор могат да бъдат изпуснати, както е показано по-долу. Добра практика е, обаче те да бъдат поставяни, дори при наличието на само един оператор. Целта е програмният код да бъде по-четим.

Ето един пример, в който изпускането на къдравите скоби води до объркване:

|  |
| --- |
| int a = 6;if (a > 5) Console.WriteLine("Променливата а е по-голяма от 5."); Console.WriteLine("Този код винаги ще се изпълни!");// Bad practice: unreadable code |

В горния пример кодът е форматиран заблуждаващо и създава впечат­ление, че и двете печатания по конзолата се отнасят за тялото на **if** блока, а всъщност това е вярно само за първия от тях.

|  |  |
| --- | --- |
| clip_image001[4] | **Винаги слагайте къдрави скоби { } за тялото на if блоко­вете, дори ако то се състои само от един оператор!** |

**Условна конструкция if-else**

В C#, както и в повечето езици за програмиране, съществува условна конструкция с **else** клауза: конструкцията **if**-**else**. Нейният формат е, както следва:

|  |
| --- |
| if (булев израз) { тяло на условната конструкция;} else { тяло на else-конструкция;} |

Форматът на **if-else** конструкцията включва: запазена дума **if**, булев израз, тяло на условната конструкция, запазена дума **else**, тяло на **else**-конструкция. Тялото на **else**-конструкцията може да се състои от един или няколко оператора, заградени в къдрави скоби, също както тялото на условната конструкция.

Тази конструкция работи по следния начин: изчислява се изразът в скобите (булевият израз). Резултатът от изчислението трябва да е булев – **true** или **false**. В зависимост от резултата са възможни два пътя, по които да продължи потока от изчисленията. Ако булевият израз се изчисли до **true**, се изпълнява тялото на условната конструкция, а тялото на **else**-конструкцията се пропуска и операторите в него не се изпъл­няват. В обратния случай, ако булевият израз се изчисли до **false**, се изпълнява тялото на **else**-конструкцията, а основното тяло на условната конструкция се пропуска и операторите в него не се изпълняват.

**Условна конструкция if-else – пример**

Нека разгледаме следния пример, за да покажем в действие как работи **if-else** конструкцията:

|  |
| --- |
| static void Main(){ int x = 2; if (x > 3) { Console.WriteLine(**"x е по-голямо от 3"**); } else { Console.WriteLine(**"x не е по-голямо от 3"**); }} |

Програмният код може да бъде интерпретиран по следния начин: ако x>3, то резултатът на изхода е: "**x е по-голямо от 3**", иначе (**else**) резултатът е: "**x не е по-голямо от 3**". В случая, понеже x=2, след изчислението на булевия израз ще бъде изпълнен операторът от **else**-конструкцията. Резултатът от примера е:

|  |
| --- |
| **x не е по-голямо от 3** |

На следващата блок-схема е показан графично потокът на изчисленията от този пример:

![clip_image003[4]]()

**Вложени if конструкции**

Понякога е нужно програмната логика в дадена програма или приложение да бъде представена посредством **if**-конструкции, които се съдържат една в друга. Наричаме ги **вложени** **if** или **if-else** конструкции.

Влагане наричаме поставянето на **if** или **if**-**else** конструкция в тялото на друга **if** или **else** конструкция. В такива ситуации всяка **else** клауза се отнася за най-близко разположе­ната предходна **if** клауза. По този начин разбираме коя **else** клауза към коя **if** клауза се отнася.

Не е добра практика нивото на влагане да бъде повече от три, тоест не трябва да бъдат влагани повече от три условни конструкции една в друга. Ако поради една или друга причина се наложи да бъде направено влагане на повече от три конструкции, то част от кода трябва да се изнесе в отделен метод (вж. главата [Методи](http://www.introprogramming.info/intro-csharp-book/read-online/glava9-metodi)).

**Вложени if конструкции – пример**

Следва пример за употреба на вложени **if** конструкции:

|  |
| --- |
| int first = 5;int second = 3; if (first == second){ Console.WriteLine(**"These two numbers are equal."**);}else{ if (first > second) { Console.WriteLine(**"The first number is greater."**); } else { Console.WriteLine(**"The second number is greater."**); }} |

В примера се разглеждат две числа и се сравняват на две стъпки: първо се сравняват дали са равни и ако не са, се сравняват отново, за да се установи кое от числата е по-голямо. Ето го и резултата от работата на горния код:

|  |
| --- |
| The first number is greater. |

**Поредици if-else-if-else-...**

Понякога се налага да ползваме поредица от **if** конструкции, в **else** клаузата на които има нова **if** конструкция. Ако ползваме вложени **if** конструкции, кодът ще се отмести прекаленo навътре. Затова в такива ситуации е допустимо след **else** веднага да следва нов **if**, дори е добра практика. Ето един пример:

|  |
| --- |
| char ch = **'X'**;if (ch == **'A'** || ch == **'a'**){ Console.WriteLine(**"Vowel [ei]"**);}else if (ch == **'E'** || ch == **'e'**){ Console.WriteLine(**"Vowel [i:]"**);}else if (ch == **'I'** || ch == **'i'**){ Console.WriteLine(**"Vowel [ai]"**);}else if (ch == **'O'** || ch == **'o'**){ Console.WriteLine(**"Vowel [ou]"**);}else if (ch == **'U'** || ch == **'u'**){ Console.WriteLine(**"Vowel [ju:]"**);}else{ Console.WriteLine(**"Consonant"**);} |

Програмната логика от примера последователно сравнява дадена промен­лива, за да провери дали тя е някоя от гласните букви на латинската азбука. Всяко следващо сравнение се извършва само в случай че пред­ходното сравнение не е било истина. В крайна сметка, ако никое от **if** условията не е изпълнено, се изпълнява последната **else** клауза, заради което резултатът от примера е следният:

|  |
| --- |
| Consonant |

**If конструкции – добри практики**

Ето и някои съвети, които е препоръчително да бъдат следвани при писането на **if** конструкции:

-     Използвайте блокове, заградени с къдрави скоби **{** **}** след **if** и **else** с цел избягване на двусмислие.

-     Винаги форматирайте коректно програмния код чрез отместване на кода след **if** и след **else** с една табулация навътре, с цел да бъде лесно четим и да не позволява двусмислие.

-     Предпочитайте използването на **switch**-**case** конструкция вместо поредица **if-else-if-else-…** конструкции или серия вложени **if-else** конструкции, когато това е възможно. Конструкцията **switch-case** ще разгледаме в [следващата секция](http://www.introprogramming.info/intro-csharp-book/read-online/glava5-uslovni-konstrukcii/%22%20%5Cl%20%22_%D0%A3%D1%81%D0%BB%D0%BE%D0%B2%D0%BD%D0%B0_%D0%BA%D0%BE%D0%BD%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%86%D0%B8%D1%8F_switch-case).

**Условна конструкция switch-case**

В следващата секция ще бъде разгледана условната конструкция **switch** за избор измежду списък от възможности.

**Как работи switch-case конструкцията?**

Конструкцията **switch-case** избира измежду части от програмен код на базата на изчислената стойност на определен израз (най-често цело­числен). Форматът на конструкцията за избор на вариант е следният:

|  |
| --- |
| switch (селектор){ case целочислена-стойност-1: конструкция; break; case целочислена-стойност-2: конструкция; break; case целочислена-стойност-3: конструкция; break; case целочислена-стойност-4: конструкция; break; // … default: конструкция; break;} |

Селекторът е израз, връщащ като резултат някаква стойност, която може да бъде сравнявана, например число или **string**. Операторът **switch** сравнява резултата от селек­тора с всяка една стойност от изброените в тялото на switch конструкцията в **case** етикетите. Ако се открие съвпа­дение с някой **case** етикет, се изпълнява съответната конструкция (проста или съставна). Ако не се открие съвпадение, се изпълнява **default** конструкцията (когато има такава). Стой­ността на селектора трябва задължително да бъде изчис­лена преди да се сравнява със стойностите вътре в **switch** конструк­цията. Етикетите не трябва да имат една и съща стойност.

Както се вижда, че в горната дефиниция всеки **case** завършва с оператора **break**, което води до преход към края на тялото на **switch** конструкцията. C# компилаторът задължително изисква да се пише **break** в края на всяка **case**-секция, която съдържа някакъв код. Ако след дадена **case**-конструк­ция липсва програмен код, **break** може да бъде пропуснат и тогава изпълнението преминава към следващата **case**-конструкция и т.н. до срещането на оператор **break**. След **default** конструкцията, **break** е задължителен.

Не е задължително **default** конструкцията да е на последно място, но е препоръчително да се постави накрая, а не в средата на **switch** конструкцията.

**Правила за израза в switch**

Конструкцията **switch** е един ясен начин за имплементиране на избор между множество варианти (тоест, избор между няколко различни пътища за изпълнение на програмния код). Тя изисква селектор, който се изчислява до някаква конкретна стойност. Типът на селектора може да бъде цяло число, **string** или **enum**. Ако искаме да използваме, например, низ или число с плаваща запетая като селектор, това няма да работи в **switch** конструкция. За нецелочислени типове данни трябва да използваме пос­ледователност от **if** конструкции.