

O'REILLY®

# C# 8.0

Leksykon kieszonkowy



Joseph Albahari  
Ben Albahari

Helion 

Tytuł oryginału: C# 8.0 Pocket Reference: Instant Help for C# 8.0 Programmers

Tłumaczenie: Przemysław Szeremiota

ISBN: 978-83-283-6687-9

© 2020 Helion SA

Authorized Polish translation of the English edition of C# 8.0 Pocket Reference ISBN 9781492051213 © 2020 Joseph Albahari and Ben Albahari

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Helion SA dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Helion SA nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel. 32 231 22 19, 32 230 98 63

e-mail: [helion@helion.pl](mailto:helion@helion.pl)

WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/cha8lk>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzje.

Printed in Poland.

- Kup książkę
- Poleć książkę
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

---

# Spis treści

Pierwszy program w C#	5
Składnia	9
System typów	12
Typy liczbowe	21
Typ wartości logicznych i operatory logiczne	29
Znaki i ciągi znaków	31
Tablice	35
Zmienne i parametry	40
Operatory i wyrażenia	49
Operatory na typach z dopuszczalną wartością pustą	54
Instrukcje	56
Przestrzenie nazw	65
Klasy	70
Dziedziczenie	85
Typ object	94
Struktury	98
Modyfikatory dostępu	100
Interfejsy	102
Typy wyliczeniowe	107
Typy zagnieżdżone	109
Uogólnienia	110
Delegaty	119
Zdarzenia	125
Wyrażenia lambda	130
Metody anonimowe	135
Wyjątki i instrukcja try	136
Enumeratory i iteratory	145

Typy z dopuszczalną wartością pustą	149
Zabezpieczanie pustych referencji (C# 8.0)	155
Metody rozszerzające	157
Typy anonimowe	159
Krotki	160
LINQ	162
Wiązanie dynamiczne	186
Przeciążanie operatorów	194
Atrybuty	197
Atrybuty wywołania	201
Funkcje asynchroniczne	203
Wskaźniki i kod nienadzorowany	213
Dyrektywy preprocesora	217
Dokumentacja XML	220
O autorach	224

## Tablice

Tablica reprezentuje zestaw elementów konkretnego typu o ustalonej liczbie. Elementy w tablicy są przechowywane zawsze w ciągłym obszarze pamięci, jeden obok drugiego — dzięki temu dostęp do nich jest wysoce efektywny.

Składnia deklaracji tablicy zawiera nazwę typu z parą nawiasów kwadratowych. Poniższa instrukcja deklaruje tablicę sześciu znaków:

```
char[] vowels = new char[6];
```

Nawiasy kwadratowe służą też do *indeksowania* odwołań do elementów tablicy, w odwołaniach do konkretnych elementów tablicy:

```
vowels [0] = 'a'; vowels [1] = 'e'; vowels [2] = 'i';  
vowels [3] = 'o'; vowels [4] = 'u';  
vowels [5] = 'y';  
Console.WriteLine (vowels [1]); // e
```

Powyższa instrukcja wypisze na konsoli znak e, ponieważ indeksy w tablicy są liczone od 0. Do przejrzania wszystkich elementów tablicy można zastosować pętlę for. W poniższym przykładzie pętla for przechodzi przez wszystkie elementy tablicy, odwołując się do nich poprzez indeks i zmieniający się od 0 do 5:

```
for (int i = 0; i < vowels.Length; i++)  
    Console.Write (vowels [i]); // aeiouy
```

Tablice implementują interfejs `IEnumerable<T>` (patrz podrozdział „Enumeratory i iteratory”), więc można je przeglądać również za pomocą instrukcji `foreach`:

```
foreach (char c in vowels) Console.Write(c); // aeiouy
```

Wszystkie odwołania do elementów tablic przez ich indeksy podlegają kontroli w czasie wykonania; użycie nieprawidłowego indeksu doprowadzi do rzucenia wyjątku `IndexOutOfRangeException`:

```
vowels[6] = 'z'; // błąd czasu wykonania
```

Właściwość `Length` obiektu tablicy zwraca rozmiar tablicy, czyli liczbę jej elementów. Po utworzeniu tablicy jej rozmiar pozostaje stały i nie może być zmieniany. W przestrzeni nazw `System.Collections` i przestrzeniach w niej zagnieżdżonych można znaleźć struktury zbiorcze wyższego poziomu, w tym tablice dynamiczne i słowniki.

Do jednoczesnego deklarowania i wypełniania tablic elementami służą *wyrażenia inicjalizacji tablicy*:

```
char[] vowels = new char[] {'a', 'e', 'i', 'o', 'u', 'y'};
```

albo prościej:

```
char[] vowels = {'a', 'e', 'i', 'o', 'u', 'y'};
```

Wszystkie tablice to wartości typów dziedziczących po klasie `System.Array`, definiującej metody i właściwości wspólne dla wszystkich tablic. Wśród nich znajdują się właściwości instancji takie jak `Length` (rozmiar tablicy) czy `Rank` (liczba wymiarów tablicy) oraz metody statyczne służące do:

- dynamicznego tworzenia tablicy (`CreateInstance`),
- odczytywania wartości i ustawiania wartości elementów niezależnie od typu tablicy (`GetValue` i `SetValue`),
- wyszukiwania w tablicy posortowanej (`BinarySearch`) albo nieposortowanej (`IndexOf`, `LastIndexOf`, `Find`, `FindIndex`, `FindLastIndex`),
- sortowania tablicy (porządkowania elementów według wartości — `Sort`),
- kopiowania tablicy (`Copy`).

## Domyślna inicjalizacja elementów tablic

Przy tworzeniu tablicy zawsze następuje inicjalizacja elementów na bazie ich wartości domyślnych, odpowiednich dla typu elementu. Domyślna wartość danego typu wartości wynika z efektu operacji bitowego wyzerowania pamięci obiektu. Weźmy za przykład tablicę liczb całkowitych typu `int`. Ponieważ `int` to typ wartościowy, tablica będzie zawierać 1000 wartości tego typu, sąsiadujących ze sobą w ciągłym obszarze pamięci. Domyślną wartością każdego elementu takiej tablicy będzie 0:

```
int[] a = new int[1000];  
Console.WriteLine(a[123]); // 0
```

W przypadku elementów typu referencyjnego wartością domyślną przy inicjalizowaniu tablic jest referencja pusta (`null`).

Tablica *sama w sobie* jest zawsze wartością typu referencyjnego, niezależnie od typu zawieranych elementów. Poniższa deklaracja jest więc jak najbardziej prawidłowa:

```
int[] a = null;
```

## Indeksy i zakresy (C# 8.0)

W C# 8.0 pojawiły się mechanizmy ułatwiające pracę z pojedynczymi elementami albo całymi zakresami elementów tablic.

---

### Uwaga

Indeksy i zakresy działają również z typami `Span<T>` i `ReadOnlySpan<T>`, co zapewnia efektywny niskopoziomowy dostęp do zarządzanej i niezarządzanej pamięci programu.

Do obsługi indeksów i zakresów można dostosowywać również własne typy — odbywa się to poprzez definiowanie indeksatora typu `Index` albo `Range` (zobacz punkt „Indeksatory”).

---

## Indeksy

Indeksy w C# 8.0 pozwalają na odnoszenie się do elementów tablicy poprzez określanie ich pozycji względem *końca* tablicy, z użyciem operatora `^`. Zapis `^1` oznacza ostatni element tablicy, `^2` to element przedostatni i tak dalej:

```
char[] vowels = new char[] { 'a', 'e', 'i', 'o', 'u', 'y' };
char lastElement = vowels[^1]; // 'y'
char secondToLast = vowels[^2]; // 'u'
```

(`^0` oznacza tutaj rozmiar tablicy, a więc wyrażenie `vowels[^0]` spowoduje błąd programu).

W języku C# indeksy są zaimplementowane na bazie typu `Index`, który też możemy wykorzystywać w adresowaniu elementów tablic:

```
Index first = 0;
Index last = ^1;
char firstElement = vowels[first]; // 'a'
char lastElement = vowels[last]; // 'y'
```

## Zakresy

Mechanizm zakresów pozwala „wyciąć” podzbiór elementów tablicy za pomocą nowego operatora `..`:

```
char[] firstTwo = vowels[..2]; // 'a', 'e'
char[] lastThree = vowels[2..]; // 'o', 'u', 'y'
char[] middleOne = vowels[2..3] // 'o'
```

Drugi (końcowy) wyznacznik zakresu określa zakres *wyłączający*, co oznacza, że zapis `..2` opisuje elementy tablicy aż do (ale z pominięciem) elementu `[2]`.

W wyznaczaniu zakresów można się też posługiwać operatorem indeksu `^`. Poniższy zapis obejmuje dwa ostatnie znaki z tablicy `vowels`:

```
char[] lastTwo = vowels [^2..^0]; // 'u', 'y'
```

(`^0` jest tu dozwolone, ponieważ końcowy wyznacznik zakresu jest wyłączający).

W C# zakresy są implementowane na bazie typu `Range`, co pozwala na stosowanie następujących konstrukcji:

```
Range firstTwoRange = 0..2;  
char[] firstTwo = vowels [firstTwoRange]; // 'a', 'e'
```

## Tablice wielowymiarowe

Tablice wielowymiarowe można podzielić na dwa rodzaje: regularne i „wyszczerbione”. Tablice regularne (prostokątne) reprezentują  $n$ -wymiarowe bloki pamięci; tablice wyszczerbione to tablice tablic.

### Tablice regularne

*Tablice regularne* (ang. *rectangular arrays*) są jawnie deklarowane jako wielowymiarowe, z wymienianymi po przecinku rozmiarami w kolejnych wymiarach. Poniższy kod przykładowy deklaruje tablicę dwuwymiarową o rozmiarach  $3 \times 3$ :

```
int[,] matrix = new int [3, 3];
```

Metoda `GetLength` tablicy zwraca rozmiar tablicy w zadanym wymiarze (wymiarzy są numerowane od 0):

```
for (int i = 0; i < matrix.GetLength (0); i++)  
    for (int j = 0; j < matrix.GetLength (1); j++)  
        matrix [i, j] = i * 3 + j;
```

Tablica regularna może być inicjalizowana jak poniżej (wypełnimy ją wartościami identycznymi jak w poprzednim przykładzie):

```
int[,] matrix = new int [3,3]  
{  
    {0,1,2},  
    {3,4,5},  
    {6,7,8},  
};
```



(Kod wyróżniony pogrubieniem można pominąć jako niewymagany — rozmiar tablicy zostanie wywnioskowany z liczby elementów wyrażenia inicjalizującego).

## Tablice wyszczerbione

Tablice wyszczerbione (albo zagnieżdżone, ang. *jagged arrays*) są deklarowane jako tablice tablic, tj. za pomocą sekwencji par nawiasów kwadratowych reprezentujących poszczególne wymiary. Oto przykład deklaracji wyszczerbionej tablicy dwuwymiarowej, w której wymiar tablicy zewnętrznej to 3:

```
int [][] matrix = new int [3] [];
```

Rozmiar tablic wewnętrznych nie został tutaj wprost określony, ponieważ (inaczej niż w tablicach regularnych) w tablicy wyszczerbionej każda tablica wewnętrzna może mieć dowolny rozmiar, niezależnie od rozmiarów pozostałych tablic na tym poziomie zagnieżdżenia. Tablice wewnętrzne są pierwotnie niejawnie inicjalizowane referencjami pustymi. Każda tablica wewnętrzna musi być potem jawnie utworzona ręcznie:

```
for (int i = 0; i < matrix.Length; i++)  
{  
    matrix[i] = new int [3]; // utworzenie tablicy wewnętrznej  
    for (int j = 0; j < matrix[i].Length; j++)  
        matrix[i][j] = i * 3 + j;  
}
```

Tablica wyszczerbiona może być także inicjalizowana za pomocą następującej składni (utworzymy tablicę identyczną jak powyżej, ale z dodatkowym elementem na końcu):


```
int[][] matrix = new int [][]  
{  
    new int[] {0,1,2},  
    new int[] {3,4,5},  
    new int[] {6,7,8,9},  
};
```

(Kod wyróżniony pogrubieniem można pominąć jako niewymagany — rozmiar tablicy zostanie wywnioskowany z liczby elementów wyrażenia inicjalizującego).



# PROGRAM PARTNERSKI

— GRUPY HELION —

- 
1. ZAREJESTRUJ SIĘ
  2. PREZENTUJ KSIĄŻKI
  3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW w działający bankomat!

**Dowiedz się więcej i dołącz już dzisiaj!**

<http://program-partnerski.helion.pl>

GRUPA  
**Helion** 

# C#. Nie czekaj, programuj!

C# to obiektowy język programowania ogólnego przeznaczenia z kontrolą typów. Jest dojrzały, wyjątkowo wszechstronny i prosty w stosowaniu. Jego twórcy chcieli przede wszystkim zapewnić programistom jak największą efektywność, co znalazło odzwierciedlenie w prostocie języka, ekspresywności kodu i wydajności działania. Wersja C# 8.0 została dostosowana do współpracy ze środowiskiem uruchomieniowym Microsoft .NET Core 3 oraz z .NET Standard 2.1.

Ta książka jest zwięzłym kompendium, w którym znajdziesz wszystko, co jest potrzebne do pracy z C#. Została pomyślana w taki sposób, aby maksymalnie ułatwić przeglądanie i odnajdywanie potrzebnych treści. Jest nieocenioną pomocą dla osób, które znają już inne języki programowania, takie jak C++ czy Java, i postanowiły nabrać wprawy w pracy z C#. Poszczególne zagadnienia przedstawiono w przejrzysty, treściwy i równocześnie esencjonalny sposób. To pozycja, która powinna znajdować się tuż obok klawiatury każdego programisty C#!

## W tej książce:

- podstawy języka z uwzględnieniem nowych cech C# w wersji 8.0
- zagadnienia zaawansowane, w tym przeciążanie operatorów, typy z wartością pustą, wyrażenia lambda i domknięcia
- LINQ: sekwencje, leniwe wykonanie, standardowe operatory zapytań

**Joseph Albahari** jest autorem kilku cenionych książek o programowaniu. Jest też twórcą LINQPad, popularnego narzędzia pomocnego w implementowaniu zapytań do baz danych w LINQ.

**Ben Albahari** jest współzałożycielem serwisu Auditionist, wirtualnej sceny castingowej dla aktorów. Przez pięć lat pracował w Microsoftzie. Współzakładał firmę Genamics, która dostarcza narzędzia dla programistów C# i J++. Jest współautorem kilku książek o programowaniu w C#.

**Helion** 

 [helion.pl](http://helion.pl)

 **HELION SA**  
ul. Kościuski 1c  
44-100 Gliwice  
tel.: 32 230 98 63  
[helion@helion.pl](mailto:helion@helion.pl)

*Sprawdź nasze szkolenia!*

**SZKOLENIA**



**AKADEMIA IT & BUSINESS**

**HELIONSZKOLENIA.PL**

**KOD KORZYŚCI**  
Sięgnij po więcej ▶



ISBN 978-83-283-6687-9



**INFORMATYKA W NAJLEPSZYM WYDANIU**

Cena: 39,90 zł