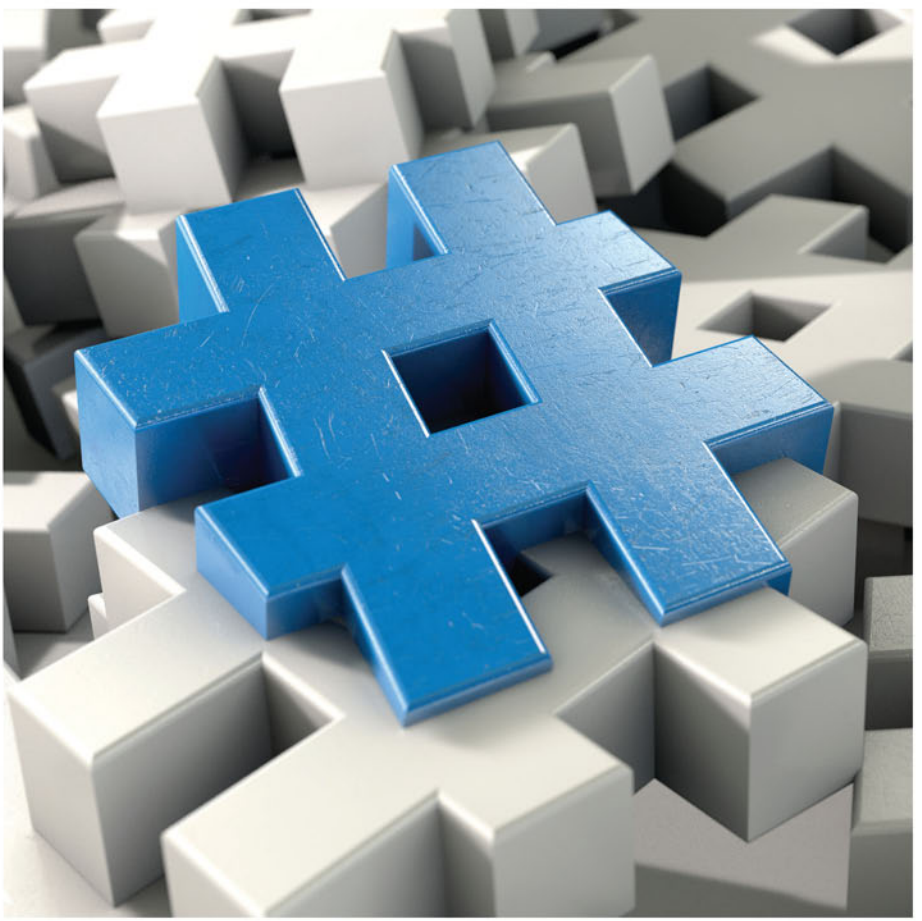
C#

ĆWICZENIA

Wydanie IV



Marcin Lis

Helion

Opieka redakcyjna: Ewelina Burska
Projekt okładki: Studio Gravite/Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *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/cwcsh4
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

ISBN: 978-83-283-1455-9

Printed in Poland.

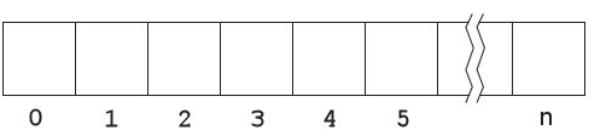
Spis treści

Część II
Programowanie obiektowe

Część III
Programowanie w Windows

5
Tablice

Tablice to jedne z podstawowych struktur danych; znane są zapewne nawet początkującym programistom. Warto jednak w kilku słowach przypomnieć podstawowe wiadomości i pojęcia z nimi związane. Tablica to stosunkowo prosta struktura danych pozwalająca na przechowanie uporządkowanego zbioru elementów danego typu — można ją sobie wyobrazić tak, jak zaprezentowano na rysunku 5.1. Składa się z ponumerowanych kolejno komórek, a każda taka komórka może przechowywać pewną porcję danych.



Rysunek 5.1. *Schemat struktury tablicy*

Jakiego rodzaju będą to dane, określa typ tablicy. Jeśli zatem zadeklarujemy tablicę typu całkowitoliczbowego (`int`), będzie mogła ona zawierać liczby całkowite, a jeśli będzie to typ znakowy (`char`), poszczególne komórki będą mogły zawierać różne znaki. Należy zwrócić uwagę, że w C# (podobnie jak w większości współczesnych popularnych języków programowania) numerowanie komórek zaczyna się od 0, czyli pierwsza komórka ma indeks 0, druga — indeks 1 itd.

Deklarowanie tablic

Przed skorzystaniem z tablicy należy zadeklarować zmienną tablicową. Ponieważ w C# tablice są obiektami, należy również utworzyć odpowiedni obiekt. Schematycznie robi się to w sposób następujący:

```
typ_tablicy[] nazwa_tablicy = new typ_tablicy[liczba_elementów];
```

Deklarację zmiennej tablicowej oraz przypisanie jej nowo utworzonego elementu można przy tym wykonać w osobnych instrukcjach, np. w ten sposób:

```
typ_tablicy[] nazwa_tablicy;
// tutaj mogą się znaleźć inne instrukcje
nazwa_tablicy = new typ_tablicy[liczba_elementów];
```

Pisząc zatem:

```
int tablica[];
```

zadeklarujemy odniesienie do tablicy, która będzie mogła zawierać elementy typu `int`, czyli 32-bitowe liczby całkowite. Samej tablicy jednak jeszcze nie będzie (odwrotnie niż w przypadku prostych typów wartościowych, takich jak `int`, `byte` czy `char`) i konieczne jest jej utworzenie.

ĆWICZENIE

5.1 Utworzenie tablicy

Zadeklaruj i zainicjalizuj tablicę elementów typu całkowitego. Przypisz pierwszemu elementowi tablicy dowolną wartość. Wyświetl zawartość tego elementu na ekranie.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tablica = new int[5];
    tablica[0] = 10;
    Console.WriteLine("Pierwszy element tablicy: " + tablica[0]);
  }
}
```

Wyrażenie `new tablica[5]` oznacza utworzenie nowej, jednowymiarowej, 5-elementowej tablicy liczb typu `int`. Ta nowa tablica została przy-

pisana zmiennej odnośnikowej o nazwie tablica. Od miejsca tego przypisania można odwoływać się do kolejnych elementów tej tablicy, pisząc:

```
tablica[index]
```

W tym przypadku pierwszemu elementowi (o indeksie 0) została przypisana wartość 10. O tym, że takie przypisanie faktycznie miało miejsce, przekonaliśmy się, wyświetlając wartość tej komórki na ekranie.

Warto w tym miejscu ponownie przypomnieć, że elementy tablicy numerowane są od 0, a nie od 1. Oznacza to, że pierwszy element tablicy 10-elementowej ma indeks 0, a ostatni 9 (nie 10!). Co się stanie, jeśli nieprzyzwyczajeni do takiego sposobu indeksowania odwołamy się do indeksu o numerze 10?

ĆWICZENIE

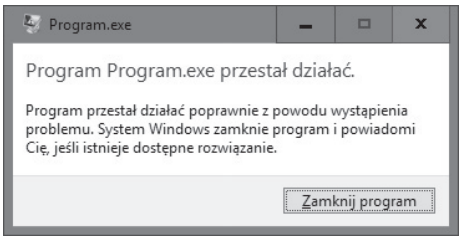
5.2 Odwołanie do nieistniejącego elementu tablicy

Zadeklaruj i zainicjalizuj tablicę 10-elementową. Spróbuj przypisać elementowi o indeksie 10 dowolną liczbę całkowitą.

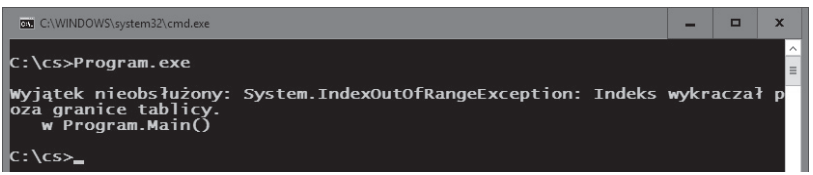
```
using System;

public class Program
{
  public static void Main()
  {
    int[] tablica = new int[10];
    tablica[10] = 1;
    Console.WriteLine("Element o indeksie 10 to: " + tablica[10]);
  }
}
```

Powyższy kod da się bez problemu skompilować, jednak przy próbie uruchomienia takiego programu na ekranie zobaczymy okno z informacją o wystąpieniu błędu. Może ono mieć różną postać, w zależności od tego, w jakiej wersji systemu została uruchomiona aplikacja. Na rysunku 5.2 jest widoczne okno z systemu Windows 8. Również na konsoli (w Windows XP dopiero po zamknięciu okna dialogowego) ujrzymy komunikat podający konkretne informacje o typie błędu oraz miejscu programu, w którym wystąpił (rysunek 5.3).



Rysunek 5.2. *Próba odwołania się do nieistniejącego elementu tablicy powoduje błąd aplikacji*



Rysunek 5.3. *Systemowa informacja o błędzie*

Wbrew pozorom nie stało się nic strasznego. Program co prawda nie działa, ale błąd został wychwycony przez środowisko uruchomieniowe. Konkretnie mówiąc, został wygenerowany tzw. wyjątek i aplikacja zakończyła działanie. Taki wyjątek można jednak przechwycić i tym samym zapobiec niekontrolowanemu zakończeniu wykonywania kodu. To jednak odrębny temat, który zostanie przedstawiony w rozdziale 6. Ważne jest to, że próba odwołania się do nieistniejącego elementu została wykryta i to odwołanie nie wystąpiło! Program nie naruszył więc obszaru pamięci niezarezerwowanej dla niego.

Inicjalizacja tablic

Tablicę można zainicjalizować już w momencie jej tworzenia. Dane, które mają się znaleźć w poszczególnych komórkach, podaje się w nawiasie klamrowym po deklaracji tablicy. Schematycznie wygląda to następująco:

```
typ[] nazwa = new typ [liczba_elementów]{dana1, dana2,…,danaN}
```

Jeśli zatem chcielibyśmy utworzyć 5-elementową tablicę liczb całkowitych i od razu zainicjalizować ją liczbami od 1 do 5, możemy zrobić to w taki sposób:

```
int[] tablica = new int[5] {1, 2, 3, 4, 5};
```

ĆWICZENIE

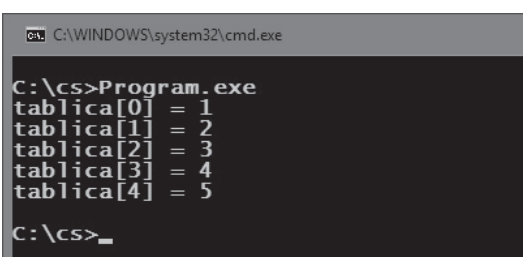
5.3 Inicjalizacja tablicy

Zadeklaruj tablicę 5-elementową typu `int` i zainicjalizuj ją liczbami od 1 do 5. Zawartość tablicy wyświetl na ekranie.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tablica = new int[5]{1, 2, 3, 4, 5};
    for(int i = 0; i < 5; i++)
    {
      Console.WriteLine("tablica[{0}] = {1}", i, tablica[i]);
    }
  }
}
```

Wynik działania kodu z powyższego ćwiczenia widoczny jest na rysunku 5.4. Nie jest niespodzianką, że wyświetlone zostały liczby od 1 do 5, natomiast indeksy kolejnych komórek zmieniają się od 0 do 4. Powstała tu bowiem 5-elementowa tablica liczb typu `int`. Skoro ma 5 elementów, to pierwszy z nich ma indeks 0, a ostatni — 4. Dlatego zmienna sterująca pętli `for`, która odczytuje dane z tablicy, ma początkową wartość 0, a warunek zakończenia pętli to `i < 5`. Tym samym `i` zmienia się też od 0 do 4.



Rysunek 5.4. *Zawartość kolejnych komórek tablicy utworzonej w ćwiczeniu 5.3*

Kiedy inicjalizowana jest tablica o z góry znanej liczbie elementów, dopuszcza się pominięcie fragmentu kodu związanego z tworzeniem obiektu. Kompilator sam wykona odpowiednie uzupełnienia. Zamiast pisać:

typ[] *nazwa* = new *typ* [*liczba_elementów*]{*dana1*, *dana2*,…,*danaN*}

można zatem równie dobrze użyć konstrukcji:

```
typ[] nazwa = {dana1, dana2,…,danaN}
```

Oba sposoby są równoważne i należy używać tego, który jest wygodniejszy.

ĆWICZENIE

5.4 Bezpośrednia inicjalizacja tablicy

Zadeklaruj tablicę 5-elementową typu `int` i zainicjalizuj ją liczbami od 6 do 2. Użyj drugiego z poznanych sposobów inicjalizacji. Zawartość tablicy wyświetl na ekranie.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tablica = {6, 5, 4, 3, 2};
    for(int i = 0; i < 5; i++)
    {
      Console.WriteLine("tablica[{0}] = {1}", i, tablica[i]);
    }
  }
}
```

Rozmiar tablicy

Każda tablica posiada właściwość `Length`, która określa bieżącą liczbę komórek. Aby uzyskać tę informację, piszemy:

```
tablica.Length
```

Przy tym dopuszczalny jest tylko odczyt, czyli prawidłowa jest konstrukcja:

```
int rozmiar = tablica.Length;
```

ale nieprawidłowy jest zapis:

```
tablica.Length = 10;
```

ĆWICZENIE

5.5 Odczyt rozmiaru tablicy

Utwórz tablicę o dowolnym rozmiarze. Odczytaj wartość właściwości `Length` i wyświetl ją na ekranie.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tablica =
    {
      10, 9, 8, 7, 6, 5, 4, 3, 2, 1
    };
    Console.Write("Liczba elementów tablicy: ");
    Console.WriteLine(tablica.Length);
  }
}
```

ĆWICZENIE

5.6 Właściwość Length i pętla for

Utwórz tablicę zawierającą pewną liczbę wartości całkowitych. Zawartość tablicy wyświetl na ekranie za pomocą pętli `for`. Do określenia rozmiaru tablicy użyj właściwości `Length`.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tab =
    {
      10, 9, 8, 7, 6, 5, 4, 3, 2, 1,
      1, 2, 3, 4, 5, 6, 7, 8, 9, 10
    };
    for(int i = 0; i < tab.Length; i++)
    {
      Console.WriteLine("tab[" + i + "] = " + tab[i]);
    }
  }
}
```

Zasada odczytu danych w tym przykładzie jest taka sama jak w ćwiczeniach 5.3 i 5.4, z tą różnicą, że rozmiar tablicy jest określany za

pomocą właściwości `Length` (`tab.Length`). Dzięki temu można np. dopisać dowolną liczbę nowych danych w instrukcji inicjalizującej tablicę, a kod pętli `for` nie będzie wymagał żadnych zmian. Nowy rozmiar zostanie uwzględniony automatycznie.

Do zapisywania danych (podobnie jak do odczytu) w tablicach często używa się pętli (przedstawionych w rozdziale 2.). Jest to wręcz niezbędne, gdyż trudno się spodziewać, że można byłoby „ręcznie" zapisać wartości z więcej niż kilkunastu czy kilkudziesięciu komórek. Wielkość tablicy nie musi też być z góry znana, może wynikać z danych uzyskanych w trakcie działania programu. Z tablicami mogą współpracować dowolne rodzaje pętli. W niektórych przypadkach bardzo wygodna jest omówiona w kolejnym podrozdziale pętla `foreach`.

ĆWICZENIE

5.7 Użycie pętli do zapisu danych w tablicy

Użyj pętli `for` do zapisania w 10-elementowej tablicy 10 kolejnych liczb całkowitych.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tab = new int[10];
    for(int i = 0; i < tab.Length; i++)
    {
      tab[i] = i + 1;
    }
    Console.WriteLine("Zawartość tablicy:");
    for(int i = 0; i < tab.Length; i++)
    {
      Console.WriteLine("tab[{0}] = {1}", i, tab[i]);
      // w C# 6.0 można również tak
      //Console.WriteLine($"tab[{i}] = {tab[i]}");
    }
  }
}
```

Powstała 10-elementowa tablica liczb typu `int`. Mamy w niej zapisać wartości od `1` do `10`, czyli komórka o indeksie `0` ma mieć wartość `1`, o indeksie `1` — wartość `2` itd. A zatem wartość komórki ma być zawsze

o 1 większa niż wartość indeksu (zmiennej `i`). Dlatego instrukcja wewnątrz pętli ma postać:

```
tab[i] = i + 1;
```

Druga pętla `for` służy tylko do wyświetlania danych zawartych w tablicy. Jej konstrukcja jest taka sama jak w pierwszym przypadku. Wewnątrz pętli znajduje się instrukcja wyświetlająca wartości kolejnych komórek.

Pętla foreach

Dotychczas poznaliśmy trzy rodzaje pętli: `for`, `while` i `do…while` (była o nich mowa w rozdziale 3.). W przypadku tablic (jak również kolekcji, które w tej książce nie były omawiane[1]) można również skorzystać z pętli typu `foreach`. Jest ona bardzo wygodna, gdyż umożliwia prostą iterację po wszystkich elementach tablicy; nie trzeba wtedy wprowadzać dodatkowej zmiennej iteracyjnej. Pętla `foreach` ma następującą postać:

```
foreach(typ identyfikator in wyrażenie)
{
  // instrukcje
}
```

Jeżeli zatem mamy tablicę o nazwie `tab` zawierającą liczby typu `int`, możemy zastosować konstrukcję:

```
foreach(int val in tab)
{
  // instrukcje
}
```

Wtedy w kolejnych przebiegach pętli pod `val` będą podstawiane kolejne elementy tablicy. Słowo `val` jest tu identyfikatorem odczytywanej wartości (można je traktować jak zmienną). Oczywiście nic nie stoi na przeszkodzie, aby zmienić je na dowolne inne.

[1] Ściślej rzecz ujmując, pętli `foreach` można użyć z każdym obiektem udostępniającym tzw. iterator. Ten temat nie będzie jednak poruszany w książce.

ĆWICZENIE

5.8 Użycie pętli foreach do wyświetlenia zawartości tablicy

Wykorzystaj pętlę `foreach` do wyświetlenia wszystkich elementów tablicy przechowującej liczby całkowite.

```
using System;

public class Program
{
  public static void Main()
  {
    int[] tab = new int[10];
    for(int i = 0; i < 10; i++)
    {
      tab[i] = i;
    }
    foreach(int i in tab)
    {
      Console.WriteLine(i);
    }
  }
}
```

Tablica `tab` została zainicjalizowana w pętli `for` kolejnymi liczbami od `0` do `9`. Do wyświetlenia danych została natomiast użyta pętla `foreach`. W każdym jej przebiegu pod identyfikator `i` jest podstawiana wartość kolejnego elementu tablicy. W pierwszym przebiegu jest to pierwszy element (o indeksie `0`), w drugim — drugi element (o indeksie `1`) itd. Pętla kończy się po osiągnięciu ostatniego elementu (o indeksie `9`).

ĆWICZENIE

5.9 Zliczanie wartości w pętli foreach

Wypełnij tablicę losowymi liczbami całkowitymi typu `int`. Wykorzystaj pętlę `foreach` do sprawdzenia, ile jest w tej tablicy liczb parzystych, a ile nieparzystych.

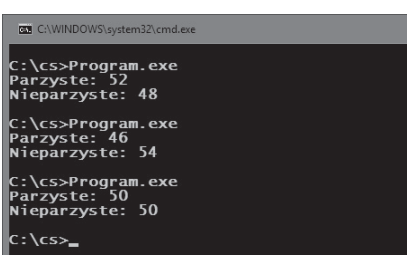
```
using System;

public class Program
{
  public static void Main()
  {
    int[] tab = new int[100];
    int parzyste = 0, nieparzyste = 0;
    Random rand = new Random();
```

```
        for(int i = 0; i < 100; i++)
        {
          tab[i] = rand.Next();
        }

        foreach(int i in tab)
        {
          if(i % 2 == 0)
          {
            parzyste++;
          }
          else
          {
            nieparzyste++;
          }
        }
        Console.WriteLine("Parzyste: {0}", parzyste);
        Console.WriteLine("Nieparzyste: {0}", nieparzyste);
      }
    }
```

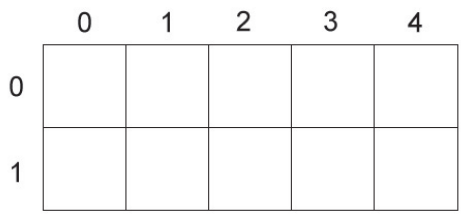
Powstała tablica `tab` typu `int`, 100-elementowa. Do wypełnienia jej danymi zostały użyte pętla `for` oraz obiekt `rand` typu `Random`, za pomocą którego uzyskujemy wartości pseudolosowe. Dokładniej rzecz ujmując, kolejną pseudolosową liczbę całkowitą otrzymujemy, wywołując metodę `Next` tego obiektu. W pętli `foreach` badamy, które z komórek tablicy `tab` zawierają wartości parzyste, a które — nieparzyste. Aby to stwierdzić, używamy operatora dzielenia modulo `%` (reszty z dzielenia; por. tabela 2.4 z rozdziału 2. i ćwiczenia 3.10 – 3.11 z rozdziału 3.). Gdy wynikiem tego dzielenia jest 0, dana komórka zawiera liczbę parzystą (jest wtedy zwiększana wartość pomocniczej zmiennej `parzyste`), natomiast gdy wynik dzielenia jest różny od 0, komórka zawiera wartość nieparzystą (jest wtedy zwiększana wartość pomocniczej zmiennej `nieparzyste`). Po zakończeniu pętli na ekranie wyświetlany jest komunikat z poszukiwaną informacją, co pokazano na rysunku 5.5 (w komunikatach używane są wartości pobrane ze zmiennych `parzyste` i `nieparzyste`).



Rysunek 5.5. *Wynik kilku uruchomień programu zliczającego liczbę wartości parzystych i nieparzystych*

Tablice wielowymiarowe

Tablice nie muszą być jednowymiarowe, jak było w dotychczas prezentowanych przykładach. Tych wymiarów może być więcej, np. dwa — otrzymujemy wtedy strukturę widoczną na rysunku 5.6, czyli rodzaj tabeli o zadanej liczbie wierszy i kolumn. W tym przypadku są dwa wiersze oraz cztery kolumny. Łatwo zauważyć, że aby w takiej sytuacji jednoznacznie wyznaczyć komórkę, trzeba podać dwie liczby: indeks wiersza i indeks kolumny.



Rysunek 5.6. *Przykład tablicy dwuwymiarowej*

W jaki sposób można zadeklarować tego typu tablicę? Zacznijmy od deklaracji samej zmiennej tablicowej. Dla tablicy dwuwymiarowej ma ona postać:

```
typ_tablicy[,] nazwa_tablicy;
```

Samą tablicę tworzy się za pomocą instrukcji:

```
new int[wiersze, kolumny];
```

Przykładową dwuwymiarową tablicę widoczną na rysunku 5.6 utworzymy następująco (przy założeniu, że ma przechowywać liczby całkowite):

```
int[,] tablica = new tablica[2, 5];
```

Inicjalizacja komórek może odbywać się — podobnie jak było w przypadku tablic jednowymiarowych — już w trakcie deklaracji:

```
typ_tablicy[,] nazwa_tablicy =
{
  (dana1, dana2),
  (dana3, dana4),
   …,
  (danaM, danaN)
};
```

Zobaczmy na konkretnym przykładzie, jak będzie to wyglądało.

Poznaj C# — to Ci się przyda!

- **Język programowania, czyli nauka podstawowych elementów C# i ich zastosowania**
- **Programowanie obiektowe, czyli zaawansowane mechanizmy i struktury danych**
- **Programowanie w Windows, czyli tworzenie aplikacji z graficznym interfejsem użytkownika**

C# to obiektowy język programowania, dość podobny do Javy i C++, choć nieco od nich prostszy. Przy tym jego możliwości są naprawdę imponujące. Coraz większa popularność i wygoda używania sprawiają, że programiści chętnie sięgają po to narzędzie. Niezależnie od tego, czy umiesz już programować w innych językach, czy dopiero przystępujesz do nauki programowania, ta książka pomoże Ci szybko opanować najważniejsze elementy C#!

Znajdziesz tu bez mała półtorej setki ćwiczeń, które pozwolą Ci zrozumieć, jak używać zmiennych i operatorów, stosować instrukcje i wprowadzać dane. Zobaczysz, jak deklarować klasy i tablice, zapewniać obsługę wyjątków i błędów, tworzyć obiekty i całe, działające aplikacje — konsolowe i z graficznym interfejsem użytkownika. Możesz to zrobić w sposób całkowicie praktyczny i od razu zobaczyć efekty swoich działań. Sprawdź, jakie cuda da się stworzyć w języku C#, i wykorzystaj tę wiedzę w swojej pracy!

- **Pierwsza aplikacja**
- **Zmienne i typy danych**
- **Instrukcje, pętle i wprowadzanie danych**
- **Klasy, obiekty i tablice**
- **Wyjątki i obsługa błędów**
- **Interfejsy**
- **Aplikacje okienkowe**
- **Delegacje i zdarzenia**
- **Komponenty**

Przećwicz C# i zacznij w nim programować!

Helion

41165 numer katalogowy

księgarnia internetowa

http://helion.pl

zamówienia telefoniczne

0 801 339900

0 601 339900

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-283-1455-9

9 788328 314559

Informatyka w najlepszym wydaniu

cena: 34,90 zł