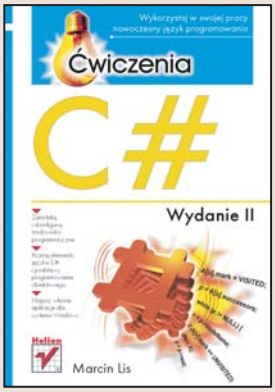
C#. Ćwiczenia. Wydanie II

Autor: Marcin Lis
ISBN: 83-246-0595-9
Format: A5, stron: 216
Przykłady na ftp: 65 kB



C# to jeden z najmłodszych języków programowania. Opracowany w firmie Microsoft język jest połączeniem najlepszych cech C++ i Javy. Stanowi świetne narzędzie do tworzenia aplikacji dla systemu Windows i urządzeń mobilnych. C# jest stosunkowo łatwy do opanowania, ma prostą i czytelną składnię. ścisła integracja z platformą .NET pozwala programistom korzystać z oferowanych przez nią klas i komponentów. Bardzo dużym atutem najnowszej wersji języka C# jest to, iż wszyscy, którzy chcą poznać jego możliwości, mogą skorzystać z dostępnego nieodpłatnie środowiska programistycznego Microsoft Visual Studio Express Edition.

Dzięki książce „C#. Ćwiczenia. Wydanie II" poznasz podstawy języka C# i nauczysz się korzystać ze środowiska Visual Studio Express. Dowiesz się, z jakich elementów składają się programy napisane w C# i na czym polega programowanie obiektowe. Przeczytasz o obsłudze wyjątków, projektowaniu okien dialogowych, przetwarzaniu danych i programowaniu sterowanym zdarzeniami. Wykonując kolejne ćwiczenia, poznasz zasady tworzenia aplikacji dla systemu Windows z wykorzystaniem komponentów platformy .NET.

- Obsługa środowiska Visual Studio Express
- Typy danych
- Operatory
- Instrukcje warunkowe i pętle
- Programowanie obiektowe
- Obsługa błędów
- Wyświetlanie okien w systemie Windows
- Korzystanie z komponentów platformy .NET

Rozpocznij przygodę z programowaniem

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Tablice

Tablice to jedne z podstawowych struktur danych i znane są z pewnością nawet początkującym programistom. Przypomnijmy jednak na wstępie podstawowe wiadomości i pojęcia z nimi związane. Tablica to stosunkowo prosta struktura danych pozwalająca na przechowanie uporządkowanego zbioru elementów danego typu. Składa się z ponumerowanych kolejno komórek, a każda taka komórka może przechowywać pewną porcję danych.

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.

Deklarowanie tablic

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

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

Oczywiście deklaracji zmiennej tablicowej oraz przypisania jej nowo utworzonego elementu można dokonać w osobnych instrukcjach, np. pisząc:

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

Pisząc zatem:

```
int tablica[];
```

zadeklarujemy odniesienie do tablicy, która będzie zawierała elementy typu `int`, czyli 32-bitowe liczby całkowite. Samej tablicy jednak jeszcze wcale nie ma. Przekonamy się o tym, wykonując kolejne ćwiczenie.

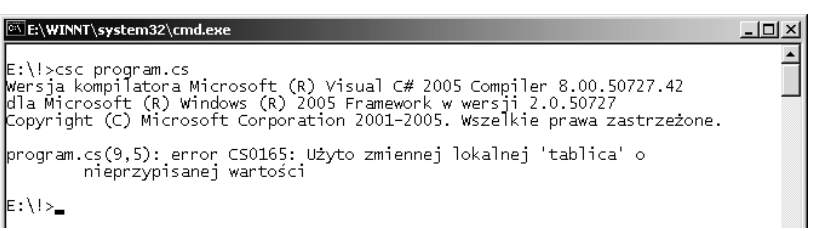
ĆWICZENIE

5.1 Deklaracja zmiennej tablicowej

Zadeklaruj tablicę elementów typu całkowitego. Przypisz zerowemu elementowi tablicy dowolną wartość. Spróbuj wyświetlić zawartość tego elementu na ekranie.

```
using System;

public
class main
{
  public static void Main()
  {
    int[] tablica;
    tablica[0] = 11;
    Console.WriteLine("Zerowy element tablicy to: " + tablica[0]);
  }
}
```

Już przy próbie kompilacji kompilator uprzejmie poinformuje nas, że chcemy odwołać się do zmiennej, która prawdopodobnie nie została zainicjalizowana, tak jak jest to widoczne na rysunku 5.1.



Rysunek 5.1. *Próba użycia niezainicjowanej zmiennej tablicowej*

Skoro więc wystąpił błąd, należy go natychmiast naprawić.

ĆWICZENIE

5.2 Utworzenie tablicy

Zadeklaruj i zainicjalizuj tablicę elementów typu całkowitego. Przypisz zerowemu elementowi tablicy dowolną wartość. Spróbuj wyświetlić zawartość tego elementu na ekranie.

```
using System;

public
class main
{
  public static void Main()
  {
    int[] tablica = new int[1];
    tablica[0] = 10;
    Console.WriteLine("Zerowy element tablicy to: " + tablica[0]);
  }
}
```

Wyrażenie `new tablica[1]` oznacza stworzenie nowej, jednowymiarowej, jednoelementowej tablicy liczb typu `int`. Ta nowa tablica została przypisana zmiennej odnośnikowej o nazwie `tablica`. Od tej chwili można odwoływać się do kolejnych elementów tej tablicy, pisząc:

```
tablica[index]
```

Warto przy tym zauważyć, że elementy tablicy numerowane są od zera, 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

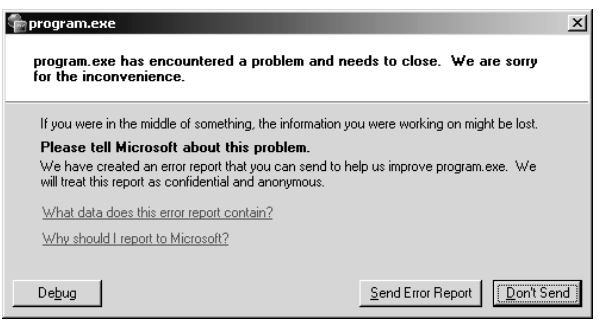
5.3 Odwołanie do nieistniejącego elementu tablicy

Zadeklaruj i zainicjuj tablicę dziesięcioelementową. Spróbuj przypisać elementowi o indeksie 10 dowolną liczbę całkowitą.

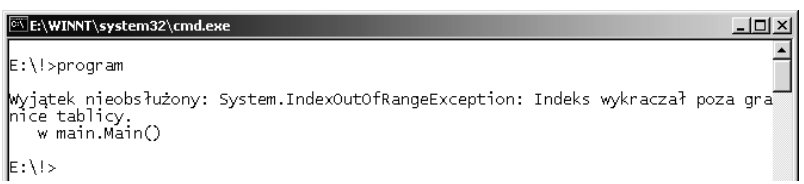
```
using System;

public
class main
{
```

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

Powyższy kod daje się bez problemu skompilować, jednak przy próbie uruchomienia takiego programu na ekranie zobaczymy okno widoczne na rysunku 5.2 informujące o wystąpieniu błędu. Chwilę później (po kliknięciu *Don't send*) ujrzymy na konsoli komunikat podający konkretne informacje o typie błędu oraz miejscu programu, w którym on 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 jednak nic strasznego. Program co prawda nie działa, ale błąd został wychwycony przez środowisko uruchomieniowe. Konkretnie mówiąc, został wygenerowany tak zwany wyjątek i aplikacja zakończyła działanie. Taki wyjątek możemy jednak przechwycić i tym samym zapobiec niekontrolowanemu zakończeniu wykonywania kodu. To jednak odrębny temat, którym zajmiemy się w rozdziale 6. Ważne jest to, że próba odwołania się do nieistniejącego elementu została wykryta i to odwołanie tak naprawdę nie wystąpiło! Program nie naruszył więc obszaru pamięci niezarezerwowanej dla niego.

Inicjalizacja

Tablice można zainicjalizować już w momencie ich tworzenia. Dane, które mają się znaleźć w poszczególnych komórkach, podajemy w nawiasach klamrowych po deklaracji tablicy. Schematycznie wygląda to następująco:

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

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

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

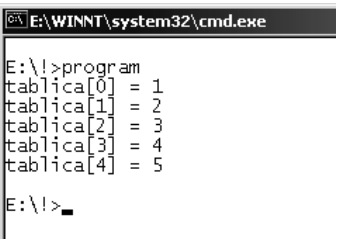
ĆWICZENIE

5.4 Inicjalizacja tablicy

Zadeklaruj tablicę pięcioelementową typu `int` i zainicjuj ją liczbami od `1` do `5`. Zawartość tablicy wyświetl na ekranie.

```
using System;

public
class main
{
  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 kolejne liczby od `1` do `5`.



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

Kiedy inicjujemy tworzoną tablicę z góry znaną liczbą elementów, możemy pominąć fragment kodu związany z tworzeniem obiektu, a kompilator doda go za nas. Zamiast pisać:

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

możemy równie dobrze użyć konstrukcji:

typ[] *nazwa* = {*dana1*, *dana2*,...,*danaN*}

Oba sposoby są sobie równoważne i możemy używać tego, który jest dla nas wygodniejszy.

ĆWICZENIE

5.5 Bezpośrednia inicjalizacja tablicy

Zadeklaruj tablicę pięcioelementową typu `int` i zainicjuj ją liczbami od `1` do `5`. Użyj drugiego z poznanych sposobów inicjalizacji. Zawartość tablicy wyświetl na ekranie.

```
using System;

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

Pętla foreach

Dotychczas poznaliśmy trzy rodzaje pętli: `for`, `while` i `do…while`. W przypadku tablic (jak również kolekcji, którymi się w niniejszej publikacji nie zajmujemy) możemy również skorzystać z pętli typu `foreach`. Jest ona bardzo wygodna, gdyż umożliwia prostą iterację po wszystkich elementach tablicy; nie musimy też wprowadzać dodatkowej zmiennej iteracyjnej. Pętla `foreach` ma postać następującą:

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

Konkretnie, jeżeli mamy tablicę o nazwie `tab` zawierającą liczby typu `int`, możemy zastosować konstrukcję:

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

W tym wypadku w kolejnych przebiegach pętli pod `i` będą podstawiane kolejne elementy tablicy.

ĆWICZENIE

5.6 Wykorzystanie pętli foreach do wyświetlenia zawartości tablicy

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

```
using System;

public
class main
{
  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);
    }
  }
}
```

ĆWICZENIE

5.7 Zliczanie wartości w pętli foreach

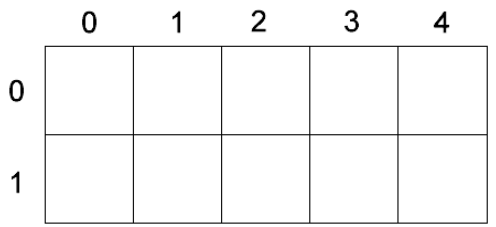
Wykorzystaj pętlę `foreach` do sprawdzenia, ile jest liczb parzystych, a ile nieparzystych w tablicy z elementami typu `int`.

```
using System;

public
class main
{
  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("Tablica zawiera {0} liczb parzystych i {1} liczb
    nieparzystych", parzyste, nieparzyste);
  }
}
```

Tym razem przy wypełnianiu tablicy danymi korzystamy z obiektu klasy `Random`, która udostępnia wartości pseudolosowe. Dokładniej mówiąc, kolejną pseudolosową liczbę całkowitą uzyskujemy, wywołując metodę `Next` tejże klasy. Do stwierdzenia, czy kolejna komórka tablicy zawiera liczbę parzystą czy nieparzystą, wykorzystujemy operator dzielenia modulo. Oczywiście liczba parzysta modulo dwa daje wynik zero i taki też warunek sprawdzamy w instrukcji `if`.

Tablice wielowymiarowe

Tablice nie muszą być jednowymiarowe, jak w dotychczas prezentowanych przykładach. Tych wymiarów może być więcej, na przykład dwa — otrzymujemy wtedy strukturę widoczną na rysunku 5.5, czyli rodzaj tabeli o zadanej ilości wierszy i kolumn. W tym przypadku mamy dwa wiersze oraz cztery kolumny. Oczywiście w takiej sytuacji, aby jednoznacznie wyznaczyć komórkę, trzeba podać dwie liczby.



Rysunek 5.5. *Przykład tablicy dwuwymiarowej*

Musimy się teraz dowiedzieć, w jaki sposób zadeklarować tego typu tablicę. Zacznijmy od deklaracji samej zmiennej tablicowej. W przypadku tablicy dwuwymiarowej ma ona postać:

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

Samą tablicę tworzymy natomiast za pomocą instrukcji:

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

Przykładowo, dwuwymiarową tablicę widoczną na rysunku 5.5 utworzymy następująco (zakładając, że ma ona przechowywać liczby całkowite):

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

Inicjalizacja samych komórek może odbywać się, podobnie jak w przypadku tablic jednowymiarowych, już w trakcie deklaracji:

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

Zobaczmy, jak wygląda to na konkretnym przykładzie.

ĆWICZENIE

5.8 Tworzenie tablicy dwuwymiarowej

Zadeklaruj tablicę dwuwymiarową typu `int` o dwóch wierszach i pięciu kolumnach i zainicjuj ją kolejnymi liczbami całkowitymi. Zawartość tablicy wyświetl na ekranie.

```
using System;

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

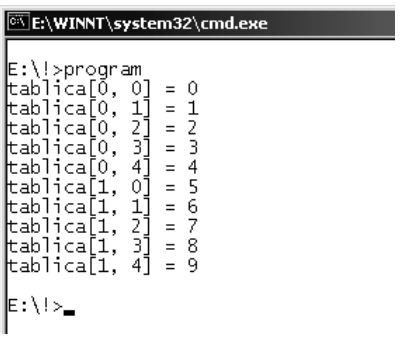
Jak widać, do wypełniania tablicy używamy dwóch zagnieżdżonych pętli `for`. Pierwsza, zewnętrzna, odpowiada za iterację po indeksach wierszy tablicy, druga za iterację po indeksach kolumn. Zmienna `counter` służy nam jako licznik i jest w każdym przebiegu zwiększana o jeden, dzięki czemu w kolejnych komórkach uzyskujemy kolejne liczby całkowite. Po wypełnieniu danymi nasza tablica ma postać widoczną na rysunku 5.6.

Rysunek 5.6. *Tablica z ćwiczenia 5.8 po wypełnianiu danymi*

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9

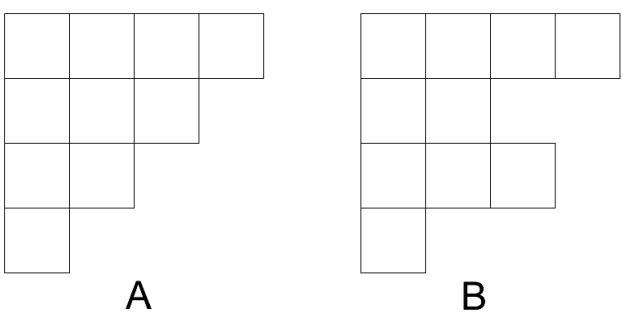
Do wyświetlenia danych używamy analogicznej konstrukcji z dwoma zagnieżdżonymi pętlami. Po uruchomieniu kodu na ekranie zobaczmy widok przedstawiony na rysunku 5.7. Jak widać, dane te zgodne są ze strukturą przedstawioną na rysunku 5.6.

Rysunek 5.7. *Wynik działania programu z ćwiczenia 5.6*



Tablica dwuwymiarowa nie musi mieć wcale, tak jak w poprzednich przykładach, kształtu prostokątnego, tzn. takiego, gdzie liczba komórek w każdym wierszu i każdej kolumnie jest stała. Równie dobrze możemy stworzyć tablicę o kształcie trójkąta (rysunek 5.8.A) lub zupełnie nieregularną (rysunek 5.8.B). Przy tworzeniu struktur nieregularnych musimy się jednak nieco więcej napracować, gdyż każdy wiersz zazwyczaj trzeba tworzyć ręcznie, pisząc odpowiednią linię kodu.

Rysunek 5.8. *Przykłady bardziej skomplikowanych tablic dwuwymiarowych*



Postarajmy się utworzyć strukturę przedstawioną na rysunku 5.8.B. Zauważmy, że każdy wiersz można traktować jako oddzielną tablicę jednowymiarową. Zatem tak naprawdę jest to jednowymiarowa tablica, której poszczególne komórki zawierają inne jednowymiarowe

tablice. Inaczej mówiąc, jest to tablica tablic. Wystarczy zatem zadeklarować zmienną tablicową o odpowiednim typie, a następnie poszczególnym jej elementom przypisać nowo utworzone tablice jednowymiarowe o zadanej długości. To całe rozwiązanie problemu.

Pytanie brzmi: co to znaczy „odpowiedni typ tablicy". Pomyślmy — jeśli w tablicy (jednowymiarowej) chcieliśmy przechowywać liczby całkowite typu `int`, to typem tej tablicy było `int`. Pisaliśmy wtedy:

```
int[] tablica;
```

Jeśli zatem typem nie jest `int`, ale tablica typu `int`, którą oznaczamy jako `int[]`, należy napisać:

```
int[][] tablica;
```

Z kolei utworzenie czteroelementowej tablicy zawierającej tablice z liczbami całkowitymi wymaga wpisu:

```
new tablica[4][];
```

Te wiadomości powinny nam wystarczyć do wykonania kolejnego ćwiczenia.

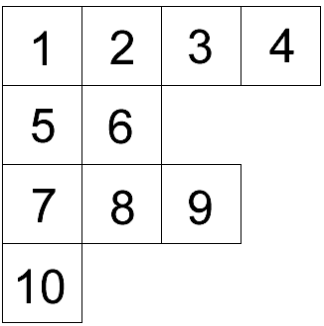
ĆWICZENIE

5.9 Budowa tablicy nieregularnej

Napisz kod tworzący strukturę tablicy widocznej na rysunku 5.8.B przechowującej liczby całkowite. W kolejnych komórkach powinny znaleźć się kolejne liczby całkowite, zaczynając od `1`.

```
using System;

public
class main
{
  public static void Main()
  {
    int[][] tablica = new int[4][];
    tablica[0] = new int[4]{1, 2, 3, 4};
    tablica[1] = new int[2]{5, 6};
    tablica[2] = new int[3]{7, 8, 9};
    tablica[3] = new int[1]{10};
  }
}
```

W tej chwili nasza struktura została wypełniona danymi, tak jak widoczne jest to na rysunku 5.9. Jak sobie jednak poradzić z jej wyświetleniem na ekranie. Oczywiście możemy zrobić to ręcznie, pisząc kod oddzielnie dla każdego wiersza. W przypadku tak małej tablicy nie będzie to problemem. Czy jednak tej czynności nie da się zautomatyzować? Najwygodniej byłoby przecież wyprowadzać dane na ekran w zagnieżdżonej pętli, tak jak w przypadku ćwiczenia 5.8.



Rysunek 5.9. *Tablica z ćwiczenia 5.9 wypełniona przykładowymi danymi*

Okazuje się, że jest to jak najbardziej możliwe, a z nieregularnością naszej tablicy poradzimy sobie w bardzo prosty sposób. Każda tablica jest obiektem; posiada właściwość `Length`, dzięki czemu możemy sprawdzić jej długość. To całkowicie rozwiązuje problem wyświetlenia danych nawet z tak nieregularnej struktury jak obecnie opisywana.

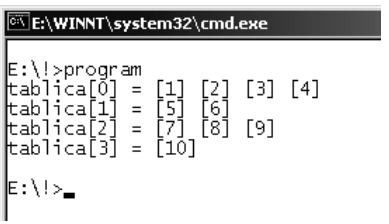
ĆWICZENIE

5.10 Wyświetlanie danych z tablicy nieregularnej

Zmodyfikuj kod z ćwiczenia 5.9 w taki sposób, aby dane zawarte w tablicy zostały wyświetlone na ekranie (rysunek 5.10). W tym celu użyj zagnieżdżonych pętli `for`.

```
using System;

public
class main
{
  public static void Main()
  {
    int[][] tablica = new int[4][];
    tablica[0] = new int[4]{1, 2, 3, 4};
    tablica[1] = new int[2]{5, 6};
    tablica[2] = new int[3]{7, 8, 9};
    tablica[3] = new int[1]{10};
```

```
        for(int i = 0; i < tablica.Length; i++){
          Console.Write("tablica[{0}] = ", i);
          for(int j = 0; j < tablica[i].Length; j++){
            Console.Write("[{0}] ", tablica[i][j]);
          }
          Console.WriteLine("");
        }
      }
    }
```



Rysunek 5.10. *Wyświetlenie danych z nieregularnej tablicy w ćwiczeniu 5.10*