

- Tablice w CSharp
 - Tablica w językach programowania
 - Podstawy
 - Tablice są obiektami
 - Tablice jednowymiarowe i wielowymiarowe
 - Deklaracja zmiennej tablicowej
 - Utworzenie i inicializacja tablicy
 - Dostęp do elementów tablicy
 - Iterowanie po elementach tablicy
 - Tablice postrzepione
 - Tablice postrzepione *versus* prostokątne
 - Pętla `foreach` w tablicach postrzepionych
 - Klasa `Array` i przydatne właściwości i metody
 - Przykład 1 - sortowanie, odwracanie
 - Przykład 2 - konwersja wszystkich elementów na inny typ
 - Przykład 3 - wyszukiwanie elementu w tablicy
 - Przykład 4 - szybkie wyszukiwanie, algorytm `binsearch`

Tablice w CSharp

tablice jednowymiarowe, wielowymiarowe, postrzepione, klasa `Array` i jej metody użytkowe

Tablica w językach programowania

Tablica (*array*) to systematyczne rozmieszczenie podobnych obiektów, zazwyczaj w formie wierszy i kolumn.

W programowaniu termin te odnosi się do *struktury danych* oraz do *typu* (typu tablicowego).

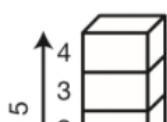
- Tablica jako struktura danych - kolekcja elementów (wartości lub zmiennych), każdy identyfikowany przez co najmniej jeden indeks (lub klucz). Istotą tablic jest:
 - jednakowy typ elementów,
 - umieszczenie w spójnym bloku pamięci, dzięki czemu pozycja dowolnego elementu tablicy może być obliczona za pomocą prostego wzoru matematycznego. zapewniony jest stały i "natychmiastowy" dostęp do elementów tablicy.
- Tablica jako typ danych - sposób notacji typu, definiowania zmiennych typu tablicowego, odwoływanie się do elementów tablic

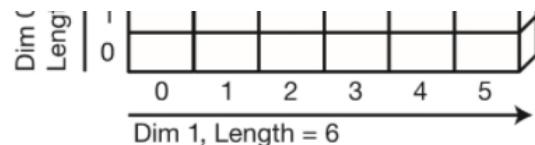
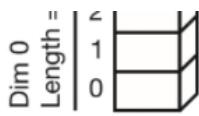
Podstawy

- **Element** - składnik tablicy. Wszystkie elementy muszą być tego samego typu lub wywodzić się z tego samego typu bazowego (prog. obiektywe)
- **Wymiar (rząd)** - (*dimension/rank*). Tablica ma określony wymiar, np. 1-wymiarowa, 2-wymiarowa (macierz), 3-wymiarowa ...
W C# numer wymiaru nazywany jest rzędem (*rank*).
- **Rozmiar (dla wymiaru)** - (*dimension length*). Każdy wymiar tablicy ma określony rozmiar (*length*)
- **Rozmiar tablicy** (dla całej tablicy) - (*array length*). Liczba elementów umieszczonych w tablicy.

...

- W C# z chwilą, gdy tablica zostanie utworzona, nie może zmienić swoich rozmiarów. W C# nie ma tablic dynamicznych - zamiast nich stosujemy wyspecjalizowane struktury danych (np. zbiory, listy, słowniki, ...).
- W C# tablice indeksowane są zawsze od `0` (*0-based indexing*) i nie można tego zmienić. Zatem, dla tablicy 1-wymiarowej o rozmiarze `n` elementy indeksowane są od `0` do `n-1`.





One-Dimensional Array, `int[5]`

- Rank = 1
- Length of Array = 5

Two-Dimensional Array, `int[3,6]`

- Rank = 2
- Length of Array = 18

...

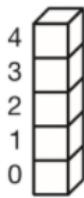
W C# rozróżnia się dwa rodzaje tablic:

- jednowymiarowe - postrzegane jak wektor (matematyczny)
Przykład: `int[] tab = new int[5];`
- wielowymiarowe - składające się z tablicy tablic (tzn. jest to główna tablica 1-wymiarowa, której elementami są inne tablice, tzw. *podtablice*).

Ze względów ułatwień notacyjnych tablice wielowymiarowe dzieli się na:

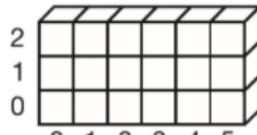
- regularne, gdy *podtablice* są tych samych rozmiarów: np. 2-wymiarowe kwadratowe, prostokątne, np. 3-wymiarowe sześciennie, prostopadłościenne, ...
Przykład: `int[,] macierz = new int[3,5];`
- postrzępione (*jagged arrays*), gdy *podtablice* są wyraźnie oznaczonymi elementami tablicy głównej i są być może różnych różnych rozmiarów
Przykład: `int[][] jagged = new int[4][];`

One-Dimensional Arrays

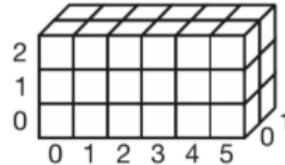


One-Dimensional
`int[5]`

Rectangular Arrays

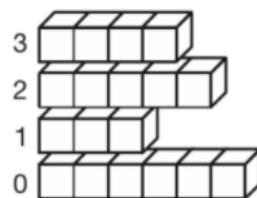


Two-Dimensional
`int[3,6]`



Three-Dimensional
`int[3,6,2]`

Jagged Arrays



Jagged Array
`int[4][]`

Tablice są obiektami

Język C# jest językiem obiektowym - wszystkie typy wywodzą się z typu 'object' - również typ tablicowy.

Tablice (konkretnie) są więc obiektami określonego typu tablicowego (np. `int[]`, `double[,]`, `string[]`, ... - nieskończanie wiele typów tablicowych).

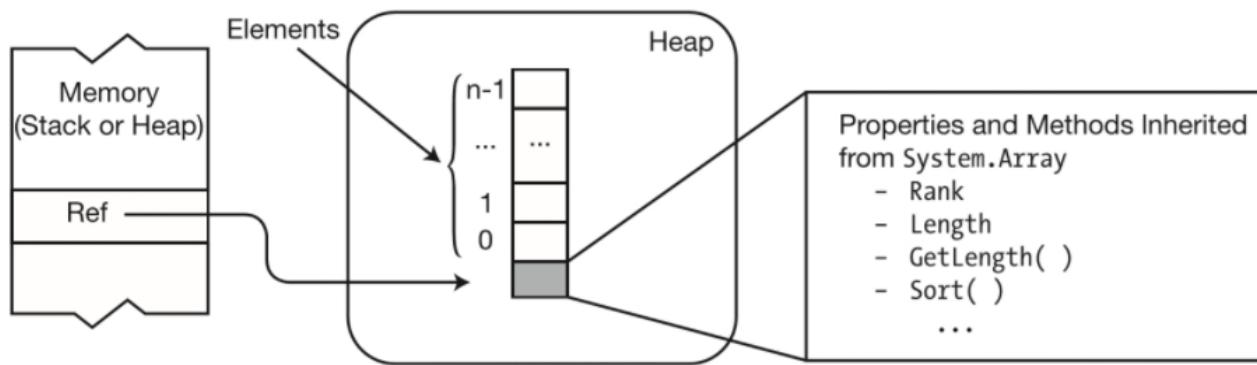
Typ danej tablicy dziedziczy ze specjalnej, prototypowej klasy `System.Array`. Specjalnej - ponieważ w specjalny sposób obsługiwanej notacyjnie z poziomu języka (przyjęto zwyczajową notację wywodzącą się z języków C/C++/Java).

Skoro tablice dziedziczą ze specjalnej bazowej klasy w bibliotece klas (BCL), zatem dziedziczą również pewne składniki. Najważniejszymi są:

- `Rank` - właściwość zwracająca liczbę wymiarów tablicy (1 dla jednowymiarowej, 2 dla prostokątnej, 3, ...)
- `Length` - właściwość zwracająca rozmiar, całkowitą liczbę elementów tablicy.
- oraz cały szereg innych, ułatwiających działania na tablicach.

Tablice są typu referencyjnego, i - tak jak inne obiekty - przechowywane są na stercie. Sama referencja do tablicy

przechowywana jest na stosie.

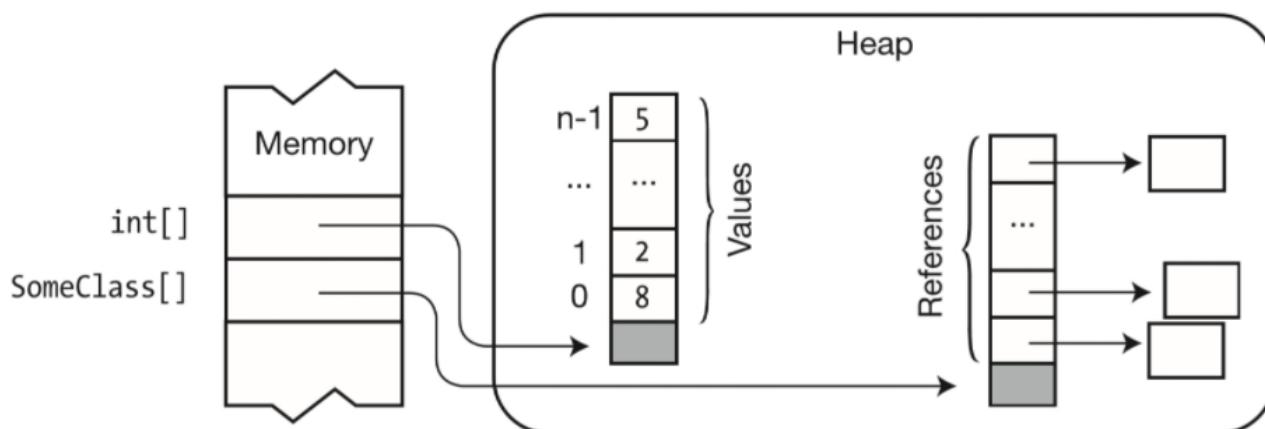


Elementami tablic mogą być wartości (np. `int`, `double`, `bool`, ...) jak i inne obiekty (np. `string`, `Osoba`, ...)- a formalnie referencje do tych obiektów.

Stąd, czasami mówi się o:

- *value type array* - jeśli jej elementy są typu wartościowego,
- *reference type array* - jeśli elementy są typu referencyjnego (są obiektami).

Bez względu na to, czy elementy tablicy są typu wartościowego, czy referencyjnego, znajdują się one na stercie.



Tablice jednowymiarowe i wielowymiarowe

Z punktu widzenia składni języka tablice 1-wymiarowe i wielowymiarowe są traktowane tak samo. Inny sposób zapisu pojawia się w przypadku tablic postrzępionych.

Deklaracja zmiennej tablicowej

Deklaracja zmiennej tablicowej składa się z określenia typu tablicowego (a więc typu elementów i wymiaru), a następnie nazwy zmiennej.

```
int[] tab1D;  
int[,] tab2D;  
int[,,] tab3D;
```

Ważne: przy deklaracji zmiennej tablicowej nie podajemy rozmiarów (pojawi się błąd kompilacji):

```
int[5] tab1D;      // źle  
int[2,5] tab2D;    // źle  
int[2,3,4] tab3D; // źle
```

Typ elementów i wymiar są składnikami typu tablicowego, ale nie fizyczny rozmiar tablicy - który będzie ustalony dopiero w chwili jej tworzenia.

Sama deklaracja tablicy powoduje:

1. utworzenie na stosie zmiennej tablicowej,
 2. zmienna ta jest nieokreślona (tzn. jest null),
 3. nie powoduje fizycznego zarezerwowania pamięci na stercie czy utworzenia tablicy.

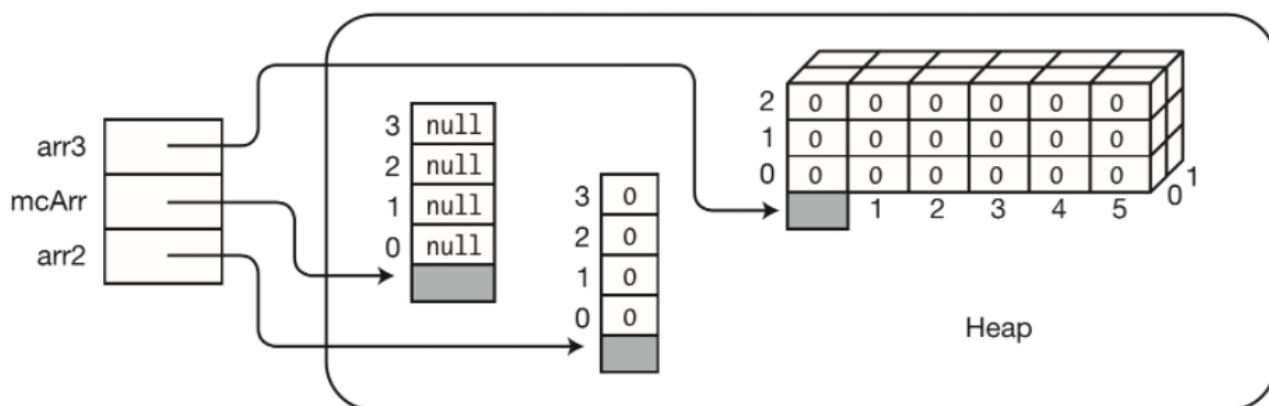
Uwaga: W językach C/C++/Java możliwa jest deklaracja tablicy postaci: `int tab[]`. Zapis ten oznacza deklarację zmiennej `tab`, która jest tablicą liczb całkowitych. W C# taki zapis jest niepoprawny. Nawiązki kwadratowe w deklaracji mogą być użyte wyłącznie w określeniu typu (czyli: `int[] tab`).

Utworzenie i inicjalizacja tablicy

Do utworzenia tablicy jedno- lub wielowymiarowej (instancji typu tablicowego), stosujemy **wyrażenie tworzące tablicę** (ang. *array-creation expression*), używając operatora **new** - podobnie jak w przypadku tworzenia obiektów - oraz określając jawnie rozmiar tablicy (rozmiary w poszczególnych wymiarach):

```
int[] arr2 = new int[4];
string[] mcArr = new string[4];
int[,] arr3 = new int[3,6,2];
```

- zmienne tablicowe (`arr2`, `mcArr`, `arr3`) przechowują referencje do fizycznych tablic (obiektów typu tablicowego) i tworzone są na stosie,
 - tablice (obiekty typu tablicowego) tworzone są na stercie i wypełniane domyślną wartością dla typu składowego (np. `0` w przypadku typów liczbowych, `null` w przypadku typów referencyjnych),
 - raz utworzona tablica nie może zmienić swojego rozmiaru.

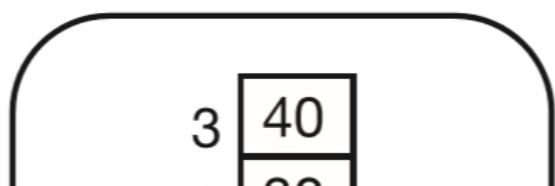
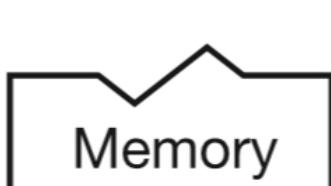


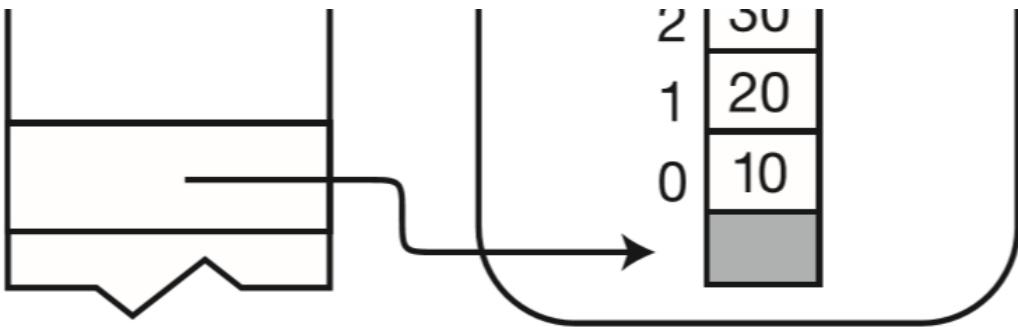
W trakcie tworzenia tablicy można określić wartości jej elementów (jeśli tablica nie jest zbyt duża - z praktycznych powodów). Proces ten nazywany jest *jawną inicjalizacją tablicy* (ang. explicit initialization) i realizowany jest poprzez podanie tzw. **listy inicjalizacyjnej**.

Poniższe instrukcje deklaracji i inicializacji tablic jednowymiarowych są sobie równoważne:

```
int[] tab1Da = new int[4] { 10, 20, 30, 40 };
int[] tab1Db = new int[] { 10, 20, 30, 40 };
int[] tab1Dc =
Console.WriteLine( $"tab1Da: {tab1Da}, Length={tab1Da.Length}" );
Console.WriteLine( $"tab1Db: {tab1Db}, Length={tab1Db.Length}" );
Console.WriteLine( $"tab1Dc: {tab1Dc}, Length={tab1Dc.Length}" );
```

```
tab1Da: System.Int32[], Length=4  
tab1Db: System.Int32[], Length=4  
tab1Dc: System.Int32[], Length=4
```

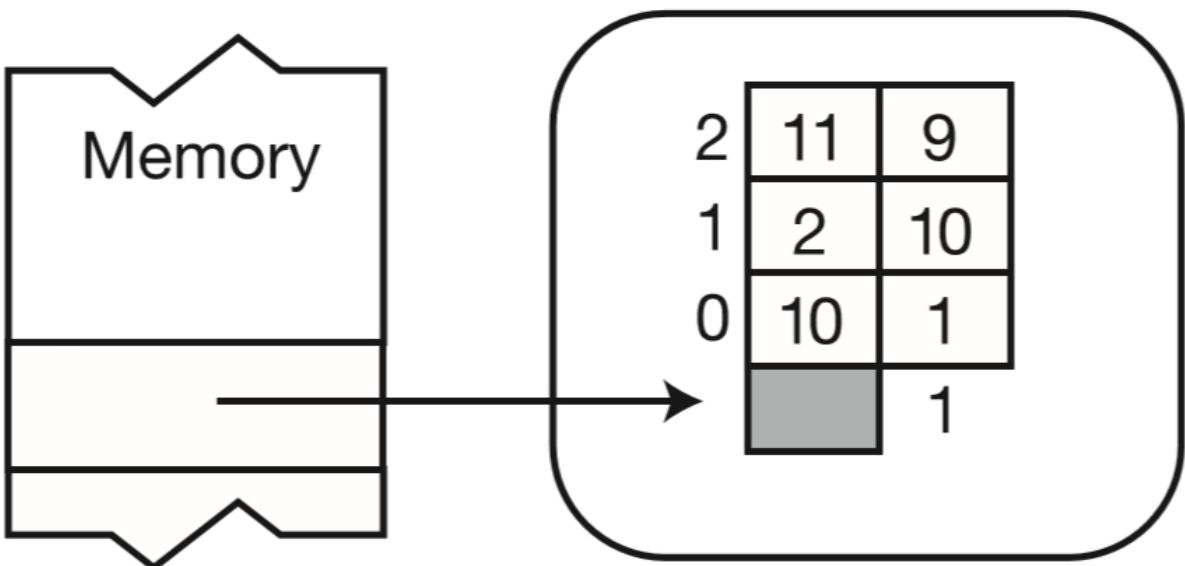




Listy inicjalizacyjne możemy również stosować dla tablic wielowymiarowych. Poniższe instrukcje są sobie równoważne:

```
int[,] tab2Da = new int[3,2] { {10, 1}, {2, 10}, {11, 9} };
int[,] tab2Db = new int[,] { {10, 1}, {2, 10}, {11, 9} };
int[,] tab2Dc =
    { {10, 1}, {2, 10}, {11, 9} };
int[,] tab2Dd = {
    {10, 1},
    {2, 10},
    {11, 9}
};
Console.WriteLine($"tab2Da: \n" +
    $" Type = {tab2Da.GetType()} \n" +
    $" Length = {tab2Da.Length} \n" +
    $" Rank = {tab2Da.Rank} \n" +
    $" GetLength(0) = {tab2Da.GetLength(0)} \n" +
    $" GetLength(1) = {tab2Da.GetLength(1)}");
```

```
tab2Da:
Type = System.Int32[,]
Length = 6
Rank = 2
GetLength(0) = 3
GetLength(1) = 2
```



Dla wygody możemy zapisywać listę inicjalizacyjną tablic wielowymiarowych "wierszami" - jak w przypadku `tab2Dd`.

Zmienne lokalne możemy deklarować z użyciem słowa kluczowego `var`. Jeśli tablice będą lokalne i na podstawie ich deklaracji i inicjalizacji można wywnioskować ich typ, również i je można będzie deklarować niejawnie (*implicitly*):

```
var tab1 = new int[] { 1, 2, 3 }; //OK
var tab2 = new [] { 1, 2, 3 }; //OK
var tab3 = new { 1, 2, 3 }; //ile - nie wiadomo,
                           //czy tablica, czy inna kolekcja
```

Dostęp do elementów tablicy

Elementy tablic numerowane są kolejnymi liczbami całkowitymi począwszy od 0. Najczęściej typ indeksu to int.

Formalnie, zgodnie z dokumentacją C#, tablice można indeksować dowolnym typem całkowitym: int, uint, long, ulong lub każdym innym niejawnie konwertowanym do podanych. Jednakże istnieje ograniczenie dotyczące wielkości obiektu: 2GB.

Zatem, można teoretycznie zapisać:

```
int[] bigTable[9_223_372_036_854_775_800]
```

tylko, czy CLR na stercie znajdzie taki wielki spójny obszar pamięci.

Do elementów tablic odwołujemy się, podając nazwę zmiennej oraz w nawiasie kwadratowym jej indeks (indeksy):

```
var tab = new int[] { 1, 2, 3 };
tab[0] = 4; // zmiana wartości elementu o indeksie 0
            //teraz tablica składa się z elementów {4, 2, 3}
tab.SetValue( 5, 1 ); // ustawiasz wartość 5 dla elementu o indeksie 1
                      //teraz tablica składa się z elementów {4, 5, 3}
Console.WriteLine( $"tab[0]={tab[0]}");
Console.WriteLine( $"tab[1]={tab[1]}");
Console.WriteLine( $"tab[2]={tab[2]}");
```

```
tab[0]=4
tab[1]=5
tab[2]=3
```

Nawias kwadratowy [] można traktować jako specjalny operator (operator wskazujący na wartość o numerze podanym w nawiasach kwadratowych).

Operator ten nazywa się indekserem. W C#, projektując własne typy danych, możemy definiować własne indeksery (czyli określać zasady dostępu do elementów własnych kolekcji).

Iterowanie po elementach tablicy

Ponieważ tablice zawierają zazwyczaj wiele elementów, najczęściej chcesz te tablice w jakiś określony sposób przeglądać (i ewentualnie na elementach przeglądanych wykonać pewne działania). Wykorzystujesz w tym celu instrukcje pętli.

Przeglądanie indeksowane "w przód":

```
var tab = new int[] { 1, 2, 3 };
for(int i = 0; i < tab.Length; i++)
{
    Console.WriteLine( $"tab[{i}]={tab[i]}");
```

```
tab[0]=1
tab[1]=2
tab[2]=3
```

Przeglądanie indeksowane "wstecz":

```
var tab = new int[] { 1, 2, 3 };
for(int i = tab.Length-1; i >= 0; i--)
{
    Console.WriteLine( $"tab[{i}]={tab[i]}");
```

```
tab[2]=3
tab[1]=2
tab[0]=1
```

Przeglądanie co drugiego elementu:

```
var tab = new int[] { 1, 2, 3, 4 };
for(int i = 0; i < tab.Length; i=i+2 )
{
    Console.WriteLine( $"tab[{i}]={tab[i]}");
```

```
tab[0]=1  
tab[2]=3
```

Przeglądanie tablicy 2-wymiarowej (zagnieźdżenie pętli):

```
int[,] tab2D = {  
    {10, 1},  
    {2, 10},  
    {11, 9}  
};  
  
for( int row = 0; row < tab2D.GetLength(0); row++ )  
{  
    for( int col = 0; col < tab2D.GetLength(1); col++ )  
    {  
        Console.WriteLine( tab2D[row, col] + " " );  
    }  
    Console.WriteLine();  
}
```

```
10 1  
2 10  
11 9
```

UWAGA: odwołanie się do elementu o indeksie spoza dozwolonego zakresu spowoduje zgłoszenie wyjątku

`IndexOutOfRangeException`:

```
int[,] tab2D = {  
    {10, 1},  
    {2, 10},  
    {11, 9}  
};  
  
for( int row = 0; row < 2; row++ ) // omyłkowo, zamiast 3 wstawiono 2  
    for( int col = 0; col < 3; col++ ) // omyłkowo, zamiast 2 wstawiono 3  
        Console.WriteLine( tab2D[row, col]);
```

```
10  
1  
System.IndexOutOfRangeException: Index was outside the bounds of the array.  
  at Submission#0.<>d__0.MoveNext() in C:\Predmioty\CScharp-2019\Arrays\arrays-wyklad\gepefzysa_code_chunk:line 9  
--- End of stack trace from previous location where exception was thrown ---  
  at Dotnet.Script.core.ScriptRunner.Execute[TReturn](String dllPath, IEnumerable`1 commandLineArgs) in C:\Users\appveyor\AppData\Local\Temp\tmp382F\Dotnet.Script.core\ScriptRunner.cs:line 100
```

Jeśli **nie zależy Ci** na informacji o położeniu elementu w tablicy oraz **nie będziesz** zmieniał wartości elementu w tablicy, a chcesz przeglądając **wszystkie** elementy tablicy, możesz zastosować pętlę `foreach`.

Zalety:

- Wiesz, że wszystkie elementy zostaną przeglądnięte.
- Nie przekroczysz zakresu indeksów (nie pojawi się wyjątek).

Wady:

- Nie możesz zmodyfikować elementu tablicy.
- Nie znasz położenia elementu w tablicy.

Przeglądanie tablicy i wypisanie elementów parzystych:

```
int[,] tab2D = {  
    {10, 1},  
    {2, 10},  
    {11, 9}  
};  
  
foreach( var x in tab2D )  
{  
    if( x % 2 == 0 )
```

```
        Console.WriteLine(x + " ");
    }
```

```
10 2 10
```

Ważne: Pętla `foreach` jest konstrukcją języka C# umożliwiającą przeglądanie elementów dowolnych *kolekcji* (formalnie sekwencji wartości typu `IEnumerable`), ale zapewniająca **tylko odczyt**. Mogą to być tablice 1-wymiarowe lub wielowymiarowe. Przeglądanie to zapewnia tzw. *enumerator* (w innych językach zwany *iteratorem*).

Tablice postrzępione

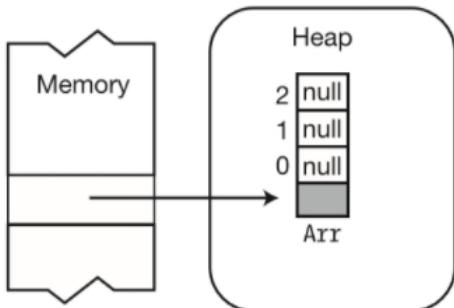
Tablice postrzępione (ang. *jagged arrays*) to tablice, których elementami są tablice. W przeciwieństwie do tablic prostokątnych, w tablicach postrzępionych podtablice mogą być różnych rozmiarów.

Tablic postrzępionych **nie można** utworzyć "za jednym krokiem".

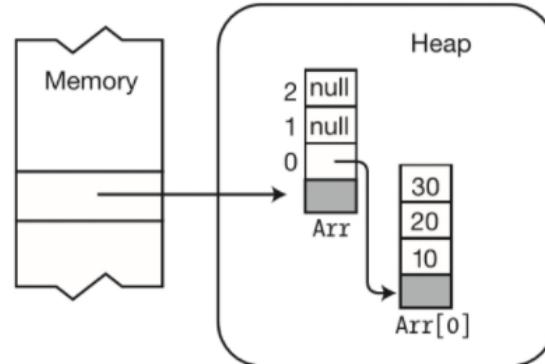
Przykład:

```
int[][] Arr; //deklaracja - tablica postrzępiona
             //      liczb całkowitych
             //      o wymiarze 2
Arr = new int[3][]; //1. Instancja najwyższego poziomu
Arr[0] = new int[] {10, 20, 30}; //2. Instancja podtablicy
Arr[1] = new int[] {40, 50, 60, 70}; //3. Instancja kolejnej podtablicy
Arr[2] = new int[] {80, 90, 100, 110, 120}; //4. Instancja ostatniej podtablicy
```

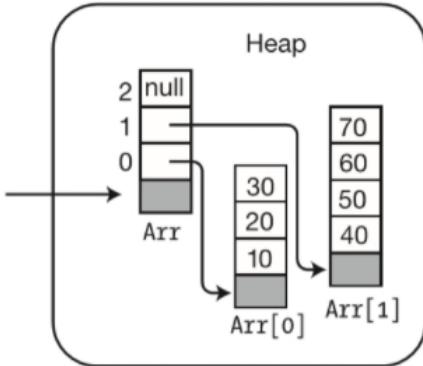
1.



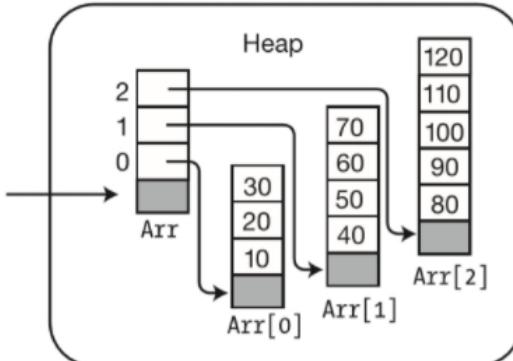
2.



3.



4.



Taka organizacja elementów w tablicach postrzępionych wymaga uważnego ich przeglądania. Możemy bowiem napotkać `null` zamiast oczekiwanej podtablicy oraz podtablice mogą być różnych rozmiarów.

```
int[][] Arr = new int[3][]
{
    new int[] {1, 2, 3},
    null,
```

```

        new int[] {5, 6}
    );
for( int i = 0; i < Arr.Length; i++ )
{
    Console.Write( $"Arr[{i}] = " );
    if( Arr[i] is null )
        Console.WriteLine("null");
    else
        for( int j = 0; j < Arr[i].Length; j++ )
            Console.Write( Arr[i][j] + " " );

    Console.WriteLine();
}

```

```

Arr[0] = 1 2 3
Arr[1] = null
Arr[2] = 5 6

```

Oczywiście możliwe jest utworzenie tablicy, zawierającej tablice 2-wymiarowe (np. tablica zdjęć).

```

int[][] Arr = new int[3][][] {new int[,] {{10, 20}, {100, 200}},  

                            new int[,] {{30, 40, 50}, {300, 400, 500}},  

                            new int[,] {{60, 70, 80, 90}, {600, 700, 800, 900}}}  

                            );  

// Wyswietla elementy tablicy:  

for (int i = 0; i < Arr.GetLength(0); i++)  

{
    for (int j = 0; j < Arr[i].GetLength(0); j++)  

    {
        for (int k = 0; k < Arr[i][j].GetLength(1); k++)
            Console.WriteLine($"Arr[{i}] [{j},{k}]: {Arr[i][j][k]}");
        Console.WriteLine();
    }
    Console.WriteLine();
}

```

```

Arr[0] [0,0]: 10  Arr[0] [0,1]: 20  

Arr[0] [1,0]: 100  Arr[0] [1,1]: 200  
  

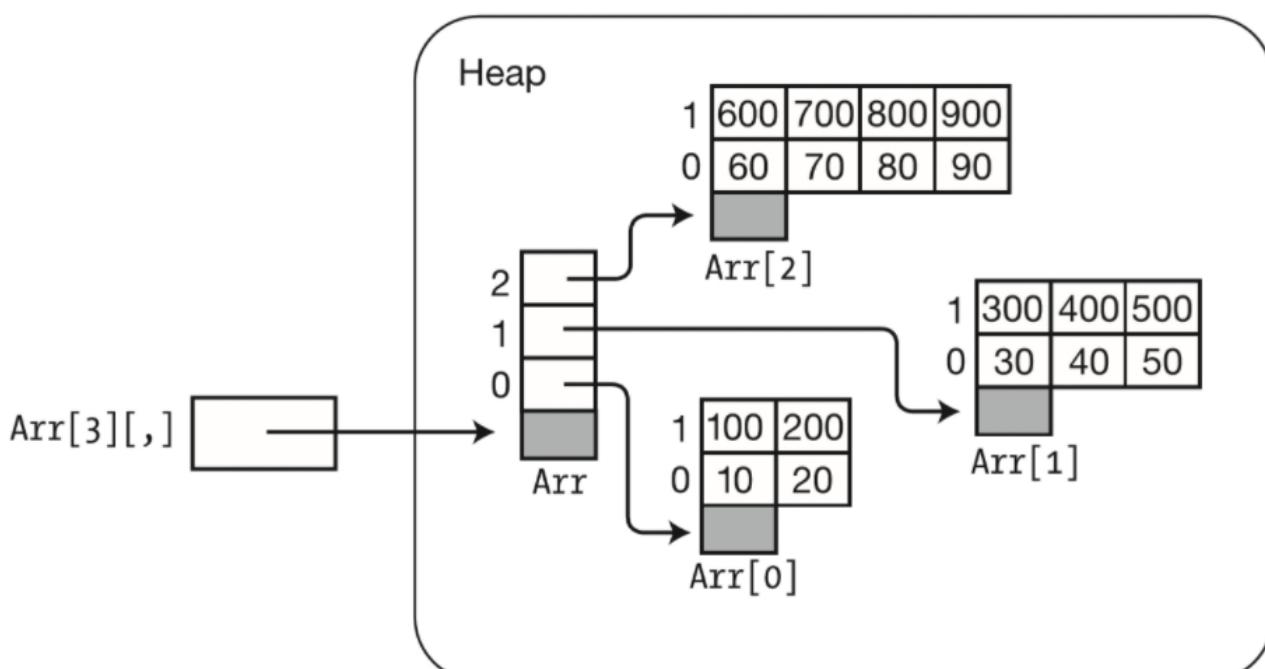
Arr[1] [0,0]: 30  Arr[1] [0,1]: 40  Arr[1] [0,2]: 50  

Arr[1] [1,0]: 300  Arr[1] [1,1]: 400  Arr[1] [1,2]: 500  
  

Arr[2] [0,0]: 60  Arr[2] [0,1]: 70  Arr[2] [0,2]: 80  Arr[2] [0,3]: 90  

Arr[2] [1,0]: 600  Arr[2] [1,1]: 700  Arr[2] [1,2]: 800  Arr[2] [1,3]: 900

```



Tablice postrzebione versus prostokatne

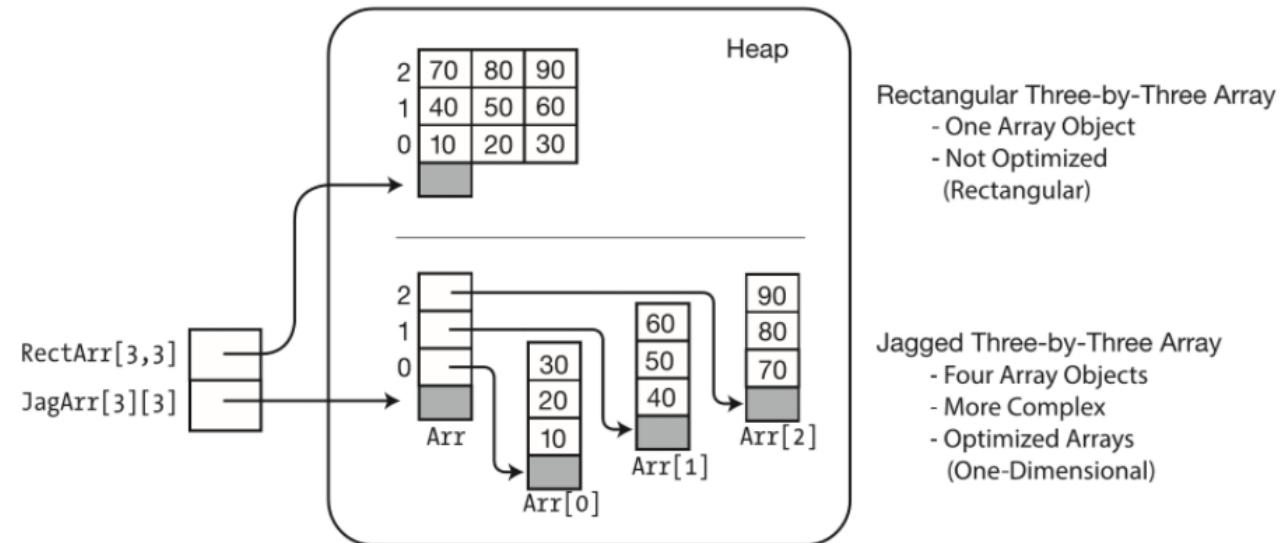
Struktura tablic postrzepionych i prostokatnych jest znacząco inna.

UWAGA: Błędem jest zapisanie:

```
int[][] JagArr = new int[3][3]; //błąd kompilacji
```

nawet, jeśli mielibyśmy utworzyć tablicę o wymiarach 3×3 .

Tablica prostokątna jest pojedynczą tablicą (obiektem), tymczasem tablica postrzepiona to 4 tablice (obiekty).



Tablice 1-wymiarowe są optymalizowane w trakcie kompilacji (pamięć, szybkość odwołania do elementów). Tablice prostokątne nie są optymalizowane na tym samym poziomie. Dlatego, czasami może się okazać, że tablice postrzepione - jako tablice tablic jednowymiarowych - są bardziej wydajne (w niektórych zastosowaniach), niż tablice prostokątne (ale w innych wcale nie muszą być). O tym, które rozwiązanie jest wydajniejsze dla rozwiązania określonego problemu decyduje programista, po wykonaniu szeregu testów wydajnościowych.

Pętla `foreach` w tablicach postrzepionych

Ponieważ tablice postrzepione są tablicami tablic, należy oddzielnie przetwarzać je pętlami `foreach`.

```
int[][] Arr = new int[3][]
{
    new int[] {1, 2, 3},
    null,
    new int[] {5, 6}
};

foreach( int[] subArray in Arr )
{
    int sum = 0;
    if( subArray is null )
        continue;
    foreach( var x in subArray )
    {
        sum += x;
    }
    Console.WriteLine( $"sum = {sum}" );
}
```

```
sum = 6
sum = 11
```

Klasa `Array` i przydatne właściwości i metody

Member	Type	Lifetime	Meaning
--------	------	----------	---------

Rank	Property	Instance	Gets the number of dimensions of the array
Length	Property	Instance	Gets the total number of elements in all the dimensions of the array
GetLength	Method	Instance	Returns the length of a particular dimension of the array
Clear	Method	Static	Sets a range of elements to 0 or <code>null</code>
Sort	Method	Static	Sorts the elements in a one-dimensional array
BinarySearch	Method	Static	Searches a one-dimensional array for a value, using binary search
Clone	Method	Instance	Performs a shallow copy of the array—copying only the elements, both for arrays of value types and reference types
IndexOf	Method	Static	Returns the index of the first occurrence of a value in a one-dimensional array
Reverse	Method	Static	Reverses the order of the elements of a range of a one-dimensional array
GetUpperBound	Method	Instance	Gets the upper bound at the specified dimension

W klasie `Array` zdefiniowane są przydatne algorytmy operujące na tablicach jednowymiarowych.

Przykład 1 - sortowanie, odwracanie

```
int[] arr = new int[] { 15, 20, 5, 25, 10 };
// String.Join() łączy separator z każdym elementem tablicy
Console.WriteLine("Oryginalna: " + String.Join(", ", arr));

Array.Sort(arr);
Console.WriteLine("Po sortowaniu: " + String.Join(", ", arr));

Array.Reverse(arr);
Console.WriteLine("Po odwroceniu: " + String.Join(", ", arr));

Console.WriteLine();
Console.WriteLine($"Rank = { arr.Rank }, Length = { arr.Length }");
Console.WriteLine($"GetLength(0)      = { arr.GetLength(0) }");
Console.WriteLine($"GetType()        = { arr.GetType() }");
```

Oryginalna: 15, 20, 5, 25, 10
Po sortowaniu: 5, 10, 15, 20, 25
Po odwroceniu: 25, 20, 15, 10, 5

Rank = 1, Length = 5
GetLength(0) = 5
GetType() = System.Int32[]

Przykład 2 - konwersja wszystkich elementów na inny typ

Zamiana tekstu zawierającego ciąg liczb na tablicę liczb

```
string napis1 = "1 23 3 2 15 6 3 8";
string[] tablicaNapisow1 = napis1.Split( new char[] { ' ' },
                                         StringSplitOptions.RemoveEmptyEntries
                                         );
int[] tablica1 = Array.ConvertAll( tablicaNapisow1, int.Parse );
Console.WriteLine( String.Join(", ", tablica1) );

string napis2 = "1, 23, 3, 2, 15, 6, 3, 8";
string[] tablicaNapisow2 = napis2.Split( new char[] { ',', ' ' },
                                         StringSplitOptions.RemoveEmptyEntries
                                         );
int[] tablica2 = Array.ConvertAll( tablicaNapisow2, int.Parse );
Console.WriteLine( String.Join(", ", tablica2) );
```

1, 23, 3, 2, 15, 6, 3, 8
1, 23, 3, 2, 15, 6, 3, 8

Przykład 3 - wyszukiwanie elementu w tablicy

- `Find`, `FindAll`, `FindLast` - zwraca znaleziony element, lub - jeśli nie znaleziono - wartość domyślnej typu.
- `FindIndex`, `FindLastIndex` - zwraca index znalezioneego elementu, lub - jeśli nie znaleziono - zwraca `-1`.
- `Exists` - sprawdza, czy element spełniający określony predykat jest w tablicy

W powyższych funkcjach należy podać **predykat** - funkcję sprawdzającą określony warunek logiczny i zwracającą `true` lub `false`. Najczęściej podaje się ja w zapisie *lambda*, np. `x => (x==10)`.

```
var los = new Random();
int[] tab = new int[100];
for(int i = 0; i < tab.Length; i++)
    tab[i] = los.Next(1, 10); //liczba pseudolosowa z zakresu 1..9

Console.WriteLine( Array.Find( tab, x => (x==10) ) ); //nie znaleziono, wynik default(int)

Console.WriteLine( Array.FindIndex( tab, x => (x==10) ) ); //nie znaleziono, wynik -1

Console.WriteLine( Array.Exists( tab, x => (x==10) ) ); //nie znaleziono, wynik false

Console.WriteLine( Array.Find( tab, x => (x > 8) ) );

Console.WriteLine( "Liczba wystapien liczby 8 = " +
    Array
        .FindAll( tab, x => (x == 1) )
        .Length
    );
```

```
0
-1
False
9
Liczba wystapien liczby 8 = 5
```

Wszystkie wymienione funkcje wymagają przeglądnięcia **wszystkich** elementów tablicy. Mówimy, że ich złożoność wynosi `O(n)`, czyli czas działania jest wprost proporcjonalny do rozmiaru zbioru (liniowo zależny).

Przykład 4 - szybkie wyszukiwanie, algorytm `binsearch`

Dla n -elementowej tablicy nieuporządkowanej, przeszukanie jej wymaga n iteracji.

Jeśli jednak tablica jest posortowana, proces wyszukiwania elementu można skrócić znaczco ($O(\log_2(n))$), stosując algorytm wyszukiwania połówkowego `BinarySearch`. Jeśli elementu nie ma, zwracany jest wynik `-1`.

UWAGA: dla tablicy nieposortowanej wynik działania algorytmu może być (najczęściej jest) błędny.

```
var los = new Random();
int[] tab = new int[100];
for(int i = 0; i < tab.Length; i++)
    tab[i] = los.Next(1, 10000); //liczba pseudolosowa z zakresu 1..99999

Console.WriteLine( Array.Find( tab, x => (x==10) ) );

Console.WriteLine( Array.FindIndex( tab, x => (x==10) ) );

Console.WriteLine( Array.Exists( tab, x => (x==10) ) );

Array.Sort( tab );
Console.WriteLine( Array.BinarySearch( tab, 10 ) );
```

```
0
-1
False
-1
```