Модуль 2. Організація і обробка складених типів даних

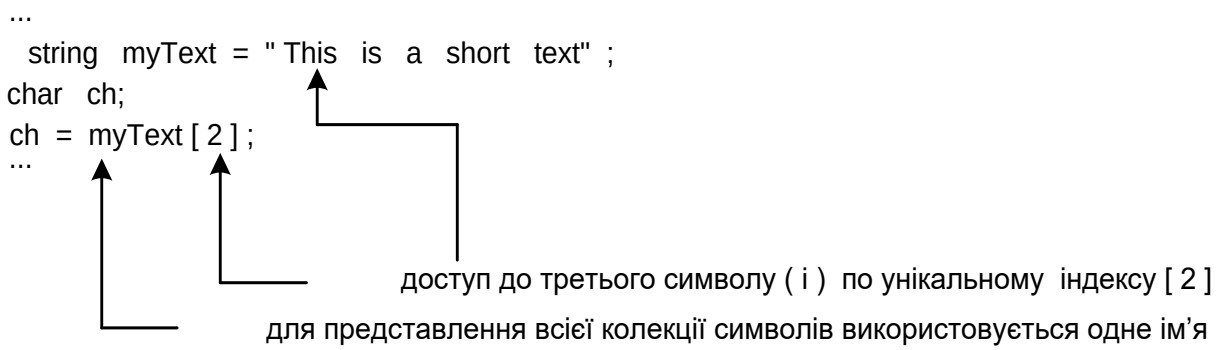
Тема 4. Масиви

4.1. Загальні відомості про масиви

4.1.1. Оголошення й визначення масиву

Тип масиву дуже схожий на тип **string** — саме тому тип **string** часто іменується як масив символів.

Змінна типу **string** може містити набір чітко розташованих і проіндексованих символів. Доступ до кожного з них здійснюється наступним чином (рис. 5.1):



або:

```
string myText = "To b or not to be said the bee" ;
int bCounter = 0;
for ( int i=0 ; i<myText . Length ; i + + )   ← ініціалізуючи змінну i значенням 0 і інкрементуючи її в кожному циклі:
{
        ↓ програма звертається до кожного конкретного символу:
    if ( myText [ i ] . ToString() . ToUpper () == "B" )
    bCounter + +;   ← і підраховує кількість символів b
}
Console . WriteLine ( "Number of b ' s in text: " + bCounter) ;
```

Рис. 4.1. **Інтерпретація типу string як масиву символів**

У цьому фрагменті коду підраховується кількість елементів **b** в рядку **myText** і виводиться повідомлення:

Number of b's in text: 3

Рядок **string** і масив є як *типами-класами*, так і *посилальними* типами.

Як **System.String** є базовим класом для типу **string**, так і **System.Array** представляє собою основу для масиву.

Отже, масив — це об'єкт, що, подібно об'єкту **string**, має багато вбудованих методів, використовуваних для доступу до елементів колекцій даних та їхньої обробки.

Між рядками **string** і масивами є чимало подібності, але в той час як тип **string** обмежується лише представленням колекцій символів, *масив може містити колекцію величин будь-якого типу*, включаючи який-небудь із простих типів **int**, **long**, **decimal** і т. д., а також будь-яку групу об'єктів.

У загальному випадку **масив** є іменованою структурою даних (або об'єктом), для якої задана відповідність між множиною індексів і комірок, званих **елементами масиву**.

Усі елементи масиву повинні належати тому самому типу. Тип елементів масиву називається типом масиву, або **базовим типом масиву.**

Елементи масиву іноді називають **індексованими змінними**, або просто елементами.

Змінна типу **array** оголошується у вихідній програмі за допомогою зазначення масиву, що супроводжується порожньою парою квадратних дужок і ім'ям масиву. В наступному рядку

decimal [ ] accountBalances;

оголошується змінна **accountBalances**, що здатна зберігати посилання на об'єкт масиву, що містить колекцію чисел типу **decimal**.

Попереднім оголошенням створюється порожній контейнер, здатний зберігати посилання на об'єкт масиву. Сам об'єкт масиву з колекцією величин типу **decimal** ще не створений, і пам'ять під нього не виділена.

Об'єкт типу **array**, подібно будь-якому іншому класу (за винятком **string**), повинен створюватися із застосуванням ключового слова **new**.

В рядку коду на рис. 5.2. створюється об'єкт масиву (класу **System.Array**), що містить колекцію з п'яти величин типу **decimal**. Посилання на нього присвоюється змінній **accountBalances**.

Рис. 4.2. **Створення об'єкта масиву з п'ятьома елементами**

Оператори оголошення й присвоювання можна об'єднати в одному рядку, як показано нижче:

decimal [ ] accountBalances = new decimal [5];

Іноді можна зустріти синтаксис оголошення змінних масиву і створення нових об'єктів, що відрізняється від розглянутого тут.

У ньому прямо використовуються посилання на імена класів і методів .NET Framework, наприклад:

```
System.Array rainfall = System.Array.CreateInstance(
                        Type.GetType(“System.Decimal”), 5 );
```

де оголошується змінна **rainfall**, якій присвоюється посилання на новий об'єкт базового типу **decimal**.

Після того як оголошена змінна масиву **accountBalances** і їй присвоєний новий об'єкт масиву, одержуємо поточний стан, показаний на рис. 5.3.

Тепер **accountBalances** містить посилання, що вказує на конкретний об'єкт класу **System.Array**, розташований в оперативній пам'яті. Він може містити колекцію з п'яти величин типу **decimal**. Оскільки кожна з них вимагає 128 біт (або 16 байт), ця частина об'єкта масиву займе в пам'яті 5 x 128 = 640 біт (або 80 байт). Саме посилання вимагає 4 байти пам'яті.

Розмір масиву (або довжина масиву) – це загальне число елементів, які масив може містити.

При створенні масиву можна визначати елементи і присвоювати їм початкові значення.

Коли масив створюється без будь-яких початкових значень (див. попередні приклади) елементи *автоматично ініціалізуються* значеннями за замовчуванням. Значення залежать від типу елементів масиву:

Числовим величинам типів **short**, **int**, **float**, **decimal** і т. д. присвоюється значення **нуль**.

Величинам типу **char** присвоюється символ **\u0000**.

Величини типу **bool** ініціалізуються значенням **false**.

Величини посилальних типів ініціалізуються значенням **null**.

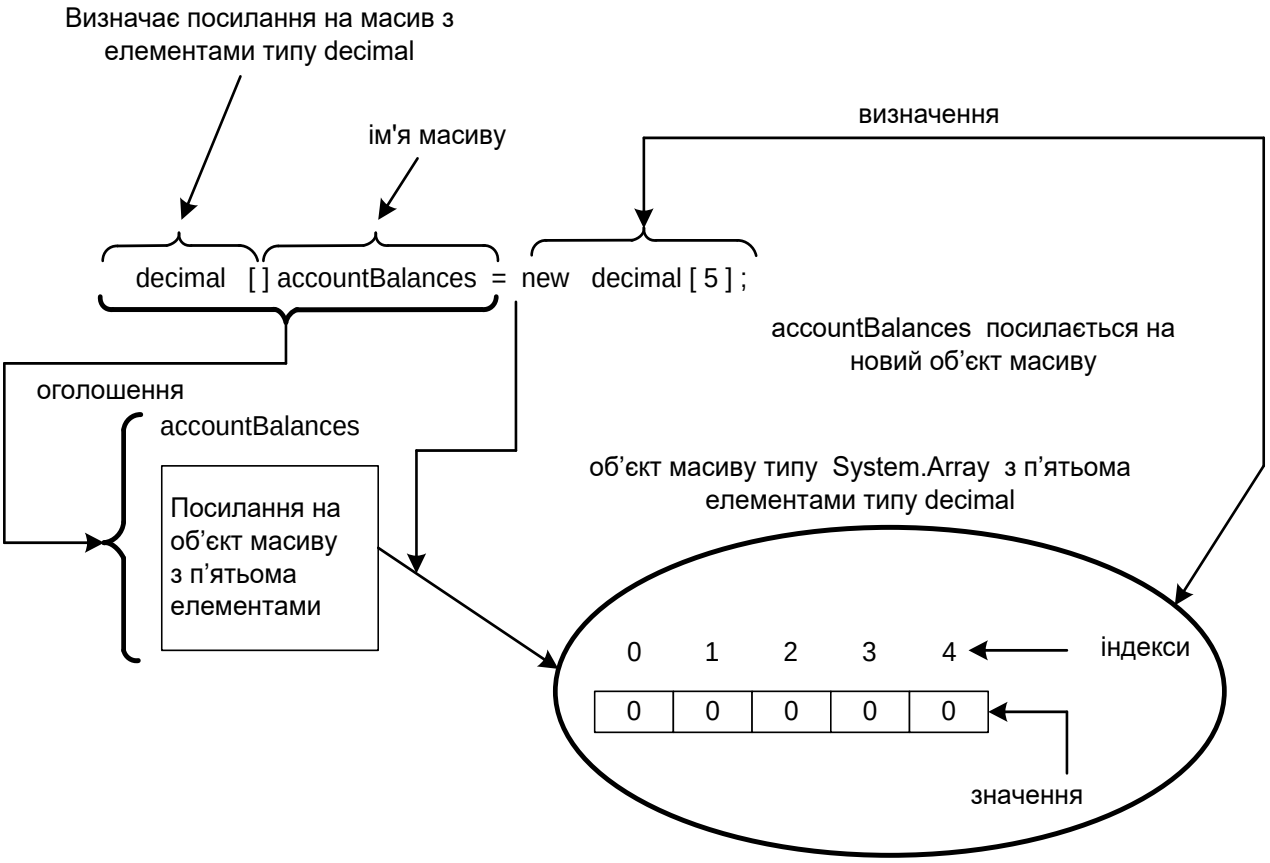


Рис. 4.3. **Посилання на об'єкт масиву**

У наведеному прикладі тип елемента масиву — **decimal**, тому всі величини ініціалізуються нулем, як показано на рис. 5.3.

Гарна методика програмування передбачає явну ініціалізацію в програмі, а не на етапі компіляції.

Значення null

Коли посилання не вказує на конкретний об'єкт, воно має значення **null**. Воно сумісне з усіма посилальними типами, представлено в мові C# ключовим словом **null**.

Якщо потрібне посилання, що ні на що не вказує, значення **null** можна присвоїти йому прямо: **eievator3 = null;**

Загальний синтаксис для оголошення й створення масиву наведений в наступному синтаксичному блоці.

Оголошення масиву й присвоювання йому значення

Оголошення_масиву ::=
<Базовий_тип> [ ] <Ідентифікатор масиву>;

Створення_масиву ::= new <Базовий_тип> [<Довжина_масиву>];
Присвоювання_посилання_на_масив ::= <Ідентифікатор_масиву> =
new <Базовий_тип> [<Довжина_масиву>];

Присвоювання_посилання_на_масив ::= <Оголошення_масиву>
new <Базовий_тип> [<Довжина_масиву>];

<Базовий_тип> в оголошенні повинен бути ідентичним **<Базовий_тип>** в операторі породження об'єкта.

<Довжина_масиву> повинна бути позитивною й належати типу, що неявно перетворюється до **int**. Це може бути літерал, константа або змінна.

Квадратні дужки **[ ]** в даному випадку є частиною синтаксису; вони не вказують на необов'язковість заключених в них елементів синтаксису.

Коли масив оголошений і присвоєне посилання на його об'єкт, можна звертатися і до його окремих елементів.

4.1.2. Доступ до окремих елементів масиву

Доступ до індивідуальних елементів масиву здійснюється так само, як до окремих символів змінної типу **string** — за ім'ям змінної з наступною парою квадратних дужок, що містять індекс. В наступному рядку коду

accountBalances[0] = 1000m;

величина **l000** типу **decimal** присвоюється першому елементу масиву, на який посилається **accountBalances**. Індекс може бути будь-яким числовим вираженням, що дає в результаті ненегативну величину типу, який неявно перетворюється в **int**.

Індекс повинен мати значення не більше, ніж довжина масиву мінус 1.

Рядки (типу **string**) *є незмінними*: будь-який символ в рядку не можна замінити. Масив же дозволяє присвоювати різні значення своїм елементам.

Елемент **accountBalances[0]** можна розглядати як звичайну змінну типу **decimal**: присвоювати та читати її значення, а також використовувати її в будь-якому вираженні.

Наприклад, щоб вивести суму десятивідсоткового нарахування на **accountBalances[0**], можна скористатися оператором:

Console.WriteLine(accountBalances[0] * 0.1m);

Приклад. Нарахування відсотків.

У вихідному коді, представленому нижче, оголошений масив з п'яти балансів рахунків. Суми присвоюються двом першим елементам, потім по них нараховується десять відсотків, і результат виводиться на екран. Інші три елементи масиву **accountBalances[2]**, **accountBalances[3]** і **accountBalances[4]** в цій програмі не використовуються.

```
1: using System;
2:
3: class SimpleAccountBalances
4: {
5:     public static void Main()
6:     {
7:         const decimal interestRate = 0.1m;
8:         decimal [ ] accountBalances;
9:
10:        accountBalances = new decimal [5];
11:
12:        Console.WriteLine("Please enter two account balances: ");
13:        Console.Write("First balance: ");
```

```
14:          accountBalances[0] = Convert.ToDecimal(Console.ReadLine());
15:          Console.Write("Second balance: ");
16:          accountBalances[1] = Convert.ToDecimal(Console.ReadLine());
17:
18:          accountBalances[0] = accountBalances[0] +
accountBalances[0] * interestRate;
19:          accountBalances[1] = accountBalances[1] +
accountBalances[1] * interestRate;
20:
21:          Console.WriteLine("New balances after interest: ");
22:          Console.WriteLine("First balance: {0:N2}", accountBalances[0]);
23:          Console.WriteLine("Second balance: {0:N2}",
accountBalances[1]);
24:    }
25:}
```

Результат роботи програми:

```
Please enter two account balances:
First balance: 1034,5
Second balance: 667,3
New balances after interest:
First balance: 1 137,95
Second balance: 734,03
Press any key to continue
```

У рядку 8 оголошується змінна **accountBalances**, що містить посилання на масив з елементами типу **decimal**.

У рядку 10 створюється об'єкт класу **System.Array**, виділяється пам'ять під нього, а посилання на нього присвоюється змінній **accountBalances**.

У рядках 14 і 16 елементам масиву з індексами 0 і 1 присвоюються задані користувачем значення. Якщо користувач вводить суми 1000 і 2000, об'єкт масиву має вигляд як на рис. 5.4, де представлений синтаксис доступу до кожного елемента масиву.

У рядках 18 і 19 на два рахунки нараховується відсоток, визначенний **const interestRate** в рядку 7.

Нові значення виводяться в рядках 22 і 23.

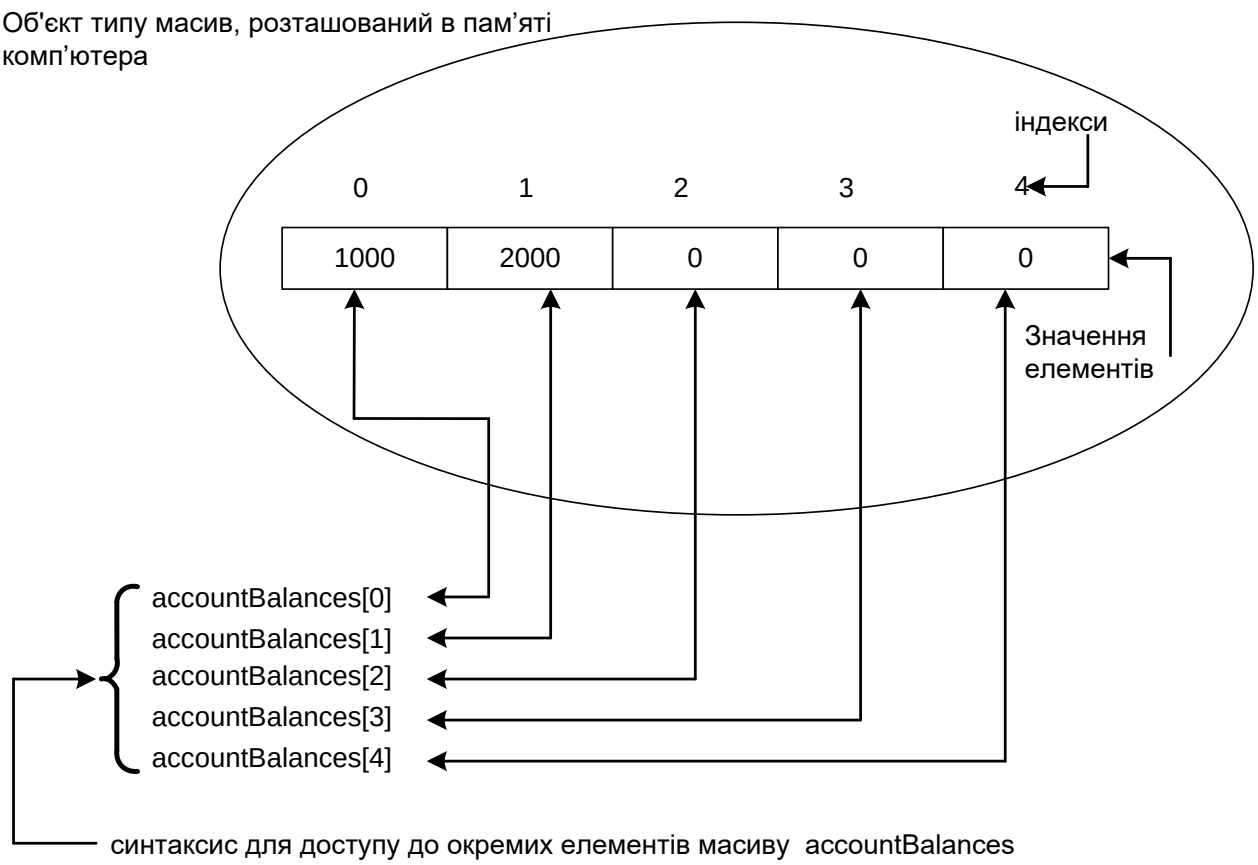


Рис. 4.3. **Два значення, що присвоюються accountBalances**

Синтаксис для доступу до окремих елементів масиву формалізований в наступному синтаксичному блоці.

Синтаксичний блок доступу до елементів масиву

Доступ_до_елемента_масиву ::=

<Ідентифікатор_масиву> [<Числове_вираження>]

Квадратні дужки [ ] в даному випадку не вказують на необов'язковість заключених в них елементів синтаксису, фактично в них розміщується індекс у формі **<Числове_вираження>.**

<**Числове_вираження**> може бути будь-яким числовим вираженням з типом, що неявно перетворюється в **int**. Тут можна використовувати вираження типу **sbyte**, **byte**, **short**, **ushort** і **char**. Будь-який інший тип повинен або мати визначений користувачем шлях явного перетворення, або бути тим, що приводиться до потрібного типу неявно.

Важливо розрізняти три випадки, в яких квадратні дужки використовуються з масивами:

1). для оголошення змінної типу масив:

decimal [ ] accountBalances;

2). для зазначення довжини масиву при створенні об'єкта типу масив:

new decimal [5];

3). для доступу до індивідуальних елементів масиву:

accountBalances[0]

Приклад. Нарахування відсотків (ускладнений варіант).

Розглянемо застосування масиву в сполученні з оператором циклу. Функціональні властивості цієї програми аналогічні програмі з попереднього прикладу, однак їхні реалізації розрізняються. Об'єднання в програмі циклу **for** з масивом **accountBalances** дозволяє користувачеві:

ввести п'ять балансів рахунків;

додавати відсотки до кожного рахунку;

вивести на екран п'ять результатів.

```
1: using System;
2:
3 :class AccountBalanceTraversal
4: {
5:     public static void Main()
6:     {
7:         const decimal interestRate = 0.1m;
8:
9:             decimal [ ] accountBalances;
10:
11:        accountBalances = new decimal [5];
12:
13:        Console.WriteLine("Please enter {0} account balances:", accountBalances.Length);
14:        for (int i = 0; i < accountBalances.Length; i++)
15:        {
16:            Console.Write("Enter balance with index {0}: ", i);
```

```
17:                 accountBalances[i] =
Convert.ToDecimal(Console.ReadLine());
18:         }
19:
20:         Console.WriteLine("\nAccount balances after adding
interest\n");
21:         for (int i = 0; i < accountBalances.Length; i++)
22:         {
23:             accountBalances[i] = accountBalances[i]
24:                 + (accountBalances[i] * interestRate);
25:             Console.WriteLine("Account balance with index {0}:
{1:N2}",
26:                 i, accountBalances[i]);
27:         }
28:   }
29:}
```

Результат роботи програми:

```
Please enter 5 account balances:
Enter balance with index 0: 10000
Enter balance with index 1: 20000
Enter balance with index 2: 15000
Enter balance with index 3: 50000
Enter balance with index 4: 100000

Account balances after adding interest
Account balance with index 0:11 000,00
Account balance with index 1:22 000,00
Account balance with index 2:16 500,00
Account balance with index 3:55 000,00
Account balance with index 4:110 000,00
Press any key to continue
```

Тут (як і раніше) **accountBalances** посилається на об'єкт масиву, що містить 5 величин типу **decimal** (рядки 9 і 11).

Цикл **for**, розташований в рядках 14 – 18, повторюється п'ять разів. Значення змінної **i** починається з 0 і збільшується на 1 при кожному повторенні тіла циклу.

Виконання зупиняється, коли умова циклу стає рівною **false**.

Вираження **accountBalances.Length** еквівалентне властивості **Length** класу **string** і є одним із багатьох корисних вбудованих властивостей і методів об'єкта **System.Array**.

Властивість **Length** повертає довжину масиву, в даному випадку 5, і робить вираження **i < accountBalances.Length** рівним **false**, коли **i** стає рівним 5. Отже, коли тіло циклу виконується востаннє, **i** дорівнює 4.

Це добре узгоджується з функціональністю, що реалізується в тілі циклу в рядках 16 і 17:

отримати нову суму від користувача і присвоїти її елементу масиву з індексом, що починається з 0 і збільшується на 1 для кожної нової суми.

Кожному з усіх 5-ти елементів масиву користувач присвоює нову суму.

Цикл **for** в рядках 21 – 27 містить ті ж оператори ініціалізації, умови і оновлення циклу, як і попередній оператор **for**.

Нарахування відсотків здійснює оператор в рядках 23 і 24. Виведення нового балансу відбувається в рядках 25 і 26.

Слід зазначити, що область видимості змінної **i**, оголошеної в рядку 14, обмежується рядками 14 – 18. Ця змінна відрізняється від **i**, оголошеної в рядку 21, яка має область видимості в межах рядків 21 – 27.

У визначенні довжини масиву в програмі варто використовувати властивість **Length**, а не літерали або константи. Це дозволяє програмі самостійно контролювати зміну розміру масиву й вимагає зміни коду лише в єдиному місці.

4.1.3. Ініціалізація масивів

Елементи масиву можна ініціалізувати будь-якими значеннями під час його створення.

Спочатку потрібно оголосити змінну масиву. Але замість створення об'єкта з ключовим словом **new**, як в наступному прикладі:

new int [6];

потрібно явно визначити список ініціалізуємих величин, указуючи їх в дужках після оператора присвоювання (рис. 5.4).

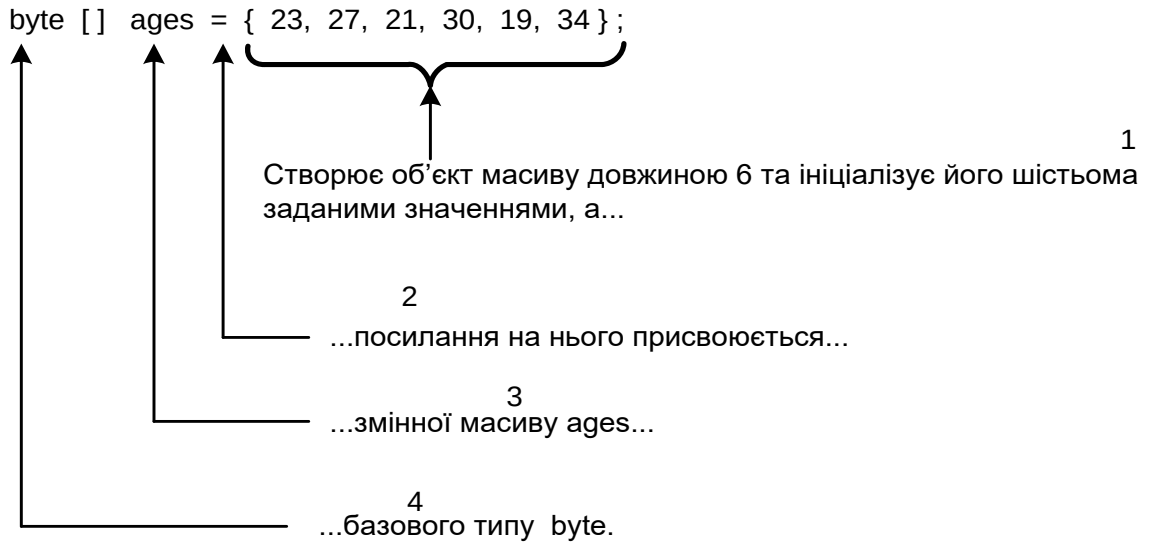


Рис.4.4. **Приклад явної ініціалізації елементів масиву**

Об'єкт масиву автоматично створюється з кількістю елементів (в цьому прикладі 6), необхідних для зберігання значень у фігурних дужках. У даному випадку немає необхідності визначати довжину масиву явно. В дужках задаються значення тільки того типу, що може бути явно перетворений в базовий тип, визначений в оголошенні масиву (в даному випадку **byte**).

Довжину масиву можна визначити і явно (хоча за замовчуванням вона задається кількістю елементів в фігурних дужках). Наступний рядок еквівалентний попередньому (див. рис. 5.4):

byte [ ] ages = new byte [6] { 23, 27, 21, 30, 19, 34} ;

new byte [6] тут не є обов'язковим, однак ця конструкція має дві переваги:

довжина масиву видна відразу при читанні коду – це зручно, коли список значень в дужках досить довгий;

компілятор порівнює довжину масиву, зазначену в квадратних дужках, з кількістю елементів в фігурних дужках. Будь-які невідповідності приводять до повідомлення про помилку (це дозволяє автоматично знайти пропущені або зайві значення).

При використанні фігурних дужок для ініціалізації масиву необхідно задавати значення всіх елементів. Якщо деякі елементи потрібно пропустити, можна скористатися одним із наступних прийомів:

задати ініціалізуючі значення в дужках, вказавши при цьому для деяких елементів значення за замовчуванням (приміром, 0). Якщо, скажімо, в прикладі, наведеному раніше, значення трьох останніх елементів невідомі, можна використовувати такий оператор:

byte [ ] ages = { 23, 27, 21, 0, 0, 0};

Синтаксичний блок оголошення і явної ініціалізації масиву

Оголошення_і_явна_ініціалізація масиву ::=

<Базовий_тип> [ ] <Ідентифікатор_масиву> =
new <Базовий_тип> [<Довжина_масиву>]
{ <Значення1>, <Значення2>,...<ЗначенняN>
} ;

<Значення1>, **<Значення2>** і т. д. повинні явно приводитися до **<Базовий_тип>**.

Кількість значень, що присвоюється масиву (позначена тут **N**), дорівнює довжині масиву.

4.1.4. Перебір елементів масиву за допомогою оператора foreach

Крім циклу **for**, для проходження по всьому масиву можна скористатися оператором **foreach**.

Наприклад, для виведення значення кожного елемента масиву **childbirths,** оголошеного й визначеного як

uint [ ] childbirths = {1340, 3240, 1003, 4987, 3877};

може використовуватися оператор **foreach**:

```
foreach ( uint  temp   in   childbirths )
{
        Console.WriteLine(temp);
}
```

який дає виведення:

1340 3240 1003 4987 3877

Розглянемо докладно цей спосіб.

Оператор **foreach** складається із заголовка й тіла (рис. 5.5). Тіло циклу може бути одиночним або складеним оператором.

Перші два слова в круглих дужках заголовка є, відповідно, типом і ідентифікатором. Разом вони оголошують ітераційну змінну оператора **foreach**. У даному випадку вона називається **temp** (від слова **temporary** – тимчасовий).

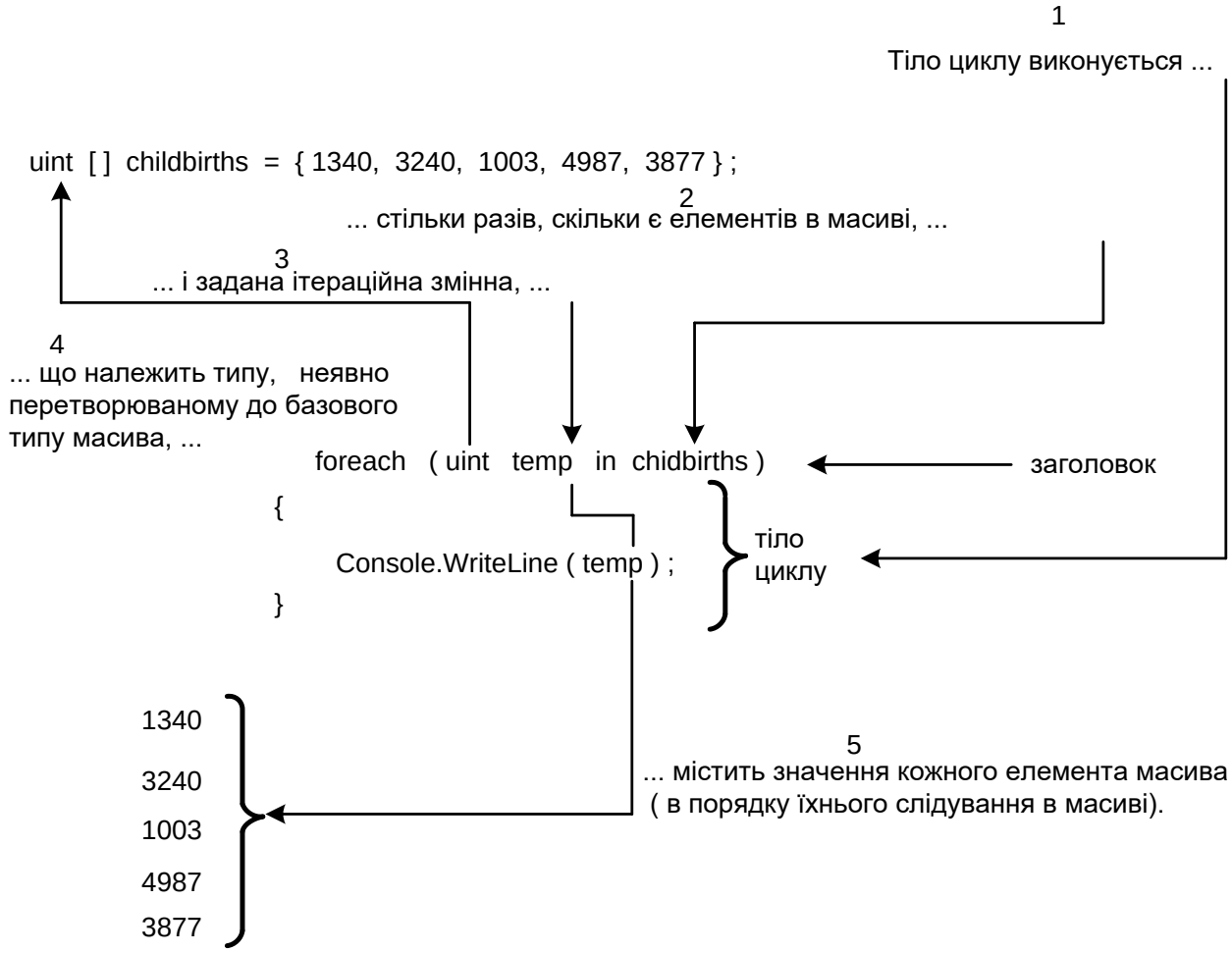


Рис. 4.5. **Оператор foreach**

Праворуч від ітераційної змінної знаходиться ключове слово **in**, за яким іде масив (у даному випадку **childbirths**). Важливо, щоб тип ітераційної змінної міг бути неявно перетворений в базовий тип масиву.

Тіло циклу виконується один раз для кожного елемента. Ітераційну змінну можна використовувати й у тілі циклу. При першому проході (виконанні) вона дорівнює першому елементу масиву й т. д.

Синтаксичний блок оператора foreach

Оператор foreach ::=
foreach (<Тип> <Ідентифікатор_ітераційної_змінної>
in <Ідентифікатор_масиву>)
[< Оператор > | <Складений_оператор>]

Виконуючи оператор **foreach**, C# автоматично визначає кількість ітерацій. При цьому кожний елемент масиву присвоюється ітераційній змінній без необхідності явної індексації.

Лічильник, умова і оновлення циклу (необхідні в стандартному циклі **for**), в даному випадку непотрібні, що забезпечує простоту та ясність коду.

4.2 Приклади обробки одномірних масивів

*4.2.1. Обчислення функції y = a*x^2 + sin(x)*

```
using System;
	class Class1
	{
		static void Main()
		{
			const double a = 10.5;
			double [ ] mas = new double[7];
//		double [ ] mas = {-1, -0.93, -0.49, 0, 1.13, 0.96, 1.75};
//		double [ ] mas = new double[7]{-1, -0.93, -0.49, 0, 1.13, 0.96,
1.75};
			double y;

			// Введення масиву
			Console.WriteLine("Введіть значення елементів
масиву");
			for ( int i=0; i < mas.Length; i++)
			{
				Console.Write("mas[{0}] = ",i);
				mas[i] = Convert.ToDouble(Console.ReadLine());
			}
```

```
                    // Контрольне виведення масиву
                    Console.WriteLine("Вихідний масив:");
                    for ( int i=0; i < mas.Length; i++)
                    {
                            Console.Write("{0} ",mas[i]);
                    }
                    Console.WriteLine(); // переведення рядка

                    //  Обчислення функції
                    for ( int i=0; i < mas.Length; i++)
                    {
                    y = a*mas [ i ] * mas [ i ] - Math.Sin ( mas [ i ] );
                    Console.WriteLine("При  значенні  x={0},  y={1:N4}",mas[i],
y);
                    }
            }
        }
```

Результат роботи програми:

Введіть значення елементів масиву

mas[0] = -1

mas[1] = -0,93

mas[2] = -0,49

mas[3] = 0

mas[4] = 1,13

mas[5] = 0,96

mas[6] = 1,75

Вихідний масив:

-1 -0,93 -0,49 0 1,13 0,96 1,75

При значенні x=-1, y=11,3415

При значенні x=-0,93, y=9,8831

При значенні x=-0,49, y=2,9917

При значенні x=0, y=0,0000

При значенні x=1,13, y=12,5030

При значенні x=0,96, y=8,8576

При значенні x=1,75, y=31,1723

Press any key to continue

4.2.2. Пузиркове сортування

```
using System;
class Class1
{
	static void Main()
	{
		const double a = 10.5;
		double [ ] mas = new double[7];
		// double [ ] mas = {-1, -0.93, -0.49, 0, 1.13, 0.96, 1.75};
		// double [ ] mas = new double[7] {-1, -0.93, -0.49, 0, 1.13, 0.96,
1.75};
		double y;

		// Введення масиву
		Console.WriteLine("Введіть значення елементів масиву");
		for ( int i=0; i < mas.Length; i++)
		{
			Console.Write("mas[{0}] = ",i);
			mas[i] = Convert.ToDouble(Console.ReadLine());
		}

		// Контрольне виведення масиву
		Console.WriteLine("Вихідний масив:");
		for ( int i=0; i < mas.Length; i++)
		{
			Console.Write("{0}  ",mas[i]);
		}
		Console.WriteLine(); // переведення рядка
		//  Обчислення функції
		double t;
		for ( int i=0; i < mas.Length-1; i++)
			for ( int j=0; j < mas.Length-1; j++)
				if(mas[j] < mas[j+1])      // варіант if(mas[j] >
mas[j+1])
					{
```

```
                        t=mas[j];
                        mas[j] = mas[j+1];
                        mas[j+1]=t;
                    }
            // Контрольне виведення масиву
            Console.WriteLine("Масив після сортування:");
            for ( int i=0; i < mas.Length; i++)
            {
                Console.Write("{0}  ",mas[i]);
            }
            Console.WriteLine(); // переведення рядка
        }
}
```

Результат роботи програми:

```
Введіть значення елементів масиву
mas[0] = 1,3
mas[1] = -34,12
mas[2] = 5,23
mas[3] = 6
mas[4] = 0,23
mas[5] = -0,56
mas[6] = 65,3
Вихідний масив:
1,3  -34,12  5,23  6  0,23  -0,56  65,3
Масив після сортування:
65,3  6  5,23  1,3  0,23  -0,56  -34,12
Press any key to continue
```

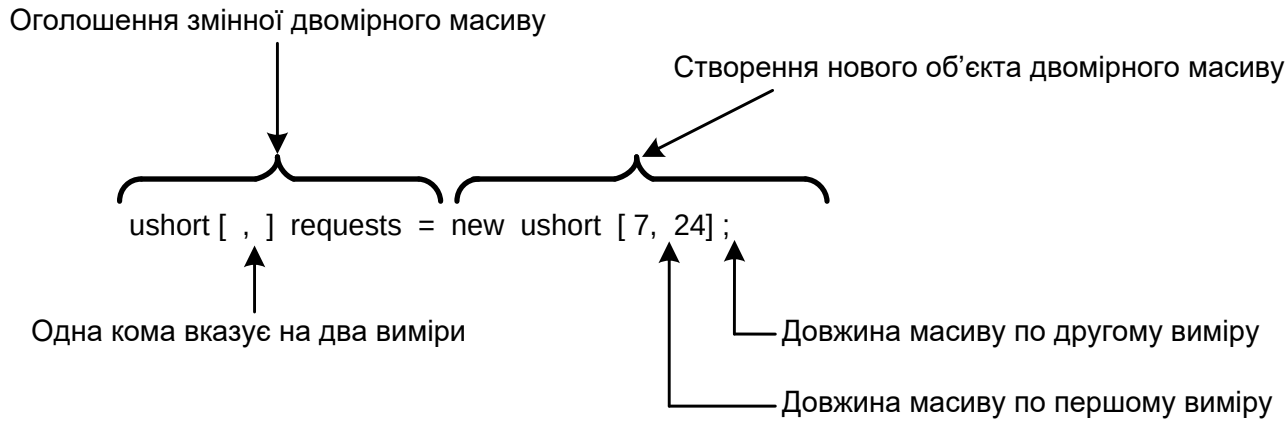
4.3. Багатомірні масиви

Оголошення й визначення двомірного масиву

Мова C# дозволяє визначати двомірні масиви, де для ідентифікації елемента потрібні два індекси. (Фактично в C# можна визначати масиви будь-якої розмірності).

На рис. 5.6 наведений приклад оголошення й визначення двомірного масиву.

Оголошення змінної масиву, створення нового об'єкта і присвоювання змінній посилання на об'єкт (перший рядок на рис. 5.6) можна (як і у випадку одномірного масиву) розбити на два оператори. Результат показаний в нижній частині рис. 5.6.



Попередній оператор, як і у випадку одномірних масивів, можно розбити на два рядки:

```
ushort [ , ]  requests ;
requests  =  new  ushort  [  7 , 24 ] ;
```

Рис. 4.6. **Приклад оголошення й визначення двомірного масиву**

Індекс першого виміру змінюється від 0 до 6, а другого – від 0 до 23.

Доступ до елементів двомірного масиву

Після оголошення змінної масиву і присвоювання їй посилання на двомірний об'єкт можна звертатися до його окремих елементів.

За винятком того, що для звертання до елементів двомірного масиву потрібен додатковий індекс, вони використовуються аналогічно елементам одномірного масиву. Отже, будь-який з них можна використовувати так само, як і окрему змінну базового типу. Наприклад:

requests[0,0] = (ushort) 89;

Тут застосовується операція приведення до типу (ushort), оскільки він є базовим типом requests. Воно необхідно, тому що літерал 89 належить до типу int, що не може бути неявно перетворений в ushort.

Представлення двомірного масиву як масиву масивів

Два виміри масиву requests можна розглядати як масив масивів. Якщо звернутися до першого виміру requests (що представляє, наприклад, дні тижня), сім днів можна вважати одномірним масивом (див. рис. 5.6). Якщо тепер включити в розгляд години, то кожний

елемент "день" можна вважати складеним з одномірного масиву "години".

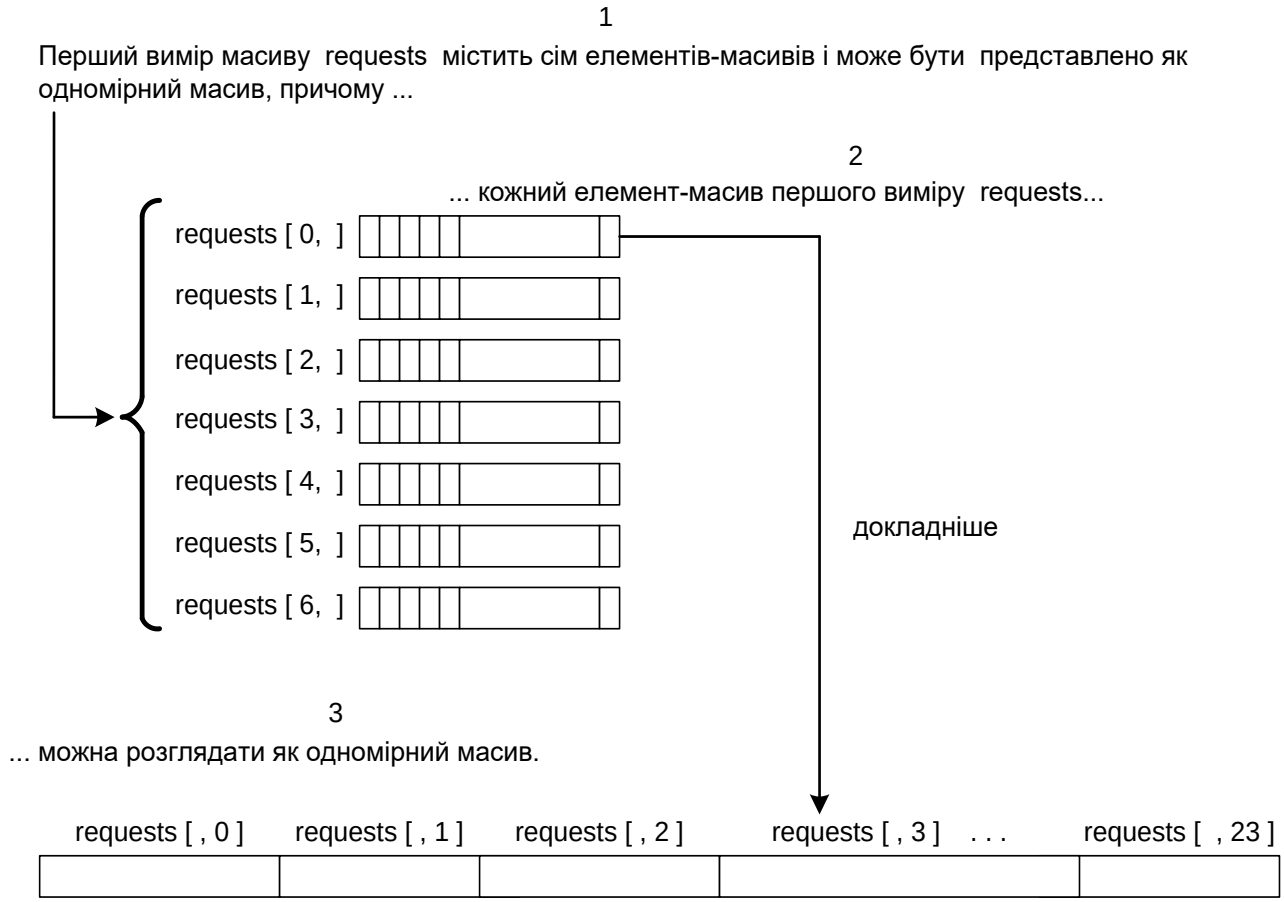


Рис. 4.7. **Двомірний масив можна представити як два одномірних масиви**

Приклад. Обробка матриці зарплати.

```
// Обробка матриці зарплати
using System;
class Class1
{
        static void Main()
        {
                int kol_rab;          // кіл-ть робітників в бригаді
                int kol_brigad;     // кіл-ть бригад
                char vibor;           // для організації діалогу
                do
                {
                        //       Введення матриці із клавіатури
```

```
                Console.WriteLine("Введіть кількість робітників в
бригаді...");
                kol_rab=Convert.ToInt32(Console.ReadLine());
                Console.WriteLine("Введіть кількість бригад...");
                kol_brigad=Convert.ToInt32(Console.ReadLine());

                double  [,] Matr = new double[kol_rab,kol_brigad];

                for( int i=0; i < kol_rab;  i++ )
                {
                        for (int j=0; j < kol_brigad;  j++)
                        {
                        Console.WriteLine("Введіть зарплату {0} робітника
з {1} бригади", i+1,j+1);
                        Console.WriteLine("Matr[{0},{1}]=?",i,j);
                        Matr[i,j]=Convert.ToDouble(Console.ReadLine());
                        }
                }
                // Контрольне виведення матриці
                Console.WriteLine("Матриця зарплат:");
                for( int i=0; i < kol_rab;  i++ )
                {
                        for (int j=0; j < kol_brigad;  j++)
                        {
                                Console.Write("{0}  ",Matr[i,j]);
                        }
                        Console.WriteLine();  // переведення рядка
                }
                // Обробка матриці
                Console.WriteLine("Розрахунок максимальної зарплати
по бригадах:\n");
                Console.WriteLine("Номер бригади     Номер робітника
Зарплата");
                int  n_rab = 0,       // номер робітника
                        n_brigad = 0;        // номер бригади
                double max_zarplata = 0;
                for( int j=0; j < kol_brigad;  j++ )
```

```
                {
                        for (int i=0; i < kol_rab ;  i++)
                        {
                                if(Matr[i,j]>max_zarplata)
                                {
                                        max_zarplata = Matr[i,j];
                                        n_rab = i;
                                        n_brigad = j;
                                }
                        }
                        Console.WriteLine("{0,7}{1,15}{2,15}",
n_brigad+1,n_rab+1,max_zarplata );
                        max_zarplata = 0;  // підготовка до наступної
ітерації
                }
                // Діалог для продовження роботи
                Console.WriteLine(" Будете продовжувати роботу? Так -
<Y>, Hi - <N> " );
                vibor=Convert.ToChar(Console.ReadLine());
        } while( (vibor == 'y') || (vibor == 'Y') );
        Console.WriteLine("Роботу завершено!" );
        }
}
```

Робота програми:

```
Введіть кількість робітників в бригаді...
4
Введіть кількість бригад...
2
Введіть зарплату 1 робітника з 1 бригади
Matr[0,0]=?
345,7
Введіть зарплату 1 робітника з 2 бригади
Matr[0,1]=?
654,1
Введіть зарплату 2 робітника з 1 бригади
Matr[1,0]=?
123,9
```

```
Введіть зарплату 2 робітника з 2 бригади
Matr[1,1]=?
765,1
Введіть зарплату 3 робітника з 1 бригади
Matr[2,0]=?
342,1
Введіть зарплату 3 робітника з 2 бригади
Matr[2,1]=?
765,3
Введіть зарплату 4 робітника з 1 бригади
Matr[3,0]=?
943,3
Введіть зарплату 4 робітника з 2 бригади
Matr[3,1]=?
342,6
Матриця зарплат:
345,7  654,1
123,9  765,1
342,1  765,3
943,3  342,6
Розрахунок максимальної зарплати по бригадах:

Номер бригади   Номер робітника   Зарплата
     1                  4                943,3
     2                  3                765,3
Будете продовжувати роботу? Так - <Y>, Ні - <N>
n
Роботу завершено!
Press any key to continue
```

Питання для самоконтролю

1. У чому полягають особливості призначення, оголошення й визначення масиву.

2. Як відтворюється доступ до окремих елементів масиву?

3. Наведіть приклад двох варіантів ініціалізації масиву.

4. Опишіть загальну схему перебору елементів масиву за допомогою оператора foreach.

5. Що таке розмір і ранг масиву?

6. Наведіть приклади алгоритмів пошуку заданих елементів масиву.

7. Наведіть приклади алгоритмів перетворення масиву.

8. У чому суть алгоритму сортування елементів масиву методом «Пузирку»?

9. Як реалізується доступ до елементів двомірного масиву?

10. У чому суть представлення двомірного масиву як масиву масивів?

11. Наведіть приклади обробки матриць.