Основні операції з масивами

Основними операціями з масивами є

- ініціювання початкових значень масиву, яке полягає у наданні кожному елементу масиву одного і того ж значення, яке відповідає базовому типу;
- введення-виведення значень масиву, яке полягає у організації такої структури циклів, коли значення усіх елементів масиву у визначеній послідовності будуть введені з клавіатури (з файлу) або виведені на екран (у файл).

Ініціювання початкових значень елементів масиву зручно використовувати у циклі або вкладених циклах. Розглянемо приклади ініціювання масивів, що описані вище.

```
{Ініціювання одномірних масивів}
{
    string[] Group = new string[5];
    for (int i=0; i<=4;i++) Group [i]="NoName";
}

{Ініціювання двомірних масивів}
{
    int[,] Box = new int[3,5];
    for (int i = 0; i < 3; i++)
      for (int j = 0; j < 5; j++) Box[i, j] = -1;
}
```

Введення-виведення значень масиву можна організовувати в різному порядку, але при цьому слід забезпечити, щоб значення, яке присвоюється, було присвоєно саме тому елементу, на який розраховує користувач. Тому зручно при виконанні цих дій виводити додаткову інформацію про індекси відповідних елементів. Найчастіше для введення-виведення масивів використовуються цикли. Розглянемо приклади введення-виведення масивів, описаних вище.

```
{Введення значень масивів. При введенні конкретних значень
 з клавіатури слід дотримуватись діапазону базового типу масиву}
{одномірні масиви}

string[] Group = new string[5];
for (int i = 0; i < 5; i++)
  {
    Console.Write("Введіть прізвище {0}-го студента: ",i+1);
    Group[i] = Console.ReadLine();
  }
    Console.ReadKey();

{двомірні масиви}

for (int i = 0; i < 3; i++)
  for (int j = 0; j < 5; j++)
    {
      Console.Write("element {0} ,{1} = ", i+1, j+1);
      Box[i, j] = Convert.ToInt32(Console.ReadLine());
    }
{Виведення значень масивів. При введенні значень елементів
 масивів рекомендується використовувати формати}

{Одномірні масиви}
for (int i = 0; i < 5; i++)
Console.WriteLine("{0}. {1}", i + 1, Group[i]);
```

```
{Двомірні масиви}
for (int i = 0; i < 3; i++)
{
   for (int j = 0; j < 5; j++) Console.Write("{0,5}",Box[i, j]);
   Console.WriteLine();
}
```

Пошук інформації у масивах

При вирішенні багатьох задач виникає потреба визначити, містить масив певну інформацію чи ні. Наприклад, чи є у матриці нульові елементи. Задачі такого типу називають пошуком у масиві. Для організації пошуку у масиві можна використовувати різні алгоритми. Найбільш простий, але не самий продуктивний - алгоритм простого (повного) перебору. Пошук здійснюється послідовним порівнянням до тих пір, поки не буде знайдений елемент, який задовольняє умові пошуку, або не будуть перевірені усі елементи масиву. Цей алгоритм застосовується коли елементи масиву не упорядковані.

Приклад 1. Пошук у числовому масиві

Завдання: Створити програму, яка в одномірному цілочисловому масиві буде відшукувати число, значення якого користувач уводить з клавіатури.

Рішення: У програмі визначимо масив цілих чисел. Змінна `i` буде параметром циклу, `n` - значення, яке вводиться з клавіатури і відшукується у масиві, `Cons` - логічна змінна, що визначає збіг числа `n` з елементом масиву `M`.

Пошук відбувається у циклі `do…while`, у тілі якого оператор `if` порівнює поточний елемент масиву із уведеним числом. Якщо такий збіг буде знайдено, логічній змінній `Cons` буде присвоєне логічне значення `true`, в іншому випадку буде збільшено значення параметру циклу і повторена перевірка умови із новим елементом.

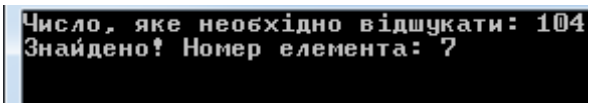
Цикл завершується, якщо у масиві знайдено елемент, який дорівнює зразку, або у випадку, коли перебрані усі елементи масиву. Після завершення циклу змінна `Cons` визначає було знайдено збіг чи ні, а у випадку позитивної відповіді змінна `i` зберігатиме номер позиції у якій знайдено збіг. Зверніть також увагу на те, що програма відшукає лише першу появу числа у масиві а не усі позиції, як це могло бути із числом 27.

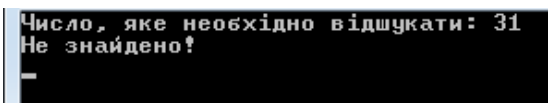
Лістинг

```
using System;
class Program
    {
        static void Main()
        {
            int[] M = new int[] { 7, 12, 142, -8, 94, 55, 104, 5, 10, 27 };
            Console.Write("Число, яке необхідно відшукати: ");
            int N = Convert.ToInt32(Console.ReadLine());
            int i=0; bool Cons = false;
            do
              if (M[i] == N) Cons = true; else i++;
            while (!Cons && i<10);
            if (Cons) Console.WriteLine("Знайдено! Номер елемента: {0}",i+1);
else Console.WriteLine("Не знайдено!");
            Console.ReadKey();
```

```
        }
    }
```

Результати виконання програми:





Приклад 2. Підрахунок у числовому масиві

Завдання: Створити програму, яка у двомірному масиві цілих чисел від -99 до 99 буде підраховувати кількість нульових елементів, кількість одноразрядних і дворозрядних чисел.

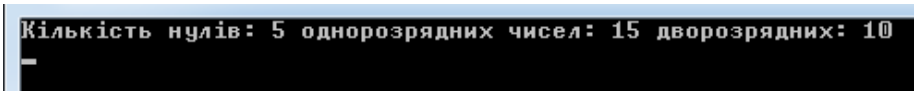
Рішення: Задача підрахунку кількості елементів, значення яких відповідають певній умові є різновидом задачі пошуку. У такому випадку слід створити спеціальні змінні, у яких буде накопичуватись кількість таких елементів.

У масиві `M` зберігаються числа. Змінні `Zero, OneDigit, TwoDigit` будуть відповідно накопичувати кількість нульових одно- і дворозрядних чисел. Присвоївши їм на початку роботи програми нульові значення, увійдемо у цикли повного перебору всіх значень масиву `M`. Приналежність числа до нульового, одно- або дворозрядного визначається операторами `if`, у яких поступово збільшуються лічильники кількості чисел. Після завершення циклу залишається лише вивести отримані значення на екран.

Текст програми:

```
using System;
    class Program
    {
        static void Main()
        {
            int[,] M = new int[,] {{11,1,24,1,32,0},
                                   {-11,2,0,-5,3,1},
                                   {33,55,21,1,3,1},
                                   {0,0,0,2,3,4},
                                   {44,42,3,21,1,-1}};
            int zero = 0, oneDigit = 0, twoDigits = 0;
            for (int i = 0; i < 5; i++)
                for (int j = 0; j < 6; j++)
                   if (M[i, j] == 0) zero++;
                   else
                     if (Math.Abs(M[i, j]) < 9) oneDigit++; else twoDigits++;
            Console.WriteLine("Кількість нулів: {0} одноразрядних чисел: {1}
дворозрядних: {2}",zero,oneDigit, twoDigits);
            Console.ReadKey();
        }
    }
```

Результати виконання програми:



Приклад 3. Пошук максимального (мінімального) елемента масиву

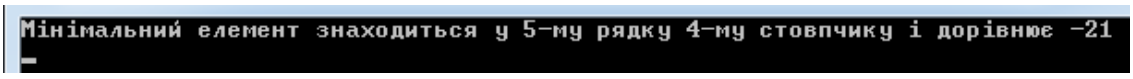
Завдання: Створити програму, яка у двомірному масиві цілих чисел від -99 до 99 буде знаходити мінімальний елемент. На екран вивести значення цього елемента і його координати у матриці.

Рішення: Алгоритм пошуку мінімально або максимального елемента масиву є доволі очевидним: слід припустити, що перший елемент масиву є мінімальним після чого порівнювати його із рештою елементів, випадку знаходження ще меншого елемента присвоювати його значення мінімуму.

Текст програми:

```
using System;
namespace ConsoleApplication3
{
    class Program
    {
        static void Main()
        {
            int[,] M = new int[,] {{11,1,24,1,32,0},
                                   {-11,2,0,-5,3,1},
                                   {33,55,21,1,3,1},
                                   {0,0,0,2,3,4},
                                   {44,42,3,-21,1,-1}};
            int iMin = 0, jMin = 0, min = M[iMin, jMin];
            for (int i=0; i<5; i++)
                for (int j=0; j<6; j++)
                    if (M[i, j] < min)
                    {
                        iMin = i;
                        jMin = j;
                        min = M[i, j];
                    }
            Console.WriteLine("Мінімальний елемент знаходиться у {0}-му рядку
{1}-му стовпчику і дорівнює {2}",iMin+1, jMin+1,min);
            Console.ReadKey();
        }
    }
}
```

Результати виконання програми:



Сортування інформації у масивах

Задачі сортування масивів розповсюджені в інформаційних системах и використовуються як попередній етап задач пошуку, оскільки пошук в упорядкованому масиві відбувається набагато швидше, аніж у неупорядкованому. Під сортуванням масиву розуміється процес перестановки елементів масиву, метою якого є розміщення елементів масиву у певному порядку, тоді:

```
a[1]<=a[2]<=...<=a[N]  - збільшення елементів
a[1]>=a[2]>=...>=a[N]  - зменшення елементів
```

Існує декілька методів сортування масивів. Розглянемо два з них: метод повного перебору і метод прямого обміну.

Приклад 4. Сортування методом прямого перебору

Завдання: Є масив випадкових цілих чисел. Відсортувати масив у порядку збільшення, використовуючи метод прямого перебору.

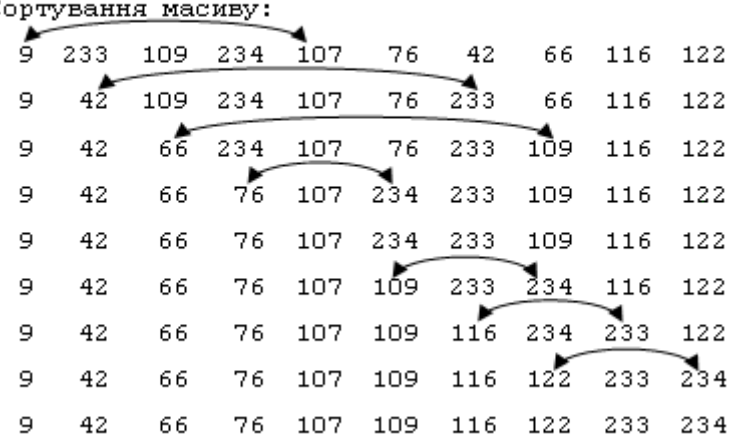
Рішення: Алгоритм сортування масиву у порядку збільшення методом прямого перебору може бути представлений у такій послідовності дій. Спочатку, переглядаючи масив від першого до останнього елемента, знайти мінімальний елемент, поставити такий елемент на перше місце, а перший - на місце мінімального. Далі переглянути масив від другого елемента до останнього, у цій групі знайти найменший елемент, поставити цей елемент на друге місце а другий - на місце знайденого. Таку послідовність дій слід виконувати до передостаннього елемента.
Зазначимо, що час роботи алгоритму не залежить від початкового стану масиву. Працює алгоритм довго, навіть якщо для остаточного сортування масиву слід виконати одну перестановку, все рівно буде виконано `N-1` циклів.

Текст програми:

```
namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int N = 9;
            Random a = new Random();
            int[] M = new int[N];
            for (int i = 0; i < N; i++)
            {
                M[i] = a.Next(100);
                Console.Write(M[i] + " ");
            }
            Console.WriteLine("Сортування");
            int min, t;
            for (int i = 0; i < N - 1; i++)
            {
                min = i;
                for (int j = i + 1; j < N; j++) if (M[j] < M[min]) min = j;
                t = M[min];
                M[min] = M[i];
                M[i] = t;
                for (int j = 0; j < N; j++) Console.Write(M[j]+ " ");
                Console.WriteLine();

            }
                Console.ReadKey();
        }
    }
}
```

Результат виконання програми:

```
Початковий масив:

107  233  109  234    9   76   42   66  116  122

Сортування масиву:
```



```
  9  233  109  234  107   76   42   66  116  122
  9   42  109  234  107   76  233   66  116  122
  9   42   66  234  107   76  233  109  116  122
  9   42   66   76  107  234  233  109  116  122
  9   42   66   76  107  234  233  109  116  122
  9   42   66   76  107  109  233  234  116  122
  9   42   66   76  107  109  116  234  233  122
  9   42   66   76  107  109  116  122  233  234
  9   42   66   76  107  109  116  122  233  234

Масив відсортований!
```

Приклад 5. Сортування методом прямого обміну ("бульбочковий" метод)

Завдання: Є масив випадкових цілих чисел. Відсортувати масив у порядку зменшення, використовуючи метод прямого обміну.

Рішення: В основі алгоритму лежить обмін сусідніх елементів масиву. При сортуванні у порядку зменшення обмін буде відбуватись за умови, що наступний елемент є більшим за поточний. Таким чином менші ("легші") числа начебто спливають у кінець масиву. Такий процес нагадує процес підняття бульбашок у рідині, тому такий алгоритм ще називають "бульбочковим". Тривалість алгоритму залежить від початкового стану масиву, чим далі будуть "легкі" елементи від кінця і чим невпорядкованішим буде масив, тим довшою буде робота з упорядкування.

У програму введена логічна змінна changed, яка перед виконанням циклу отримує значення false. Процес сортування завершується, якщо після чергового циклу жоден з елементів не може бути обмінений місцями із сусіднім.

Текст програми:

```
using System;

namespace ConsoleApplication2
{
    class Program
    {
        static void Main(string[] args)
        {
            int N = 9;
            Random a = new Random();
            int[] M = new int[N];
            for (int i = 0; i < N; i++)
            {
                M[i] = a.Next(100);
                Console.Write(M[i] + " ");
            }
            bool changed; int t;
            Console.WriteLine();
            Console.WriteLine("Сортування");
            do
```

```
            {
                changed = false;
                for (int i=0; i<N-1; i++)
                    if (M[i] < M[i+1])
                    {
                    t = M[i];
                    M[i] = M[i+1];
                    M[i+1]=t;
                    changed=true;
                    }
                for (int i=0; i<N; i++) Console.Write (M[i]+" ");
                Console.WriteLine();
            }
            while (changed);
            Console.ReadKey();
            Console.WriteLine("Масив відсортований!");
        }

    }
}
```

Результат виконання програми:

```
Початковий масив:

174  225   78   99   16  128  224   68  171   42

Сортування масиву:
225  174   99   78  128  224   68  171   42   16
225  174   99  128  224   78  171   68   42   16
225  174  128  224   99  171   78   68   42   16
225  174  224  128  171   99   78   68   42   16
225  224  174  171  128   99   78   68   42   16
225  224  174  171  128   99   78   68   42   16

Масив відсортований!
```

Приклад 6. Бінарний пошук

Завдання: Є масив цілих чисел, відсортований у порядку збільшення. З клавіатури вводиться ціле число. Перевірити, чи є уведене число у масиві.

Рішення: Для пошуку інформації в упорядкованому масиві застосовують інші, більш ефективні у порівнянні з методом прямого перебору алгоритми, одним із яких є **метод бінарного пошуку**.

В упорядкованому масиві мінімальним (`Min`) елементом буде перший елемент у масиві, а максимальним (`Max`) - останній. На першому етапі слід порівняти зразок (`Find`) із середнім за номером (`Sered`) елементом масиву (див. рис. 14.1, а). Якщо трапилось, що зразок дорівнює середньому елементу, тоді задача вирішена. Якщо зразок є більшим за середній елемент, це означає, що відшукуваний елемент розташований нижче. У такому випадку мінімальним елементом, від якого слід шукати число буде елемент з номером `Sered+1`, а максимальним залишиться те саме значення (див. рис. 14.1, б).

Якщо зразок є меншим за середній елемент, це означає, що відшукуваний елемент розташований вище. У такому випадку максимальним елементом, до якого слід шукати елемент буде елемент з номером `Sered-1`, а мінімальним залишиться те саме значення (див. рис. 14.1, в).

Після визначення частини (верхня або нижня) масиву, у якій слід продовжувати пошук, значення `Sered` повинно лежати на середині відстані між `Min` і `Max`, тобто, `Sered =(Max-Min)/2+Min`.
Алгоритм бінарного пошуку, блок-схема якого представлена на рис. 14.2, закінчує свою роботу, якщо відшукуваний елемент знайдено, або у випадку, коли перед виконанням чергового циклу отримуємо `Min>Max`.

Рис. 14.1. Вибір чергового елемента при бінарному пошуку

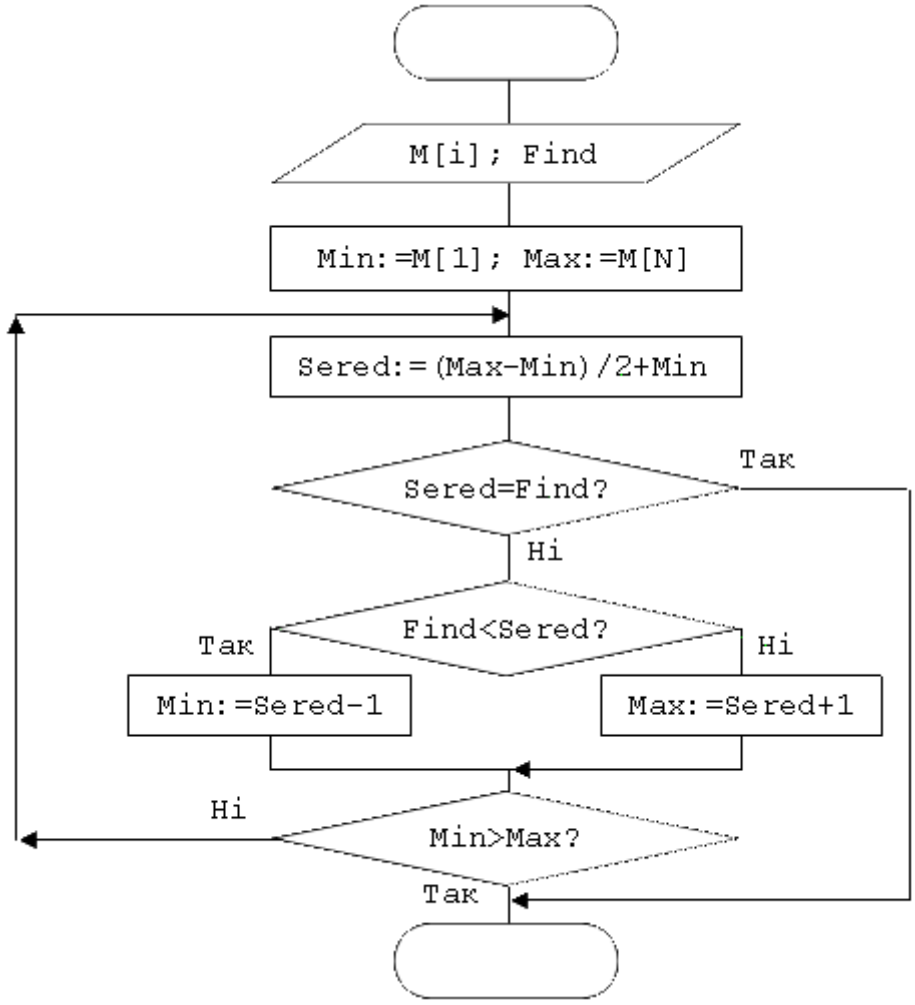


Рис. 14.2. Алгоритм бінарного пошуку у масиві, упорядкованому за збільшенням

Текст програми:

```
using System;
    class Program
    {
        static void Main(string[] args)
        {
         int[] M = new int[] { -8, -5, -1, 0, 1, 22, 55, 145, 200, 205,300 };
         Console.Write("Число, яке необхідно відшукати: ");
         int find = Convert.ToInt32(Console.ReadLine());
         int min = 0, max = 10;
         int Sered;
         do
            {
                Sered = (max-min)/2+min;
                if (find == M[Sered]) { Console.WriteLine("Значення {0}
знайдено під номером {1}", find, Sered+1); break; }
                else if (find < M[Sered]) max = Sered-1; else min=Sered+1;
            }
            while (min<=max);
            if (min > max) Console.WriteLine("Не знайдено!");
            Console.ReadKey();
        }
    }
```

Результати виконання програми: