2 ЦИКЛИ. МАСИВИ. МЕТОДИ. СТРУКТУРИ

2.1 Мета роботи

Навчитися використовувати умовні конструкції, масиви і цикли при створенні програмних засобів, які вирішують найпростіші задачі; навчитися використовувати тип даних enum, створювати свої методи і структури засобами мови C#.

2.2 Організація самостійної роботи студентів

Під час підготовки до виконання лабораторної роботи необхідно вивчити роботу з циклами, масивами, створення та використання методів, створення структур.

Масиви

Масив являє собою набір однотипних змінних. Створення масиву схоже с оголошенням змінної: *тип_даних*[] *назва масиву*;. Наприклад:

```
int[] nums = new int[4];
nums[0] = 1;
nums[1] = 2;
nums[2] = 3;
nums[3] = 5;
Console.WriteLine(nums[3]);
```

Спочатку ми створили масив *nums*, який буде зберігати дані типу *int*. Далі використовуючи операцію *new*, ми виділили пам'ять для 4 елементів масиву: *new int*[4]. Число 4 ще називається довжиною масиву.

Відлік елементів масиву починається з 0, тому в даному випадку, щоб звернутися к четвертому елементу в масиві треба використовувати вираз *nums*[3].

Існують також альтернативні шляхи ініціалізації масивів:

```
int[] nums2 = new int[] { 1, 2, 3, 5 };
int[] nums3 = { 1, 2, 3, 5 };
```

В цьому випадку ми одразу вказуємо всі елементи масиву.

Масиви бувають одномірними і багатовимірними. В прикладах вище створювалися одномірні масиви. Приклад створення багатовимірного масиву:

```
int[] nums1 = new int[] { 0, 1, 2, 3, 4, 5 };
int[,] nums2 = { { 0, 1, 2 }, { 3, 4, 5 } };
```

Від багатовимірних масивів слід відрізняти масив масивів або так званий «зубчатий масив»:

```
int[][] nums = new int[3][];
nums[0] = new int[2];
nums[1] = new int[3];
nums[2] = new int[5];
```

Дві групи квадратних дужок вказують, що це масив масивів, тобто такий масив, який в свою чергу містить в собі інші масиви. Причому розмірність кожного з цих масивів може не співпадати.

Деякі методи і властивості масивів:

– властивість *Length*: дозволяє отримати кількість елементів масиву;

– властивість *Rank*: дозволяє отримати розмірність масиву;

– метод Array.Reverse: змінює порядок елементів масиву на зворотній;

– метод Array.Sort: сортирує елементи масиву.

Умовні конструкції

Умовні конструкції – один з базових компонентів багатьох мов програмування, які направляють роботу програми за одним із шляхів в залежності від певних умов.

В мові C# використовуються наступні умовні конструкції: *if..else* і *switch..case*.

Конструкція *if/else* перевіряє істинність деякої умови і в залежності від результатів перевірки виконує деякий код:

```
int num1 = 8;
int num2 = 6;
if (num1 > num2)
{
   Console.WriteLine("Число {0} більше числа {1}", num1, num2);
}
else { Console.WriteLine("Число {0} менше числа {1}", num1, num2);}
```

Після ключового слова *if* ставиться умова. І якщо ця умова виконується, то спрацьовує код, який поміщено в блоці *if* після фігурних дужок. В якості умов виступають операції порівняння.

В даному випадку перше число більше другого, тому вираз *num1>num2* істинний і повертає *true*, тому управління переходить до строки *Console.WriteLine("Число {0} більше числа {1}", num1, num2);*.

Конструкція *switch/case* дозволяє обробити відразу декілька умов:

```
Console.WriteLine("Нажмите Y или N");
string selection = Console.ReadLine();
switch (selection)
{
   case "Y":
      Console.WriteLine("Ви натиснули букву Y");
      break;
   case "N":
      Console.WriteLine("Ви натиснули букву N");
      break;
   default:
      Console.WriteLine("Ви натиснули невідому букву ");
      break;
}
```

Після ключового слова *switch* в дужках вказується змінна для порівняння. Значення цієї змінної послідовно порівнюється зі значеннями, які розміщені після оператора *case*. Якщо збіг буде знайдено, то буде виконуватися відповідний блок *case*. В кінці блоку *case* ставиться оператор *break*, щоб запобігти виконанню інших блоків. Щоб опрацювати ситуацію, коли збіг не буде знайдено, треба додати блок *default*.

Цикли

Цикли також є управляючими конструкціями, які дозволяють в залежності від певних умов виконувати деякі дії декілька разів. В C# існують наступні види циклів: *for*, *foreach*, *while*, *do…while*.

Цикл *for* має наступне формальне визначення:

for ([ініціалізація лічильника];[умова];[зміна лічильника])
{
    //операції
}

Розглянемо стандартний цикл *for*:

```
for (int i = 0; i < 9; i++)
{
   Console.WriteLine("Квадрат числа {0} равен {1}", i, i * i);
}
```

Перша частина створення циклу – *int i = 0* створює і ініціює лічильник *i*. Лічильник необов'язково повинен бути типу *int*. Перед виконанням циклу його значення буде дорівнювати 0. Друга частина – умова, при якій буде виконуватися цикл. В даному випадку цикл буде виконуватися до тих пір, поки *i* не досягне 9. Третя частина – збільшення лічильника на 1. В даному прикладі цикл виконається 9 разів.

Цикл *foreach* призначений для перебору елементів в контейнерах. Формальне оголошення циклу:

foreach (тип_даних назва_змінної in контейнер)
{
   // операції
}

Наприклад:

```
int[] array = new int[] { 1, 2, 3, 4, 5 };
foreach (int i in array)
{
   Console.WriteLine(i);
}
```

Цикл *for* більш гнучкий, ніж *foreach*. Якщо *foreach* послідовно бере елементи контейнера і тільки для зчитування, то в циклі *for* можна перескочити на декілька елементів вперед в залежності від зміни лічильника, а також змінювати елементи.

В циклі *do/while* спочатку виконується код циклу, а потім виконується перевірка умови в інструкції *while*. До тих пір поки умова істинна цикл повторюється. Наприклад:

```
int i = 6;
do
{
   Console.WriteLine(i);
   i--;
}
while (i > 0);
```

В цьому прикладі цикл виконається 6 разів, поки *i* не дорівнює нулю. Цикл *do/while* гарантує хоча б одне виконання дій, навіть якщо умова в інструкції *while* не є істиною.

На відміну від циклу *do/while* цикл *while* спочатку перевіряє істинність деякої умови, і якщо умова є істиною, то код циклу виконується:

```
int i = 6;
while (i > 0)
{
   Console.WriteLine(i);
   i--;
}
```

Методи (функції)

Методи представляють собою набір операторів, які виконують певні дії. Загальне оголошення методу виглядає наступним чином:

```
[модифікатори] тип_поверненого_значення назва_методу ([параметри])
{
   // тіло методу
}
```

Модифікатори і параметри не є обов'язковими. Розглянемо на прикладі методу *Main*:

```
static void Main(string[] args)
{
   Console.WriteLine("Hello world!");
}
```

Ключове слово *static* є модифікатором. Далі йде тип поверненого значення. В даному випадку ключове слово *void* вказує на те, що метод нічого не повертає. Такий метод ще називають процедурою. Потім вказується назва методу «*Main*» і в дужках параметри – *string[] args*. В фігурних дужках заключено тіло методу – всі дії, які він виконує.

В функції в якості типу поверненого значення замість *void* використовується будь-який інший тип даних. Вони також обов'язково повинні містити оператор *return*, після якого ставиться значення, що повертається. Тип значення, що стоїть після *return* повинно співпадати з типом вказаним в прототипі функції.

Створивши методи, можна використовувати їх в програмі. Щоб викликати метод, потрібно вказати його ім'я, а після нього в дужках значення для параметрів.

```
static void Main(string[] args)
{
   string message = Hello(); // виклик першого методу
   Console.WriteLine(message);
   Sum(); // виклик другого методу
   Console.ReadLine();
}
static string Hello()
{
   return "Hello World!"; }
```

```
static void Sum()
{
   int x = 2;
   int y = 3;
   Console.WriteLine("{0} + {1} = {2}", x, y, x+y);
}
```

Перший метод *Hello* повертає значення типу *string*. Тому ми можемо присвоїти значення деякій змінній типу *string*. Другий метод (процедура *Sum)* виконує додавання двох чисел і виводить результат на екран.

Передача параметрів в метод

В мові C# існує два способи передачі параметрів в метод: за значенням і за посиланням. Найбільш простий спосіб передачі параметрів є передача за значенням:

```
static int Sum(int x, int y)
{
   return x + y;
}
```

При викликанні цього методу в програмі обов'язково потрібно передати на місце параметрів значення, які відповідають типу параметрів:

```
int z = Sum(x, 15);
```

Параметри можуть бути також вихідними. Щоб зробити параметр вихідним, треба перед ним поставити модифікатор *out*.

```
static void Sum(int x, int y, out int a)
{
   a = x + y;
}
```

В цьому прикладі результат повертається не через оператор *return*, а через вихідний параметр. Виклик методу в програмі:

```
Sum(x, 15, out z);
```

Слід звернути увагу, що модифікатор *out* вказується, як при створенні методу, так і при його викликанні в методі *Main*. Також методи, які використовують вихідні параметри обов'язково повинні присвоїти їм певне значення.

При передачі параметрів за посиланням використовується модифікатор *ref*:

```
Addition(ref x, y); // виклик методу
```

Як і в випадку з *out* ключове слово *ref* використовується як при створенні методу, так і при його викликанні. При передачі параметрів за значенням метод отримує не саму змінну, а її копію. При передачі параметру за посиланням метод отримує адресу змінної в пам'яті. Якщо метод змінює значення параметра, який передається за посиланням, то також змінюється і значення змінної, яка передається на його місце.

C# дозволяє використовувати необов'язкові параметри. Для таких параметрів необхідно задати значення за замовчуванням. Також слід враховувати, що після необов'язкових параметрів всі наступні параметри також повинні бути необов'язковими.

```
static int OptionalParam(int x, int y, int z=5, int s=4)
{
```

```
    return x + y + z + s;
}
```

Так як останні два параметри оголошені як необов'язкові, то можна опустити один з них, або обидва.

```
static void Main(string[] args)
{
    int rez = OptionalParam(2, 3);
    rez = OptionalParam(2,3,10);
}
```

Використовуючи ключове слово *params*, можна передати невизначену кількість параметрів.

```
static void Addition(params int[] integers)
{
    int result = 0;
    for (int i = 0; i < integers.Length; i++)
    {
        result += integers[i];
    }
    Console.WriteLine(result);
}
static void Main(string[] args)
{
    Addition(1, 2, 3, 4, 5);
    int[] array = new int[] { 1, 2, 3, 4 };
    Addition(array);
    Addition();
    Console.ReadLine();
}
```

Причому на місце параметру з модифікатором *params* можна передавати як окреме значення, так і масив значень, або зовсім не передавати параметри. Після параметру з модифікатором *params* не можна вказувати інші параметри.

Перерахування enum

Крім примітивних типів даних в C# є такий тип даних як *enum* або перерахування. Перерахування являють собою набір констант, зв'язаних логікою. Створення перерахування відбувається за допомогою оператора *enum*. Після вказується назва перерахування і тип перерахування (обов'язково цілочисельний). Якщо тип явно не вказаний, то за замовчуванням використовується тип *int*. Далі вказується список елементів перерахування через кому.

```
enum Days
{
    Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday
}
enum Time : byte
{
    Morning, Afternoon, Evening, Night
}
```

В даних перерахуваннях кожному елементу присвоюється ціле значення, причому перший елемент має значення 0.

Структури

Крім базових елементарних типів і перерахувань в C# є складений тип даних, який називається структурою. Структури можуть вміщувати в собі звичайні змінні і методи.

Для прикладу створимо структуру *Book*, в якій будуть зберігатися змінні для назви, автора і року публікації книги. Крім того, структура буде містити метод для виведення інформації про книгу на екран.

```
struct Book
{
   public string name;
   public string author;
   public int year;

   public void Info()
   {
      Console.WriteLine("Книга ‘{0}’ (автор {1}) була видана в {2} році", name, author, year);
   }
}
```

Крім звичайних методів структура може мати спеціальний метод – конструктор, який виконує деяку початкову ініціалізацію об’єкту, наприклад, присвоює всім полям деякі значення за замовчуванням. Якщо не використовувати конструктор, то спочатку потрібно буде проініціювати всі поля структури:

```
book1.name = "Война и мир";
book1.author = "Л. Н. Толстой";
book1.year = 1869;
```

Виклик конструктора за замовчуванням дозволяє автоматично проініціювати всі поля структури значеннями за замовчуванням.

```
Book book2=new Book(); // використання конструктора
```

Виклик конструктора має наступний синтаксис: *new* назва_структури ([список_параметрів]). Конструктор є звичайним методом, тільки не має значення, що повертається, і його назва завжди співпадає з іменем структури.

```
Book book=new Book("Война и мир", "Л. Н. Толстой", 1869);
book.Info();
```

Тепер не потрібно вручну присвоювати полям структури – їх ініціалізацію виконав конструктор.

2.3 Порядок виконання роботи

1. Ознайомитися з теоретичним матеріалом.
2. Виконати індивідуально завдання з пункту 2.4.
3. Оформити звіт.
4. Здати практичну частину.

2.4 Індивідуальні завдання

Завдання №1

Дано три цілих числа. Знайти кількість додатних і кількість від'ємних чисел.

Завдання №2

Дано три змінні дійсного типу: *A*, *B*, *C*. Якщо їх значення впорядковані за зростанням (*A*<*B*<*C*) або за спаданням (*A*>*B*>*C*), то збільшити їх в два рази. Інакше замініть значення кожної змінної на протилежне. Виведіть нові значення змінних на екран.

Завдання №3

Дано ціле число *N* (*N*>0). Створіть і виведіть масив розміром *N*, який вміщує N перших додатних непарних чисел: 1, 3, 5,…

Завдання №4

Створіть двомірний масив 5x5, заповніть будь-яким способом. Виведіть його елементи, які розміщені в стовбцях з непарними номерами (1, 3, 5).

Завдання №5

Дано два цілих числа *A* і *B* (*A*<*B*). Знайдіть суму квадратів всіх цілих чисел від *A* до *B* включно.

Завдання №6

Дано цілі додатні числа *A* і *B* (*A*<*B*). Виведіть всі цілі числа від *A* до *B* включно; при цьому кожне число повинно виводитись таку кількість разів, що дорівнює значенню числа (наприклад, число 3 виводиться 3 рази).

Завдання №7

Дано ціле число *N* (*N*>0). Якщо воно є степенем числа 3, то виведіть TRUE, інакше виведіть FALSE.

Завдання №8

Спортсмен-лижник почав тренування, пробігши в перший день 10 км. Кожен наступний день він збільшував довжину пробігу на *P* процентів від пробігу попереднього дня (*P* – дійсне, 0<*P*<50). За даним *P* визначте, після якого дня сумарний пробіг лижника за всі дні перевищить 200 км, і виведіть найдену кількість днів *K* (ціле) і сумарний пробіг *S* (дійсне).

Завдання №9

Напишіть метод, який запитує ввести два числа і потім повертає суму цих чисел.

Завдання №10

Напишіть метод *InvertDigits*(*K*), який змінює порядок слідування цифр цілого додатного числа *K* на зворотній (*K* – параметр цілого типу).

Завдання №11

Напишіть метод *TrianglePS*(*a*, *P*, S*),* який розраховує за стороною *a* рівностороннього трикутника його периметр *P* = 3 і площу *S* = *a***a**(√3)/4 (*a* – вхідний параметр, *P* і *S* – вихідні параметри; всі параметри являються дійсними).

Завдання №12

Напишіть метод *MinMax*(*X*, *Y*), який записує в змінну *X* мінімальне зі значень *X* і *Y*, а в змінну *Y* – максимальне із цих значень (*X* і *Y* – дійсні параметри, які одночасно є вхідними і вихідними).

Завдання №13

Напишіть метод, який приймає будь-яку кількість параметрів цілого типу і повертає їх суму.

Завдання №14

Напишіть рекурсивний метод *DigitSum*(*K*) цілого типу, який знаходить суму цифр цілого числа *K*, не користуючись операторами циклу.

Завдання №15

Напишіть перерахування з арифметичними операціями: *add*, *sub*, *mul*, *div*. Напишіть метод з трьома параметрами:

1 параметр – дійсне число;

2 параметр – дійсне число;

3 параметр – змінна типу перерахування, яке було створено раніше.

В залежності від значення третього параметра, метод виконує конкретну операцію над першим і другим параметром. Після цього повертає результат.

Завдання №16

Створіть структуру, яка зберігає інформацію про товари (назва товару, дата надходження, маса, ціна, назва постачальника, максимальний термін збереження) деякого складу. Структура повинна бути описана в окремому файлі проекту.

Створіть масив з трьох структур, заповніть першу структуру за допомогою звернення до змінних, другу і третю структури заповніть за допомогою конструктора. Після цього виведіть всю інформацію на екран.

2.5 Зміст звіту

Звіт має містити:

– мету роботи;

– індивідуально виконані завдання;

– код програми;
– результат виконання програми;
– висновки.

2.6 Контрольні питання та завдання

1. Що таке умовна конструкція? Які види умовних конструкцій існують?
2. Що таке цикл? Описати усі види циклів, які існують в мові C#?
3. Як створити одномірний і двомірний масиви? Чим відрізняється синтаксис оголошення масиву масивів від оголошення багатовимірного масиву?
4. Опишіть загальний синтаксис оголошення масивів?
5. В чому суть модифікатора *out*?
6. Для чого використовується модифікатор *ref*?
7. Що таке необов'язкові параметри?
8. Для чого використовують ключове слово *params*?
9. Що таке перерахування і для чого вони використовуються?
10. Що таке структура? В яких випадках її використовують?
11. Якими способами можна ініціювати структуру?
12. Чи може структура мати методи?