МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

**Звіт**

про виконання лабораторної роботи № 5 на тему:

«Структури в С#»

**Виконав:**

студент групи ФеП-12

Козачок А.А.

**Перевірив:**

Щербак С.С.

**Львів 2020**

**Мета роботи:** розглянути структури, вивчити поняття перерахування, навчитися їх застосовувати.

**Обладнання:** ноутбук, інтегроване середовище розробки програмного забезпечення Microsoft Visual Studio (2019).

**Теоретичні відомості** Структура - простіша версія класів.

Всі структури успадковуються від базового класу System.ValueType і є типами значень, тоді як класи - посилальні типи. Структури відрізняються від класів наступними речами:

* Структура не може мати конструктора без параметрів (конструктора за замовчуванням);
* Поля структури не можна ініціалізувати, крім випадків, коли поля статичні. private int x = 0; // в структурі неприпустимо;
* Примірники структури можна створювати без ключового слова new; - Структури не можуть успадковуватися від інших структур або класів. Класи не можуть успадковуватися від структур. Структури можуть реалізовувати інтерфейси;
* Так як структури це типи значень, вони мають всі властивості подібних типів

(передача в метод за значенням і т.д.), на відміну від посилальних типів; - Структура може бути nullable типом.

Структури оголошуються за допомогою ключового слова struct:

public struct Book

{ public string Name; public string Year; public string Author;

}

Примірник структури можна створювати без ключового слова new:

static void Main(string[] args)

{

Book b;

b. Name = "BookName";

}

Структури підходять для створення нескладних типів, таких як точка, колір, коло. Якщо необхідно створити безліч екземплярів подібного типу, використовуючи структури, ми економимо пам'ять, яка могла б виділятися під посилання у випадку з класами.

Прикладами структур в стандартній бібліотеці класів .Net є такі типи як int, float, double, bool та інші. Також DateTime, Point (точка), Color.

Перерахування (Enumeration) - це визначений користувачем цілочисельний тип, який дозволяє уточняти набір допустимих значень, і призначити кожному зрозуміле ім'я. Для оголошення перерахування використовується ключове слово enum. Загальна структура оголошення перерахування виглядає так:

enum [ім'я\_перерахування] { [ім'я1], [ім'я2], … };

Наприклад, перерахування Directions, яке буде відповідати напрямам руху:

enum Directions { Left, Right, Forward, Back };

Оголосивши таким чином перерахування, кожній символічно позначеній константі присвоюється цілочисельне значення, починаючи з 0 (Left = 0, Right = 1 ...). Це цілочисельне значення можна задавати і самому:

enum Directions { Left, Right = 5, Forward = 10, Back };

Back в цьому прикладі буде мати значення 11.

Приклад програми з використанням перерахування:

enum Directions { Left, Right, Forward, Back }; // оголошення перерахування class Program

{ public static void GoTo(Directions direction)

{ switch (direction)

{

case Directions.Back:

Console.WriteLine("Go back"); break; case Directions.Forward:

Console.WriteLine("Go forward"); break; case Directions.Left:

Console.WriteLine("Turn left"); break; case Directions.Right:

Console.WriteLine("Turn right "); break;

}

}

static void Main(string[] args)

{

Directions direction = Directions.Forward;

GoTo(direction); // "Go forward"

Console.ReadKey();

}

}

Щоб отримати ціле значення певного елемента перерахування, досить цей елемент явно привести до цілого типу:

enum Directions : byte { Left, Right, Forward, Back }; class Program

{ static void Main(string[] args)

{

Console.WriteLine((int)Directions.Forward); // 2

Console.ReadKey();

}

}

За умовчанням як цілого типу для enum використовується int. Цей тип можна змінити на будь-який інший цілий тип (крім char), вказавши після імені перерахування необхідний тип і розділивши двокрапкою:

enum Directions : byte { Left, Right, Forward, Back };

Головні переваги, які дають перерахування це:

* Гарантія того, що змінним будуть призначатися допустимі значення із заданого набору;
* Дозволяє заощадити деякий час, і нагадує, які значення можна використовувати;
* Код стає читабельнішим, коли в ньому присутні зрозумілі імена, а не числа.

Перерахування дуже широко використовуються в самій бібліотеці класів .NET. Наприклад, при створенні файлового потоку (FileStream) використовується перерахування FileAccess, за допомогою якого ми вказуємо з яким режимом доступу відкрити файл (читання / запис).

Типи значень зберігаються в стеці. Стек - це область пам'яті, яка використовується для передачі параметрів в методи і зберігання визначених у межах методів локальних змінних. Дані змінної типу значення зберігаються в самій змінної.

До типів значень відносяться:

* Цілочисельні типи (byte, sbyte, char, short, ushort, int, uint, long, ulong);
* Типи з плаваючою комою (float, double);
* Тип decimal;
* Тип bool;
* Призначені для користувача структури (struct);
* Перерахування (enum).

Код нижче показує, що при присвоєнні значення однієї змінної значимого типу інший, подальша зміна однієї з змінних не впливає на іншу. Так тому, що зберігання даних значимого типу відбувається в самій змінної:

static void Main(string[] args)

{ int a = 1; int b = 2; b = a; a = 3;

Console.WriteLine(a); // 3

Console.WriteLine(b); // 1

}

Змінна посилального типу містить не дані, а посилання на них.

Самі дані в цьому випадку вже зберігаються в купі.

Купа - це область пам'яті, в якій розміщуються керовані об'єкти, і працює збирач сміття. Складальник сміття звільняє всі ресурси і об'єкти, які вже не потрібні.

До посилальних типів відносяться:

* Класи (class);
* Інтерфейси (interface);
* Делегати (delegate);
* Тип object;
* Тип string.

У C # значення змінних за замовчуванням передаються за значенням (в метод передається локальна копія параметра, який використовується при виклику). Це означає, що ми не можемо всередині методу змінити параметр із зовні:

public static void ChangeValue(object a)

{ a = 2;

}

static void Main(string[] args)

{ int a = 1;

ChangeValue(a);

Console.WriteLine(a); // 1

Console.ReadLine();

}

Щоб передавати параметри по посиланню, і мати можливість впливати на зовнішню змінну, використовуються ключові слова ref і out.

Щоб використовувати ref, це ключове слово варто вказати перед типом параметра в методі, і перед параметром при виклику методу:

public static void ChangeValue(ref int a)

{

a = 2; } static void Main(string[] args)

{ int a = 1;

ChangeValue(ref a);

Console.WriteLine(a); // 2

Console.ReadLine();

}

Особливістю ref є те, що змінна, яку ми передаємо в метод, обов'язково повинна бути проініціалізувати значенням, інакше компілятор видасть помилку «Use of unassigned local variable 'a'». Це є головною відмінністю ref від out.

Out використовується точно таким же чином як і ref, за винятком того, що параметр не зобов'язаний бути ініціалізованим першим перед передачею, але при цьому в методі переданому параметру обов'язково повинно бути присвоєно нове значення:

public static void ChangeValue(out int a)

{ a = 2; } static void Main(string[] args)

{ int a;

ChangeValue(out a);

Console.WriteLine(a); // 2

Console.ReadLine();

}

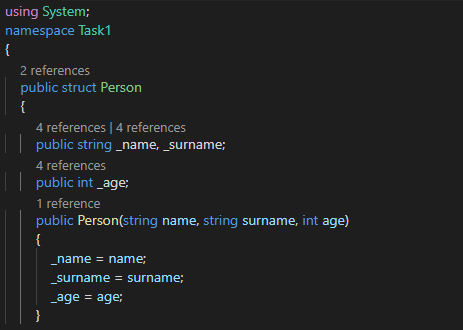
Якщо не присвоїти нове значення параметру out, ми отримаємо помилку «The out parameter 'a' must be assigned to before control leaves the current method»

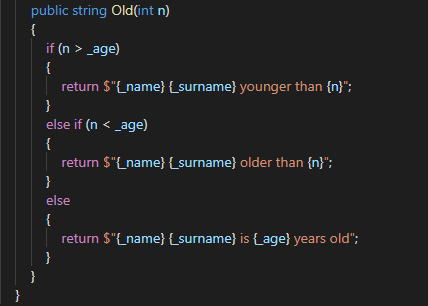
З огляду на той факт, що за замовчуванням в метод передаються параметри за значенням і створюються їх копії в стеці, при використанні складних типів даних (призначені для користувача структури), або якщо метод викликається багато разів, це погано позначиться на продуктивності. В такому випадку також варто використовувати ключові слова ref і out.

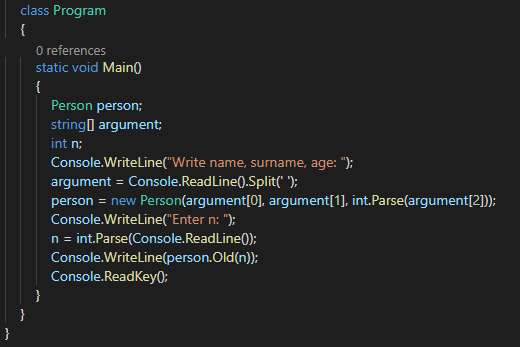
Якщо говорити в цілому про посилальні типи і типи значень, то продуктивність програми впаде, якщо використовувати тільки посилальні типи. На створення змінної посилального типу в купі виділяється пам'ять під дані, а в стеку під посилання на ці дані. Для типів значень пам'ять виділяється тільки в стеку. Час на розміщення даних в стеку менше, ніж в купі, це також йде в плюс типам значень в плані продуктивності.

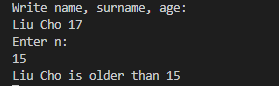
**Хід роботи**

**Завдання 1**

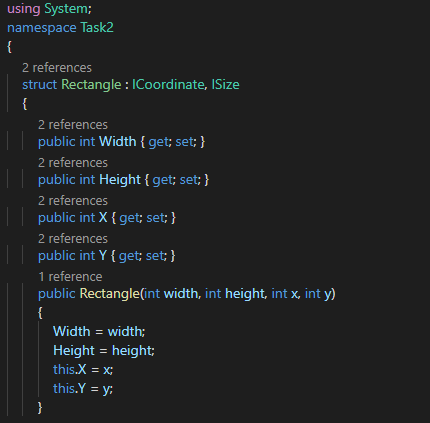


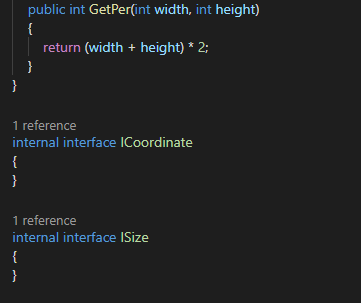


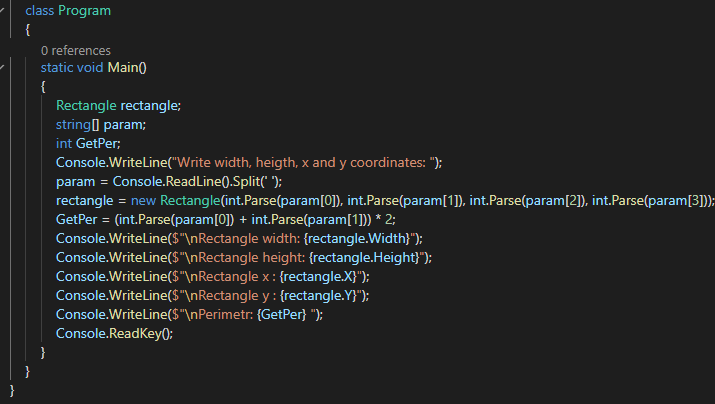


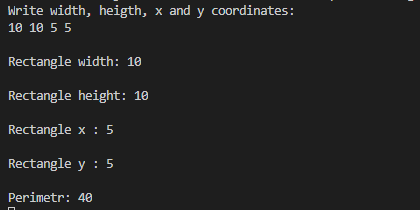


**Завдання 2**

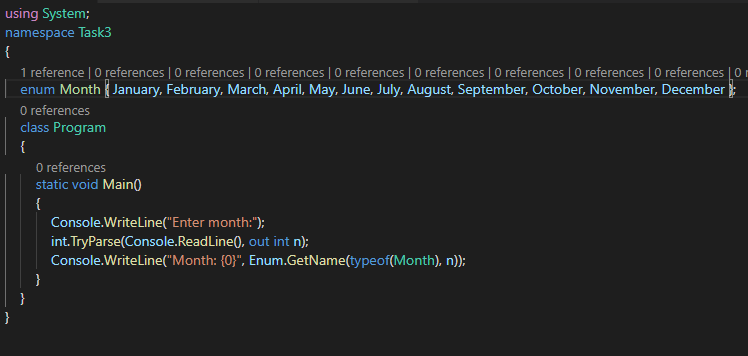






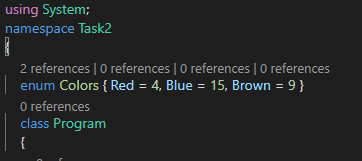


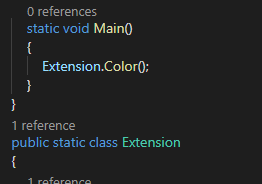
**Завдання 3**

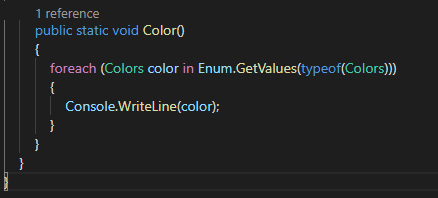




**Завдання 4**

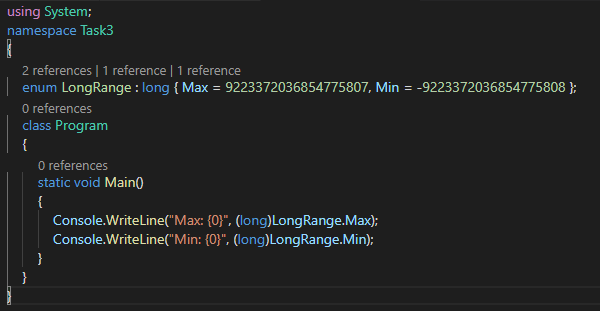








**Завдання 5**





**Висновок:** на даній лабораторній роботі я ознайомився з структурами в мові С# .