МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ІВАНА ФРАНКА

Звіт

про виконання лабораторної роботи № 5

на тему:

«Структури в C#»

Викона(в/ла):

Студент(ка) групи ФеП-12

Шита М.О.

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.

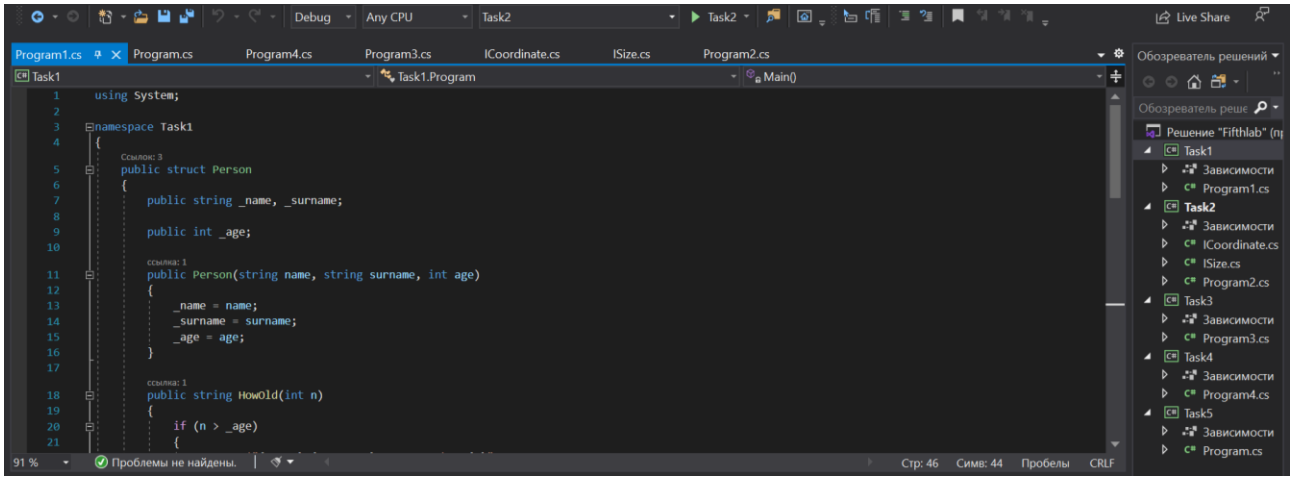
Хід роботи

Завдання 1:

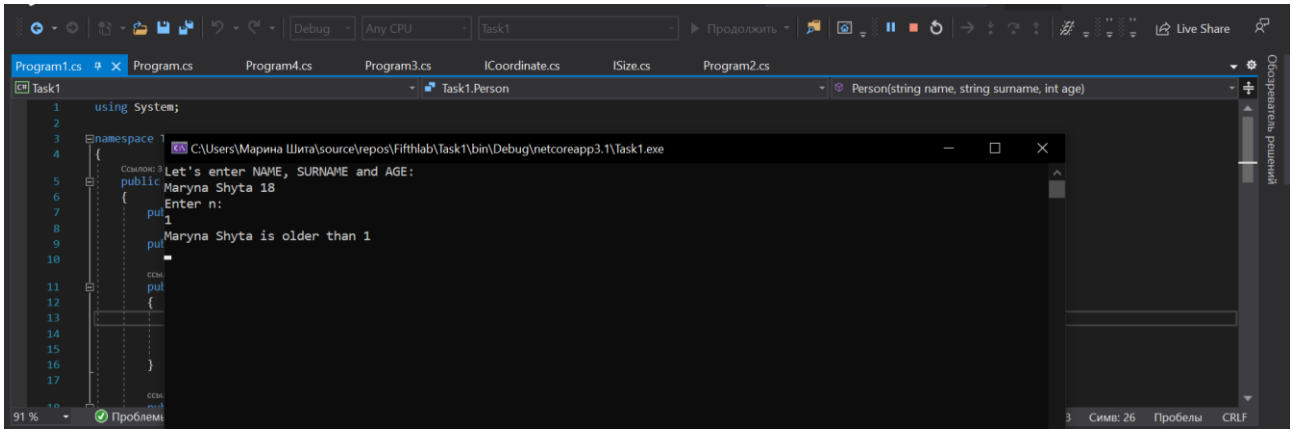
Реалізувала структуру Person в якій зберігається ім'я, прізвище та вік людини. Реалізувала метод для створеної структури, який приймає ціле значення “n” (n > 0) та повертає відформатований string:

- "{Name} {Surname} older than {n}" якщо вік персони більший від заданого “n“

- "{Name} {Surname} younger than {n}" якщо вік персони менший від заданого “n”



зображення коду



вигляд програми

Приклад коду:

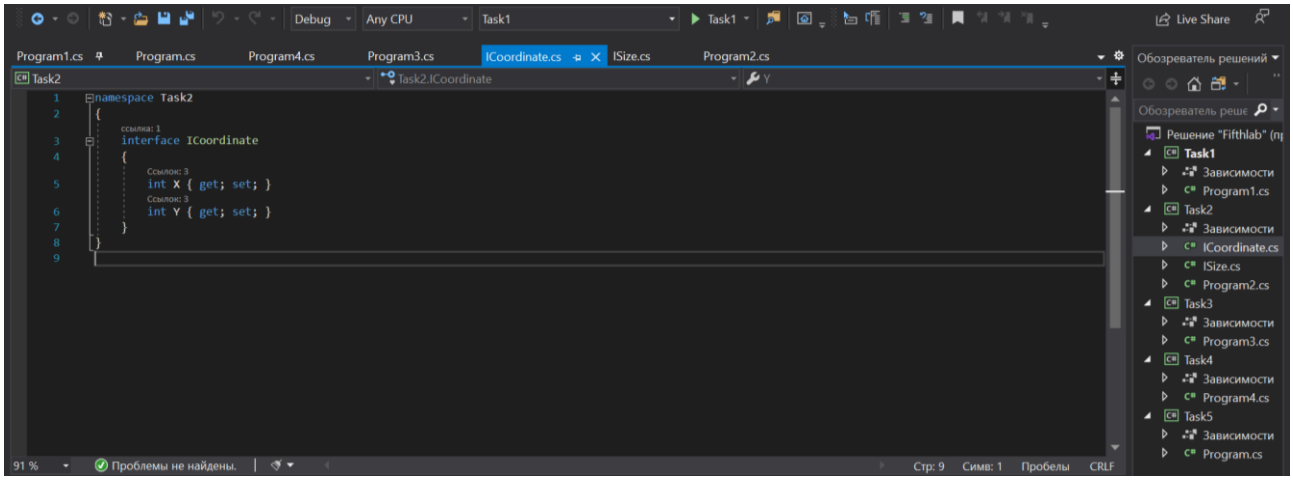
```
using System;

namespace Task1
{
    public struct Person
    {
        public string _name, _surname;

        public int _age;

        public Person(string name, string surname, int age)
        {
            _name = name;
            _surname = surname;
            _age = age;
        }

        public string HowOld(int n)
        {
            if (n > _age)
            {
                return $"{_name} {_surname} younger than {n}";
            }
            else if (n < _age)
            {
                return $"{_name} {_surname} is older than {n}";
            }
            else
            {
                return $"{_name} {_surname} is {_age} years old";
            }
        }
    }

    class Program
    {
        static void Main()
        {
            Person person;
            string[] argument;
            int n;

            Console.WriteLine("Let's enter NAME, SURNAME and AGE: ");
            argument = Console.ReadLine().Split(' ');
            person = new Person(argument[0], argument[1], int.Parse(argument[2]));
```

```
            Console.WriteLine("Enter n: ");
            n = int.Parse(Console.ReadLine());
            Console.WriteLine(person.HowOld(n));
            Console.ReadKey();
        }
    }
}
```

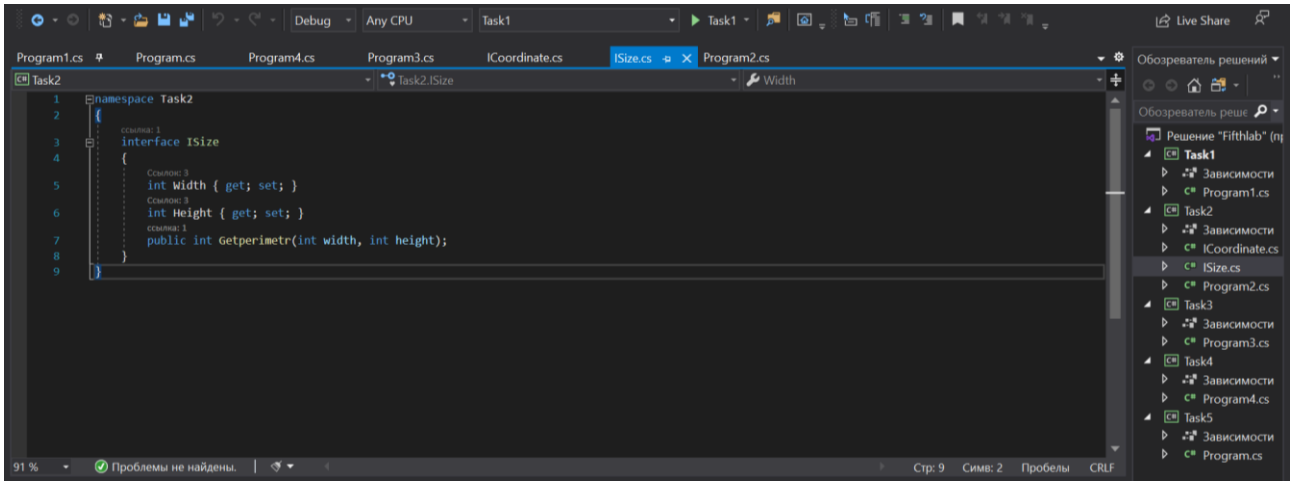
Завдання 2:

Створила інтерфейси ISize (з властивостями Width та Height і методом Perimeter) та ICoordinates (з властивостями X, Y).

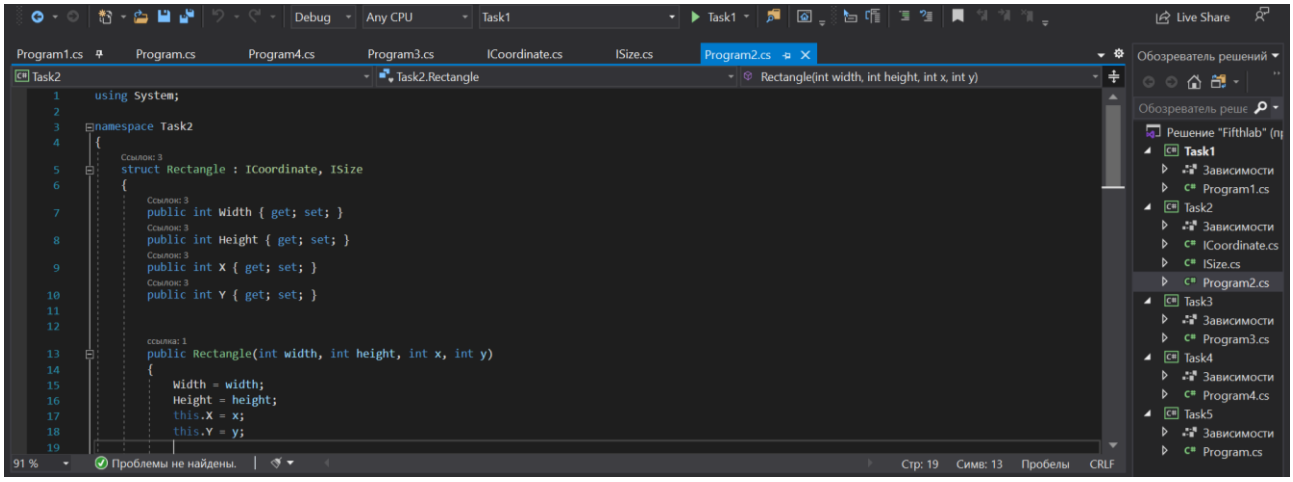
Створила структуру Rectangle, що реалізує дані інтерфейси.



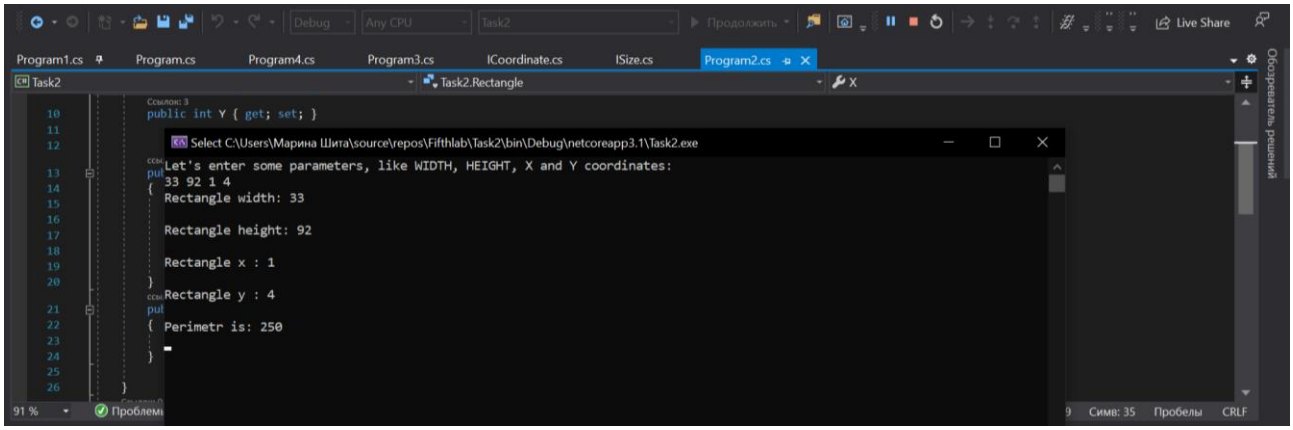
зображення коду(додаток 1)



зображення коду(додаток 2)



зображення коду



вигляд програми

Приклад коду:

```
using System;

namespace Task2
{
   struct Rectangle : ICoordinate, ISize
   {
      public int Width { get; set; }
      public int Height { get; set; }
      public int X { get; set; }
      public int Y { get; set; }


      public Rectangle(int width, int height, int x, int y)
      {
         Width = width;
         Height = height;
         this.X = x;
         this.Y = y;

      }
      public int Getperimetr(int width, int height)
      {
         return (width + height) * 2;
      }

   }
   class Program
   {
      static void Main()
      {
         Rectangle rectangle;
         string[] parameters;
         int Getperimetr;
         Console.WriteLine("Let's enter some parameters, like WIDTH, HEIGHT, X
and Y coordinates: ");
         parameters = Console.ReadLine().Split(' ');

         rectangle = new Rectangle(int.Parse(parameters[0]), int.Parse(parameters[1]),
int.Parse(parameters[2]), int.Parse(parameters[3]));
         Getperimetr = (int.Parse(parameters[0]) + int.Parse(parameters[1])) * 2;

         Console.WriteLine($"Rectangle width: {rectangle.Width}\n");
         Console.WriteLine($"Rectangle height: {rectangle.Height}\n");
         Console.WriteLine($"Rectangle x : {rectangle.X}\n");
         Console.WriteLine($"Rectangle y : {rectangle.Y}\n");
```

```
            Console.WriteLine($"Perimetr is: {Getperimetr} ");
            Console.ReadKey();

        }
    }
}
```

Додаток (interface ICoordinate **):**

```
namespace Task2
{
   interface ICoordinate
   {
      int X { get; set; }
      int Y { get; set; }
   }
}
```

Додаток (interface ISize **):**

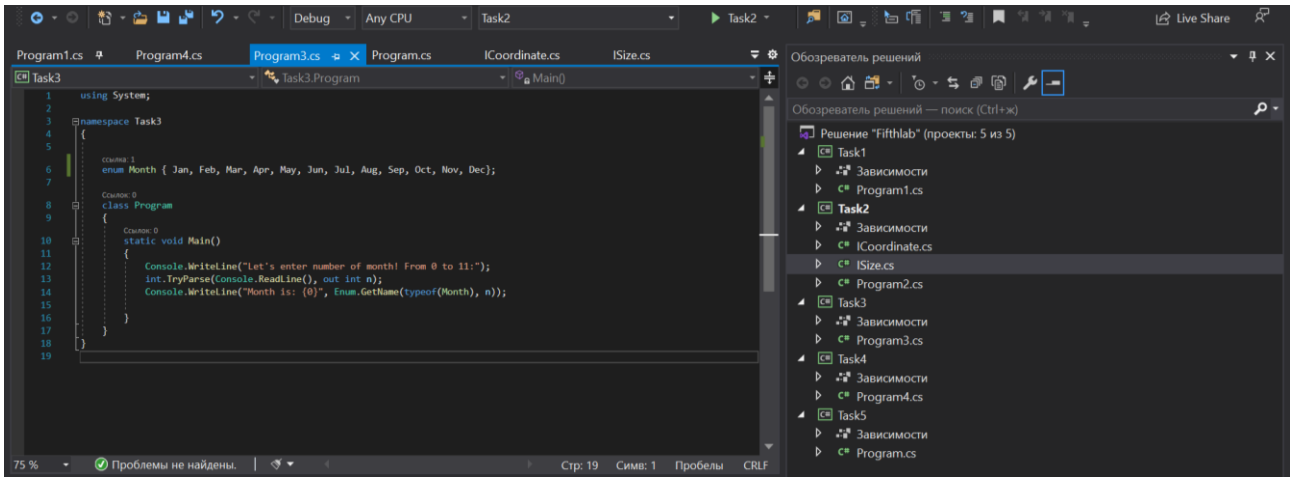
```
namespace Task2
{
   interface ISize
   {
      int Width { get; set; }
      int Height { get; set; }
      public int Getperimetr(int width, int height);
   }
}
```

Завдання 3:

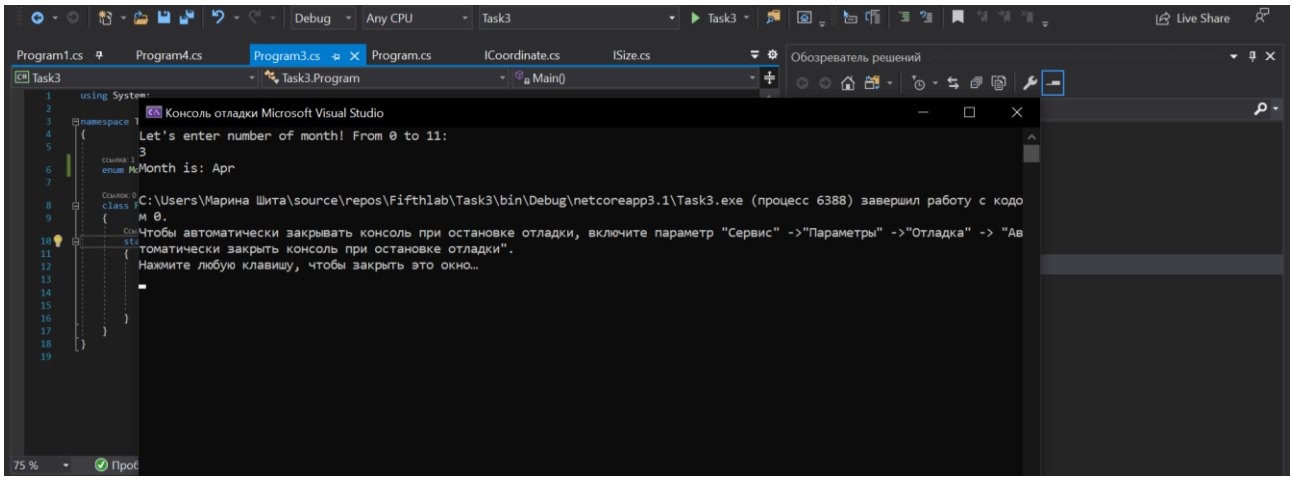
Створила enum який включає в себе всі місяці року.

Значення "n" зчитуються з консолі (0 <= n < 12).

На консоль виводиться місяць, який відповідає значенню "n".



зображення коду



вигляд програми

Приклад коду:

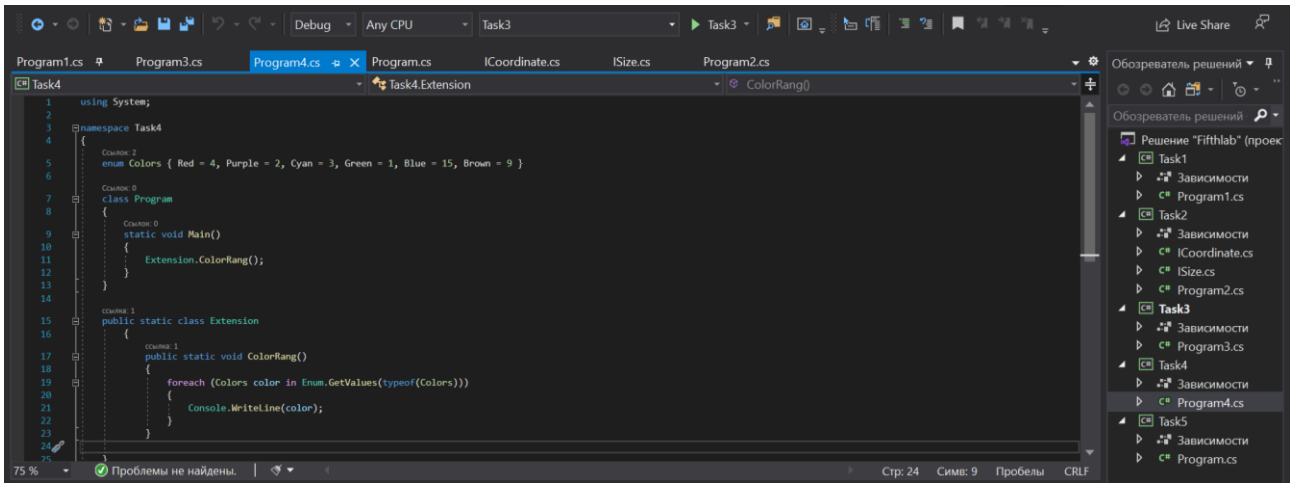
```
using System;

namespace Task3
{

   enum Month { Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, Dec};

   class Program
   {
      static void Main()
      {
         Console.WriteLine("Let's enter number of month! From 0 to 11:");
         int.TryParse(Console.ReadLine(), out int n);
         Console.WriteLine("Month is: {0}", Enum.GetName(typeof(Month), n));

      }
   }
}
```

Завдання 4:

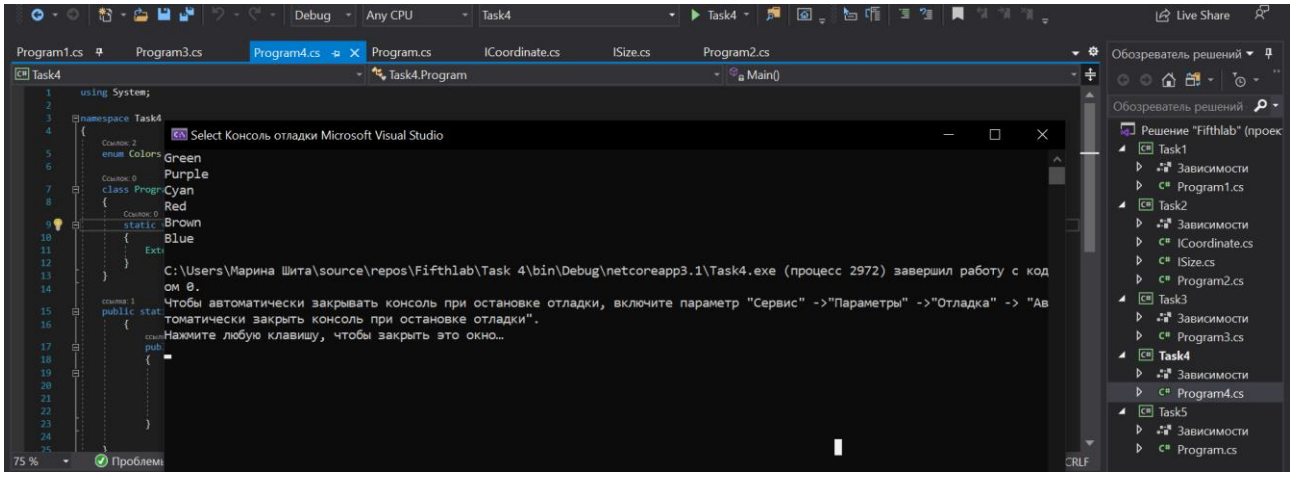
Створила enum який включає в себе декілька кольорів.

Замінила стандартні значення на випадкові (наприклад "Red=4, Blue=15, Green=1...").

Та написала розширювальний метод, котрий виведе на консоль всі значення зі створеного переліку у порядку зростання (наприклад "Green=1, Red=4, Blue=15, ...").



зображення коду



вигляд програми

Приклад коду:

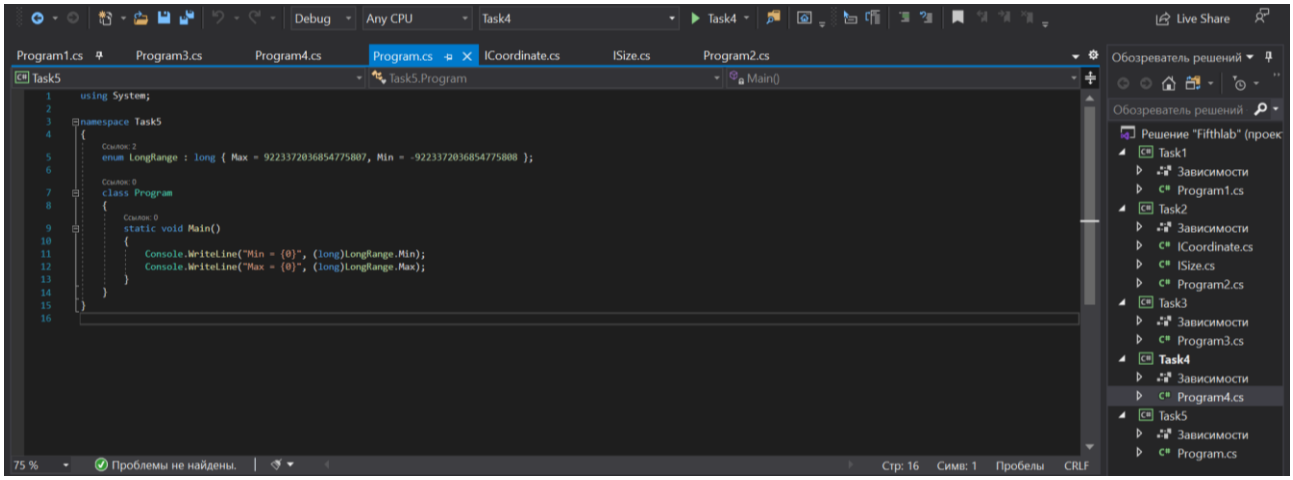
```
using System;

namespace Task4
{
   enum Colors { Red = 4, Purple = 2, Cyan = 3, Green = 1, Blue = 15, Brown = 9 }

   class Program
   {
      static void Main()
      {
         Extension.ColorRang();
      }
   }

   public static class Extension
      {
         public static void ColorRang()
         {
            foreach (Colors color in Enum.GetValues(typeof(Colors)))
            {
               Console.WriteLine(color);
            }
         }

   }
}
```

Завдання 5:

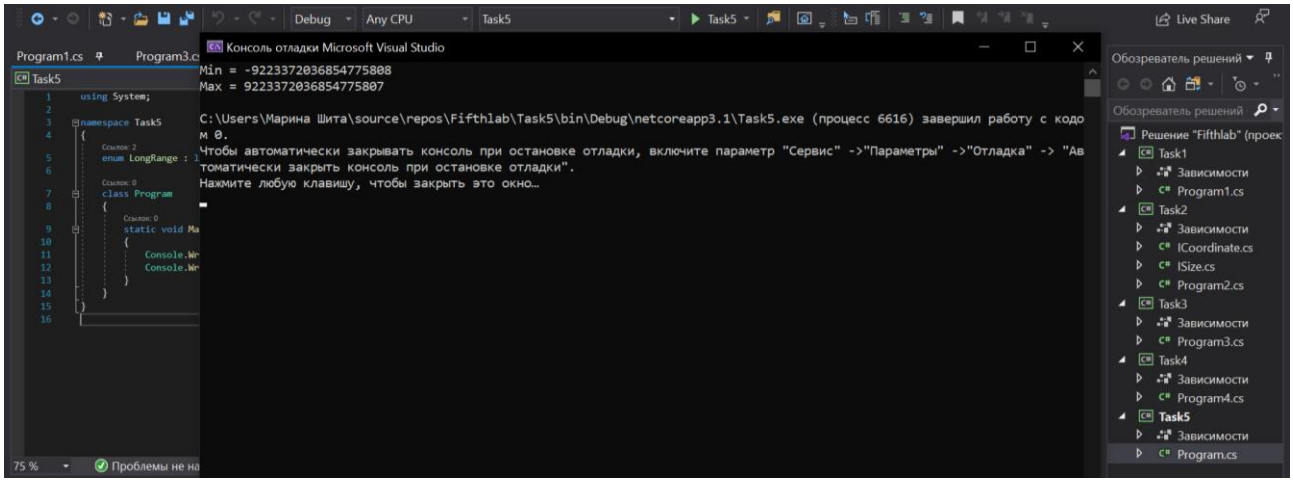
Створила enum LongRange в якому знаходяться мінімальне і максимальне значення типу long ("Max=9223372036854775807,

Min= - 9223372036854775808").

Ці значення виводяться на консоль за допомогою створеного enum.



зображення коду



вигляд програми

Приклад коду:

```
using System;

namespace Task5
{
   enum LongRange : long { Max = 9223372036854775807, Min = -
9223372036854775808 };

   class Program
   {
      static void Main()
      {
         Console.WriteLine("Min = {0}", (long)LongRange.Min);
         Console.WriteLine("Max = {0}", (long)LongRange.Max);
      }
   }
}
```

Висновок: протягом виконання даної лабораторної роботи я ознайомилася з структурами в мові C#, а також вивчила поняттям перерахування та навчилася їх застосовувати.