ЖИТОМИРСЬКА ПОЛІТЕХНІКА
ДЕРЖАВНИЙ УНІВЕРСИТЕТ

C#. Структури

Лекція 05

План

1. Класи і структури
2. Структури в C#
3. Модифікатори доступу структур
4. Ініціалізація структури
5. Конструктори
6. Дії над структурами

Класи і структури

Класи і структури — це користувацькі типи даних, які дозволяють об'єднати дані та функції в єдине ціле.

Класи частіше використовуються для опису об'єктів з більш складною логікою і взаємозв'язками, в той час як структури краще підходять для зберігання простих наборів даних.

Класи працюють з типами-посиланнями це означає, що клас змінного типу містить посилання на об'єкт в пам'яті. Структури працюють з типами-значеннями це означає, що структура змінного типу містить копію самого значення.

Структури в C#

Структура (struct) в C# – це користувацький тип даних, який дозволяє об'єднати різні типи даних в одну логічну одиницю.

Синтактис:

```
struct ім'я_структури : [інтерфейси]
{
    // оголошення членів (полів) та методів структури
}
```

Структури в C#

struct - ключове слово, яке використовується для оголошення типу значення, який називається структурою.

ім'я структури – назва на основі якої будуть оголошуватись об'єкти (змінні, екземпляри структури);

: - використовується для вказівки, що структура реалізує один або декілька інтерфейсів;

інтерфейси – перелік інтерфейсів, методи яких потрібно реалізувати в тілі структури. Якщо інтерфейсів не один, вони розділяються комами.

Структури в C#

```
struct PhoneBook
{
    public String name;
    public String telephone;
    public String email;
}
```

Модифікатор доступу public забезпечує можливість доступу до полів структури

Структура може містити і методи.

В методах можна звертатися до полів та інших методів цієї ж структури безпосередньо за іменем.

```
public struct Person
{
    public string name;
    public int age;

    public void Print()
    {
        Console.WriteLine($"Ім'я: {name} Вік: {age}");
    }
}
```

Модифікатори доступу структур

Модифікатори доступу в C# визначають, з яких частин програми можна отримати доступ до елементів структури (полів, методів, властивостей). Існують наступні модифікатори доступу:

public - доступний з будь-якої частини програми.

private - доступний тільки всередині тієї ж структури

В структурах НЕ ВИКОРИСТОВУЮТЬСЯ:
~~protected~~
~~protected internal~~

Приклад

```
public struct Circle
{
    public double Radius;

    public Circle(double radius){
        Radius = radius;
    }

    public double GetArea(){
        return Math.PI * Radius * Radius;
    }

    public double Getlengthi(){
        return 2 * Math.PI * Radius;
    }
}
```

Конструктор призначений для ініціалізації полів структури при створенні змінної

Метод для обчислення площі кола за визначеною формулою

Метод для обчислення довжини кола за визначеною формулою

Приклад

```
Circle myCircle = new Circle(15);
```

Створюємо екземпляр структури *Circle* з радіусом 15.

```
double area = myCircle.GetArea();
double lengthi = myCircle.Getlengthi();
```

Виклик методів *GetArea()* та *GetLength()* для обчислень кола та зберігання результатів у відповідних змінних.

```
Console.WriteLine($"Площа кола: {area}");
Console.WriteLine($"Довжина кола: {lengthi}");
```

Виведення обчислень на консоль..

Конструктор

Конструктор - це метод, що викликається середовищем виконання при створенні екземпляра структури або класу.

Конструктор – має теж ім'ям, що і його тип. Його сигнатура може включати *модифікатор доступу, ім'я методу, список параметрів* але він не включає тип значення, що повертається.

Структура або клас може мати кілька конструкторів, які приймають різні аргументи.

Конструктори дозволяють забезпечити коректність екземплярів типу, коли вони створюються.

Приклад

```
public struct Circle{

    public double Radius;

    public Circle(double radius){
        Radius = radius;
    }

    public Circle(double x, double y, double radius){
        Radius = radius;
    }

    public Circle(Circle other){
        Radius = other.Radius;
    }

    public double GetArea(){return Math.PI * Radius * Radius;}

    public double Getlengthi(){return 2 * Math.PI * Radius;}
}
```

Конструктор звичайний, створює новий об'єкт

Конструктор з параметрами (x та y, не використовується, але їх можна використати для розширення функціональності, наприклад, для зберігання координат)

Конструктор копіювання

Приклад

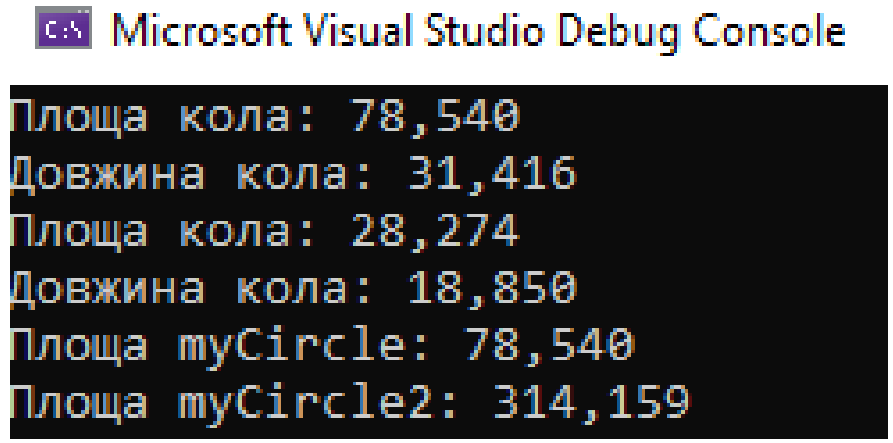
```
Circle myCircle = new Circle(5);
Circle myCircle1 = new Circle(10, 20, 3);
Circle myCircle2 = new Circle(myCircle);

double area = myCircle.GetArea();
double lengthi = myCircle.Getlengthi();
Console.WriteLine($"Площа кола: {area}");
Console.WriteLine($"Довжина кола: {lengthi}");

area = myCircle1.GetArea();
lengthi = myCircle1.Getlengthi();
Console.WriteLine($"Площа кола: {area}");
Console.WriteLine($"Довжина кола: {lengthi}");

myCircle2.Radius = 10;
Console.WriteLine($"Площа myCircle: {myCircle.GetArea()}");
Console.WriteLine($"Площа myCircle2: {myCircle2.GetArea()}");
```

Конструктори в структурах повинні обов'язково ініціалізувати всі поля структури.



В структурі можна реалізовувати:

- ✓ поля;
- ✓ методи;
- ✓ конструктори (крім конструктора за замовчуванням);
- ✓ інтерфейси;
- ✓ індексатори;
- ✓ властивості;
- ✓ події;

Приклад

Розробити структуру Student з полями *прізвище, ім'я, вік* , а таж 2 конструктора з параметрами.

```
struct Student
{
    public string FirstName;
    public string LastName;
    public int Age;
    public Student(string firstName, string lastName, int age)
    {
        FirstName = firstName;
        LastName = lastName;  Age = age;
    }
    public Student(string firstName, string lastName)
    {
        FirstName = firstName;
        LastName = lastName; Age = 18;
    }
```

Приклад

Додамо метод PrintInfo() до структури Student, який буде виводити на консоль повну інформацію про студента (ім'я, прізвище, вік)

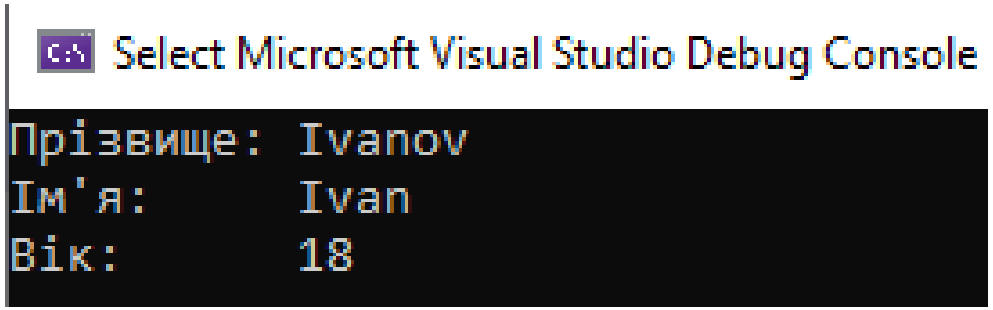
```
public void PrintInfo()
{
    Console.WriteLine($"Прізвище: {LastName}\nІм'я:      {FirstName}\nВік:
{Age}");

}
```

Приклад 1

Щоб використати структуру потрібно створити змінну типу структури:

```
Student stud;
stud.LastName = "Іванов";
stud.FirstName = "Іван";
stud.Age = 18;
stud.PrintInfo();
```



або

```
Student stud = new Student("Ivan", "Ivanov", 18);

stud.PrintInfo();
```

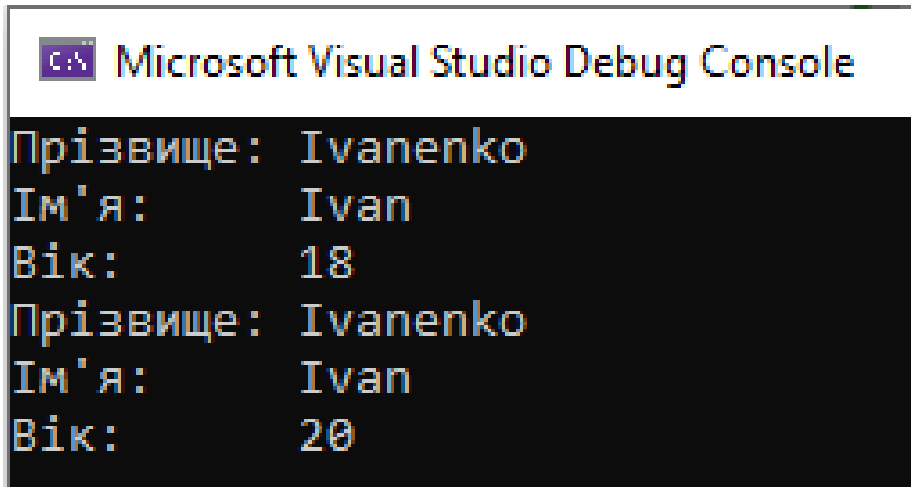
Викликається конструктор, якому передаються три параметри

Приклад 2

Щоб використати структуру потрібно створити змінну типу структури:

```
Student st2 = new Student("Ivan", "Ivanenko");
Student st3 = new Student("Ivan", "Ivanenko", 20);

st2.PrintInfo();
st3.PrintInfo();
```



При створенні екземпляра структури за допомогою new автоматично підбиратиметься конструктор для виклику

Створення екземпляра структури

Як здійснюється підбір конструктору для виклику?

```
Student st2 = new Student("Ivan", "Ivanenko");
Student st3 = new Student("Ivan", "Ivanenko", 20);

struct Student {
  …
  public Student(string firstName, string lastName, int age) {
    …
  }
  public Student(string firstName, string lastName) {
    …
  }
}
```

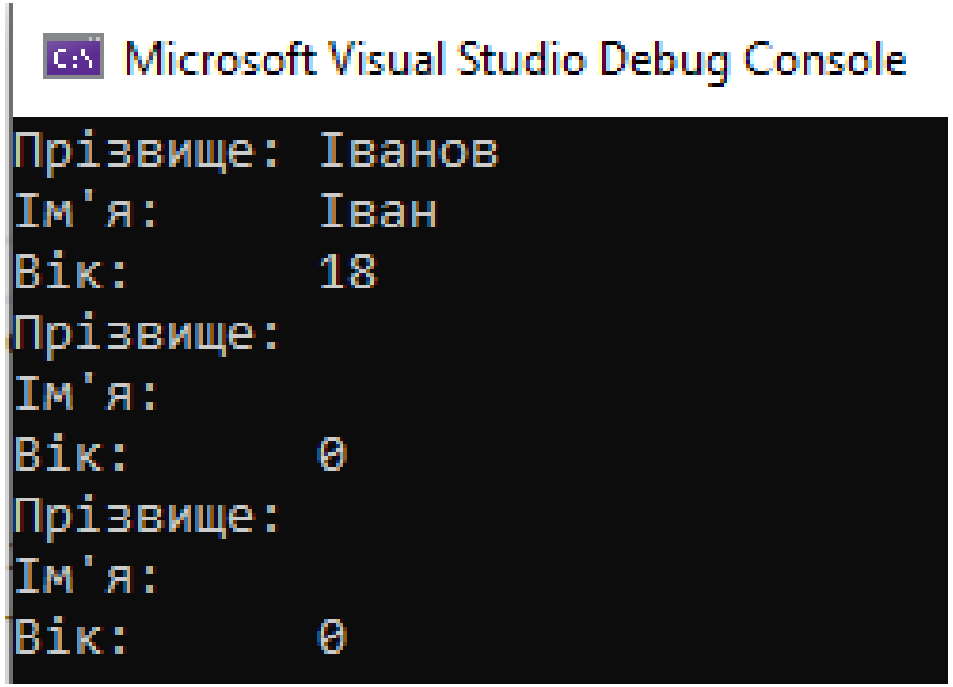
Приклад масиви структур (1)

Можна створювати масиви структур:

```
Student[] stud = new Student[3];

stud[0].LastName = "Іванов";
stud[0].FirstName = "Іван";
stud[0].Age = 18;


for (int i = 0; i < stud.Length; i++)
    stud[i].PrintInfo();
```

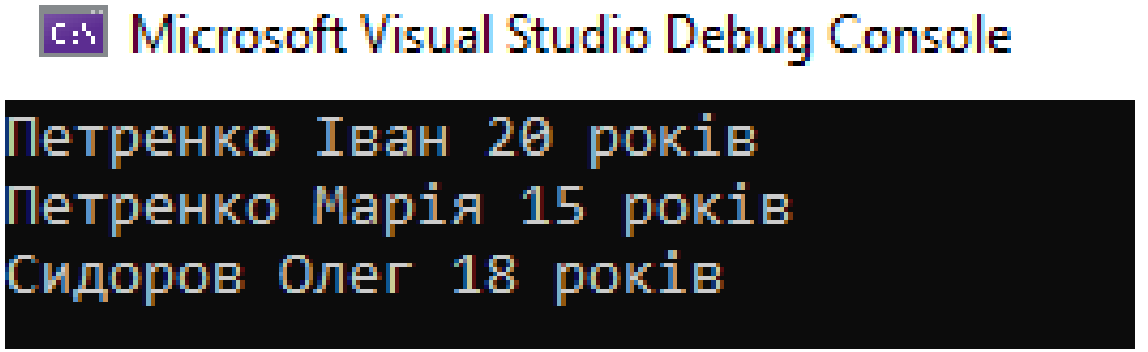


Приклад масиви структур (2)

Можна створювати масиви структур наступним чином:

```
Student[] studentArray = new Student[] {
            new Student { FirstName = "Петренко", LastName = "Іван", Age = 20 },
            new Student {FirstName = "Петренко", LastName = "Марія", Age = 15 },
            new Student {FirstName = "Сидоров", LastName = "Олег", Age = 18 }
        };

        foreach (var stud2 in studentArray)
    Console.WriteLine($"{stud2.FirstName} {stud2.LastName} {stud2.Age} років");
```



Приклад

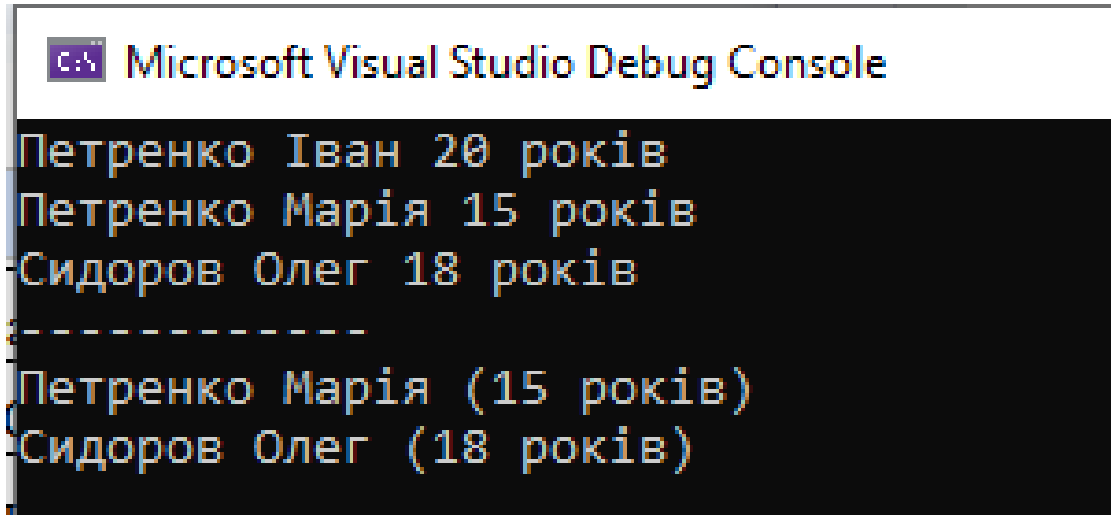
Вибрати з масиву студентів, яким менше або = 18 років.

```
List <Student> Mstudent = new List<Student>();

foreach (var student in studentArray){

     if (student.Age <= 18){
       Mstudent.Add(student);
       Console.WriteLine($"{student.FirstName} {student.LastName}
             ({student.Age} років)");
     }

}
```



Приклад

Вибрати з масиву студентів, яким більше 18 років.

Додамо ще один метод до структури:

```
public bool IsAdult()
{
    return Age > 18;
}
```

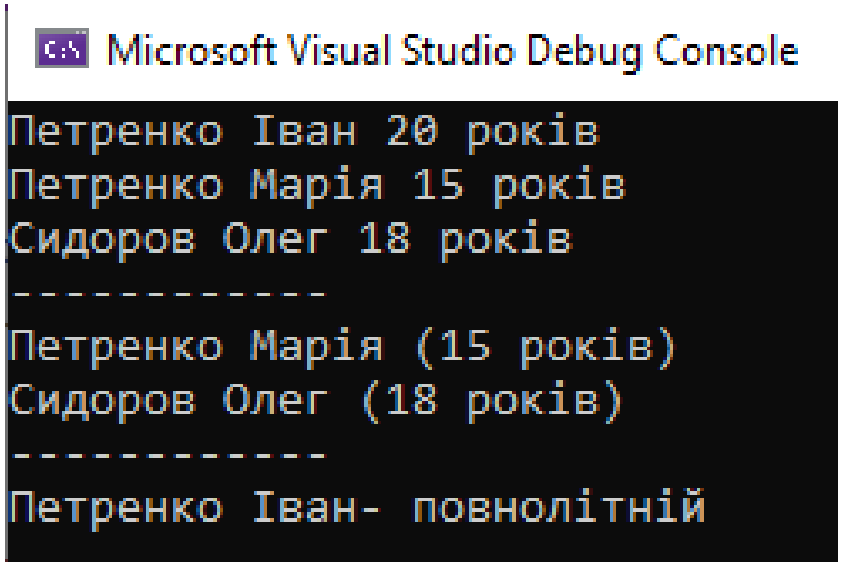
Приклад

```
 for (int i = 0; i < studentArray.Length; i++)
{

    if (studentArray[i].IsAdult())
    {

        Console.WriteLine($"{studentArray[i].FirstName}
              {studentArray[i].LastName}- повнолітній");

    }
}
```



Структури для роботи з датою та часом

Для роботи з датою та часом в середовищі .Net реалізовано структури DateTime та TimeSpan

Структура *DateTime* представляє дату та час в діапазоні від 00:00:00 1 січня 0001 року (н. е.) до 23:59:59 31 грудня 9999 року (н. е.).

Значення часу вимірюється в 100-наносекундних одиницях, що називаються *тактами*, і точна дата представляється числом тактів від 00:00:00 1 січня 0001 року (н.е.) по григоріанському календарю.

Переваги та обмеження

Переваги struct:

- ✓ підходить для представлення простих об'єктів, що містять невелику кількість даних;
- ✓ зберігаються в стеку, що забезпечує швидший доступ до них порівняно з класами;
- ✓ є типами значень, тому їх копіювання не змінює оригінальний об'єкт.

Обмеження struct:

- ✓ не підтримують спадкування;
- ✓ не можуть містити віртуальні методи;
- ✓ не можуть мати конструктор без параметрів.