# **Structure types**

Наряду с классами структуры представляют еще один способ создания собственных типов данных в C#. Более того многие примитивные типы, например, ***int***, ***double*** и т.д., по сути являются структурами.

**Определение структуры**

Для определения структуры применяется ключевое слово **struct**:

struct имя\_структуры

{

// элементы структуры

}

После слова **struct** идет название структуры и далее в фигурных скобках размещаются элементы структуры - поля, методы и т.д.

Например, определим структуру, которая будет называться ***Person*** и которая будет представлять человека:

struct Person

{

}

Как и классы, структуры могут хранить состояние в виде полей (переменных) и определять поведение в виде методов. Например, добавим в структуру ***Person*** пару полей и метод:

struct Person

{

public string name;

public int age;

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

}

В данном случае определены две переменные - name и age для хранения соответственно имени и возраста человека и метод Print для вывода информации о человеке на консоль.

И как и в случае с классами, для обращения к функциональности структуры - полям, методам и другим компонентам структуры применяется точечная нотация - после объекта структуры ставится точка, а затем указывается компонент структуры:

1.объект.поле\_структуры

2.объект.метод\_структуры(параметры\_метода)

**Создание объекта структуры**

**Инициализация с помощью конструктора**

Для использования структуры ее необходмо инициализировать. Для инициализации создания объектов структуры, как и в случае с классами, применяется вызов конструктура с оператором new. Даже если в коде стуктуры не определено ни одного конструктора, тем не менее имеет как минимум один конструктор - конструктор по умолчанию, который генерируется компилятором. Этот конструктор не принимает параметров и создает объект структуры со значениями по умолчанию.

new название\_структуры();

Например, создадим объект структуры ***Person*** с помощью конструктора по умолчанию:

Person tom = new Person(); // вызов конструктора

// или так

// Person tom = new();

tom.name = "Tom"; // изменяем значение по умолчанию в поле name

tom.Print(); // Имя: Tom Возраст: 0

struct Person

{

public string name;

public int age;

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

}

В данном случае создается объект ***tom***. Для его создания вызывается конструктор по умолчанию, который устанавливает значения по умолчанию для его полей. Для числовых данных это значение 0, поэтому поле ***age*** будет иметь значение 0. Для строк это значение null, которое указывает на отсутствие значения. Но далее, если поля доступны (а в данном случае поскольку они имеют модификатор public они доступны), мы можем изменить их значения. Так, здесь полю name присваивается строка "Tom". Соответственно при выполнении метода Print() мы получим следующий консольный вывод:

Имя: Tom Возраст: 0

**Непосредственная иницилизация полей**

Если все поля структуры доступны (как в случае с полями структуры Person, который имеют модификатор public), то структуру можно инициализировать без вызова конструктора. В этом случае необходимо присвоить значения всем полям структуры перед получением значений полей и обращением к методам структуры. Например:

Person tom; // не вызываем конструктор

// инициализация полей

tom.name = "Sam";

tom.age = 37;

tom.Print(); // Имя: Sam Возраст: 37

struct Person

{

public string name;

public int age;

public void Print()

{

Console.WriteLine($"Имя: {name} Возраст: {age}");

}

}

**Конструкторы структуры**

Как и класс, структура может определять конструкторы. Например, добавим в структуру Person конструктор:

Person tom = new();

Person bob = new("Bob");

Person sam = new("Sam", 25);

tom.Print(); // !!!! Имя: Возраст: 0

bob.Print(); // Имя: Bob Возраст: 1

sam.Print(); // Имя: Sam Возраст: 25

struct Person

{

public string name;

public int age;

public Person(string name = "Tom", int age = 1)

{

this.name = name;

this.age = age;

}

public void Print() => Console.WriteLine($"Имя: {name} Возраст: {age}");

}

В данном случае в структуре Person определен конструктор с двумя параметрами, для которых предоставлены значения по умолчания. Однако обратите внимание на создание первого объекта структуры:

Person tom = new(); // по прежнему используется конструктор без параметров по умолчанию

tom.Print(); // !!!! Имя: Возраст: 0

Здесь по-прежнему применяется конструктор по умолчанию, тогда как при инициализации остальных двух переменных структуры применяется явно определенный конструктор.

Однако начиная с версии C# 10 мы можем определить свой конструктор без параметров:

Person tom = new();

tom.Print(); // Имя: Tom Возраст: 37

struct Person

{

public string name;

public int age;

public Person()

{

name = "Tom";

age = 37;

}

public void Print() => Console.WriteLine($"Имя: {name} Возраст: {age}");

}

В случае если нам необходимо вызывать конструкторы с различным количеством параметров, то мы можем, как и в случае с классами, вызывать их по цепочке:

Person tom = new();

Person bob = new("Bob");

Person sam = new("Sam", 25);

tom.Print(); // Имя: Tom Возраст: 1

bob.Print(); // Имя: Bob Возраст: 1

sam.Print(); // Имя: Sam Возраст: 25

struct Person

{

public string name;

public int age;

public Person() : this("Tom")

{ }

public Person(string name) : this(name, 1)

{ }

public Person(string name, int age)

{

this.name = name;

this.age = age;

}

public void Print() => Console.WriteLine($"Имя: {name} Возраст: {age}");

}

Консольный вывод программы:

Имя: Tom Возраст: 1

Имя: Bob Возраст: 1

Имя: Sam Возраст: 25

**Инициализатор структуры**

Также, как и для класса, можно использовать инициализатор для создания структуры:

Person tom = new Person { name = "Tom", age = 22 };

tom.Print(); // Имя: Tom Возраст: 22

struct Person

{

public string name;

public int age;

public void Print() => Console.WriteLine($"Имя: {name} Возраст: {age}");

}

При использовании инициализатора сначала вызывается конструктор без параметров: если мы явным образом не определили конструктор без параметров, то вызывается конструктор по умолчанию. А затем его полям присваиваются соответствующие значения.

**Копирование структуры с помощью with**

Если нам необходимо скопировать в один объект структуры значения из другого с небольшими изменениями, то мы можем использовать оператор ***with***:

Person tom = new Person { name = "Tom", age = 22 };

Person bob = tom ***with*** { name = "Bob" };

bob.Print(); // Имя: Bob Возраст: 22

В данном случае объект bob получает все значения объекта ***tom***, а затем после оператора ***with*** в фигурных скобках указывается поля со значениями, которые мы хотим изменить.

## **Члены экземпляров readonly**

Модификатор readonly также можно использовать для объявления того, что член экземпляра не изменяет состояние структуры. Если не удается объявить весь тип структуры как readonly, используйте модификатор readonly, чтобы пометить члены экземпляров, которые не изменяют состояние структуры.

В члене экземпляра readonly невозможно назначать поля экземпляра структуры. Однако член readonly может вызвать член, не являющийся readonly. В этом случае компилятор создает копию экземпляра структуры и вызывает в ней член, не являющийся readonly. В результате исходный экземпляр структуры не изменяется.

Как правило, модификатор readonly применяется к следующим типам элементов экземпляров.

* **Методы**

public readonly double Sum()

{

return X + Y;

}

Можно также применить модификатор readonly к методам, переопределяющим методы, объявленные в System.Object.

public readonly override string ToString() => $"({X}, {Y})";

* **Свойства и индексаторы**

private int counter;

public int Counter

{

readonly get => counter;

set => counter = value;

}

Eсли необходимо применить модификатор readonly к методам доступа свойства или индексатора, примените его в объявлении свойства или индексатора.

Модификатор readonly можно применить к статическим полям типа структуры, но не к другим статическим членам, таким как свойства или методы.