C# struct

The struct (structure) is like a class in C# that is used to store data. However, unlike classes, a struct is a value type.

Suppose we want to store the name and age of a person. We can create two variables: `name` and `age` and store value.
However, suppose we want to store the same information of multiple people.

In this case, creating variables for an individual person might be a tedious task. To overcome this we can create a struct that stores `name` and `age`. Now, this struct can be used for every person.

Define struct in C#

In C#, we use the `struct` keyword to define a struct. For example,

```
struct Employee  {
  public int id;
}
```

Here, `id` is a field inside the struct. A struct can include methods, etc as well.

Declare struct variable

Before we use a struct, we first need to create a struct variable. We use a struct name with a variable to declare a struct variable. For example,

```
struct Employee {
  public int id;
}
...

// declare emp of struct Employee
Employee emp;
```

In the above example, we have created a struct named `Employee`. Here, we have declared a variable `emp` of the struct `Employee`.

Access C# struct

We use the struct variable along with the `.` operator to access members of a struct. For example,

```
struct Employee {
  public int id;
}
...
// declare emp of struct Employee
Employee emp;

// access member of struct
emp.id = 1;
```

Here, we have used variable `emp` of a struct `Employee` with `.` operator to access members of the `Employee`.

```
emp.id = 1;
```

This accesses the `id` field of struct `Employee`.

> **Note**: Primitive data types like `int`, `bool`, `float` are pre-defined structs in C#.

Example: C# Struct

```
using System;
namespace CsharpStruct {

  // defining struct
  struct Employee {

    public int id;

    public void getId(int id) {
      Console.WriteLine("Employee Id: " + id);
    }
  }

  class Program {
    static void M
ain(string[] args) {

      // declare emp of struct Employee
      Employee emp;


      // accesses and sets struct field
      emp.id = 1;

      // accesses struct methods
      emp.getId(emp.id);

      Console.ReadLine();
    }
  }
}
```

Output

```
Employee Id: 1
```

In the above program, we have created a struct named `Employee`. It contains a field `id` and a method `getId()`.

Inside the `Program` class, we have declared a variable `emp` of struct `Employee`. We then used the `emp` variable to access fields and methods of the class.

Note: We can also instantiate a struct using the `new` keyword. For example,

```
Employee emp = new Employee();
```

Here, this line calls the parameterless constructor of the struct and initializes all the members with default values.

Constructors in C# struct

In C#, a struct can also include constructors. For example,

```
struct Employee {

  public int id;

  // constructor
  public Employee(int employeeId) {
   id = employeeId
  }
}
```

Here, we have created a parameterized constructor `Employee()` with parameter `employeeId`.

> **Note**: We cannot create parameterless constructors in C# version 9.0 or below.

Example: Constructor in C# structs

```
using System;
namespace CsharpStruct {

  // defining struct
  struct Employee {
    public int id;

    public string name;

    // parameterized constructor
    public Employee(int employeeId, string employeeName) {

      id = employeeId;
      name = employeeName;
    }

  }

  class Program {
    static void Main(string[] args) {

      // calls constructor of struct
      Employee emp = new Employee(1, "Brian");


      Console.WriteLine("Employee Name: " + emp.name);
      Console.WriteLine("Employee Id: " + emp.id);

      Console.ReadLine();
    }
  }
}
```

Output

```
Employee Name: Brian
Employee Id: 1
```

In the above example, we have created a parameterized constructor inside the `Employee` struct. Inside the constructor, we have assigned the values of fields: `id` and `name`.

Notice the line,

```
Employee emp = new Employee(1, "Brian");
```

Like in C# classes, we are using the `new` keyword to call the constructor. Here, **1** and **"Brian"** are arguments passed to the constructor, where they are assigned to the parameters `employeeID` and `employeeName` respectively."

Note: We must assign the value for every field of struct inside the parameterized constructor. For example,

```
// error code
public Employee(int employeeID, employeeName) {
  id = employeeID;
}
```

Here, we have not assigned the value for the `name` field. So the code will generate an error.

Properties in C# struct

We can also use properties inside a C# struct. For example,

```
using System;
namespace CsharpStruct {

  // defining struct
  struct Employee {
    private int id;

    // creates property
    public int Id {

      // returns id field
      get {
        return id;
      }

      // sets id field
      set {
        id = value;
      }
    }

  }

  class Program {
    static void Main(string[] args) {

      // calls the constructor of struct
      Employee emp = new Employee();

      emp.Id = 1;
      Console.WriteLine("Employee Id: " + emp.Id);

      Console.ReadLine();

    }
  }
}
```

Output

```
Employee Id: 1
```

In the above example, we have `Id` property inside the `Employee` struct. The `get` method returns the `id` field and the `set` method assigns the value to the `id` field.

Difference between class and struct in C#

In C# classes and structs look similar. However, there are some differences between them.

A class is a reference type whereas a struct is a value type. For example,

```
using System;
namespace CsharpStruct {

  // defining class
  class Employee {
    public string name;

  }

  class Program {
    static void Main(string[] args) {

      Employee emp1 = new Employee();
      emp1.name = "John";

      // assign emp1 to emp2
      Employee emp2 = emp1;

      emp2.name = "Ed";
      Console.WriteLine("Employee1 name: " + emp1.name);

      Console.ReadLine();
    }
  }
}
```

Output

```
Employee1 name: Ed
```

In the above example, we have assigned the value of `emp1` to `emp2`. The `emp2` object refers to the same object as `emp1`. So, an update in `emp2` updates the value of `emp1` automatically.
This is why a class is a **reference type**.

Contrary to classes, when we assign one struct variable to another, the value of the struct gets copied to the assigned variable. So updating one struct variable doesn't affect the other. For example,

```
using System;
namespace CsharpStruct {

  // defining struct
  struct Employee {
    public string name;

  }

  class Program {
    static void Main(string[] args) {

      Employee emp1 = new Employee();
      emp1.name = "John";

      // assign emp1 to emp2
      Employee emp2 = emp1;

      emp2.name = "Ed";
      Console.WriteLine("Employee1 name: " + emp1.name);

      Console.ReadLine();
    }
  }
}
```

Output

```
Employee1 name: John
```

When we assign the value of `emp1` to `emp2`, a new value `emp2` is created. Here, the value of `emp1` is copied to `emp2`. So, change in `emp2` does not affect `emp1`. This is why struct is a **value type**.

Using Loops in Struct

You can use loops to iterate through a collection of structs just like you would with any other data type. However, because structs are value types in C#, there are some important considerations to keep in mind when working with them in loops.

```
using System;

struct Employee
{
    public int Id;
    public string Name;

    public Employee(int id, string name)
    {
        Id = id;
        Name = name;
    }
}

class Program
{
    static void Main(string[] args)
    {
        Employee[] employees = new Employee[]
        {
            new Employee(1, "John"),
            new Employee(2, "Alice"),
            new Employee(3, "Bob")
        };

        // Using a for loop to iterate through the array of Employee structs
        for (int i = 0; i < employees.Length; i++)
        {
            Console.WriteLine($"Employee Id: {employees[i].Id}, Name:
{employees[i].Name}");
        }

        // Using a foreach loop to iterate through the array of Employee structs
        foreach (Employee employee in employees)
        {
            Console.WriteLine($"Employee Id: {employee.Id}, Name: {employee.Name}");
        }
    }
}
```

In this example, we have defined a struct **Employee**, created an array of **Employee** structs, and used both a **for** loop and a **foreach** loop to iterate through the array.