562.613 APPLIED DATA STRUCTURES

Lecture 03

Dr. M. G. Abbas Malik
muhammad.malik@manukau.ac.nz

Lecture 03

- Structure
- Collections in C#
- ArrayList
- SortedList
- Stack
- Queue
- Generics

Structure - struct

- Value Type Entity
- Share most of the same syntax as classes
- Contain constructors, constants, fields, methods, properties, indexers, operators, events and nested types
- Defined using the struct keyword

struct - Limitations

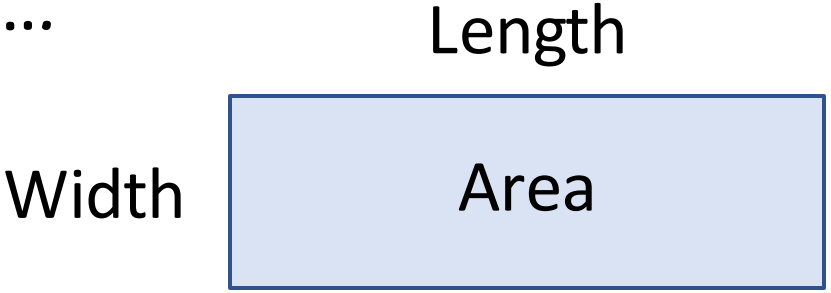
- Within a **struct** declaration, fields cannot be initialized unless they are declared as **const** or **static**
- A struct cannot declare a default constructor
- A finalizer (destructor) cannot be defined
- structs are copied on assignment
- structs are **value types** and classes are **reference types**

struct – Limitations

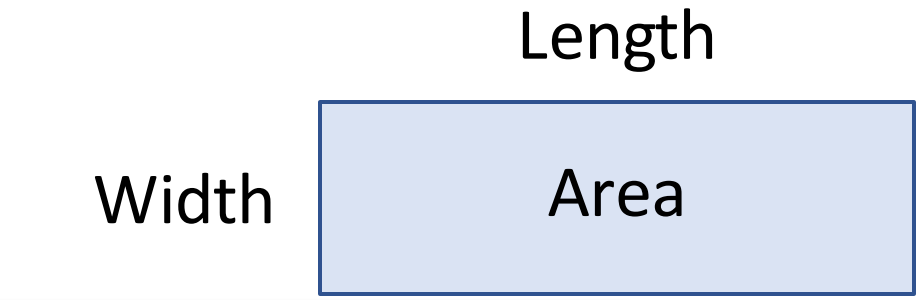
- Can be instantiated without using a **new** operator like Primitive Data Types
- No **inheritance**
- Can implement **interfaces**
- A **struct** can be used as a **nullable type** and can be assigned a **null** value
- Members cannot be specified as **abstract**, **virtual**, or **protected**

struct

- Faster than a class object – **Value Type**
- Use struct whenever you want to store data only – good for game programming
- Suitable for light-weight objects:
  - Point, Rectangle, Square, Color, …



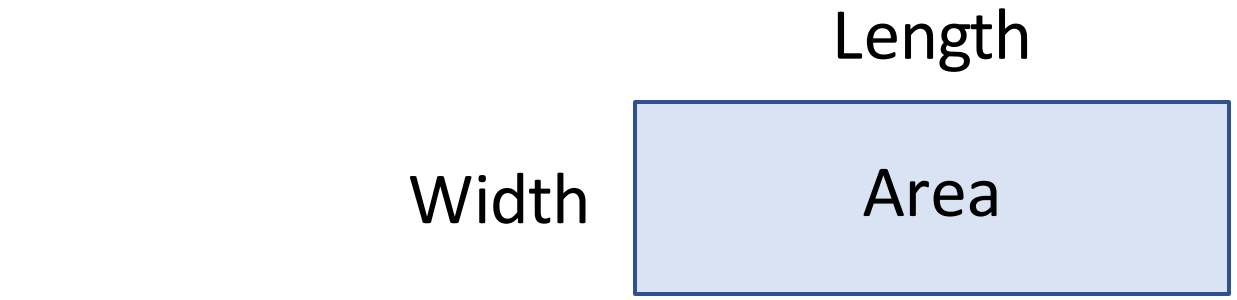
struct




```
public struct MyRectangle\n
{\n
    public int length;\n
    public int width;\n
    public MyRectangle(int len, int wid){\n
        length = len;\n
        width = wid;\n
    }\n
    public int areaRectangle(){\n
        return length * width;\n
    }\n
}\n
```

```
Area of Rectangle is 144
```

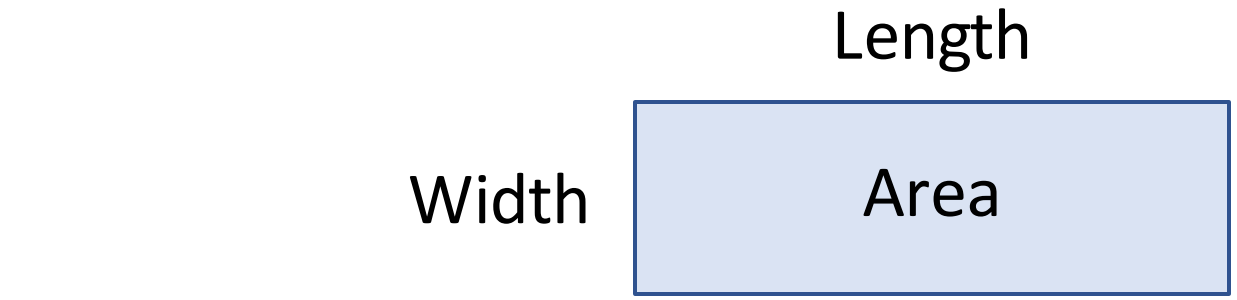
struct




```
MyRectangle myRectangle;\r\n
myRectangle.length = 12;\r\n
myRectangle.width = 12;\r\n
int area = myRectangle.areaRectangle();\r\n
Console.WriteLine("Area of Rectangle is {0}", area);
```

```
Area of Rectangle is 144
```

struct



Compiler Error

```
MyRectangle myRectangle;\r\n
Console.WriteLine("{0}", myRectangle.areaRectangle());
```

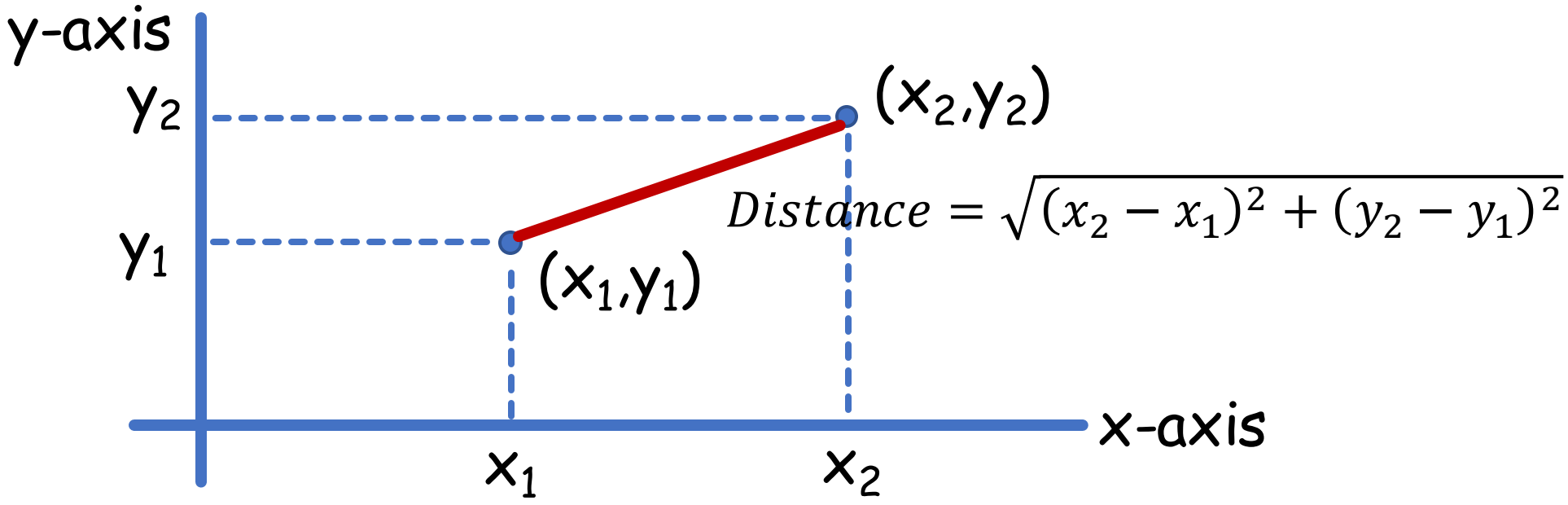
Initialize the member variables with default value, i.e. ZERO for int

```
MyRectangle myRectangle = new MyRectangle();\r\n
Console.WriteLine("Area of Rectangle is {0}",\r\
                  myRectangle.areaRectangle());
```

```
MyRectangle myRectangle = new MyRectangle(12, 8);
```

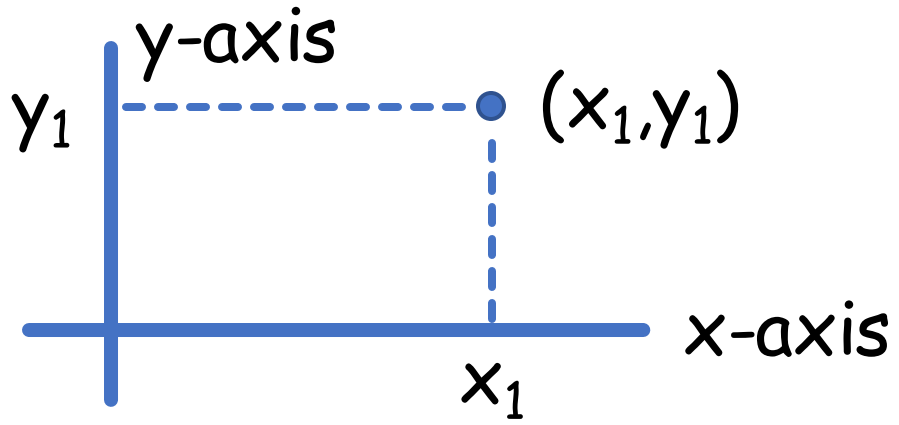
struct

- Suitable for light-weight objects:
  - **Point**, Rectangle, Square, Color, …



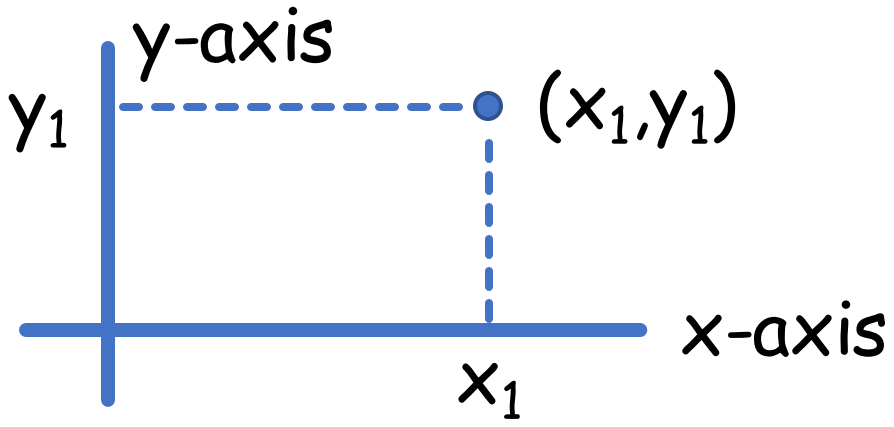
struct

```
public struct MyPoint\n
{\n
    public int x { get; set; }\n
    /*\n
     * public void set(int val){\n
     *      x = val;\n
     * }\n
     * public int get(){\n
     *      return x;\n
     * }\n
     */\n
    public int y { get; set; }\n
    public MyPoint(int a, int b){\n
        x = a;\n
        y = b;\n
    }\n
}\n
```




```
MyPoint myPoint1 = new MyPoint();
myPoint1.x = 34;\r\n
myPoint1.y = -12;\r\n
```

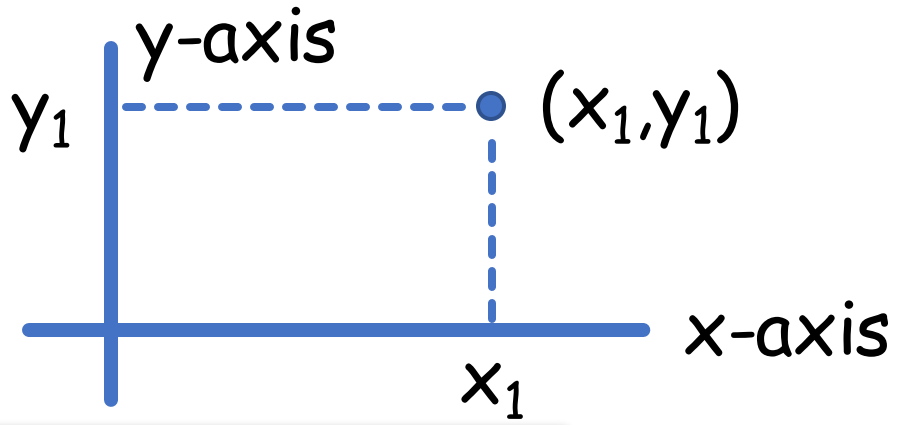
struct



$$Distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

```
public double distance(MyPoint point){
    double distancepoints;
    distancepoints = Math.Sqrt(
        Math.Pow((point.x - this.x), 2)
        + Math.Pow((point.y - this.y), 2));
    return distancepoints;
}
```

struct




```
MyPoint myPoint1 = new MyPoint();\r\n
myPoint1.x = 34;\r\n
myPoint1.y = -12;\r\n
MyPoint myPoint2 = new MyPoint(2, 8);\r\n
Console.WriteLine("Point 1 ({0}, {1})",\r\n
                  myPoint1.x, myPoint1.y);\r\n
Console.WriteLine("Point 2 ({0}, {1})",\r\n
                  myPoint2.x, myPoint2.y);\r\n
Console.WriteLine("Distance between " +\r\n
                  "Point1 & Point2 is {0})",\r\n
                  myPoint1.distance(myPoint2));
```

```
Point 1 (34, -12)
Point 2 (2, 8)
Distance between Point1 & Point2 is 37.7359245282264)
```

C# Collections

- Specialized classes that hold many values or objects in a specific series
- Two Types
  1. **Non-generic collections**
  2. **Generic Collections**
- Every collection class implements the **IEnumerable** interface – **foreach** loop

ArrayList - C# Collection

- **Non-generic collection**
- Contain **elements** of **different types**
- Don't need to specify the **size**
- **Dynamic Size** - Grows automatically as we add items in it

```
ArrayList arrayList = new ArrayList();
```

ArrayList - Properties

Property	Description
Capacity	Gets or sets the number of elements that the ArrayList contain
Count	Gets the number of elements actually in the ArrayList

```
Console.WriteLine("Array List Capacity : {0}",
                  arrayList.Capacity);\r\n
arrayList.Capacity = 5;\r\n
Console.WriteLine("Array List Capacity : {0}",
                  arrayList.Capacity);\r\n
Console.WriteLine("Array List Count : {0}",
                  arrayList.Count);\r\n
```

```
C# Collection - Non_Generic
Array List Capacity : 0
Array List Capacity : 5
Array List Count : 0
```

ArrayList - Adding Elements

```
arrayList.Add(562.613);\r\n
arrayList.Add("Applied Data Structures");
arrayList.Add("Quarter 2, 2018");\r\n
arrayList.Add(6);\r\n
arrayList.Add('L');\r\n
foreach (var val in arrayList)\r\n
    Console.WriteLine(val);\r\n
```

```
562.613
Applied Data Structures
Quarter 2, 2018
6
L
```

562.613	Applied Data Structures	Quarter 2, 2018	6	L

ArrayList - Methods

Methods	Description
AddRange()	Add a collection in the ArrayList at the end
Insert()/InsertRange()	Insert one element/a collection at an index
Remove()/RemoveRange()	Remove specified one element/a range
RemoveAt()	Removes the element at the specified index
Sort()	Sort the elements of ArrayList
Reverse()	Reverse the order of elements
IndexOf()	Search specified element and return index, if found
Contains()	Check whether specified element is in ArrayList
Clear()	Remove all elements

SortedList – C# Collection

- Stores **key-value pairs** in the **ascending order** of key by default
- **SortedList** class implements **IDictionary** & **ICollection** interfaces
- Properties: Capacity, Count, …
- Methods: Add, Remove, Contains, Clear, **GetByIndex(index)**, **GetKey(index)**, **IndexOfKey(key)**, **IndexOfValue(value)**

SortedList - Adding and Printing

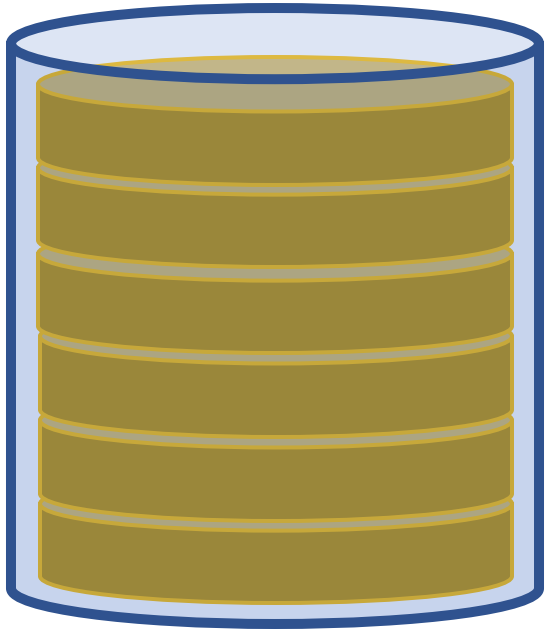
```
SortedList sortedList1 = new SortedList();
sortedList1.Add(3, "Three");\r\n
sortedList1.Add(4, "Four");\r\n
sortedList1.Add(1, "One");\r\n
sortedList1.Add(5, "Five");\r\n
sortedList1.Add(2, "Two");\r\n
```

```
foreach (DictionaryEntry de in sortedList1)\r\n
    Console.WriteLine("Key: {0} and Value: {1}",\
                      de.Key, de.Value);\r\n
for (int i = 0; i < sortedList1.Count; i++)\r\n
    Console.WriteLine("Value: {0}",\r\n
                      sortedList1.GetByIndex(i));
```

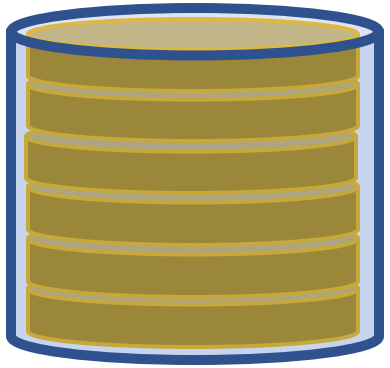
```
Key: 1 and Value: One
Key: 2 and Value: Two
Key: 3 and Value: Three
Key: 4 and Value: Four
Key: 5 and Value: Five
Value: One
Value: Two
Value: Three
Value: Four
Value: Five
```

Stack

- **Last-in-First-out (LIFO)** data structure
- **Push** – Adding an element
- **Pop** – Removing an element
- Always remove from **TOP**
- Always add from **TOP**
- Add and remove from the same end

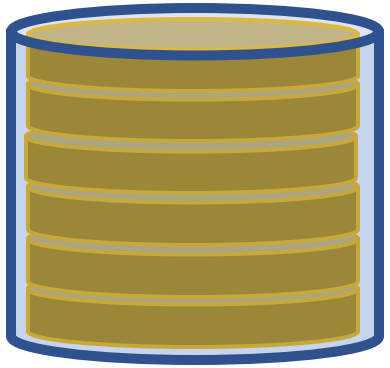


Stack – C# Collection

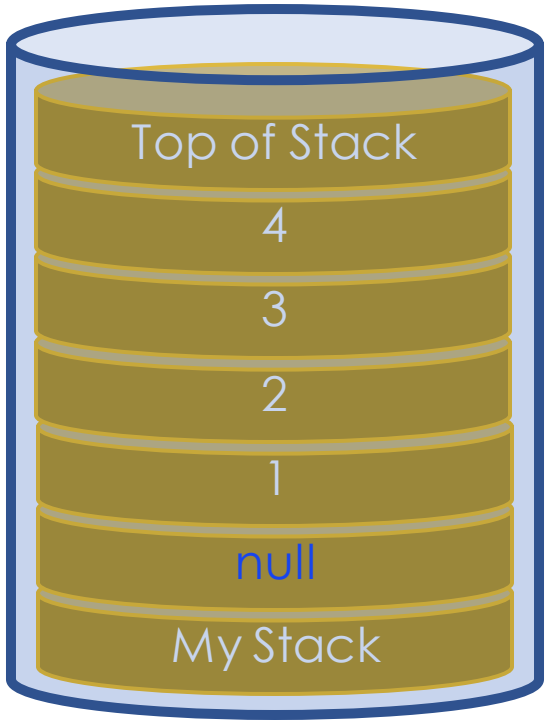


- Allows **null value** and also **duplicate values** of **different types**
- **Push()** – add an element
- **Pop()** or **Seek()** – retrieve an element
- Property: **Count** to show how many element in stack.
- Other Methods: **Clear()**, **Contains()**

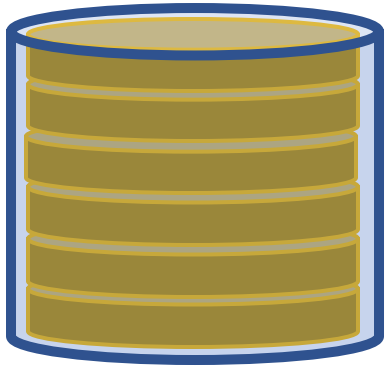
Stack - C# Collection



```
Stack stack = new Stack();\r
stack.Push("My Stack");\r\n
stack.Push(null);\r\n
stack.Push(1);\r\n
stack.Push(2);\r\n
stack.Push(3);\r\n
stack.Push(4);\r\n
stack.Push("Top of Stack");
```

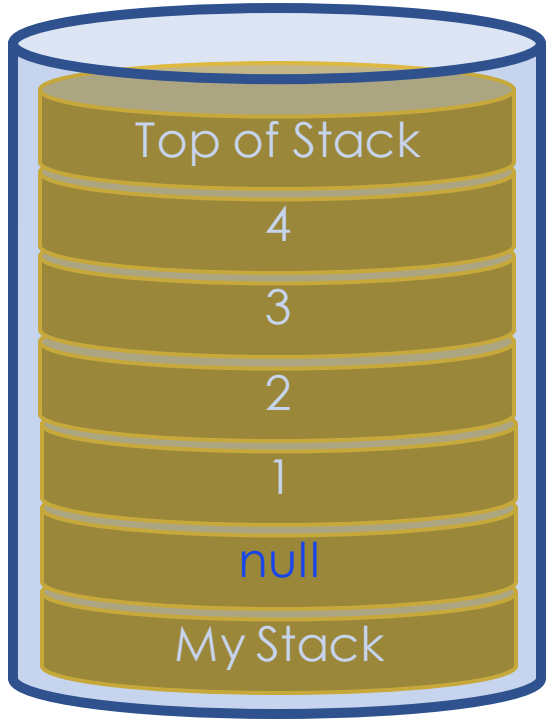


Stack - C# Collection



```
while (stack.Count != 0)\r\n
    Console.WriteLine(\r\n
        "Stack Value is {0}",
        stack.Pop());\r\n
```

Top of Stack
4
3
2
1
null
My Stack

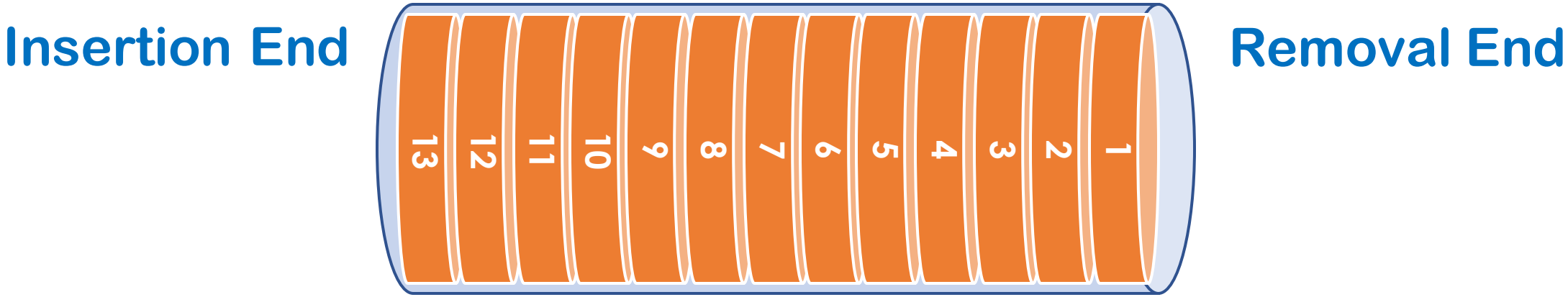


Stack – Applications

- **Reverse a word** – first push all characters, then pop them all
- **UNDO** – OS maintain the stack of user actions
- **Wearing/Removing Bangles**
- **Syntax Checking** in Programming Language
- **Recursive Stack** …

Queue

- First-in-First-out (FIFO) data structure
- EnQueue - add an element
- DeQueue - remove an element

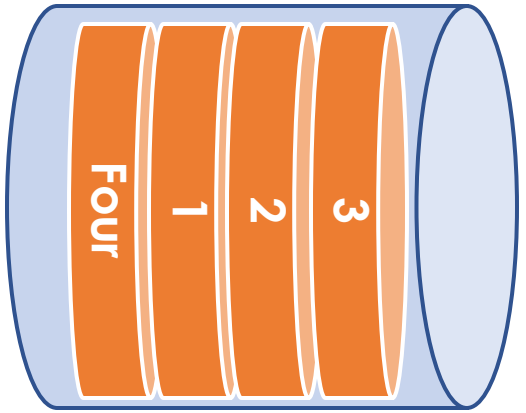


Queue – C# Collection

- Allows **null value** and also **duplicate values** of **different types**
- **Enqueue()** – add an element
- **Dequeue()** **Peek()**– retrieve an element
- Property **Count**: returns total element in queue
- Other Methods: **Clear**(), **Contains**()

Queue - C# Collection

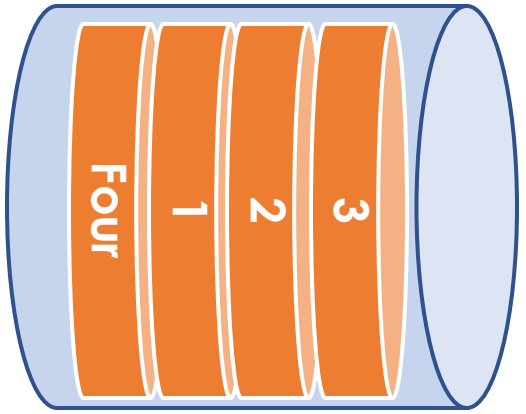
```
static void QueueTesting(){\r\n
    Queue queue = new Queue();
    queue.Enqueue(3);\r\n
    queue.Enqueue(2);\r\n
    queue.Enqueue(1);\r\n
    queue.Enqueue("Four");\r\n
```



Queue - C# Collection

```
Console.WriteLine(\r\n
    "Number of elements in the Queue: {0}",
    queue.Count);\r\n
```

Number of elements in the Queue: 4




```
while (queue.Count > 0)\r\n
    Console.WriteLine(queue.Dequeue());
```

3
2
1
Four

Generics

- **Type Parameters** – defer the specification of one or more types until the class or method is declared and instantiated by client code
- Generic type parameter **<T>**
- Specifies the **types** at compile time

```
public class myGenericClass<T>
```

```
public struct GenericRectangle<T>
```

Generics

Advantages

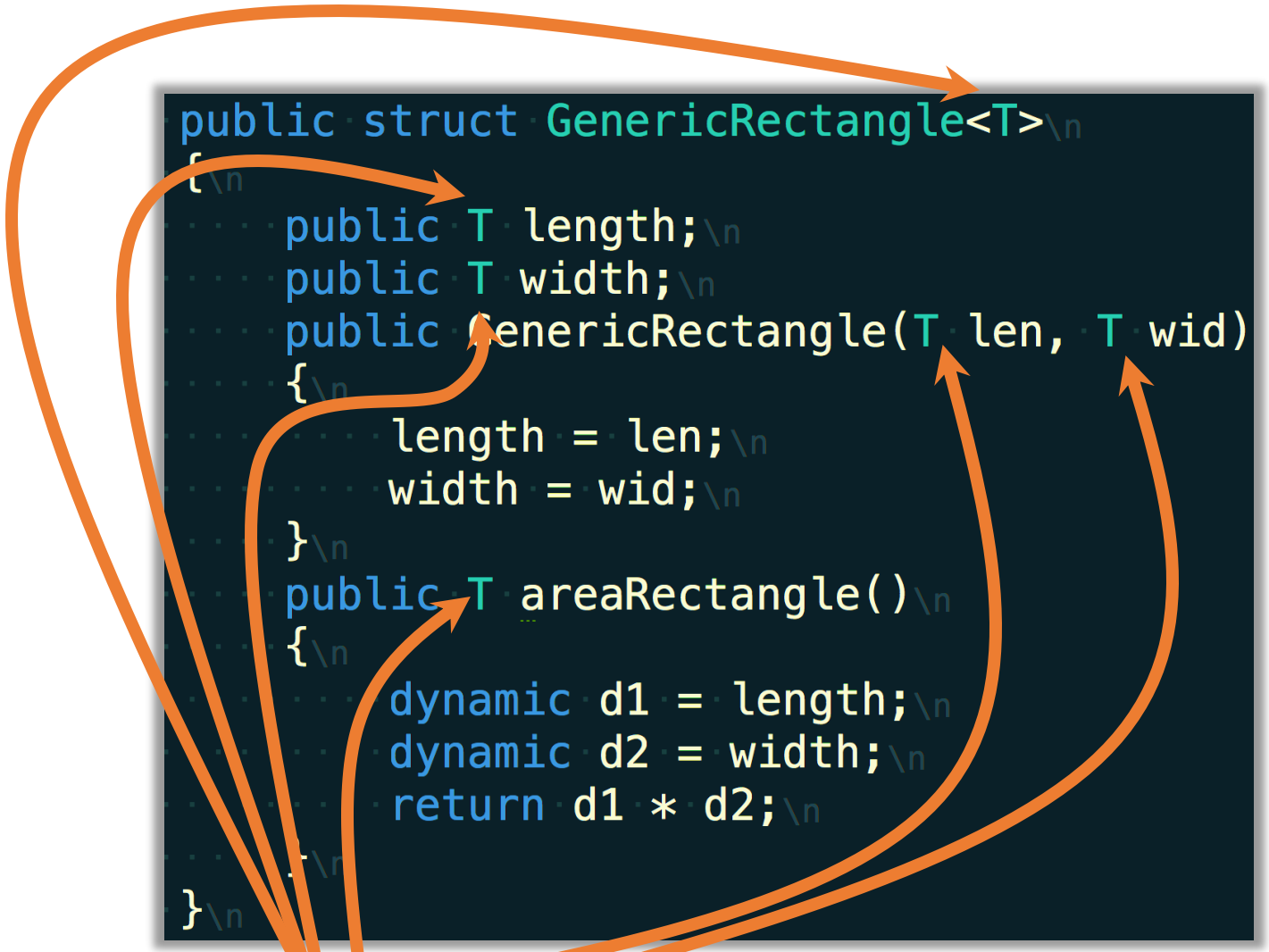
- Code reusability
- Type safety
- Performance

Can be Applied on

- → Interfaces
- → Abstract Class
- → Class
- → Method
- → Property
- → Structure
- → Event
- → Delegates
- → Operator

Generics

At **Compile** Time
Type Setting



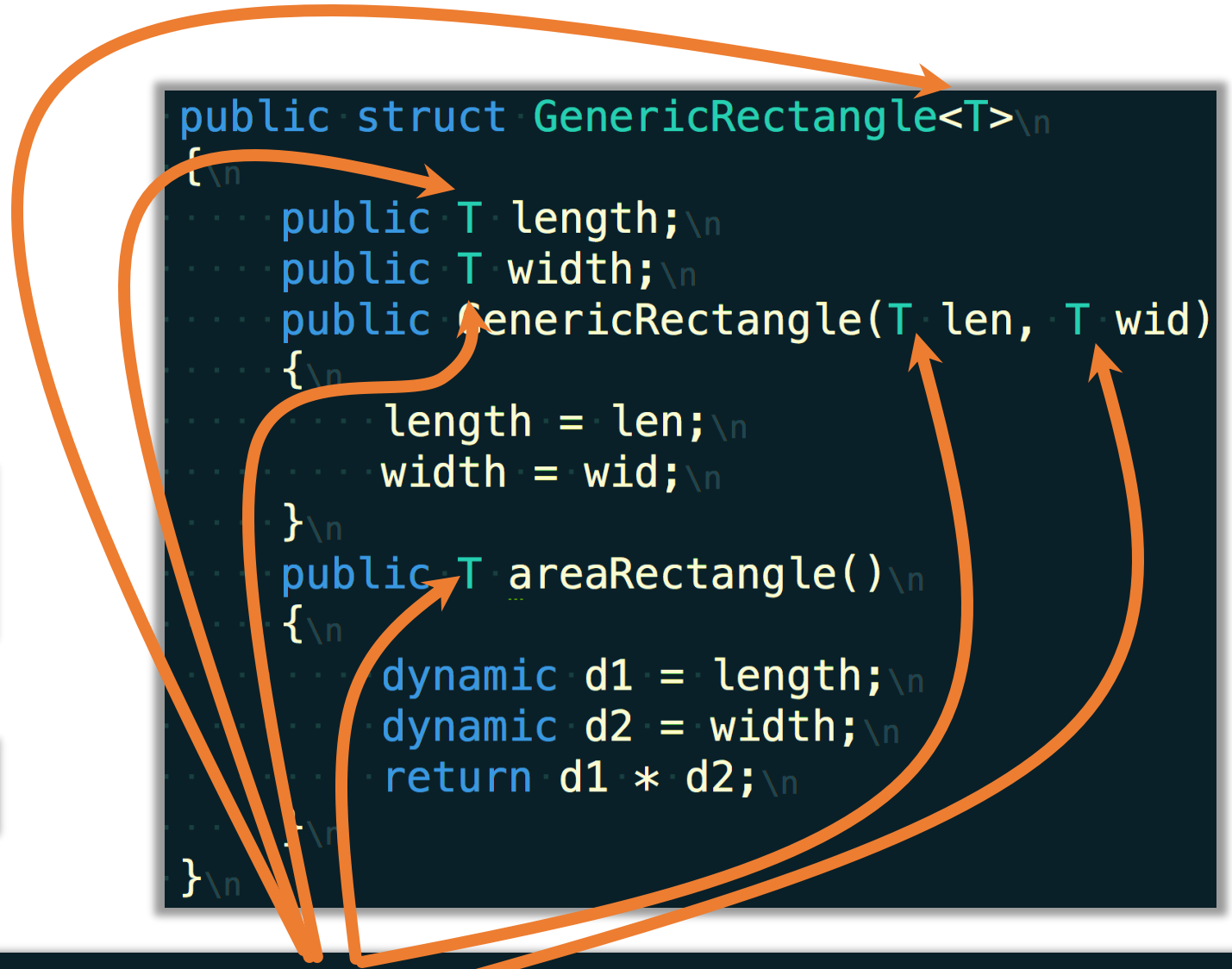
```
public struct GenericRectangle<T>
{
    public T length;
    public T width;
    public GenericRectangle(T len, T wid)
    {
        length = len;
        width = wid;
    }
    public T areaRectangle()
    {
        dynamic d1 = length;
        dynamic d2 = width;
        return d1 * d2;
    }
}
```

```
GenericRectangle<int> myGenericRectangle =
    new GenericRectangle<int>(3, 8);
```

Generics

At **Compile** Time
Type Setting

Code Reusability



```
public struct GenericRectangle<T>
{
    public T length;
    public T width;
    public GenericRectangle(T len, T wid)
    {
        length = len;
        width = wid;
    }
    public T areaRectangle()
    {
        dynamic d1 = length;
        dynamic d2 = width;
        return d1 * d2;
    }
}
```

```
GenericRectangle<double> myGenericRectangle1 =
    new GenericRectangle<double>(3.4, 8.7);
```

Generics - Constraints

- Specify which **type of placeholder** type with the generic class is **allowed**
- Compile time error - in case using type that is not allowed
- **where** is used to specify constraints

```
public class myGenericClass<T> where T: class
```

Only a reference type is allowed – no primitive types, stuct, etc.

Generics - Constraints

Constraint	Description
where T: struct	Type must be value type
where T: new()	Type must have public parameterless constructor
where T: <base class name>	Type must be or derive from the specified base class
where T: <interface name>	Type must be or implement the specified interface
where T: U	Type supplied for T must be or derive from the argument supplied for U

```
public class myGenericClass<T> where T: class where U:struct
```

Reference and Reading Material

- C# Tutorial: Link
- Struct: Link, Link
- Generics: Link, Link
- Collection: Link, Link
- ArrayList: Link
- SortedList: Link
- Stack: Link
- Queue: Link