

CS6004NI

Application Development

samyush

Samyush.maharjan@islingtoncollege.edu.np

C# Built-in Types

C# Guid

Guid is globally unique identifier (GUID), there is very low probability of being duplicated across all DB and computer.

Example:

```
1. using System;

2. // Generating a new Guid
3. Guid id = Guid.NewGuid();

4. // Converting existing Guid from string format
5. Guid itemId = new("2deeffbd-40ce-419f-9553-0cae39743c3b");

6. // Converting Guid to string value
7. string idStr = id.ToString(); // 33ea3e4a-adc5-4f45-b52a-36488100eb82

8. // Comparing Guids
9. if (id == itemId)
10. {
11.     Console.WriteLine("This will never be True.");
12. }
```

DateTime

The [DateTime](#) represents a combined date and time value for a fixed point in time and allows you to get the current time.

Example:

```
1. // Creating DateTime value
2. DateTime newYearEve = new(year: 2022, month: 12, day: 31);
3. DateTime newYearEve2 = new(year: 2022, month: 12, day: 31, hour: 23, minute: 59, second: 59);
4. DateTime newYearEve3 = DateTime.Parse("12/31/2022 23:59:59");

5. // Current DateTime value
6. DateTime nowLocal = DateTime.Now; // As determined by this computer
7. DateTime nowUtc = DateTime.UtcNow; // UTC = GMT = Z Time or Zulu Time
8. DateTime today = DateTime.Today; // Same as DateTime.Now but time is 00:00:00
9. DateTime tomorrow = today.AddDays(1);

10. if (today.Month == 4 && today.Day == 1)
11.     Console.WriteLine("April Fools!");

12. int result = DateTime.Compare(nowUtc, nowLocal); // -1
```

TimeSpan

The TimeSpan represents a length of time.

Example:

```
1. TimeSpan timeSpan1 = new TimeSpan(1, 30, 0); // 1 hour, 30 minutes, 0 seconds.
2. TimeSpan timeSpan2 = new TimeSpan(2, 12, 0, 0); // 2 days, 12 hours.
3. TimeSpan timeSpan3 = new TimeSpan(0, 0, 0, 0, 500); // 500 milliseconds.
4. TimeSpan timeSpan4 = new TimeSpan(10); // 10 "ticks" == 1 microsecond or 1000 nanoseconds

5. TimeSpan aLittleWhile = TimeSpan.FromSeconds(5);
6. TimeSpan quiteAWhile = TimeSpan.FromHours(1.5);
7. // The whole collection includes FromTicks, FromMilliseconds, FromSeconds, FromHours, and FromDays.

8. TimeSpan timeLeft = new TimeSpan(1, 30, 0);
9. Console.WriteLine($"{timeLeft.Days}d {timeLeft.Hours}h {timeLeft.Minutes}m {timeLeft.Seconds}s"); // 0d 1h 30m 0s
10. Console.WriteLine(timeLeft.TotalHours); // 1.5
11. Console.WriteLine(timeLeft.TotalMinutes); // 90

12. DateTime nextNewYear = new(2023, 1, 1);
13. TimeSpan newYearTimeLeft = nextNewYear - DateTime.Now; // 42.12:06:27.7876334
```


C# StringBuilder

The [StringBuilder](#) is the optimal way to write code when multiple string manipulation operations take place in code. StringBuilder hangs on to fragments of strings and does not assemble them into the final string until it is done.

Concatenation Example:

```
1. using System;

2. string[] todoList = {"Walk the dog.", "Buy some fruits.", "Pay the water bill."};
3. string today = DateTime.Today.ToShortDateString();
4. string printableTodo = $"Todo - {today}\n";
5. printableTodo += "=====\n";
6. foreach (var todo in todoList)
7. {
8.     printableTodo += $"[ ] {todo}\n";
9. }
10. Console.WriteLine(printableTodo);
```




```
Todo - 20/11/2022
=====
[ ] Walk the dog.
[ ] Buy some fruits.
[ ] Pay the water bill.
```

StringBuilder Example:

```
1. using System;
2. using System.Text;

3. string[] todoList = {"Walk the dog.", "Buy some fruits.", "Pay the water bill."};
4. string today = DateTime.Today.ToShortDateString();
5. StringBuilder printableTodo = new($"Todo - {today}\n");
6. printableTodo.AppendLine("=====");
7. foreach (var todo in todoList)
8. {
9.     printableTodo.AppendLine($"[ ] {todo}");
10.}

11.// Use ToString method to get the string
12.Console.WriteLine(printableTodo.ToString());
```



```
Todo - 20/11/2022
=====
[ ] Walk the dog.
[ ] Buy some fruits.
[ ] Pay the water bill.
```

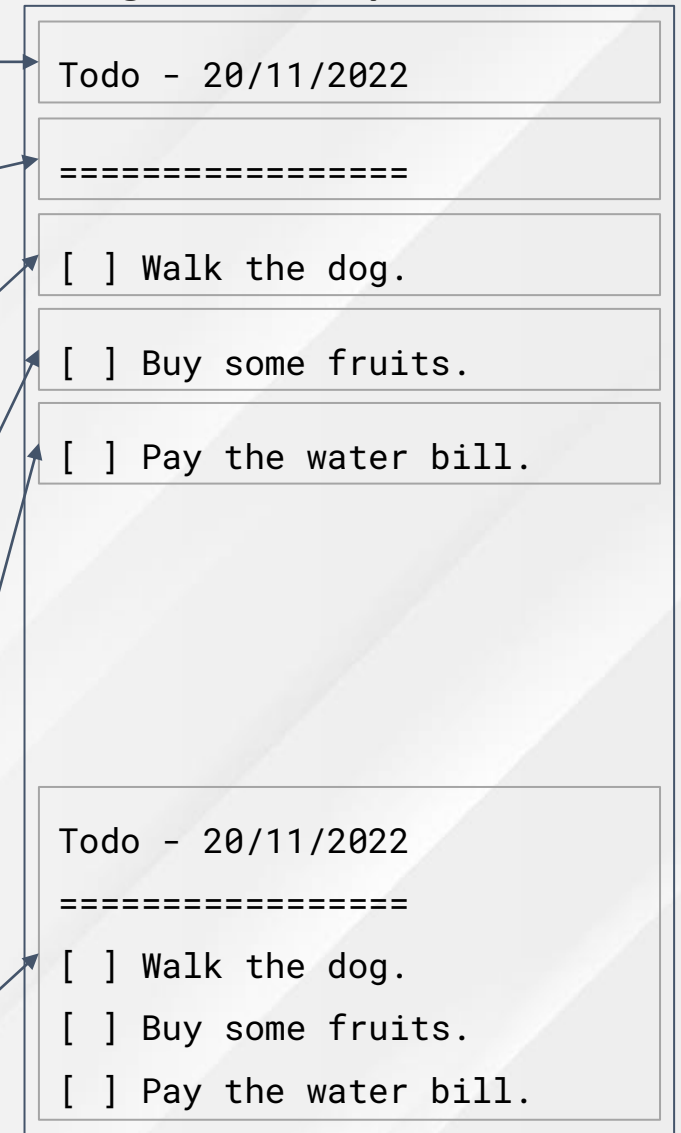
Concatenation Heap:



Execution:



StringBuilder Heap:



List<T>

List<T> is a popular and versatile generic collection. It provides methods to search, sort, and manipulate lists.

Example:

```
1. using System.Collections.Generic;

2. List<string> fruits = new(){ "apple", "banana", "pineapple", "durian" };
3. Console.WriteLine(fruits.Count); // 4
4. Console.WriteLine(fruits[2]); // pineapple
5. fruits[1] = "avocado"; // => replace an item in a list
6. fruits.Add("kiwi");
7. fruits.Insert(2, "orange");
```

List has other useful methods like **AddRange**, **InsertRange**, **Contains**, **IndexOf**, **Clear**, **Find**, **FindAll** etc.

Dictionary<TKey, TValue>

Dictionary<TKey, TValue> allows us to store a set of keys and values. The key must be unique.

Example:

```
1. using System.Collections.Generic;

2. Dictionary<string, string> acronym = new();
3. acronym.Add("ASCII", "American Standard Code for Information Interchange");
4. acronym["NASA"] = "National Aeronautics and Space Administration";

5. if (acronym.ContainsKey("BIOS"))
6.     Console.WriteLine(acronym["BIOS"]);
```

Dictionary has other useful methods like **Keys**, **Values**, **Clear**, **Count** etc.

C# Working with Files

C# Cross-platform filesystem

Paths are different for Windows, macOS, and Linux, so .NET offer some helper static properties and methods to determine the commonly used platform specific paths and directory separator.

```
1. using System;
2. using System.IO;

3. char directorySeparator = Path.DirectorySeparatorChar;
4. // Windows => '\'
5. // Mac => '/'

6. string currentDirectoryPath = Environment.CurrentDirectory;
7. // Windows => "D:\projects\ProjectName"
8. // Mac => "/Users/username/projects/ProjectName"

9. string appDataDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
10. // Windows => "C:\Users\username\AppData\Roaming"
11. // Mac => "/Users/username/.config"
```

C# Creating a Directory

Directory class provides the static methods to manage directories in the system's file system.

Example:

```
1. using System;
2. using System.IO;

3. string appDataDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
4. string appDataDirectory = Path.Combine(appDataDirectoryPath, "MyAppName");

5. if (!Directory.Exists(appDataDirectory))
6. {
7.     Directory.CreateDirectory(appDataDirectory);
8. }
```

C# Writing to a File

[File](#) class provides static methods for the creation, copying, deletion, moving, and opening of a single file.

There are various ways to write to a file like **WriteAllText**, **WriteAllLines**, **CreateText** (StreamWriter.WriteLine) method

Example:

```
1. using System;
2. using System.IO;

3. string appDataDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
4. string appDataDirectory = Path.Combine(appDataDirectoryPath, "MyAppName");
5. string todoListFile = Path.Combine(appDataDirectory, "todoList.txt");
6. string[] todoItems = {"Walk the dog.", "Buy some fruits.", "Pay the water bill."};

7. File.WriteAllLines(todoListFile, todoItems);.
```

C# Reading a File

There are various ways to write to a file like **ReadAllText**, **ReadAllLines**, **OpenText** (StreamReader.ReadLine) method

Example:

```
1. using System;
2. using System.IO;

3. string appDataDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
4. string appDataDirectory = Path.Combine(appDataDirectoryPath, "MyAppName");
5. string todoListFile = Path.Combine(appDataDirectory, "todoList.txt");

6. string[] todoItems = File.ReadAllLines(todoListFile);
7. Console.WriteLine(todoItems[0]); // Walk the dog.
```

C# Serializing with JSON

Serialization is the process of converting a live object into a sequence of bytes using a specified format. JSON (JavaScript Object Notation) is commonly used on web and mobile applications.

Example:

```
1. public class ToDoItem // Sample class
2. {
3.     public Guid Id { get; set; } = Guid.NewGuid();
4.     public string TaskName { get; set; }
5.     public bool IsDone { get; set; }
6.     public DateTime DueDate { get; set; }
7.     public DateTime CreatedAt { get; set; } = DateTime.Now;
8. }
```

Example JSON

```
1. {
2.     "Id": "151bdf68-b1f8-4d0c-ac83-85e597425482",
3.     "TaskName": "Pay Electricity bill.",
4.     "IsDone": false,
5.     "DueDate": "2022-11-26T21:50:24.215571+05:45",
6.     "CreatedAt": "2022-11-20T21:50:24.200151+05:45"
7. }
```



```
1. using System.Collections.Generic;
2. using System.IO;
3. using System.Text.Json;

4. string appDataDirectoryPath = Environment.GetFolderPath(Environment.SpecialFolder.ApplicationData);
5. string todoListFilePath = Path.Combine(appDataDirectoryPath, "MyAppName", "todoList.json");

6. List<ToDoItem> todoList = new() {
7.     new ToDoItem
8.     {
9.         TaskName = "Walk the dog.",
10.        DueDate = DateTime.Now.AddHours(2)
11.    },
12.    // ...
13. };

14. string todoListJson = JsonSerializer.Serialize(todoList);
15. File.WriteAllText(todoListFilePath, todoListJson);

16. // Deserialize
17. string jsonFromFile = File.ReadAllText(todoListFilePath);
18. List<ToDoItem> todoListFromJson = JsonSerializer.Deserialize<List<ToDoItem>>(jsonFromFile);
```

C# LINQ

Language Integrated Query

Querying and Manipulating Data Using LINQ

[LINQ](#) is a set of language extensions that add the ability to work with sequences of items and then filter, sort, and project them into different outputs.

Example:

```
1. using System.Collections.Generic;
2. using System.Linq;

3. List<ToDoItem> todoList = new() {
4.     new ToDoItem
5.     { TaskName = "Walk the dog.", DueDate = DateTime.Now.AddHours(2) },
6.     new ToDoItem
7.     { TaskName = "Buy some fruits.", DueDate = DateTime.Now.AddHours(5) },
8.     new ToDoItem
9.     { TaskName = "Pay the water bill.", DueDate = DateTime.Now.AddDays(1) }
10.};

11.todoList[0].IsDone = true;
```

Querying and Manipulating Data Using LINQ

- **Filter:** LINQ Where method filters a sequence of values based on a predicate.

```
14. List<ToDoItem> doneList = todoList.Where(x => x.IsDone).ToList();
```

```
15. List<ToDoItem> dueToday = todoList
```

```
16.     .Where(x => !x.IsDone && x.DueDate < DateTime.Today.AddDays(1)).ToList();
```

```
17. string searchTerm = "bill";
```

```
18. List<ToDoItem> searchResult = todoList.Where(x => x.TaskName.Contains(searchTerm)).ToList();
```

- **Sort:** LINQ OrderBy and OrderByDescending methods allows us to sort a list by one or more object properties.

```
21. List<ToDoItem> oldestTaskOnTop = todoList.OrderBy(x => x.CreatedAt).ToList();
```

```
22. List<ToDoItem> newestTaskOnTop = todoList.OrderByDescending(x => x.CreatedAt).ToList();
```

- **Project:** LINQ Select method allows us to projects each element of a sequence into a new form.

```
21. List<string> doneTaskNames = todoList.Where(x => x.IsDone).Select(x => x.TaskName).ToList();
```

Questions?