**EXPERIMENT NO: 06**

**Title**: String manipulation using String & String builder.

**Aim:** Study of String and StringBuilder.

**Theory:**

**String:**

In C#, string is a sequence of Unicode characters or array of characters. The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the text. So the string is the representation of the text. A string is an important concept and sometimes people get confused whether the string is a keyword or an object or a class. So let’s clear out this concept.

A string is represented by class System.String. The “string” keyword is an alias for System.String class and instead of writing System.String one can use String which is a shorthand for System.String class. So we can say string and String both can be used as an alias of System.String class. So string is an object of System.String class.

**Example:**

string s1 = “GeeksforGeeks”; // creating the string using string keyword

String s2 = “GFG”; // creating the string using String class

System.String s3 = “Pro Geek”; // creating the string using String class

The String class is defined in the .NET base class library. In other words a String object is a sequential collection of System.Char objects which represents a string. The maximum size of String object in memory is 2GB or about 1 billion characters. System.String class is immutable, i.e once created its state cannot be altered.

**Program:** To illustrate how to declare the string and initialize the string. Also, below program show the declaration and initialization of a string in a single line.

// C# program to declare string using

// string, String and System.String

// and initialization of string

using System;

class Geeks {

// Main Method

static void Main(string[] args)

{

// declare a string Name using

// "System.String" class

System.String Name;

// initialization of String

Name = "Geek";

// declare a string id using

// using an alias(shorthand)

// "String" of System.String

// class

String id;

// initialization of String

id = "33";

// declare a string mrk using

// string keyword

string mrk;

// initialization of String

mrk = "97";

// Declaration and initialization of

// the string in a single line

string rank = "1";

// Displaying Result

Console.WriteLine("Name: {0}", Name);

Console.WriteLine("Id: {0}", id);

Console.WriteLine("Marks: {0}", mrk);

Console.WriteLine("Rank: {0}", rank);

}

}

**Output:**

Name: Geek

Id: 33

Marks: 97

Rank: 1

**String Characteristics:**

* It is a reference type.
* It’s immutable( its state cannot be altered).
* It can contain nulls.
* It overloads the operator(==).

**Differences between String and System.String :**

The string is an alias for System.String. Both String and System.String means same and it will not affect the performance of the application. “string” is keyword in C#. So the main difference comes in the context, how to use these:

* The String is used for the declaration but System.String is used for accessing static string methods.
* The String is used to declare fields, properties etc. that it will use the predefined type System.String. It is the easy way to use.
* The String has to use the System.String class methods, such as String.SubString, String.IndexOf etc. The string is only an alias of System.String.

Note: In .NET, the text is stored as a sequential collection of the Char objects so there is no null-terminating character at the end of a C# string. Therefore a C# string can contain any number of embedded null characters (‘\0’).

**String arrays:** We can also create the array of string and assigns values to it. The string arrays can be created as follows:

**Syntax:**

String [] array\_variable = new String[Length\_of\_array]

**Example:** To illustrate the creation of string arrays and assigning values to it

// C# program for an array of strings

using System;

class Geeks {

// Main Method

static void Main(string[] args)

{

String[] str\_arr = new String[3];

// Initialising the array of strings

str\_arr[0] = "Geeks";

str\_arr[1] = "For";

str\_arr[2] = "Geeks";

// printing String array

for(int i = 0; i < 3; i++)

{

Console.WriteLine("value at Index position "+i+" is "+str\_arr[i]);

}

}

}

**Output:**

value at Index position 0 is Geeks

value at Index position 1 is For

value at Index position 2 is Geeks

**Reading String from User-Input:** A string can be read out from the user input. ReadLine() method of console class is used to read a string from user input.

**Example:**

// C# program to demonstrate Reading

// String from User-Input

using System;

class Geeks {

// Main Method

static void Main(string[] args)

{

Console.WriteLine("Enter the String");

// Declaring a string object read\_user

// and taking the user input using

// ReadLine() method

String read\_user = Console.ReadLine();

// Displaying the user input

Console.WriteLine("User Entered: " + read\_user);

}

}

**Input:**

Hello Geeks !

**Output:**

Enter the String

User Entered: Hello Geeks !

**Different Ways for Creating a String:**

* Create a string from a literal
* Create a string using concatenation
* Create a string using a constructor
* Create a string using a property or a method
* Create a string using formatting

Create a string from a literal: It is the most common way to create a string. In this, a user has to define string variable and then assign the value within the double quotes. We can use any type of characters within double quotes except some special character like a backslash (\).

**Program:** To illustrate the string creation using literals

// C# program to demonstrate the

// string creation using literals

using System;

class Geeks {

// Main Method

static void Main(string[] args)

{

string str1 = "GeeksforGeeks";

Console.WriteLine(str1);

// Give Error Unrecognized escape sequence \H, \G, \p

// string str3 = "X:\Home\GFG\Geeks.cs";

// Console.WriteLine(str3);

// using double slash \\

string str2 = "X:\\Home\\GFG\\program.cs";

Console.WriteLine(str2);

}

}

**Output:**

GeeksforGeeks

X:\Home\GFG\program.cs

**Create a string using concatenation:** We can create a string by using string concatenation operator “+” in C#. To create a single string from any combination of String instances and string literals, the string concatenation operator (+) is used to combine or merge one or more string.

**Program:** To illustrates the use of the string concatenation operator

// C# program to demonstrate the use of

// the string concatenation operator.

using System;

class Geeks {

// Main Method

public static void Main()

{

string s1 = "Geek";

string s2 = "s";

string s3 = "For";

string s4 = "Geek";

// using concatenation operator

string str = s1 + s2 + s3 + s4 + "s";

Console.WriteLine(str);

}

}

**Output:**

GeeksForGeeks

**Create a string using Constructor:** The String class has been several overloaded constructors which take an array of characters or bytes. Some of the constructors include pointers to character arrays or signed byte arrays as parameters.

**Program:** To illustrates Creation of a string using the constructor

// C# program to demonstrate the creation

// of string using the constructor

using System;

class Geeks {

// Main Method

public static void Main()

{

char[] chars = { 'G', 'E', 'E', 'K', 'S' };

// Create a string from a character array.

string str1 = new string(chars);

Console.WriteLine(str1);

// Create a string that consists of

// a character repeated 20 times.

string str2 = new string('G', 10);

Console.WriteLine(str2);

/\* below comment part give the error

for unsafe mode go through offline

sbyte[] bytes = { 0x41, 0x42, 0x43,

0x44, 0x45, 0x00 };

string stringtoBytes = null;

string stringtomChars = null;

unsafe

{

fixed (sbyte\* pbytes = bytes)

{

// Create a string from a pointer

// to a signed byte array.

stringFromBytes = new string(pbytes);

}

fixed (char\* pchars = chars)

{

// Create a string from a pointer

// to a character array.

stringFromChars = new string(pchars);

}

}

Console.WriteLine(stringtoBytes); // output : ABCDE

Console.WriteLine(stringtoChars); // output : GEEKS \*/

}

}

Output:

GEEKS

GGGGGGGGGG

**Create a string using a Property or a Method:** To retrieving a property or calling a method which always returns a string. For example, using methods of the String class to extract a substring from a larger string.

**Program:** To illustrate the Creation of a string using a Property or a Method

// C# program to extract a substring from a larger

// string using methods of the String class

using System;

class Geeks {

// Main Method

public static void Main()

{

string sentence = "Geeks For Geeks";

// Extract the second word.

// taking the first space position value

int startpos = sentence.IndexOf(" ") + 1;

// taking the second space position value

int endpos = sentence.IndexOf(" ", startpos) - startpos;

// now extract second word from the sentence

string wrd = sentence.Substring(startpos, endpos);

Console.WriteLine(wrd);

}

}

**Output:**

For

**Create a string using Format:** The “Format” method is used to convert the value or object to its string representation. The String.Format method returns a string.

**Program:** To illustrate the creation of string using Format method

// C# method to illustrate the creation

// of string using format method

using System;

class Geeks {

// Main Method

public static void Main()

{

int no = 10;

string cname = "BMW";

string clr = "Red";

// string creation using string.Format method

string str = string.Format("{0} {1} Cars color " +

"are {2}", no.ToString(), cname, clr);

Console.WriteLine(str);

}

}

**Output:**

10 BMW Cars color are Red

**String Class Properties:** The String class has two properties as follows:

* Chars: It is used to get the Char object at a specified position in the current String object.
* Length: It is used to get the number of characters in the current String object. To know more about the string class properties please go to String Properties in C#.

**String class:**

In C#, a string is a sequence of Unicode characters or array of characters.

The range of Unicode characters will be U+0000 to U+FFFF. The array of characters is also termed as the text. So the string is the representation of the text. A string is represented by a class System.String.

The String class is defined in the .NET base class library. In other words, a String object is a sequential collection of System.Char objects which represent a string. The maximum size of the String object in memory can be 2GB or about 1 billion characters.

**Characteristics of String Class:**

* The System.String class is immutable, i.e once created its state cannot be altered.
* With the help of length property, it provides the total number of characters present in the given string.
* String objects can include a null character which counts as the part of the string’s length.
* It provides the position of the characters in the given string.
* It allows empty strings. Empty strings are the valid instance of String objects that contain zero characters.
* A string that has been declared but has not been assigned a value is null. Attempting to call methods on that string throws a NullReferenceException.
* It also supports searching strings, comparison of string, testing of equality, modifying the string, normalization of string, copying of strings, etc.
* It also provides several ways to create strings like using a constructor, using concatenation, etc.
* **Constructor**

|  |  |
| --- | --- |
| Constructor | Description |
| **String(Char\*)** | Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters. |
| **String(Char\*, Int32, Int32)** | Initializes a new instance of the String class to the value indicated by a specified pointer to an array of Unicode characters, a starting character position within that array, and a length. |
| **String(Char, Int32)** | Initializes a new instance of the String class to the value indicated by a specified Unicode character repeated a specified number of times. |
| **String(Char[])** | Initializes a new instance of the String class to the value indicated by an array of Unicode characters. |
| **String(Char[], Int32, Int32)** | Initializes a new instance of the String class to the value indicated by an array of Unicode characters, a starting character position within that array, and a length. |
| **String(SByte\*)** | Initializes a new instance of the String class to the value indicated by a pointer to an array of 8-bit signed integers. |
| **String(SByte\*, Int32, Int32)** | Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, and a length. |
| **String(SByte\*, Int32, Int32, Encoding)** | Initializes a new instance of the String class to the value indicated by a specified pointer to an array of 8-bit signed integers, a starting position within that array, a length, and an Encoding object. |

**Example:**

// C# program to demonstrate the creation

// of string using the constructor

using System;

class Geeks {

// Main Method

public static void Main()

{

char[] chars = { 'G', 'E', 'E', 'K', 'S' };

// Create a string from a character array.

string str1 = new string(chars);

Console.WriteLine(str1);

// Create a string that consists of

// a character repeated 5 times.

string str2 = new string('E', 5);

Console.WriteLine(str2);

}

}

Output:

GEEKS

EEEEE

#### Properties:

|  |  |
| --- | --- |
| Property | Description |
| **Chars[Int32]** | Gets the Char object at a specified position in the current String object. |
| **Length** | Gets the number of characters in the current String object. |

**Example:**

// C# program to demonstrate the

// String Class Properties

using System;

class Geeks {

// Main Method

public static void Main()

{

string str = "GeeksforGeeks";

// using Chars[Int32] & Length property

for (int i = 0; i <= str.Length - 1; i++)

Console.Write("{0} ", str[i]);

}

}

**Output:**

G e e k s f o r G e e k s

**Methods:**

|  |  |
| --- | --- |
| Method | Description |
| **Clone()** | Returns a reference to this instance of String. |
| **Compare()** | Used to compare the two string objects. |
| **CompareOrdinal(String, Int32, String, Int32, Int32)** | Compares substrings of two specified String objects by evaluating the numeric values of the corresponding Char objects in each substring. |
| **CompareOrdinal(String, String)** | Compares two specified String objects by evaluating the numeric values of the corresponding Char objects in each string. |
| **CompareTo()** | Compare the current instance with a specified Object or String object. |
| **Concat()** | Concatenates one or more instances of String, or the String representations of the values of one or more instances of Object. |
| **Contains(String)** | Returns a value indicating whether a specified substring occurs within this string. |
| **Copy(String)** | Creates a new instance of String with the same value as a specified String. |
| **CopyTo(Int32, Char[], Int32, Int32)** | Copies a specified number of characters from a specified position in this instance to a specified position in an array of Unicode characters. |
| **EndsWith()** | Determines whether the end of this string instance matches a specified string. |
| **Equals()** | Determines whether two String objects have the same value. |
| **Format()** | Converts the value of objects to strings based on the formats specified and inserts them into another string. |
| **GetEnumerator()** | Retrieves an object that can iterate through the individual characters in this string. |
| **GetHashCode()** | Returns the hash code for this string. |
| **GetType()** | Gets the Type of the current instance. (Inherited from Object) |
| **GetTypeCode()** | Returns the TypeCode for class String. |
| **IndexOf()** | Reports the zero-based index of the first occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance. |
| **IndexOfAny()** | Reports the index of the first occurrence in this instance of any character in a specified array of Unicode characters. The method returns -1 if the characters in the array are not found in this instance. |
| **Insert(Int32, String)** | Returns a new string in which a specified string is inserted at a specified index position in this instance. |
| **Intern(String)** | Retrieves the system’s reference to the specified String. |
| **IsInterned(String)** | Retrieves a reference to a specified String. |
| **IsNormalized()** | Indicates whether this string is in a particular Unicode normalization form. |
| **IsNullOrEmpty(String)** | Indicates whether the specified string is null or an Empty string. |
| **IsNullOrWhiteSpace(String)** | Indicates whether a specified string is null, empty, or consists only of white-space characters. |
| **Join()** | Concatenates the elements of a specified array or the members of a collection, using the specified separator between each element or member. |
| **LastIndexOf()** | Reports the zero-based index position of the last occurrence of a specified Unicode character or string within this instance. The method returns -1 if the character or string is not found in this instance. |
| **MemberwiseClone()** | Creates a shallow copy of the current Object. (Inherited from Object) |
| **Normalize()** | Returns a new string whose binary representation is in a particular Unicode normalization form. |
| **PadLeft()** | Returns a new string of a specified length in which the beginning of the current string is padded with spaces or with a specified Unicode character. |
| **PadRight()** | Returns a new string of a specified length in which the end of the current string is padded with spaces or with a specified Unicode character. |
| **Remove()** | Returns a new string in which a specified number of characters from the current string are deleted. |
| **Replace()** | Returns a new string in which all occurrences of a specified Unicode character or String in the current string are replaced with another specified Unicode character or String. |
| **Split()** | Returns a string array that contains the substrings in this instance that are delimited by elements of a specified string or Unicode character array. |
| **StartsWith(String)** | Determines whether the beginning of this string instance matches a specified string. |
| **Substring(Int32)** | Retrieves a substring from this instance. |
| **ToCharArray()** | Copies the characters in this instance to a Unicode character array. |
| **ToLower()** | Returns a copy of this string converted to lowercase. |
| **ToLowerInvariant()** | Returns a copy of this String object converted to lowercase using the casing rules of the invariant culture. |
| **ToString()** | Converts the value of this instance to a String. |
| **ToUpper()** | Returns a copy of this string converted to uppercase. |
| **ToUpperInvariant()** | Returns a copy of this String object converted to uppercase using the casing rules of the invariant culture. |
| **Trim()** | Returns a new string in which all leading and trailing occurrences of a set of specified characters from the current String object are removed. |
| **TrimEnd(Char[])** | Removes all trailing occurrences of a set of characters specified in an array from the current String object. |
| **TrimStart(Char[])** | Removes all leading occurrences of a set of characters specified in an array from the current String object. |

**Example:**

// C# program to illustrate

// String class methods

using System;

class GFG {

static void copymethod()

{

string str1 = "GeeksforGeeks";

string str2 = "geeks";

Console.WriteLine("Original Strings: str1 = "

+ "'{0}' and str2 ='{1}'",

str1, str2);

Console.WriteLine("");

Console.WriteLine("After Copy method");

Console.WriteLine("");

// using the Copy method

// to copy the value of str1

// into str2

str2 = String.Copy(str1);

Console.WriteLine("Strings are str1 = "

+ "'{0}' and str2='{1}'",

str1, str2);

}

// Main method

static public void Main()

{

// variables

string str1 = "geeksforgeeks";

string str2 = "geeksforgeeks";

bool result;

// Compare(string, string) method return true

// because the given strings are equal

result = String.Compare(str1, str2) == 0;

Console.WriteLine("Result of Compare Method: " +result);

// calling method

copymethod();

}

}

**Output:**

Result of Compare Method: True

Original Strings: str1 = 'GeeksforGeeks' and str2 ='geeks'

After Copy method

Strings are str1 = 'GeeksforGeeks' and str2='GeeksforGeeks'

#### Operators:

|  |  |
| --- | --- |
| Operator | Description |
| **Equality(String, String)** | Determines whether two specified strings have the same value. |
| **Inequality(String, String)** | Determines whether two specified strings have different values. |

**Example:**

// C# program to illustrate the

// Equality and Inequality operator

using System;

class GFG {

// Main Method

public static void Main(string[] args)

{

string s1 = "WelcomeToGeeks";

string s2 = "WelcomeToGeeks";

bool result1, result2;

// Equality operator return true

// as both string are equal

result1 = s1 == s2;

// Inequality operator return false

// as both string are equal

result2 = s1 != s2;

Console.WriteLine(result1);

Console.WriteLine(result2);

}

}

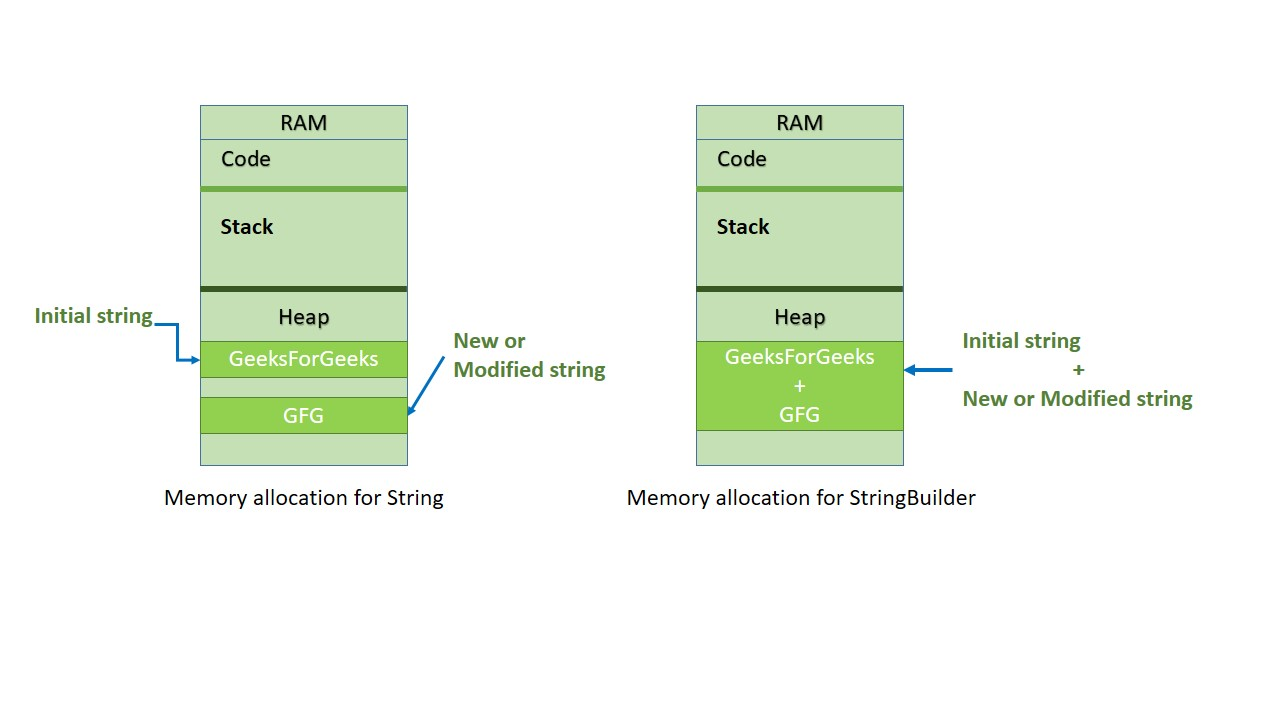
Output:

True

False

**StringBuilder in C#:**

C# StringBuilder is similar to Java StringBuilder. A String object is immutable, i.e. a String cannot be changed once created. Every time when you use any of the methods of the System.String class, then you create a new string object in memory. For example, a string “GeeksForGeeks” occupies memory in the heap, now, by changing the initial string “GeeksForGeeks” to “GFG” will create a new string object on the memory heap instead of modifying the initial string at the same memory location. In situations where you need to perform repeated modifications to a string, we need StringBuilder class. To avoid string replacing, appending, removing or inserting new strings in the initial string C# introduce StringBuilder concept. StringBuilder is a dynamic object. It doesn’t create a new object in the memory but dynamically expands the needed memory to accommodate the modified or new string.



**Declaration and Initialization of StringBuilder:**

StringBuilder can be declared and initialized the same way as class,

StringBuilder s = new StringBuilder();

or

StringBuilder s = new StringBuilder("GeeksforGeeks");

“s” is the object of StringBuilder class. Also, we can pass a string value(here “GeeksforGeeks”) as an argument to the constructor of StringBuilder.

**Defining the capacity of StringBuilder:**

Although the StringBuilder is a dynamic object that allows you to expand the number of characters in the string that it encapsulates, you can specify a value for the maximum number of characters that it can hold. This value is called the capacity of the StringBuilder object.

StringBuilder s = new StringBuilder(20);

or

StringBuilder s = new StringBuilder("GeeksForGeeks", 20);

Here,

* In the 1st statement we pass an integer value as an argument to the constructor. This is the maximum capacity of character that can hold a string.
* In the 2nd statement we pass string value with an integer value (that is the maximum capacity of character a string can hold) as an argument to the constructor.

**Important Methods of StringBuilder Class:**

* Append(string value)
* AppendFormat()
* Insert(int index, string value)
* Remove(int start, int length)
* Replace(old\_val, new|\_val)
* StringBuilder.Append(string value) Method

The Append method can be used to add or append a string value of an object to the end of a string represented by the current StringBuilder object. AppendLine() method also come under this method. This method append the string with a newline at the end.

**Example:**

// C# program to demonstrate the

// StringBuilder.Append(value) and

// StringBuilder.AppendLine(value) method

using System;

using System.Text;

class GFG {

// Main Method

public static void Main()

{

// "20" is capacity

StringBuilder s = new StringBuilder("HELLO ", 20);

s.Append("GFG");

// after printing "GEEKS"

// a new line append

s.AppendLine("GEEKS");

s.Append("GeeksForGeeks");

Console.WriteLine(s);

}

}

**Output:**

HELLO GFGGEEKS

GeeksForGeeks

**StringBuilder.AppendFormat()**

This method uses to format the input string into the specified format and then append it. This method also appends text to the end of the StringBuilder object.

// C# program to demonstrate the

// StringBuilder.AppendFormat() method

using System;

using System.Text;

class GFG {

// Main Method

public static void Main()

{

StringBuilder s = new StringBuilder("Your total amount is ");

// using the method

s.AppendFormat("{0:C} ", 50);

Console.WriteLine(s);

}

}

**Output:**

Your total amount is Â¤50.00

**StringBuilder.Insert(int index, string value)**

This method inserts the string at specified index in StringBuilder object.

**Example:**

// C# program to demonstrate the

// StringBuilder.Insert(int index,

// string value) method

using System;

using System.Text;

class GFG {

// Main Method

public static void Main()

{

// "20" is capacity

StringBuilder s = new StringBuilder("HELLO ", 20);

// "GEEKS" insert after 6th index

s.Insert(6, "GEEKS");

Console.WriteLine(s);

}

}

**Output:**

HELLO GEEKS

**StringBuilder.Remove(int start, int length)**

This method removes the specified number of characters from the current StringBuilder object. The removing process beginning at a specified index and extends up to another specified index.

**Example:**

// C# program to demonstrate the

// StringBuilder.Remove(int index,

// int length) method

using System;

using System.Text;

class GFG {

// Main Method

public static void Main()

{

// "20" is capacity

StringBuilder s = new StringBuilder("GeeksForGeeks", 20);

// remove starts from index 5

// and remove happes 3 index

// after index 5

s.Remove(5, 3);

Console.WriteLine(s);

}

}

**Output:**

GeeksGeeks

**StringBuilder.Replace(old\_val, new\_val)**

This method is used to replace characters within the StringBuilder object with another specified character.

**Example:**

// C# program to demonstrate the

// StringBuilder.Replace(string old\_val,

// string new\_val) method

using System;

using System.Text;

class GFG {

// Main Method

public static void Main()

{

// "20" is capacity

StringBuilder s = new StringBuilder("GFG Geeks ", 20);

// Replace "GFG" with "Geeks For"

s.Replace("GFG", "Geeks For");

Console.WriteLine(s);

}

}

**Output:**

Geeks For Geeks

**String vs StringBuilder:**

StringBuilder is used to represent a mutable string of characters. Mutable means the string which can be changed. So String objects are immutable but StringBuilder is the mutable string type. It will not create a new modified instance of the current string object but do the modifications in the existing string object. The complete functionality of StringBuilder is provided by StringBuilder class which is present in System.Text namespace.

**Need of the StringBuilder:** As stated above that the String class objects are immutable which means that if the user will modify any string object it will result into the creation of a new string object. It makes the use of string costly. So when the user needs the repetitive operations on the string then the need of StringBuilder come into existence. It provides the optimized way to deal with the repetitive and multiple string manipulation operations.

**Example:**

// C# program to demonstrate the

// difference between String,

// StringBuilder

using System;

using System.Text;

using System.Collections;

class GFG {

// Concatenates to String

public static void concat1(String s1)

{

// taking a string which

// is to be Concatenate

String st = "forGeeks";

// using String.Concat method

// you can also replace it with

// s1 = s1 + "forgeeks";

s1 = String.Concat(s1, st);

}

// Concatenates to StringBuilder

public static void concat2(StringBuilder s2)

{

// using Append method

// of StringBuilder class

s2.Append("forGeeks");

}

// Main Method

public static void Main(String[] args)

{

String s1 = "Geeks";

concat1(s1); // s1 is not changed

Console.WriteLine("Using String Class: " + s1);

StringBuilder s2 = new StringBuilder("Geeks");

concat2(s2); // s2 is changed

Console.WriteLine("Using StringBuilder Class: " + s2);

}

}

**Output:**

Using String Class: Geeks

Using StringBuilder Class: GeeksforGeeks

**Explanation:**

**Use of concat1 Method:** In this method, we are passing a string “Geeks” and performing “s1 = String.Concat(s1, st);” where st is “forGeeks” to be concatenated. The string passed from Main() is not changed, this is due to the fact that String is immutable. Altering the value of string creates another object and s1 in concat1() stores reference of the new string. But the references s1 in Main() and concat1() refer to different strings.

**Use of concat2 Method:** In this method, we are passing a string “Geeks” and performing “s2.Append(“forGeeks”)” which changes the actual value of the string (in Main) to “GeeksforGeeks”. This is due to the simple fact that StringBuilder is mutable and hence changes its value.

**When to use which one:**

* If a string is going to remain constant throughout the program, then use String class object because a String object is immutable.
* If a string can change (example: lots of logic and operations in the construction of the string) then using a StringBuilder is the best option.

**Problem Statement:**

Student Record Management

**Conclusion:**

**Sample Questions:**

1) What is String?

2) Explain difference between String and StringBuilder?

Problem statement code is here don’t write this code execute it

public class StudentRecordSystem

{

private string[] studentNames;d

private int[] studentAges;

private string[] deptNames;

private int totalStudents;

private const int MAX\_STUDENTS = 100; // Maximum number of students

public StudentRecordSystem()

{

studentNames = new string[MAX\_STUDENTS];

studentAges = new int[MAX\_STUDENTS];

deptNames = new string[MAX\_STUDENTS];

totalStudents = 0;

}

public void AddStudent(string name, int age, string gradeLevel)

{

if (totalStudents < MAX\_STUDENTS)

{

studentNames[totalStudents] = name;

studentAges[totalStudents] = age;

deptNames[totalStudents] = gradeLevel;

totalStudents++;

Console.WriteLine("Student added successfully.");

}

else

{

Console.WriteLine("Maximum number of students reached.");

}

}

public void UpdateStudent(string name, int age, string gradeLevel)

{

int index = FindStudentIndex(name);

if (index != -1)

{

studentAges[index] = age;

deptNames[index] = gradeLevel;

Console.WriteLine("Student record updated successfully.");

}

else

{

Console.WriteLine("Student not found.");

}

}

public void DeleteStudent(string name)

{

int index = FindStudentIndex(name);

if (index != -1)

{

for (int i = index; i < totalStudents - 1; i++)

{

studentNames[i] = studentNames[i + 1];

studentAges[i] = studentAges[i + 1];

deptNames[i] = deptNames[i + 1];

}

totalStudents--;

Console.WriteLine("Student record deleted successfully.");

}

else

{

Console.WriteLine("Student not found.");

}

}

public void SearchStudent(string name)

{

int index = FindStudentIndex(name);

if (index != -1)

{

Console.WriteLine("Name: {studentNames[index]}, Age: {studentAges[index]}, Grade Level: {studentGradeLevels[index]}");

}

else

{

Console.WriteLine("Student not found.");

}

}

public string GetAllStudentsInfo()

{

if (totalStudents == 0)

{

return "No students found.";

}

string allStudentsInfo = "";

for (int i = 0; i < totalStudents; i++)

{

allStudentsInfo += "Name:" + studentNames[i] + ", Age: " + studentAges[i] + ", department name: " + deptNames[i] + "\n";

}

return allStudentsInfo;

}

private int FindStudentIndex(string name)

{

for (int i = 0; i < totalStudents; i++)

{

if (studentNames[i] == name)

{

return i;

}

}

return -1; // Student not found

}

}

class Program

{

static void Main(string[] args)

{

StudentRecordSystem recordSystem = new StudentRecordSystem();

char choice;

do

{

Console.WriteLine("\nMenu:");

Console.WriteLine("1. Add Student");

Console.WriteLine("2. Update Student");

Console.WriteLine("3. Delete Student");

Console.WriteLine("4. Search Student");

Console.WriteLine("5. Display All Students");

Console.WriteLine("6. Exit");

Console.Write("Enter your choice: ");

choice = Console.ReadKey().KeyChar;

Console.WriteLine();

switch (choice)

{

case '1':

Console.Write("Enter name: ");

string name = Console.ReadLine();

Console.Write("Enter age: ");

int age = int.Parse(Console.ReadLine());

Console.Write("Enter Department name: ");

string gradeLevel = Console.ReadLine();

recordSystem.AddStudent(name, age, gradeLevel);

break;

case '2':

Console.Write("Enter name of student to update: ");

string nameUpdate = Console.ReadLine();

Console.Write("Enter age: ");

int ageUpdate = int.Parse(Console.ReadLine());

Console.Write("Enter Department name: ");

string gradeLevelUpdate = Console.ReadLine();

recordSystem.UpdateStudent(nameUpdate, ageUpdate, gradeLevelUpdate);

break;

case '3':

Console.Write("Enter name of student to delete: ");

string nameDelete = Console.ReadLine();

recordSystem.DeleteStudent(nameDelete);

break;

case '4':

Console.Write("Enter name of student to search: ");

string searchName = Console.ReadLine();

recordSystem.SearchStudent(searchName);

break;

case '5':

Console.WriteLine("All Students:");

Console.WriteLine(recordSystem.GetAllStudentsInfo());

break;

case '6':

Console.WriteLine("Exiting...");

break;

default:

Console.WriteLine("Invalid choice. Please enter a valid option.");

break;

}

} while (choice != '6');

}

}