// Displaying Linked List Employee objects

// iterate over the list by using the nodes

LinkedListNode<Employee> currentNode = lnkEmployee.First;

while (currentNode != null)

{

// get value in node

Console.WriteLine($"\n Employee Name: {currentNode.Value.Name}"); // convert to uppercase

Console.WriteLine($"\n Employee Salary: {currentNode.Value.Salary}");

currentNode = currentNode.Next; // get next node

} // end while

FunctionalProgramming

var values = new List<int> {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};

Console.Write("Original values: ");

values.Display(); // call Display extension method

// display the Min, Max, Sum and Average

Console.WriteLine($"\nMin: {values.Min()}");

Console.WriteLine($"Max: {values.Max()}");

Console.WriteLine($"Sum: {values.Sum()}");

Console.WriteLine($"Average: {values.Average()}");

// sum of values via Aggregate

Console.WriteLine("\nSum via Aggregate method: " +

values.Aggregate(0, (x, y) => x + y));

// sum of squares of values via Aggregate

Console.WriteLine("Sum of squares via Aggregate method: " +

values.Aggregate(0, (x, y) => x + y \* y));

// product of values via Aggregate

Console.WriteLine("Product via Aggregate method: " +

values.Aggregate(1, (x, y) => x \* y));

// even values displayed in sorted order

Console.Write("\nEven values displayed in sorted order: ");

values.Where(value => value % 2 == 0) // find even integers

.OrderBy(value => value) // sort remaining values

.Display(); // show results

// odd values multiplied by 10 and displayed in sorted order

Console.Write(

"Odd values multiplied by 10 displayed in sorted order: ");

values.Where(value => value % 2 != 0) // find odd integers

.Select(value => value \* 10) // multiply each by 10

.OrderBy(value => value) // sort the values

.Display(); // show results

// display original values again to prove they were not modified

Console.Write("\nOriginal values: ");

values.Display(); // call Display extension method

}

static class Extensions

{

// extension method that displays all elements separated by spaces

public static void Display<T>(this IEnumerable<T> data)

{

Console.WriteLine(string.Join(" ", data));

}

}

**Parent Class**

public string FirstName { get; }

public string LastName { get; }

public string SocialSecurityNumber { get; }

// three-parameter constructor

public Employee(string firstName, string lastName,

string socialSecurityNumber)

{

FirstName = firstName;

LastName = lastName;

SocialSecurityNumber = socialSecurityNumber;

}

// return string representation of Employee object, using properties

public override string ToString() => $"{FirstName} {LastName}\n" +

$"social security number: {SocialSecurityNumber}";

// abstract method overridden by derived classes

public abstract decimal Earnings(); // no implementation here

**child class properties**

// four-parameter constructor

public SalariedEmployee(string firstName, string lastName,

string socialSecurityNumber, decimal weeklySalary)

: base(firstName, lastName, socialSecurityNumber)

{

WeeklySalary = weeklySalary; // validate salary via property

}

// property that gets and sets salaried employee's salary

public decimal WeeklySalary

{

get

{

return weeklySalary;

}

set

{

if (value < 0) // validation

{

throw new ArgumentOutOfRangeException(nameof(value),

value, $"{nameof(WeeklySalary)} must be >= 0");

}

weeklySalary = value;

}

}

// return string representation of SalariedEmployee object

public override string ToString() =>

$"salaried employee: {base.ToString()}\n" +

$"weekly salary: {WeeklySalary:C}";

**Array**

private static double[] doubleValues = { 8.4, 9.3, 0.2, 7.9, 3.4 };

// sort doubleValues

Sort(doubleValues);

// iterate through the double array with an enumerator

IEnumerator enumerator = doubleValues.GetEnumerator();

while (enumerator.MoveNext())

{

Console.Write($"{enumerator.Current} ");

}

// iterate through the second int array with a foreach statement

foreach (var element in intValuesCopy)

{

Console.Write($"{element} ");

}

// search for 5 in intValues

int result = Array.BinarySearch(intValues, 5);

Console.WriteLine(result >= 0 ?

$"5 found at element {result} in intValues" :

"5 not found in intValues");

**Sorted Dictionary**

SortedDictionary<int, string> coursesDict = new SortedDictionary<int, string>();

// Adding elements to dictionary

coursesDict.Add(101, "Programming I");

// Displaying the values

foreach (KeyValuePair<int, string> pair in coursesDict)

{

//Console.WriteLine("${pair.Key},{pair.Value}");

Console.WriteLine("{0}, {1}", pair.Key, pair.Value);

}

// Searching for a given value based on key/value

if(coursesDict.ContainsKey(101))

Console.WriteLine("Value : {0}", coursesDict[101]);

if (coursesDict.ContainsValue("Java Programming"))

Console.WriteLine("\nElement present");

// Removing an element

coursesDict.Remove(212);

// Counting number of elements in the dictionary

Console.WriteLine("\nTotal elements present: {0}", coursesDict.Count);

SortedDictionary<int, Product> productDict = new SortedDictionary<int, Product>();

// Adding object of product class to dictionary

for (int i=0; i<products.Length; ++i)

{

productDict.Add(i, products[i]);

}

//finding average

private static void AverageDict(SortedDictionary<int, Product> avgDict)

{

double sum = 0;

foreach (var key in avgDict.Keys)

{

sum = sum + avgDict[key].Price;

}

Console.WriteLine($" Average Product Price: {sum/avgDict.Keys.Count}");

}

**Linked list**

string name;

double salary;

Employee employee = new Employee("Sujeet", 1200);

Console.WriteLine($"\n Employee: {employee}");

LinkedList<Employee> lnkEmployee = new LinkedList<Employee>();

// adding employees to linked list

Console.WriteLine("\nEnter Name and Salary:");

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

{

name = Console.ReadLine();

salary = double.Parse(Console.ReadLine());

employee = new Employee(name, salary);

lnkEmployee.AddLast(employee);

}

// Displaying Linked List Employee objects

// iterate over the list by using the nodes

LinkedListNode<Employee> currentNode = lnkEmployee.First;

while (currentNode != null)

{

// get value in node

Console.WriteLine($"\n Employee Name: {currentNode.Value.Name}");

// convert to uppercase

Console.WriteLine($"\n Employee Salary: {currentNode.Value.Salary}");

currentNode = currentNode.Next; // get next node

} // end while

//Adding 5 student object to linkList

AddLinkedListItem(linkStudents, student);

//Remove item from linkList

RemoveLinkedListItem(linkStudents, student1);

//Search item in LinkList

SearchLinkedListItem(linkStudents, student1);

//Remove all item from Link LIst

RemoveAllLinkedListItem(linkStudents);

PrintList(linkStudents);

private static void AddLinkedListItem(LinkedList<Students> name, Students st)

{

name.AddLast(st);

}

private static void PrintList(LinkedList<Students> name)

{

Console.WriteLine("Student List : ");

foreach (var value in name)

{

Console.Write($"{value} ");

}

Console.WriteLine("\n");

}

// delete list items

private static void RemoveLinkedListItem(LinkedList<Students> name, Students st)

{

LinkedListNode<Students> currentNode = name.Find(st);

name.Remove(currentNode); // remove next node

}

// Search list items

private static void SearchLinkedListItem(LinkedList<Students> name, Students st)

{

LinkedListNode<Students> currentNode = name.Find(st);

if (currentNode != null)

{

Console.WriteLine(currentNode.Value);

}

else

{

Console.WriteLine("\n This Student not exists in Student List \n");

}

}

// delete all list items

private static void RemoveAllLinkedListItem(LinkedList<Students> name)

{

LinkedListNode<Students> currentNode = name.First;

while (currentNode != null)

{

name.Remove(currentNode);

currentNode = name.First; // next node become first node now

}

}

**Delegate**

public delegate bool NumberPredicate(int number);

int[] numbers = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

// determine whether an int is even

private static bool IsEven(int number) => number % 2 == 0;

// determine whether an int is odd

private static bool IsOdd(int number) => number % 2 == 1;

// determine whether an int greater than 5

private static bool IsOver5(int number) => number > 5;

// display the elements of a List

private static void DisplayList(string description, List<int> list)

{

Console.Write(description); // display the output's description

// iterate over each element in the List

foreach (var item in list)

{

Console.Write($"{item} "); // print item followed by a space

}

Console.WriteLine(); // add a new line

}

// select an array's elements that satisfy the predicate

private static List<int> FilterArray(int[] intArray, NumberPredicate predicate)

{

// hold the selected elements

var result = new List<int>();

// iterate over each element in the array

foreach (var item in intArray)

{

// if the element satisfies the predicate

if (predicate(item)) // invokes method referenced by predicate

{

result.Add(item); // add the element to the result

}

}

return result; // return the result

}

**test**

// create an instance of the NumberPredicate delegate type

NumberPredicate evenPredicate = IsEven;

// call IsEven using a delegate variable

Console.WriteLine($"Call IsEven using a delegate variable: {evenPredicate(4)}");

// filter the even numbers using method IsEven

List<int> evenNumbers = FilterArray(numbers, evenPredicate);

// display the result

DisplayList("Use IsEven to filter even numbers: ", evenNumbers);

// filter the odd numbers using method IsOdd

List<int> oddNumbers = FilterArray(numbers, IsOdd);

// display the result

DisplayList("Use IsOdd to filter odd numbers: ", oddNumbers);

// filter numbers greater than 5 using method IsOver5

List<int> numbersOver5 = FilterArray(numbers, IsOver5);

// display the result

DisplayList("Use IsOver5 to filter numbers over 5: ", numbersOver5);

**Check primes**

// check whether value is a prime number

// and mark all multiples as not prime

public bool IsPrime(int value)

{

// if value is prime, mark all of multiples

// as not prime and return true

if (primes[value])

{

// mark all multiples of value as not prime

for (var i = value + value; i < primes.Length; i += value)

{

primes[i] = false; // i is not prime

}

return true;

}

else

{

return false;

}

}

// displays prime numbers in primesTextBox

private async Task<int> FindPrimes(int maximum)

{

var primeCount = 0;

// find primes less than maximum

for (var i = 2; i < maximum && !Canceled; ++i)

{

// if i is prime, display it

if (await Task.Run(() => IsPrime(i)))

{

++primeCount; // increment number of primes found

primesTextBox.AppendText($"{i}{Environment.NewLine}");

}

var percentage = (double)progressBar.Value /

(progressBar.Maximum - progressBar.Minimum + 1);

percentageLabel.Text = $"{percentage:P0}";

progressBar.Value = i + 1; // update progress

}

**FunC**

// user defined function – Max – compare three string to find max

private static string Max(string s1, string s2, string s3)

{

string max = s1;

max = (s2.CompareTo(max) > 0) ? s2 : max;

max = (s3.CompareTo(max) > 0) ? s3 : max;

return max;

} // end Max

Func<string, string, string, string> s4 = Max;

Console.WriteLine(s4("me", "she", "he"));

Console.WriteLine($"largest of three iS {s4("me", "she", "he")}");

private static void Square2(double num)

{

Console.WriteLine(num \* num);

}

Action<double> sq2 = Square2;

sq2(5);

**Aggregate Method**

var values = new List<int> {3, 10, 6, 1, 4, 8, 2, 5, 9, 7};

Console.Write("Original values: ");

values.Display(); // call Display extension method

// display the Min, Max, Sum and Average

Console.WriteLine($"\nMin: {values.Min()}");

Console.WriteLine($"Max: {values.Max()}");

Console.WriteLine($"Sum: {values.Sum()}");

Console.WriteLine($"Average: {values.Average()}");

// sum of values via Aggregate

Console.WriteLine("\nSum via Aggregate method: " +

values.Aggregate(0, (x, y) => x + y));

// sum of squares of values via Aggregate

Console.WriteLine("Sum of squares via Aggregate method: " +

values.Aggregate(0, (x, y) => x + y \* y));

// product of values via Aggregate

Console.WriteLine("Product via Aggregate method: " +

values.Aggregate(1, (x, y) => x \* y));

// even values displayed in sorted order

Console.Write("\nEven values displayed in sorted order: ");

values.Where(value => value % 2 == 0) // find even integers

.OrderBy(value => value) // sort remaining values

.Display(); // show results

// odd values multiplied by 10 and displayed in sorted order

Console.Write(

"Odd values multiplied by 10 displayed in sorted order: ");

values.Where(value => value % 2 != 0) // find odd integers

.Select(value => value \* 10) // multiply each by 10

.OrderBy(value => value) // sort the values

.Display(); // show results

static class Extensions

{

// extension method that displays all elements separated by spaces

public static void Display<T>(this IEnumerable<T> data)

{

Console.WriteLine(string.Join(" ", data));

}

}

**LinQ with array of object**

// initialize array of employees

var employees = new[] {

new Employee("Jason", "Red", 5000M),

new Employee("Ashley", "Green", 7600M),

new Employee("Matthew", "Indigo", 3587.5M),

new Employee("James", "Indigo", 4700.77M),

new Employee("Luke", "Indigo", 6200M),

new Employee("Jason", "Blue", 3200M),

new Employee("Wendy", "Brown", 4236.4M)};

// display all employees

Console.WriteLine("Original array:");

foreach (var element in employees)

{

Console.WriteLine(element);

}

// filter a range of salaries using && in a LINQ query

var between4K6K =

from e in employees

where (e.MonthlySalary >= 4000M) && (e.MonthlySalary <= 6000M)

select e;

// display employees making between 4000 and 6000 per month

Console.WriteLine("\nEmployees earning in the range " +

$"{4000:C}-{6000:C} per month:");

foreach (var element in between4K6K)

{

Console.WriteLine(element);

}

// order the employees by last name, then first name with LINQ

var nameSorted =

from e in employees

orderby e.LastName, e.FirstName

select e;

//display element sorted by last name

Console.WriteLine("employee list displayed by last name --- \n");

foreach (var item in nameSorted)

{

Console.WriteLine(item);

}

// header

Console.WriteLine("\nFirst employee when sorted by name:");

// attempt to display the first result of the above LINQ query

if (nameSorted.Any())

{

Console.WriteLine(nameSorted.First());

}

else

{

Console.WriteLine("not found");

}

// use LINQ to select employee last names

var lastNames =

from e in employees

select e.LastName;

// use method Distinct to select unique last names

Console.WriteLine("\nUnique employee last names:");

foreach (var element in lastNames.Distinct())

{

Console.WriteLine(element);

}

// use LINQ to select first and last names

var names =

from e in employees

select new { e.FirstName, e.LastName }; // or

// select new { FIRST = e.FirstName, LAST= e.LastName };

// display full names

Console.WriteLine("\nNames only:");

foreach (var element in names)

{

Console.WriteLine(element);

}

Console.WriteLine();

**How to append new line in text box**

// display message if operation was canceled

if (Canceled)

{

primesTextBox.AppendText($"Canceled{Environment.NewLine}");

}

**Get input and split input string**

Console.WriteLine("Enter a string: "); // prompt for user input

string input = Console.ReadLine(); // get input

// split input text into tokens

string[] words = Regex.Split(input, @"\s+");