



University of Malta  
Faculty of ICT  
Department of Intelligent Computer Systems

# Data Structures and Algorithms Course Assignment

by

Lazar Lazarević

Lecturer: Dr. John Abela

Study unit:

Data Structures and Algorithms 1, and Programming Paradigms

Study unit code: CIS1017

Year 2013/2014

## Table of contents:

<b>1. Assignment tasks</b>	<b>3</b>
Note:	3
<b>2. Implementation</b>	<b>4</b>
2.1 Arab to Roman number	4
2.2 RPN evaluation	4
2.3 Prime number	6
2.4 Sorting	7
2.5 Square root	8
2.6 Matrix multiplication	9
2.7 Sum of odd numbers	9
2.8 Sine & Cosine	10
2.9 Palindrome	11
<b>3. Appendix</b>	<b>12</b>
3.1 Arab to Roman number Code	12
3.2 RPN evaluation Code	12
3.3 Prime number Code	18
3.4 Sorting Code	22
3.5 Square root Code	23
3.6 Matrix multiplication Code	23
3.7 Sum of odd numbers Code	25
3.8 Sine & Cosine Code	26
3.9 Palindrome Code	27

## 1. Assignment tasks

1. Write and implement a function that accepts a positive integer and outputs a representation of the same integer as a string in Roman numerals format. The range of input should be from 1 to 1024. (Greedy algorithm?)
2. Write a program that uses an ADT Stack to evaluate arithmetic expressions in RPN format. The contents of the stack should be displayed on the screen during evaluation. The allowed arithmetic operators are +, -, x, and /.
3. Write a Boolean function that checks if a number is prime. Alternatively, implement the Sieve of Eratosthenes algorithm. Explain any optimizations made.
4. Write a program that fills an integer array of size 16384 with random integers and then sorts the array using an optimized Shell sorting algorithm and by Insertion Sort.
5. Write a program that finds an approximation to the square root of a given number  $n$  using numerical iterative methods.
6. Write and implement a procedure that multiplies two random real-valued 16 X 16 matrices. Use two 2-dimensional arrays to store the matrices and another similar array to store the result.
7. Write a recursive function to compute the sum of all the odd numbers from  $n$  to  $m$ , where  $n$  and  $m$  are two odd (check!) integers entered by the user. Also write an iterative procedure/function to perform the same task.
8. Write a function that computes cosine or sine by taking the first  $n$  terms of the appropriate infinite series.
9. Write a program that checks if a given input string is a palindrome.

### Note:

I attempted and completed **all questions**.

In chapter 2, only methods' code will be presented, for complete project code please refer to the corresponding appendix.

All programs were implemented using C# with Visual Studio 2013.

## 2. Implementation

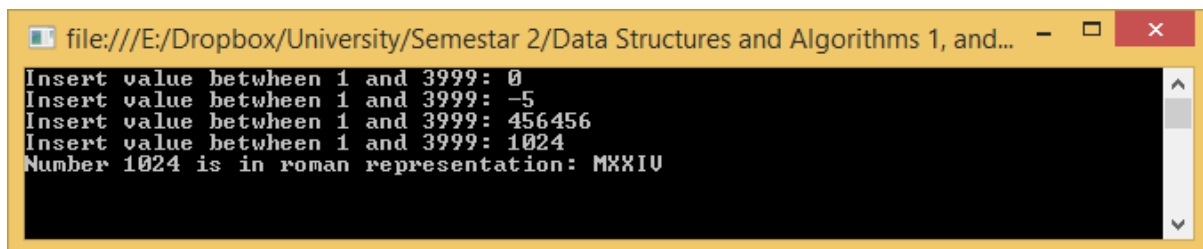
### 2.1 Arab to Roman number

Transforming an integer number to Roman form. In this case is done by recursion.

Test: I tested values than cannot be translated to Roman form to make sure that the program will not crash, and then entered the valid number “1024”.

Code:

```
public static string ArabToRoman(int num)
{
    if (num >= 1000) return "M" + ArabToRoman(num - 1000);
    if (num >= 900) return "CM" + ArabToRoman(num - 900);
    if (num >= 500) return "D" + ArabToRoman(num - 500);
    if (num >= 400) return "CD" + ArabToRoman(num - 400);
    if (num >= 100) return "C" + ArabToRoman(num - 100);
    if (num >= 90) return "XC" + ArabToRoman(num - 90);
    if (num >= 50) return "L" + ArabToRoman(num - 50);
    if (num >= 40) return "XL" + ArabToRoman(num - 40);
    if (num >= 10) return "X" + ArabToRoman(num - 10);
    if (num >= 9) return "IX" + ArabToRoman(num - 9);
    if (num >= 5) return "V" + ArabToRoman(num - 5);
    if (num >= 4) return "IV" + ArabToRoman(num - 4);
    if (num >= 1) return "I" + ArabToRoman(num - 1);
    if (num < 1) return string.Empty;
    return string.Empty;
}
```

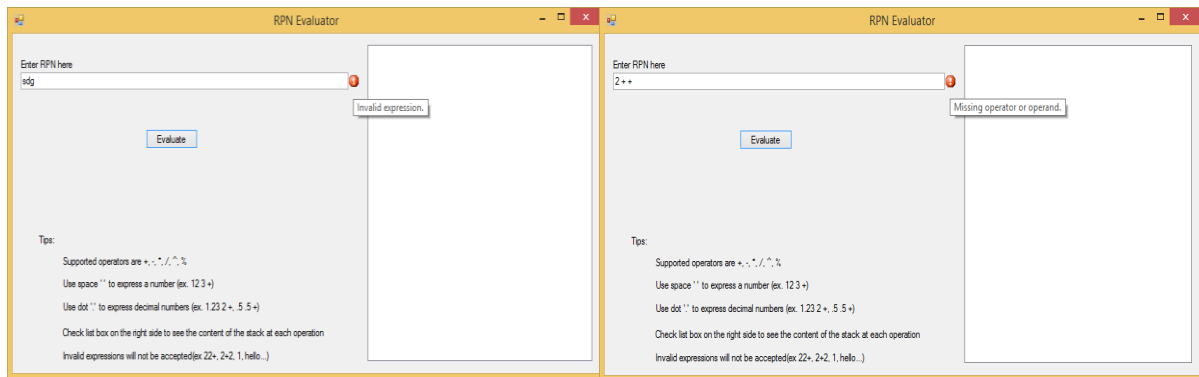


```
file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Insert value between 1 and 3999: 0
Insert value between 1 and 3999: -5
Insert value between 1 and 3999: 456456
Insert value between 1 and 3999: 1024
Number 1024 is in roman representation: MXXIV
```

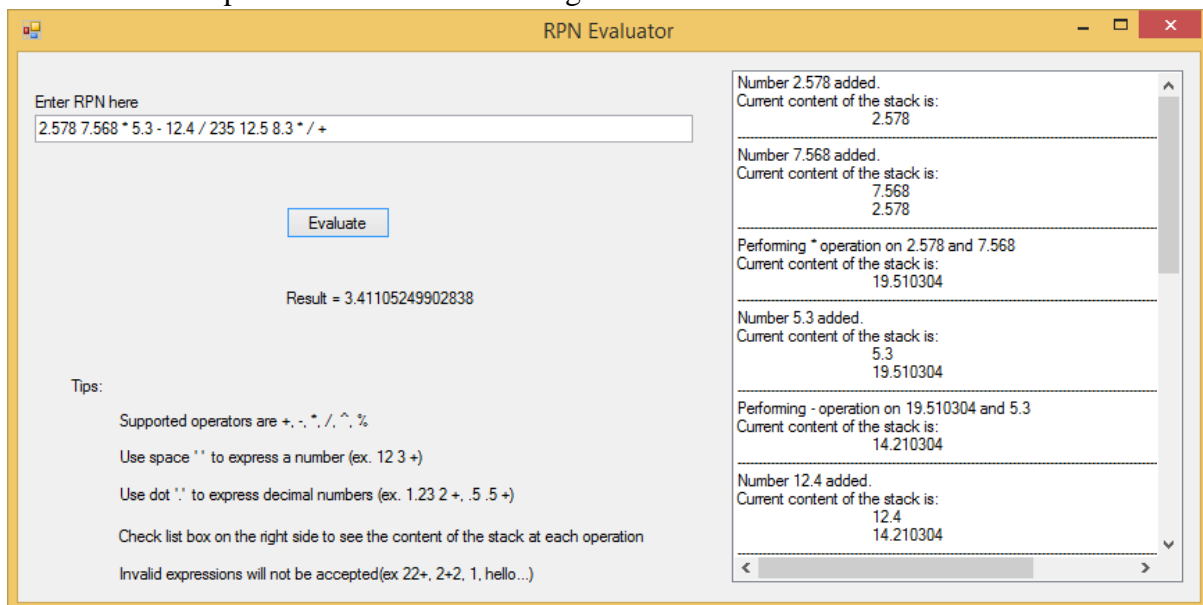
### 2.2 RPN evaluation

This program is made with Windows forms, which makes it easier to copy long expressions (into text box). Invalid expressions will not be accepted, if one tries to provide such string corresponding error will appear as shown below (Invalid expression or missing operator or operand).

Test: I entered some invalid expressions to demonstrate “Invalid expression” and “Missing operator or operand” errors. At the end, valid expression was entered and correct result appeared together with complete stack content in the list box.



When user provides expression in correct format, result will appear and the content of the stack will be printed in list box on the right side.



Code:

```
public double GetResult(List<Element> elements)
{
    double result = 0;
    stackContent.Clear();

    for(int i = 0; i < elements.Count; i++)
    {
        if(elements[i] is Number)
        {
            stack.Push(elements[i] as Number);
            stackContent.Add("Number " + elements[i].ToString() + " added.");
        }
        else
        {
            Number second = stack.Pop(), first = stack.Pop();
            stackContent.Add("Performing " + elements[i].ToString() + " operation on " + first.ToString() + " and " + second.ToString());
            stack.Push(Calculate(first, elements[i] as Operator, second));
        }
        TraverseStack();
    }
    return result = stack.Pop().Value;
}
//For complete project code refer to appendix or source code on the CD.
```

## 2.3 Prime number

This algorithm checks only whether a number is prime or not.

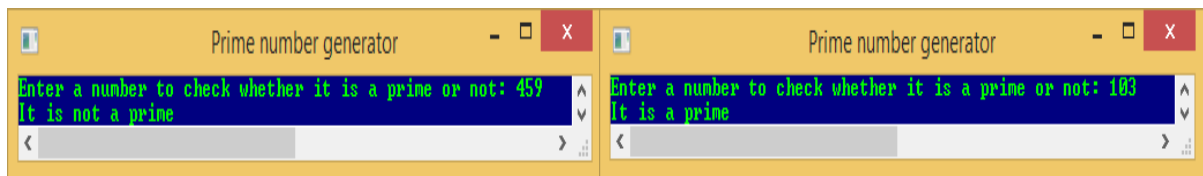
Test: One prime and one not prime was inputted to demonstrate the correctness of the program.

Code:

```
public static bool IsPrime(int num)
{
    if(num == 2 || num == 3) return true;
    if(num % 2 == 0 || num % 3 == 0) return false;

    int i = 5, w = 2;

    while(i * i <= num)
    {
        if(num % i == 0)
            return false;
        i += w;
        w = 6 - w;
    }
    return true;
}
```



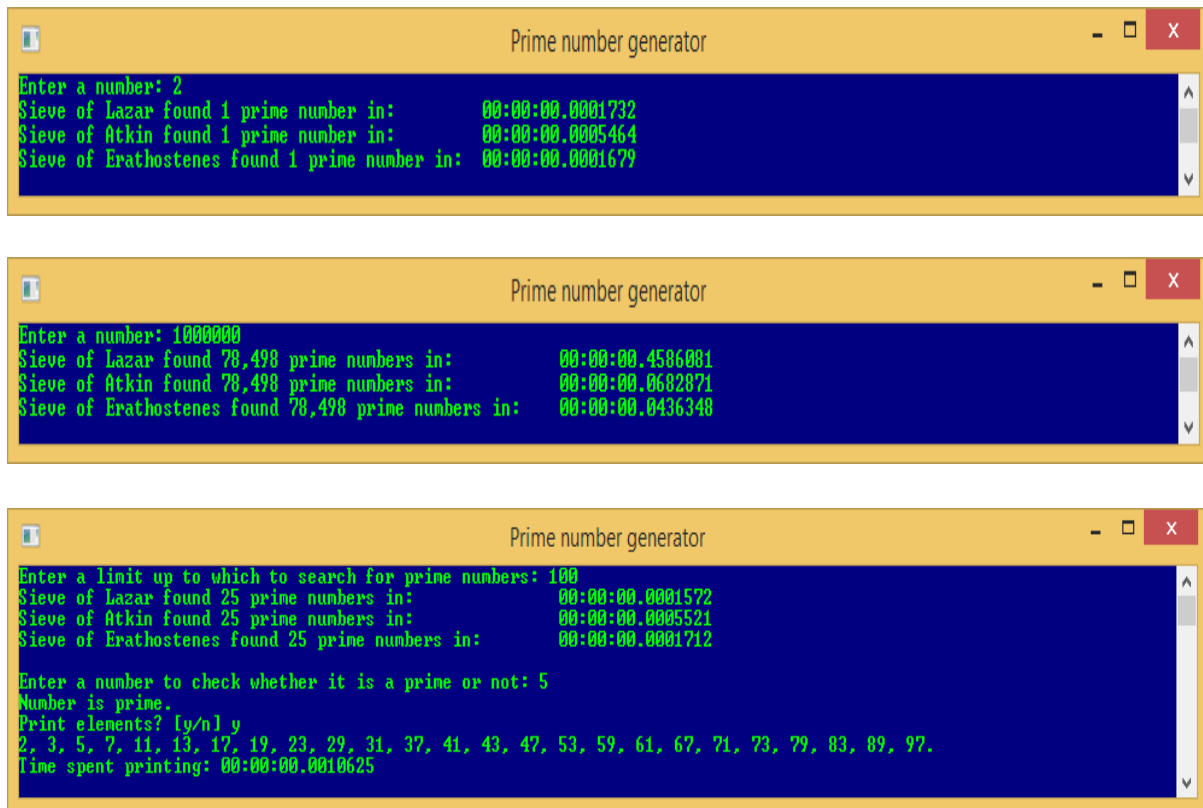
The Sieve of Eratosthenes finds all prime numbers until the specified limit is not reached.

Test: Limit was entered to demonstrate low and high limit, I've implemented two more algorithms beside Sieve of Eratosthenes, that is Sieve of Atkins and my algorithm. Time spent finding primes are printed next to each algorithm to show the difference between them.

Code:

```
sealed class Sieve_of_Eratosthenes : Primes
{
    #region Constructor
    public Sieve_of_Eratosthenes(int limit)
        : base(limit)
    {
        for (int i = 2; i <= Limit; i++)
            isPrime[i] = true;
    }
    #endregion

    #region FindPrimes FindAllPrimesTillLimit
    public override void FindAllPrimesTillLimit()
    {
        for (int i = 0; i < LimSqrt; i++)
            if (isPrime[i]) {
                for (int j = i * i; j <= Limit; j += i)
                    isPrime[j] = false;
            }
    }
    #endregion
}
//Algorithm found on http://en.wikipedia.org/wiki/Sieve\_of\_Eratosthenes
```



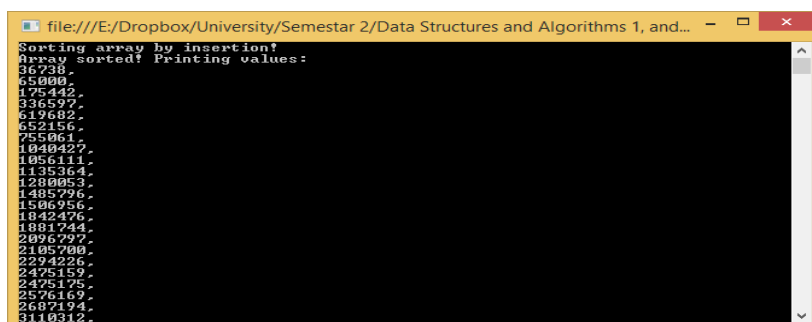
## 2.4 Sorting

Insertion sort code:

Test: random numbers were generated and stored in array which was sorted, and all values were printed after sorting.

Code:

```
public static void Insertion(int[] arr)
{
    int c = 0;
    for (int i = 1; i < arr.Length; i++)
    {
        if (i % 10000 == 0) Console.WriteLine("\r" + c++);
        int key = arr[i];
        int j = i - 1;
        while(j >= 0 && arr[j] > key)
        {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
```



Shell Sort Code:

```
public static void Shell(int[] arr)
{
    int n = arr.Length;
    int increment = n / 2;
    while (increment > 0) {
        int last = increment;
        while (last < n) {
            int current = last - increment;
            while (current >= 0) {
                if (arr[current] > arr[current + increment]) {
                    int tmp = arr[current];
                    arr[current] = arr[current + increment];
                    arr[current + increment] = tmp;
                    current -= increment;
                }
            }
            last++;
        }
        increment /= 2;
    }
}
```

Test: same as for insertion sort.

```
file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Sorting array by Shell
Array sorted, printing elements
68512,
99646,
207300,
230569,
300297,
315334,
321427,
630734,
1041555,
1214288,
1296215,
1482704,
1793716,
2030668,
2268539,
2785021,
2833355,
2858326,
2891769,
2907102,
3131586,
3186454,
3275635,
```

## 2.5 Square root

Test: A number was entered of which root will be found, next was entered a guess which algorithm requires.

Code:

```
double result = ((value / guess) + guess) / 2;

while(Math.Abs(result - guess) > epsilon) {
    guess = result;
    result = ((value / guess) + guess) / 2;
}
```

```
file:///C:/Users/Lazar/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and Progr...
Enter a number: 36
Enter a guess: 6
The approx sqrt of 36 is 6
```



## 2.6 Matrix multiplication

Assumption: Here I assumed that by multiplying matrices, cross product was ment.

Test: Random numbers were generated and stored in two 16x16 matrices, and then cross product was calculated.

Code:

```
static readonly short MAX = 16;
```

```
public static short[,] Mult(short[,] matrix1, short[,] matrix2, short[,] matrix3)
{
    for(short i = 0; i < MAX; i++) {
        for(short j = 0; j < MAX; j++) {
            short result = 0;
            for(short k = 0; k < MAX; k++) {
                result += (short)(matrix1[i, k] * matrix2[k, j]);
            }
            matrix3[i, j] = result;
        }
    }
    return matrix3;
}
```

## 2.7 Sum of odd numbers

Test: All non-acceptable values were entered to demonstrate the security, then correct numbers were inputted and result was printed for both iterative and recursive way.

Sum iteratively code:

```
public static int SumIter(int lower, int higher)
{
    int result = 0;

    for(int i = lower; i <= higher; i += 2)
        result += i;

    return result;
}
```

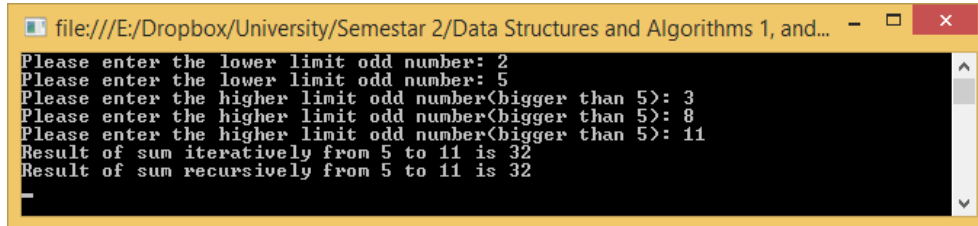
Sum recursively code:

```

public static int SumRec(int lower, int higher)
{
    if(lower > higher)
        return 0;

    return lower + SumRec(lower + 2, higher);
}

```



```

file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Please enter the lower limit odd number: 2
Please enter the lower limit odd number: 5
Please enter the higher limit odd number(bigger than 5): 3
Please enter the higher limit odd number(bigger than 5): 8
Please enter the higher limit odd number(bigger than 5): 11
Result of sum iteratively from 5 to 11 is 32
Result of sum recursively from 5 to 11 is 32

```

## 2.8 Sine & Cosine

Test: Numbers were entered and both sin and cos were calculated.

Note: Code  $-(i \& 1) \mid 1$  only change the sign  $-/+$

Sine code:

```

public static double Sin(double x, int n)
{
    double result = 0;
    x *= (Math.PI / 180); // Converting degrees to radians

    for (byte i = 0; i < n; i++)
        result +=  $-(i \& 1) \mid 1$  * (Math.Pow(x, 2 * i + 1) / Fact(2 * i + 1));

    return Math.Round(result, 10);
}

```

Cosine code:

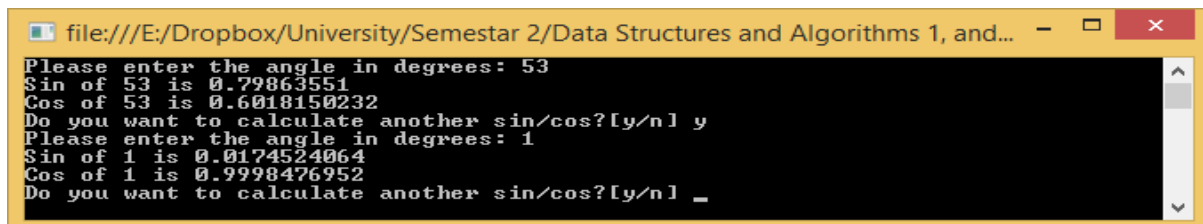
```

public static double Cos(double x, int n)
{
    double result = 0;
    x *= (Math.PI / 180); //Converting degrees to radians

    for (byte i = 0; i < n; i++)
        result +=  $-(i \& 1) \mid 1$  * (Math.Pow(x, 2 * i) / Fact(2 * i));

    return Math.Round(result, 10);
}

```



```

file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Please enter the angle in degrees: 53
Sin of 53 is 0.79863551
Cos of 53 is 0.6018150232
Do you want to calculate another sin/cos?[y/n] y
Please enter the angle in degrees: 1
Sin of 1 is 0.0174524064
Cos of 1 is 0.9998476952
Do you want to calculate another sin/cos?[y/n] _

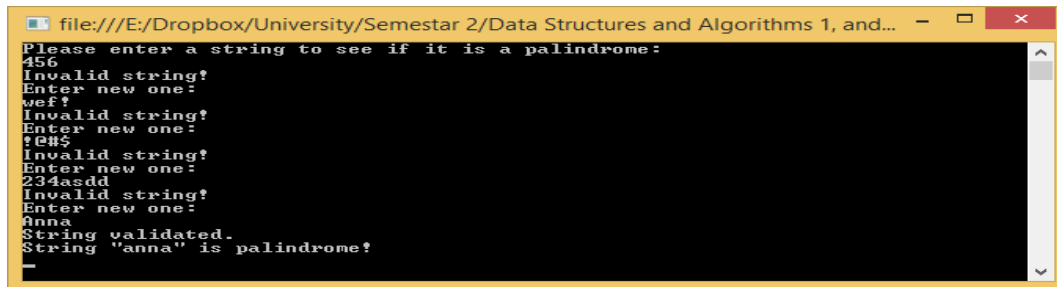
```

## 2.9 Palindrome

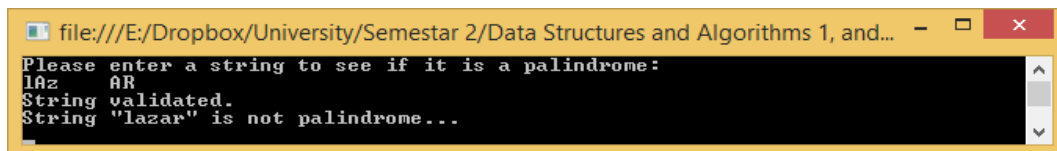
Test: All non-acceptable strings were entered to demonstrate robustness, then one non palindrome and one palindrome string were inputted to show that the program works.

Code:

```
static bool IsPalindrome(string s)
{
    for (int i = 0; i < s.Length / 2; i++)
        if (s[i] != s[s.Length - 1 - i])
            return false;
    return true;
}
```



```
file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Please enter a string to see if it is a palindrome:
456
Invalid string!
Enter new one:
wef!
Invalid string!
Enter new one:
234asdd
Invalid string!
Enter new one:
anna
String validated.
String "anna" is palindrome!
```



```
file:///E:/Dropbox/University/Semestar 2/Data Structures and Algorithms 1, and...
Please enter a string to see if it is a palindrome:
lAz AR
String validated.
String "lazar" is not palindrome...
```

## 3. Appendix

### 3.1 Arab to Roman number Code

```
using System;

namespace RomanNumbers
{
    class Program
    {
        #region Main
        static void Main(string[] args)
        {
            int num;

            do {
                Console.WriteLine("Insert value between 1 and 3999: ");
                int.TryParse(Console.ReadLine(), out num);
            } while (num <= 0 || num > 3999);

            Console.WriteLine("Number {0} is in roman representation: " + ArabToRoman(num), num);

            Console.ReadKey();
        }
        #endregion

        #region ArabToRoman
        public static string ArabToRoman(int num)
        {
            if (num >= 1000) return "M" + ArabToRoman(num - 1000);
            if (num >= 900) return "CM" + ArabToRoman(num - 900);
            if (num >= 500) return "D" + ArabToRoman(num - 500);
            if (num >= 400) return "CD" + ArabToRoman(num - 400);
            if (num >= 100) return "C" + ArabToRoman(num - 100);
            if (num >= 90) return "XC" + ArabToRoman(num - 90);
            if (num >= 50) return "L" + ArabToRoman(num - 50);
            if (num >= 40) return "XL" + ArabToRoman(num - 40);
            if (num >= 10) return "X" + ArabToRoman(num - 10);
            if (num >= 9) return "IX" + ArabToRoman(num - 9);
            if (num >= 5) return "V" + ArabToRoman(num - 5);
            if (num >= 4) return "IV" + ArabToRoman(num - 4);
            if (num >= 1) return "I" + ArabToRoman(num - 1);
            if (num < 1) return string.Empty;
            return "";
        }
        #endregion
    }
}
```

### 3.2 RPN evaluation Code

```
namespace RPNEvaluator
{
    abstract class Element
    {
    }
}

using System;

namespace RPNEvaluator
{
    class Number : Element
    {
        #region Private Variables
        /// <summary>
```

```

        /// Value of the number
        /// </summary>
private double value;
        #region Public Properties
public double Value
{
    get { return value; }
}
        #endregion

        #region Constructor Number
        /// <summary>
        /// Constructor
        /// </summary>
        /// <param name="number">String representing a number</param>
public Number(string number)
{
    double.TryParse(number, out this.value);
}
        #endregion

        #region Operator ^ overloading
        /// <summary>
        /// Operator overloading ^
        /// </summary>
        /// <param name="n1">First number</param>
        /// <param name="n2">Second number (exponential)</param>
        /// <returns>Result of Math.pow</returns>
public static double operator ^(Number n1, Number n2)
{
    return Math.Pow(n1.Value, n2.Value);
}
        #endregion

        #region ToString
        /// <summary>
        /// ToString() override
        /// </summary>
        /// <returns>Value of number in string format</returns>
public override string ToString()
{
    return value.ToString();
}
        #endregion
    }
}

namespace RPNEvaluator
{
    #region enum OperatorType
    /// <summary>
    /// Representing operators
    /// </summary>
    public enum OperatorType
    {
        ADD,
        SUB,
        DIV,
        MUL,
        POW,
        MOD,
    }
    #endregion

    class Operator : Element
    {
        #region Public
        /// <summary>
        /// OperatorType enum
        /// </summary>
        public OperatorType type;
        #endregion
    }
}

```

```

        #region Private variables
        /// <summary>
        /// Char representing a operator
        /// </summary>
private char ch;
        #endregion

        #region Constructor
        /// <summary>
        /// Creating new operator object
        /// </summary>
        /// <param name="ch">Char representing a operator</param>
public Operator(char ch)
    {
        this.ch = ch;
switch(ch)
    {
case '+': type = OperatorType.ADD; break;
case '-': type = OperatorType.SUB; break;
case '*': type = OperatorType.MUL; break;
case '/': type = OperatorType.DIV; break;
case '^': type = OperatorType.POW; break;
case '%': type = OperatorType.MOD; break;
    }
    }
        #endregion

        #region ToString
        /// <summary>
        /// Convert operator to string
        /// </summary>
        /// <returns>Operator in string format</returns>
public override string ToString()
    {
return ch.ToString();
    }
        #endregion
    }
}

using System;
using System.Collections.Generic;

namespace RPNEvaluator
{
    class Parser
    {
        #region Private
        /// <summary>
        /// List of all elements parsed from a string (numbers and operators)
        /// </summary>
public static List<Element> elements;
        #endregion

        #region Parse
        /// <summary>
        /// Parse string and store all elements (number and operators) to the list
        /// </summary>
        /// <param name="expression">String to be parsed</param>
private void Parse(string expression)
    {
for(int i = 0; i < expression.Length; i++)
    {
if(Char.IsNumber(expression[i]) || expression[i] == '.')
    elements.Add(new Number(GetNumberAt(expression, ref i)));
else if(IsOperator(expression[i]))
    elements.Add(new Operator(expression[i]));
    }
    }
        #endregion
        #region GetElementsFrom
        /// <summary>
        /// Gets all elements from string format

```

```

        /// </summary>
        /// <param name="RPNexpression"></param>
        /// <returns></returns>
public List<Element> GetElementsFrom(string RPNexpression)
{
    elements = new List<Element>();
    Parse(RPNexpression);

    return this.IsValid() ? elements : null;
}
#endregion

#region isValid
/// <summary>
/// Check if expression is valid
/// count of numbers must be for 1 greater than count of operators
/// </summary>
/// <returns>True if expression is valid, otherwise false</returns>
private bool IsValid()
{
    int num = 0, op = 0;

    foreach(Element e in elements)
    {
        if(e.GetType() == typeof(Number))
            num++;
        else op++;
    }
    return (num - op == 1);
}
#endregion

#region isOperator
/// <summary>
/// Check if the character is valid operator
/// </summary>
/// <param name="ch"></param>
/// <returns>True if it is, false if it is not operator</returns>
private bool IsOperator(char ch)
{
    if(ch == '+' || ch == '-' || ch == '*' || ch == '/' || ch == '^' || ch == '%') return true;
    else return false;
}
#endregion

#region GetNumber
/// <summary>
/// Checks if number is multiple digit number or decimal
/// if not it will return the first character of a string(signle digit number)
/// </summary>
/// <param name="expr">String expression where number is</param>
/// <param name="pos">The first index of a number in string</param>
/// <returns>String of a number. (one digit, multiple digit or decimal number)</returns>
private string GetNumberAt(string expr, ref int pos)
{
    string num;
    int index = pos;

    while(expr[index] != ' ')
        index++;

    num = expr.Substring(pos, index - pos);
    pos = index;

    return num;
}
#endregion
}

using System;
using System.Collections.Generic;

```

```

namespace RPNEvaluator
{
class RPNEvaluator
{
    #region Private
    /// <summary>
    /// Stack where will be stored numbers, and popped later for calculation purposes
    /// </summary>
private Stack<Number> stack = new Stack<Number>();
    #endregion

    #region Public
    /// <summary>
    /// List of strings that holds all operations done on stack,
    /// such as added new element, all current elements on the stack after new one is added or
removed
    /// </summary>
public List<string> stackContent = new List<string>();
    #endregion

    #region Calculate
    /// <summary>
    /// Method calculating two numbers
    /// </summary>
    /// <param name="first">First operand</param>
    /// <param name="op">Operator</param>
    /// <param name="second">Second operand</param>
    /// <returns>Calculated value of two operands</returns>
private Number Calculate(Number first, Operator op, Number second)
    {
double result = 0;

switch(op.type)
    {
case OperatorType.ADD: result = first.Value + second.Value; break;
case OperatorType.SUB: result = first.Value - second.Value; break;
case OperatorType.MUL: result = first.Value * second.Value; break;
case OperatorType.DIV: result = first.Value / second.Value; break;
case OperatorType.POW: result = first ^ second; break;
case OperatorType.MOD: result = first.Value % second.Value; break;
    }

return new Number(Convert.ToString(result));
    }
    #endregion

    #region GetResult
    /// <summary>
    /// Method that check each element in the list, if it is number push it on the stack,
    /// if it is operator pop two numbers from the stack and do some operation on them
    /// </summary>
    /// <param name="elements">List of elements previously parsed</param>
    /// <returns>Final value</returns>
public double GetResult(List<Element> elements)
    {
double result = 0;
stackContent.Clear();

for(int i = 0; i < elements.Count; i++)
    {
if(elements[i] is Number)
    {
stack.Push(elements[i] as Number);
stackContent.Add("Number " + elements[i].ToString() + " added.");
    }
else
    {
        Number second = stack.Pop(), first = stack.Pop();
stackContent.Add("Performing " + elements[i].ToString() + " operation on " + first.ToString() + " and " + second.ToString());
stack.Push(Calculate(first, elements[i] as Operator, second));
    }
    }
    TraverseStack();
    }
}
}

```



```

return result = stack.Pop().Value;
    }
    #endregion

    #region TraverseStack
    /// <summary>
    /// Traverses the stack and add content to the stack for printing
    /// </summary>
private void TraverseStack()
    {
        stackContent.Add("Current content of the stack is:");
        foreach(Number n in stack)
            stackContent.Add("\t\t " + n.ToString());

        stackContent.Add("-----");
        stackContent.Add("-----");
    }
    #endregion
}

using System;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using System.Windows.Forms;

namespace RPNEvaluator
{
    public partial class Form1 : Form
    {
        #region Private
        /// <summary>
        /// List of all elements
        /// </summary>
private List<Element> elements;

        /// <summary>
        /// Regular expression, to check if RPN is in valid format
        /// a number, decimal or not, must be followed by a space,
        /// number with multiple dots are not allowed (2.2.2), .5 is allowed = 0.5
        /// operators allowed are - * / + ^(exponential), %(modulus)
        /// other characters are not allowed
        /// </summary>
private Regex regRpn = new Regex(@"G((\d*(\.\d+)?[ ])+[ -*/+^%])+$");

        /// <summary>
        /// Regular expression to convert all two or more spaces into one space ex: " " -> " "
        /// </summary>
private Regex spaceRemover = new Regex(@"[ ]{2,}", RegexOptions.None);
private RPNEvaluator rpnEvaluator = new RPNEvaluator();

private Parser parser = new Parser();
        #endregion

        #region Form1
public Form1()
    {
        InitializeComponent();
    }
    #endregion

    #region evaluate_Click
    /// <summary>
    /// On button click event evaluate expression if valid
    /// </summary>
    /// <param name="sender"></param>
    /// <param name="e"></param>
private void evaluate_Click(object sender, EventArgs e)
    {
        errorProvider1.SetError(RPNEExpression, "");

        RPNEExpression.Text = spaceRemover.Replace(RPNEExpression.Text.Trim(), @" ");
    }
}

```

```

if (regRpn.IsMatch(RPNExpression.Text)) {
    elements = new List<Element>();
    if ((elements = parser.GetElementsFrom(RPNExpression.Text)) != null) { // if list is not empty

        finalResult.Text = "Result = " + rpnEvaluator.GetResult(elements);

    PrintStackContent();
    }
    else errorProvider1.SetError(RPNExpression, "Missing operator or operand.");
    }
    else errorProvider1.SetError(RPNExpression, "Invalid expression.");
    }
    #endregion

    #region PrintStackContent
    /// <summary>
    /// Print all elements in stack
    /// </summary>
private void PrintStackContent()
{
    StackContentListBox.Items.Clear();

    foreach (string s in rpnEvaluator.stackContent)
        StackContentListBox.Items.Add(s);
    }
    #endregion
}
}

```

### 3.3 Prime number Code

```

using System;
using System.Collections.Generic;
using System.IO;
using System.Text;

namespace Prime_Number
{
    abstract class Primes
    {
        #region Data part
        private int limit;
        private StringBuilder sb;

        protected readonly List<int> primes;
        protected readonly double LimSqrt;
        protected bool[] isPrime;
        protected int Limit
        {
            set { limit = value; }
            get { return limit; }
        }
        #endregion

        #region Constructor
        protected Primes(int limit)
        {
            this.LimSqrt = Math.Sqrt(limit);
            this.Limit = limit;
            this.isPrime = new bool[limit + 1];
            this.primes = new List<int>();
            this.sb = new StringBuilder();
        }
        #endregion

        #region Property NumberOfPrimes
        public int NumberOfPrimes
        {
            get
            {

```

```

if (primes.Count == 0) {
    int count = 0;
    for (int i = 0; i < isPrime.Length; i++)
        if (isPrime[i])
            count++;
    return count;
}
else
    return primes.Count;
}
}
#endregion

#region FindAllPrimesTillLimit
public abstract void FindAllPrimesTillLimit();
#endregion

#region IsPrime
public bool IsPrime(int num)
{
    if (this.NumberOfPrimes == 0)
        FindAllPrimesTillLimit();

    return isPrime[num];
}
#endregion

#region ToList
private void ToList()
{
    primes.Add(2);
    for (int i = 3; i <= limit; i += 2)
        if (isPrime[i])
            primes.Add(i);
}
#endregion

#region TraversePrimes()
private void TraversePrimes()
{
    foreach (int p in primes)
        sb.Append(p).Append(',').Append(' ');

    sb.Remove(sb.Length - 2, 2).Append('.');
}
#endregion

#region PrintPrimes
public void PrintPrimes()
{
    if (NumberOfPrimes == 0) {
        Console.WriteLine("No prime numbers in the list...");
        return;
    }
    ToList();
    TraversePrimes();
    using (StreamWriter stream = new StreamWriter(Console.OpenStandardOutput())) {
        stream.Write(sb);
    }
}
#endregion
}

namespace Prime_Number
{
    sealed class Sieve_of_Eratosthenes : Primes
    {
        #region Constructor
        public Sieve_of_Eratosthenes(int limit)
            : base(limit)
        {

```

```

for (int i = 2; i <= Limit; i++)
isPrime[i] = true;
    }
    #endregion

    #region FindPrimes FindAllPrimesTillLimit
public override void FindAllPrimesTillLimit()
    {
for (int i = 0; i < LimSqrt; i++)
if (isPrime[i]) {
for (int j = i * i; j <= Limit; j += i)
isPrime[j] = false;
        }
    }
    #endregion
}

}

using System;
using System.Diagnostics;
namespace Prime_Number
{
class Program
    {
        #region GetInput
public static int GetInput()
        {
int limit;
int.TryParse(Console.ReadLine(), out limit);
if(limit <= 0)
        {
Console.WriteLine("Invalid number\nApplication will terminate ...");
Console.ReadLine();
Environment.Exit(0);
        }
return limit;
        }
        #endregion

        #region IsPrime
public static bool IsPrime(int num)
        {
if(num == 2 || num == 3) return true;
if(num % 2 == 0 || num % 3 == 0) return false;

int i = 5, w = 2;

while(i * i <= num)
        {
if(num % i == 0)
return false;

            i += w;
            w = 6 - w;
        }
return true;
        }
        #endregion

        #region Main
static void Main(string[] args)
        {
            #region Console Modification
Console.Title = "Prime number generator";
Console.BufferHeight = Int16.MaxValue - 1;
Console.WindowWidth = Console.LargestWindowWidth / 2;
Console.WindowHeight = Console.LargestWindowHeight / 2;
Console.BackgroundColor = ConsoleColor.DarkBlue;
Console.ForegroundColor = ConsoleColor.Green;
Console.Clear();
            #endregion

```

```

        #region Get Input and check if prime
Console.Write("Enter a number to check whether it is a prime or not: ");
int numm = GetInput();

if(IsPrime(numm)) Console.WriteLine("It is a prime");
else Console.WriteLine("It is not a prime");
        #endregion

        #region Get Input and create new objects
Console.Write("Enter a limit up to which to search for prime numbers: ");
int limit = GetInput();

        Stopwatch sw = new Stopwatch();
        Sieve_Of_Lazar SoL = new Sieve_Of_Lazar(limit);
        Sieve_of_Atkin SoA = new Sieve_of_Atkin(limit);
        Sieve_of_Eratosthenes SoE = new Sieve_of_Eratosthenes(limit);
        #endregion

        #region Search by Sieve of Lazar
Console.Write("Searching primes with Sieve of Lazar ...");
sw.Start();
SoL.FindAllPrimesTilllimit();
sw.Stop();
Console.WriteLine("\rSieve of Lazar found {0} prime number{1} in:\t\t" + sw.Elapsed,
SoL.NumberOfPrimes.ToString("N0"), (SoL.NumberOfPrimes > 1) ? "s" : "");
sw.Reset();
        #endregion

        #region Search by Sieve of Atkin
Console.Write("Searching primes with Sieve of Atkin ...");
sw.Start();
SoA.FindAllPrimesTilllimit();
sw.Stop();
Console.WriteLine("\rSieve of Atkin found {0} prime number{1} in:\t\t" + sw.Elapsed,
SoA.NumberOfPrimes.ToString("N0"), (SoA.NumberOfPrimes > 1) ? "s" : "");
sw.Reset();
        #endregion

        #region Search by Sieve of Erathosthenes
Console.Write("Searching primes with Sieve of Erathosthenes ...");
sw.Start();
SoE.FindAllPrimesTilllimit();
sw.Stop();
Console.WriteLine("\rSieve of Erathosthenes found {0} prime number{1} in:\t\t" + sw.Elapsed +
Environment.NewLine, SoE.NumberOfPrimes.ToString("N0"), (SoE.NumberOfPrimes > 1) ? "s" : "");
        #endregion

        #region Check if number is prime
Console.Write("Enter a number to check whether it is a prime or not: ");
int num = GetInput();
if(num <= limit)
if(SoA.IsPrime(num))
Console.WriteLine("Number is prime.");
else Console.WriteLine("Number is not prime.");
else Console.WriteLine("Invalid number.");
        #endregion

        #region Print primes
Console.Write("Print elements? [y/n] ");
if(Console.Read() == 'y')
{
sw.Reset();
sw.Start();
SoA.PrintPrimes();
sw.Stop();
Console.WriteLine(Environment.NewLine + "Time spent printing: " + sw.Elapsed);
Console.WindowTop = 0;
}
else Console.WriteLine("Thank you. bye");
        #endregion

Console.ReadKey();
}
#endregion

```

```

    }
}

```

### 3.4 Sorting Code

```

using System;

namespace Sorting
{
    public class Sort
    {
        #region Insertion sort
        /// <summary>
        /// Sort array by insertion sort
        /// </summary>
        /// <param name="arr">Array of unsorted elements</param>
        public static void Insertion(int[] arr)
        {
            for (int i = 1; i < arr.Length; i++)
            {
                int key = arr[i];
                int j = i - 1;
                while(j >= 0 && arr[j] > key)
                {
                    arr[j + 1] = arr[j];
                    j--;
                }
                arr[j + 1] = key;
            }
        }
        #endregion
        #region Shell Sort
        public static void Shell(int[] arr)
        {
            int n = arr.Length;
            int increment = n / 2;
            while (increment > 0) {
                int last = increment;
                while (last < n) {
                    int current = last - increment;
                    while (current >= 0) {
                        if (arr[current] > arr[current + increment]) {
                            //swap
                            int tmp = arr[current];
                            arr[current] = arr[current + increment];
                            arr[current + increment] = tmp;
                            current -= increment;
                        }
                        else { break; }
                    }
                    last++;
                }
                increment /= 2;
            }
        }
        #endregion
    }
}

```

```

using System;
using System.Diagnostics;

namespace Sorting
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.BufferHeight = Int16.MaxValue - 1;

```

```

        Random r = new Random();
int[] arrayToBeSortByInsertion = new int[16384];
int[] arrayToBeSortByShell = new int[16384];

        //generating random numbers
for (int i = 0; i < arrayToBeSortByInsertion.Length; i++) {
    arrayToBeSortByInsertion[i] = r.Next();
}
Console.WriteLine("Sorting array by insertion!");
Sort.Insertion(arrayToBeSortByInsertion);

Console.WriteLine("Array sorted! Printing values:");
for (int i = 0; i < arrayToBeSortByInsertion.Length; i++) {
    Console.WriteLine(arrayToBeSortByInsertion[i] + ",");
}

        //generating random numbers
for (int i = 0; i < arrayToBeSortByShell.Length; i++)
{
    arrayToBeSortByShell[i] = r.Next();
}
Console.WriteLine("Sorting array by Shell");
Sort.Shell(arrayToBeSortByShell);
Console.WriteLine("Array sorted, printing elements");
foreach (int item in arrayToBeSortByShell) {
    Console.WriteLine(item + ",");
}

Console.ReadKey();

    }
}

```

### 3.5 Square root Code

```

using System;

namespace SquareRootAproximation
{
    class Program
    {
        static void Main(string[] args)
        {
            double guess, value, epsilon = 1.0e-9;

            Console.WriteLine("Enter a number root");
            if(double.TryParse(Console.ReadLine(), out value)) {
                Console.WriteLine("Enter a guess");
                if(double.TryParse(Console.ReadLine(), out guess)) {
                    double result = ((value / guess) + guess) / 2;

                    while(Math.Abs(result - guess) > epsilon) {
                        guess = result;
                        result = ((value / guess) + guess) / 2;
                    }
                    Console.WriteLine("The approx sqrt of {0} is {1}", value, result);
                }
                else Console.WriteLine("Invalid Number");
            }
            else Console.WriteLine("Invalid Number");

            Console.ReadKey();
        }
    }
}

```

### 3.6 Matrix multiplication Code

```

using System;

namespace MatrixMult
{
    class Program
    {
        static readonly short MAX = 16;

        #region Main
        static void Main(string[] args)
        {
            Console.WindowWidth = Console.LargestWindowWidth - 90;
            Console.WindowHeight = Console.LargestWindowHeight - 25;
            Console.BackgroundColor = ConsoleColor.Black;
            Console.ForegroundColor = ConsoleColor.Gray;
            Console.Clear();

            Random r = new Random();

            short[,] matrix1 = new short[MAX, MAX];
            short[,] matrix2 = new short[MAX, MAX];
            short[,] matrixResult = new short[MAX, MAX];

            for(short i = 0; i < MAX; i++) {
                for(short j = 0; j < MAX; j++) {
                    matrix1[i, j] = (short)r.Next(short.MinValue, short.MaxValue);
                    matrix2[i, j] = (short)r.Next(short.MinValue, short.MaxValue);
                }
            }
            matrixResult = Mult(matrix1, matrix2, matrixResult);

            Console.WriteLine("First matrix: ");
            Print(matrix1);

            Console.WriteLine("\nSecond matrix: ");
            Print(matrix2);

            Console.WriteLine("\nResult matrix: ");
            Print(matrixResult);

            Console.ReadKey();
        }
        #endregion

        #region Mult
        /// <summary>
        /// Method that multiplies two 2D matrices 16x16
        /// </summary>
        /// <param name="matrix1">First matrix 16x16</param>
        /// <param name="matrix2">Second matrix 16x16</param>
        /// <param name="matrix3">Matrix to be stored result</param>
        /// <returns>Returns 16x16 matrix with result</returns>
        public static short[,] Mult(short[,] matrix1, short[,] matrix2, short[,] matrix3)
        {
            for(short i = 0; i < MAX; i++) {
                for(short j = 0; j < MAX; j++) {
                    short result = 0;
                    for(short k = 0; k < MAX; k++) {
                        result += (short)(matrix1[i, k] * matrix2[k, j]);
                    }
                    matrix3[i, j] = result;
                }
            }
            return matrix3;
        }
        #endregion

        #region Print
        /// <summary>
        /// Method that prints the matrices
        /// </summary>
        /// <param name="arr">Matrix to be printed</param>
        public static void Print(short[,] arr)

```



```

    {
    for(short i = 0; i < MAX; i++) {
    for(short j = 0; j < MAX; j++) {
    Console.WriteLine(String.Format("{0,9}", arr[i, j]));
    }
    Console.WriteLine();
    }
    }
    #endregion
}
}

```

### 3.7 Sum of odd numbers Code

```

using System;

namespace Sum
{
    class Program
    {
        #region Main
        static void Main(string[] args)
        {
            int lowerLimit, higherLimit;

            GetInput(out lowerLimit, out higherLimit);

            Console.WriteLine("Result of sum iteratively from {0} to {1} is {2}", lowerLimit, higherLimit,
                SumIter(lowerLimit, higherLimit));

            Console.WriteLine("Result of sum recursively from {0} to {1} is {2}", lowerLimit, higherLimit,
                SumRec(lowerLimit, higherLimit));

            Console.ReadLine();
        }
        #endregion

        #region SumRec
        /// <summary>
        /// Summation of all odd numbers between 2 odd numbers recuresevely
        /// </summary>
        /// <param name="lowerLimit">Lower limit of odd number</param>
        /// <param name="higherLimit">Upper limit of odd number</param>
        /// <returns>Sum of numbers</returns>
        public static int SumRec(int lowerLimit, int higherLimit)
        {
            if(lowerLimit > higherLimit)
                return 0;

            return lowerLimit + SumRec(lowerLimit + 2, higherLimit);
        }
        #endregion

        #region SumIter
        /// <summary>
        /// Summation of all odd numbers between 2 odd numbers iteratively
        /// </summary>
        /// <param name="lowerLimit">Lower limit of odd number</param>
        /// <param name="higherLimit">Upper limit of odd number</param>
        /// <returns>Sum of numbers</returns>
        public static int SumIter(int lowerLimit, int higherLimit)
        {
            int result = 0;

            for(int i = lowerLimit; i <= higherLimit; i += 2)
                result += i;

            return result;
        }
    }
}

```

```

        #endregion

        #region GetInput
        /// <summary>
        /// Get input with input validation, can return only 2 odd number where first is smaller than
the second
        /// </summary>
        /// <param name="lowerLimit">Lower limit input</param>
        /// <param name="higherLimit">Upper limit input</param>
public static void GetInput(out int lowerLimit, out int higherLimit)
    {
        do {
            Console.Write("Please enter the lower limit odd number: ");
            if(!int.TryParse(Console.ReadLine(), out lowerLimit))
                continue;
            } while(lowerLimit % 2 == 0);

        do {
            Console.Write("Please enter the higher limit odd number(bigger than {0}): ", lowerLimit);
            if(!int.TryParse(Console.ReadLine(), out higherLimit))
                continue;
            } while(higherLimit % 2 == 0 || higherLimit <= lowerLimit);
        }
        #endregion
    }
}

```

### 3.8 Sine & Cosine Code

```

using System;
namespace SinCos
{
    class Program
    {
        #region Main
        static void Main(string[] args)
        {
            double angle;
            string ch;
            do
            {
                Console.Write("Please enter the angle in degrees: ");
                if(double.TryParse(Console.ReadLine(), out angle)) {
                    Console.WriteLine("Sin of {0} is " + Sin(angle, 20), angle);
                    Console.WriteLine("Cos of {0} is " + Cos(angle, 20), angle);
                }
            } else Console.WriteLine("Invalid number");

            Console.Write("Do you want to calculate another sin/cos?[y/n] ");
            ch = Console.ReadLine().ToLower();
            } while (ch[0] != 'n');
        }
        #endregion
        #region Factorial
        /// <summary>
        /// Calculate factorial of a number
        /// </summary>
        /// <param name="x">Actual number</param>
        /// <returns>Calculated factorial</returns>
public static double Fact(double x)
    {
        return (x <= 1 ? 1 : x * Fact(x - 1));
    }
        #endregion
        #region Sin
        /// <summary>
        /// Calculate the Sine of given number
        /// </summary>
        /// <param name="x">Actual number</param>

```

```

        /// <param name="n">Number of iterations</param>
        /// <returns>Sine of specified angle</returns>
public static double Sin(double x, int n)
{
    double result = 0;
    x *= (Math.PI / 180); // Converting degrees to radians
    for (byte i = 0; i < n; i++)
        result += (-1 & i | 1) * (Math.Pow(x, 2 * i + 1) / Fact(2 * i + 1));

    return Math.Round(result, 10);
}
#endregion
#region Cos
/// <summary>
/// Calculate the cosine of given number
/// </summary>
/// <param name="x">Actual number</param>
/// <param name="n">Number of iterations</param>
/// <returns>Cosine of specified angle</returns>
public static double Cos(double x, int n)
{
    double result = 0;
    x *= (Math.PI / 180); //Converting degrees to radians
    for (byte i = 0; i < n; i++)
        result += (-1 & i | 1) * (Math.Pow(x, 2 * i) / Fact(2 * i));

    return Math.Round(result, 10);
}
#endregion
}
}

```

### 3.9 Palindrome Code

```

using System;
using System.Text.RegularExpressions;

namespace Palindrome
{
    class Program
    {
        #region Main
        static void Main(string[] args)
        {
            Regex reg = new Regex("[a-zA-Z]+$");
            Console.WriteLine("Please enter a string to see if it is a palindrome:");
            string word = Console.ReadLine().ToLower().Replace(" ", string.Empty);

            while (!reg.IsMatch(word))
            {
                Console.WriteLine("Invalid string!\nEnter new one: ");
                word = Console.ReadLine().ToLower().Replace(" ", string.Empty);
            }
            Console.WriteLine("String validated.");

            if (IsPalindrome(word)) Console.WriteLine("String \"{0}\" is palindrome!", word);
            else Console.WriteLine("String \"{0}\" is not palindrome...", word);

            Console.ReadKey();
        }
        #endregion

        #region IsPalindrome
        /// <summary>
        /// Check whether the string is palindrome or not
        /// </summary>
        /// <param name="s">Actual string</param>
        /// <returns></returns>
        static bool IsPalindrome(string s)
        {
            for (int i = 0; i < s.Length / 2; i++)

```

```
if (s[i] != s[s.Length - 1 - i])  
    return false;  
    return true;  
        }  
        #endregion  
    }  
}
```