Table of Contents

[Problem 2](#_Toc52139669)

[Algorithm 2](#_Toc52139670)

[Flow Chart 2](#_Toc52139671)

[The Basics 4](#_Toc52139672)

[Write Messages to Console 4](#_Toc52139673)

[Reading Input from Console 5](#_Toc52139674)

[Wait for the User Input to Close Console 5](#_Toc52139675)

[Converting String to Number 5](#_Toc52139676)

[Order of Evaluation 6](#_Toc52139677)

[BODMAS 6](#_Toc52139678)

[String Functions 7](#_Toc52139679)

[Operators 7](#_Toc52139680)

# Problem

A user will provide two numbers and one of the four basic operations and then will see the result of the operation.

Operation

* Add
* Subtract
* Multiply
* Divide

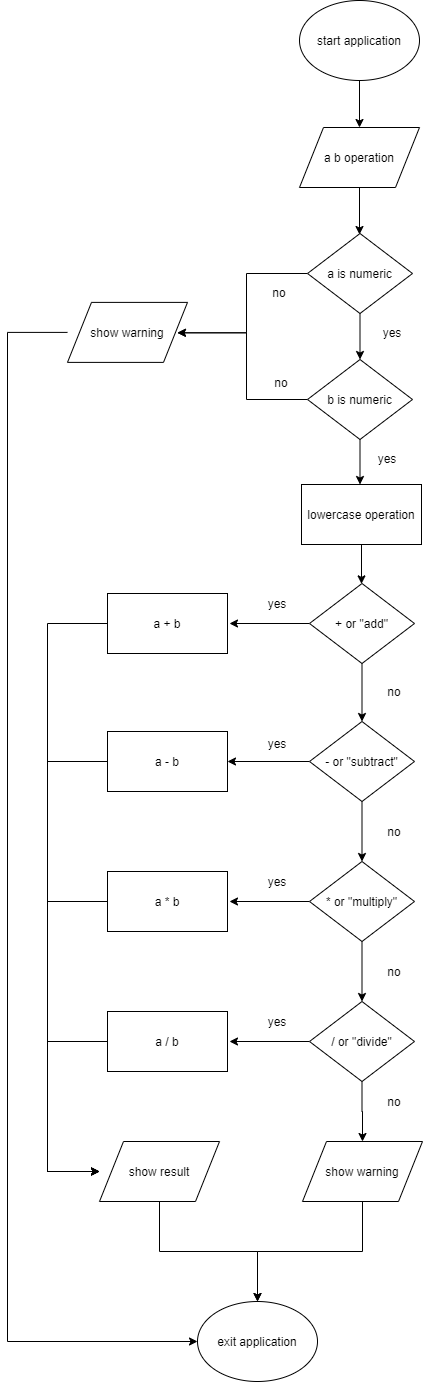
# Algorithm

1. First number = get input from the user
2. If first number not numeric then print warning
3. Second number = get input from the user
4. If second number is not numeric then print warning
5. Operation = get input from user
6. Lower case the operation for case insensitivity
7. If operation is + or “add” then
   1. Print first number + second number
8. If operation is - or “subtract” then
   1. Print first number – second number
9. If operation is \* or “multiply” then
   1. Print first number \* second number
10. If operation is / or “divide” then
    1. Print first number / second number
11. Otherwise
    1. Print warning

# Flow Chart

Flow chart is made with draw.io <https://app.diagrams.net/>.

Flowchart.xml is in the folder which you can open in draw.io or you can download the desktop app and open in it as well.



# The Basics

## Write Messages to Console

There is a helper method to write messages to console. The two main methods are write and writeLine. You can provide all the arguments or skip any.

/// <summary>

/// To display the message in the console with writeLine

/// </summary>

/// <param name="message">The message</param>

/// <param name="color">The message color</param>

/// <param name="isChangeToWhite">Change the color back to default white</param>

static void writeLine(string message, ConsoleColor? color, bool isChangeToWhite = false)

{

    changeForgroundColor(color);

    if (!string.IsNullOrWhiteSpace(message))

        Console.WriteLine(message);

    if (isChangeToWhite)

        changeForgroundColor(ColorWhite);

}

/// <summary>

/// To display the message in the console with write

/// </summary>

/// <param name="message">The message</param>

/// <param name="color">The message color</param>

/// <param name="isChangeToWhite">Change the color back to default white</param>

static void write(string message, ConsoleColor? color, bool isChangeToWhite = false)

{

    changeForgroundColor(color);

    if (!string.IsNullOrWhiteSpace(message))

        Console.Write(message);

    if (isChangeToWhite)

        changeForgroundColor(ColorWhite);

}

/// <summary>

/// Change forground color

/// </summary>

/// <param name="color">The color</param>

static void changeForgroundColor(ConsoleColor? color)

{

    if (color != null)

        Console.ForegroundColor = color.GetValueOrDefault();

}

And then a sample use would be like

private const ConsoleColor ColorYellow = ConsoleColor.Yellow;

private const ConsoleColor ColorRed = ConsoleColor.Red;

private const ConsoleColor ColorGreen = ConsoleColor.Green;

private const ConsoleColor ColorBlue = ConsoleColor.Blue;

private const ConsoleColor ColorWhite = ConsoleColor.White;

private const ConsoleColor ColorCyan = ConsoleColor.Cyan;

The display message examples are not in the code, follow this document to test.

static void Main(string[] args)

{

    writeLine("Starting Program", null);

    writeLine("Checking Permissions", null);

    writeLine("Permissions are correct", ConsoleColor.Cyan);

    writeLine("Pinging Server", ConsoleColor.White);

    writeLine("Warning:Pinging response is over 1 second", ConsoleColor.Yellow);

    writeLine("Checking database", ConsoleColor.White);

    writeLine("Error: Database not reachable", ConsoleColor.Red);

    double percentage = 0.40;

    writeLine($"Percentage = {percentage:0.0%}", ConsoleColor.Green);

    int sin = 526193526; //some SIN number, display as 526-193-526

    writeLine($"Some SIN = {sin:000-000-000}", ConsoleColor.Blue);

    long phone = 9051234560;

    writeLine($"Phone = {phone:(000) 000-0000}", ConsoleColor.Green);

    double cur = 45610.05;

    writeLine($"Currency = {cur:c}", ConsoleColor.Blue);

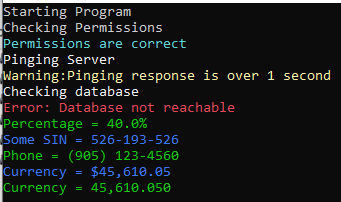
    writeLine($"Currency = {cur:n3}", ConsoleColor.Green);

    writeLine(string.Empty, ConsoleColor.White);

    Console.ReadKey();

}

*Important:* see the use of the format with the $ symbol. We are not writing String.Format anymore, even though it is still valid.



## Reading Input from Console

Use Console.ReadLine()

## Wait for the User Input to Close Console

Console.ReadKey();

## Converting String to Number

static void Main(string[] args)

{

    write("Enter a number: ", ColorYellow);

    string input = Console.ReadLine();

    //if the input is string then the converted value will be 0

    int converted;

    int.TryParse(input, out converted);

    writeLine($"Converted to number as {converted}", ColorGreen);

    Console.ReadKey();

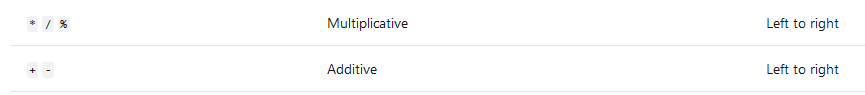
}



## Order of Evaluation

Take a look here for more details

<https://docs.microsoft.com/en-us/cpp/c-language/precedence-and-order-of-evaluation?view=vs-2019>



### BODMAS

Or look at here for BODMAS

<https://www.mathsisfun.com/operation-order-bodmas.html>

B: **B**rackets first

O: **O**rders (i.e. Powers and square roots, etc)

DM: **D**ivision and **M**ultiplication (left to right)

AS: **A**ddition and **S**ubtraction (left to right)

**Example 1**: How do you work out 3 + 6 × 2 ?

Multiplication before Addition

First 6 × 2 = 12, then 3 + 12 = 15

**Example 2**: How do you work out (3 + 6) × 2 ?

Brackets first

First (3 + 6) = 9, then 9 × 2 = 18

**Example 3**: How do you work out 12 / 6 × 3 / 2 ?

Multiplication and Division rank equally, so just go left to right:

First 12 / 6 = 2, then 2 × 3 = 6, then 6 / 2 = 3

**Example 4**: Sam threw a ball straight up at 20 meters per second, how far did it go in 2 seconds?

Sam uses this special formula that includes gravity:

height = velocity × time − (1/2) × 9.8 × time2

Sam puts in the velocity of 20 meters per second and time of 2 seconds:

height = 20 × 2 − (1/2) × 9.8 × 22

Now for the calculations

Start with: 20 × 2 − (1/2) × 9.8 × 22

Brackets first: 20 × 2 − 0.5 × 9.8 × 22

Then Orders: (22=4):20 × 2 − 0.5 × 9.8 × 4

Then the Multiplies: 40 − 19.6

Subtract and DONE: 20.4

The ball reaches 20.4 meters after 2 seconds

## String Functions

1. soemthinstring.Length: gives the length of the string
2. someString[8]: gives the character at index location 8, this is actually 9th characters since index is 0 based
3. someString.Substring(8,4): pick the section so the string starting at index location 8, but actually the 9th character. The second parm 3 means pick 4 characters.
4. someString.ToUpper(): converts the string to upper case
5. someString.ToLower(): converts the string to lower case
6. var someString = string.Empty; : creates an empty string
7. concatenating string creates a new string in memory. The better approach is using StringBuilder. Check here for more details: <https://docs.microsoft.com/en-us/dotnet/standard/base-types/stringbuilder>
8. someString.replace(“text”, “string”): it will replace the word text with string.

Above are a few string methods that are used widely. For a complete list and more reading about strings, take a look at this: <https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/strings/>

## Operators

1. Ternary Operator which is also referred as single line if statement. It is a conditional statement, allow us to change the flow of the program. It is used as ? and : . Here is a small example:

is this condition true ? yes : no.

Traditional if else statement

var x = "something";

var y = string.Empty;

if (x == "something")

{

y = "inside if";

}

else

{

y = "inside else";

}

Whereas the ternary would be like

y = x == "something" ? "inside if" : "inside else";

1. ! (null-forgiving) operator

Check this for more details: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-forgiving>

It got introduced in C# 8

This operator basically turns off warnings, or turns off the compiler null checks.

Assuming this definition:

class Person

{

public string? MiddleName;

}

The usage would be:

void LogPerson(Person person)

{

Console.WriteLine(person.MiddleName.Length); // WARNING: may be null

Console.WriteLine(person.MiddleName!.Length); // No warning

}

This is an anti-pattern try to avoid using it.

1. ?? and ??= operators

Take a look at this resource for more information: <https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/null-coalescing-operator>

The null-coalescing operator ?? returns the value of its left-hand operand if it isn't null; otherwise, it evaluates the right-hand operand and returns its result. The ?? operator doesn't evaluate its right-hand operand if the left-hand operand evaluates to non-null.

**Example 1:**

int? a = null;

int b = a ?? -1;

Console.WriteLine(b); // output: -1

**Example 2:**

Traditional way

if (variable is null)

{

variable = expression;

}

With the following code

variable ??= expression;

1. Null conditional operator ?.

static void Main(string[] args)

{

    double SumNumbers(List<double[]> setsOfNumbers, int indexOfSetToSum)

    {

        return setsOfNumbers?[indexOfSetToSum]?.Sum() ?? double.NaN;

    }

    var sum = SumNumbers(null, 0);

    writeLine(sum.ToString(), ColorRed);  // output: NaN

    Console.ReadKey();

}



1. :: operator name space alias operator

Take a look at this page for more details: <https://docs.microsoft.com/en-us/previous-versions/visualstudio/visual-studio-2012/c3ay4x3d(v=vs.110)?redirectedfrom=MSDN>

# Helper Classes

We’ll create the three helper classes to do everything. All of these classes have been created in the root. The code in the classes has been properly commented for readability purposes.

## ConsoleEngine

This will be used to read the input from the console. Also, all the current functionality has been moved to this engine.

## InputConverter

This will convert the input to number.

## CalculatorEngine

This will be our calculator engine

# Unit Tests

We’ll create a MS unit test project inside our solution with a name “SC.SimpleCalculator.Test.Unit”.

Also, add a reference to “SC.SimpleCalculator” in the unit test project.

## CalculatorEngineTest

Then create a new unit test class “CalculatorEngineTest”.

Take a look at the test class for more information.

I have added two tests to it

* AddTwoNumbersTest
* AddTwoNumbersSymbolTest

You can add additional tests for the rest of the operations.

Now run the unit test by clicking Test menu and then clicking Run Unit Tests.

## InputConverterTest

Then create a new unit test class “InputConverterTest”.

Take a look at the test class for more information.

I have added two tests to it

* ConvertValidStringToDoubleTest
* FailsToConvertInvalidNumberTest

You can add additional tests for the rest of the operations.

Now run the unit test by clicking Test menu and then clicking Run Unit Tests.