

Visual Basic

Programming and Design Principles

with Visual Studio

Ray O'Connor



Using Visual Studio

History of programming languages

A programming language is a formal constructed language designed to communicate instructions to a machine, particularly a computer. Programming languages can be used to create programs to control the behavior of a machine or to express algorithms.

Evolution of Programming Language

- <http://www.howtogeek.com/94917/the-evolution-of-computer-programming-languages-infographic/>

Procedural programming – languages such as Pascal, Fortran, Cobol, etc

Object Oriented Programming – OOP programming languages such as Java, Visual Basic, Python, etc

Scripting languages are programming languages that don't require an explicit compilation step.

For example, in the normal case, you have to compile a C program before you can run it. But in the normal case, you don't have to compile a JavaScript program before you run it. So JavaScript is sometimes called a "scripting" language.

Syntax of a programming language

Syntax refers to the spelling and grammar of a programming language. Computers are inflexible machines that understand what you type only if you type it in the exact form that the computer expects. The expected form is called the syntax.

Programming languages - what they have in common

- e.g. strict syntax rules, data storage, input statements, output statements, branching, looping.
- Planning Phase – Design Phase – Testing Phase

Programming languages - their differences

- e.g. different syntax, different structures, different focus
- procedural versus object oriented (OOP)

Video <https://www.youtube.com/watch?v=qmksVfulV0o>

Machine code is a computer programming language consisting of binary or hexadecimal instructions which a computer can respond to directly.

Low-level programming language

In computer science, a low-level programming language is a programming language that provides little or no abstraction from a computer's instruction set architecture—commands or functions in the language map closely to processor instructions. Generally this refers to either machine code or assembly language.

Low-level languages are useful because written in them can be crafted to run very fast and with a very small memory footprint. However, they are considered more difficult to utilize because they require a deeper knowledge of machine language.

Languages such as C and C++ are considered "lower-level" — they provide a minimal amount of abstraction at the smallest possible cost to performance and efficiency. These abstractions, such as classes, lambda functions and macros, allow programmers to use complex functionality without writing overly complex code. For this reason, lower-level languages are used in projects where abstractions are necessary to keep code highly readable and maintainable, but where maximum performance is still paramount. Many operating systems and high-frame rate computer games are a good example of this.

High level programming languages

A high-level language is a computer programming language that isn't limited by the computer, designed for a specific job, and is easier to understand. It is more like human language and less like machine language. However, for a computer to understand and run a program created with a high-level language, it must be compiled into machine language.

The first high-level languages were introduced in the 1950's. Today, there are many high-level languages in use, including BASIC, C, C++, Cobol, FORTRAN, Java, Pascal, Perl, PHP, Python, Ruby, and Visual Basic.

Difference between Low-Level & High-Level Language

High-level Language

- 1. Learning** - High-level languages are easy to learn.
- 2. Understanding** – High level languages are near to human languages.
- 3. Execution** - Programs in high-level languages are slow in execution.
- 4. Modification** - Programs in high-level languages are easy to modify.
- 5. Facility at hardware level** - High-level languages do not provide much facility at hardware level.
- 6. Knowledge of hardware Deep** - Knowledge of hardware is not required to write programs.
- 7. Uses** - These languages are normally used to write application programs.

Low-level languages

- 1. Learning** - Low-level languages are difficult to learn.
- 2 Understanding** - Low-level languages are far from human languages.
- 3. Execution** - Programs in low-level languages are fast in execution.
- 4. Modification** - Programs in low-level languages are difficult to modify.
- 5. Facility at hardware level** - Low-level languages provide facility to write programs at hardware level.
- 6. Knowledge of hardware Deep** - Deep knowledge of hardware is required to write programs.
- 7. Uses** - These languages are normally used to write hardware programs.

What is an Algorithm?

- A process or set of rules to be followed in calculations or other problem-solving operations, especially by a computer.

Notes

- [0 Program Algorithms.ppt](#) (Powerpoint file in Software Development - 2_Programming_Design_Principles folder on hard drive)
- [10 Algorithms.pdf](#) (in Software Development - 2_Programming_Design_Principles folder on hard drive)
- <https://www.khanacademy.org/computing/computer-science/algorithms>

Creating an algorithm

For example how do we find the biggest number in the list 2,7,8,34,29,11. The biggest number is 34. We need to apply some logical thinking to solve this.

Sample algorithm

- Store first list item in a variable Biggest
- For each item in the list
 - Store item in variable Current
 - If Current > Biggest then
 - Biggest = Current

Good, logical programming is developed through good pre-code planning and organization. This is assisted by the use of pseudocode and program flowcharts.

Flowcharts are written with program flow from the top of a page to the bottom. Each command is placed in a box of the appropriate shape, and arrows are used to direct program flow. The following shapes are often used in flowcharts:

Sample Algorithm

Step 1: Start

Step 2: Create a variable to receive the user's email address

Step 3: Clear the variable in case it's not empty

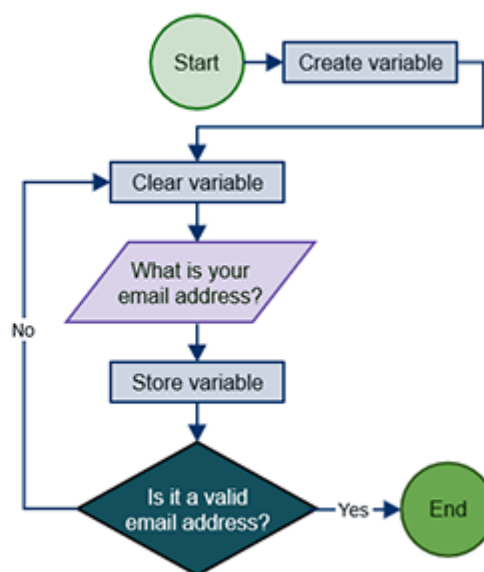
Step 4: Ask the user for an email address

Step 5: Store the response in the variable

Step 6: Check the stored response to see if it is a valid email address

Step 7: Not valid? Go back to Step 3.

Step 8: End





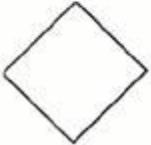
An oval indicates beginning or end of a program.



A parallelogram is a point where there is input to or output from the program.



A rectangle indicates the assignment of a value to a variable, constant, or parameter. The assigned value can be the result of a computation. The computation would also be included in the rectangle.



A diamond indicates a point where a decision is made.



An open-ended rectangle contains comment statements. The comment is connected to the program flow via a dashed line.



A hexagon indicates the beginning of a repetition.



The double-lined rectangle indicates the use of an algorithm specified outside the program, such as a subroutine.



Circles can be used to combine flow lines.



Arrows indicate the direction and order of program execution.

Pseudocode is a method of describing computer algorithms using a combination of natural language and programming language. It is essentially an intermittent step towards the development of the actual code. It allows the programmer to formulate their thoughts on the organization and sequence of a computer algorithm without the need for actually following the exact coding syntax. Although pseudocode is frequently used there are no set of rules for its exact implementation. In general, here are some rules that are frequently followed when writing pseudocode:

- The usual Fortran symbols are used for arithmetic operations (+, -, *, /, **).
- Symbolic names are used to indicate the quantities being processed.
- Certain Fortran keywords can be used, such as PRINT, WRITE, READ, etc.
- Indentation should be used to indicate branches and loops of instruction.

Here is an example problem, including a flowchart, pseudocode, and the final Fortran 90 program. This problem and solution are from Nyhoff, pg 206:

For a given value, *Limit*, what is the smallest positive integer *Number* for which the sum

$$Sum = 1 + 2 + \dots + Number$$

is greater than *Limit*. What is the value for this *Sum*?

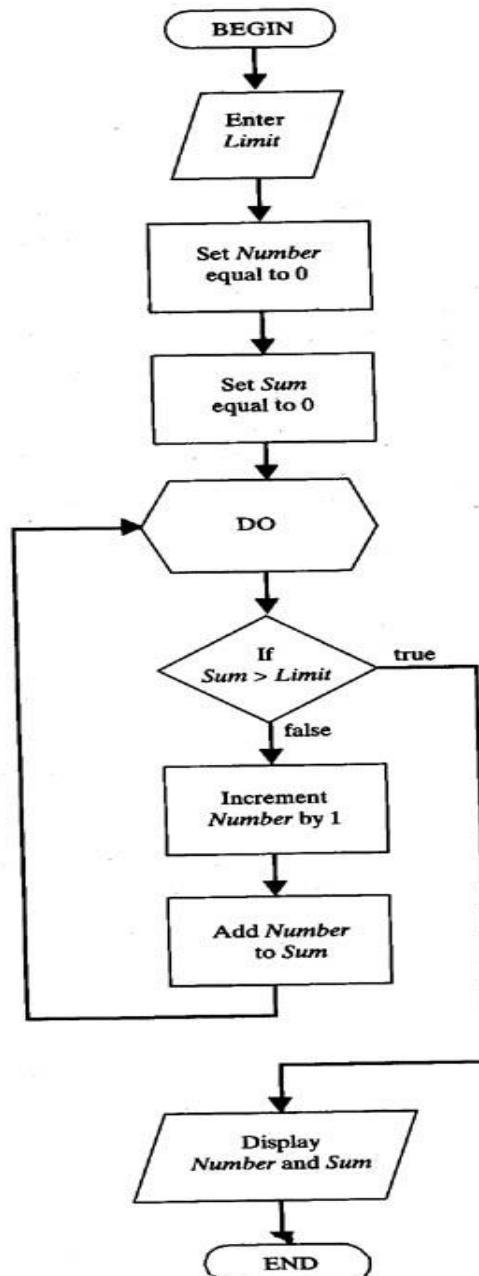
Pseudocode:

Input: An integer *Limit*

Output: Two integers: *Number* and *Sum*

1. Enter *Limit*
2. Set *Number* = 0.
3. Set *Sum* = 0.
4. Repeat the following:
 - a. If *Sum* > *Limit*, terminate the repetition, otherwise.
 - b. Increment *Number* by one.
 - c. Add *Number* to *Sum* and set equal to *Sum*.
5. Print *Number* and *Sum*.

Flowchart:



Fortran 90 source code:

PROGRAM Summation

! Program to find the smallest positive integer Number
! For which Sum = 1 + 2 + ... + Number
! is greater than a user input value Limit.

IMPLICIT NONE

! Declare variable names and types

INTEGER :: Number, Sum, Limit

! Initialize Sum and Number

Number = 0

Sum = 0

! Ask the user to input Limit

PRINT *, "Enter the value for which the sum is to exceed:"

READ *, Limit

! Create loop that repeats until the smallest value for Number is found.

DO

IF (Sum > Limit) EXIT ! Terminate repetition once Number is found

! otherwise increment number by one

Number = Number + 1

Sum = Sum + 1

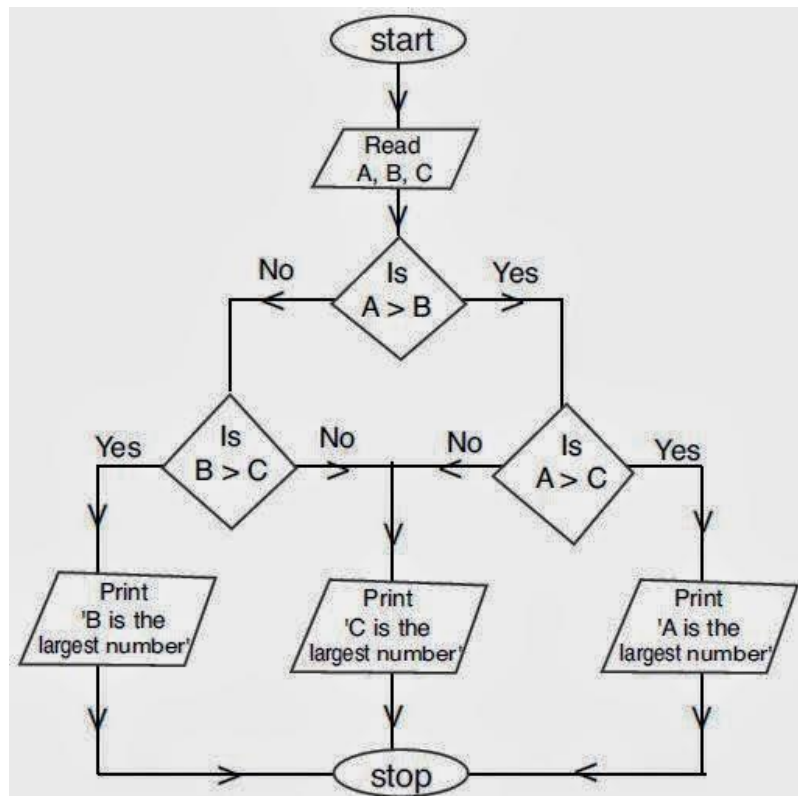
END DO

! Print the results

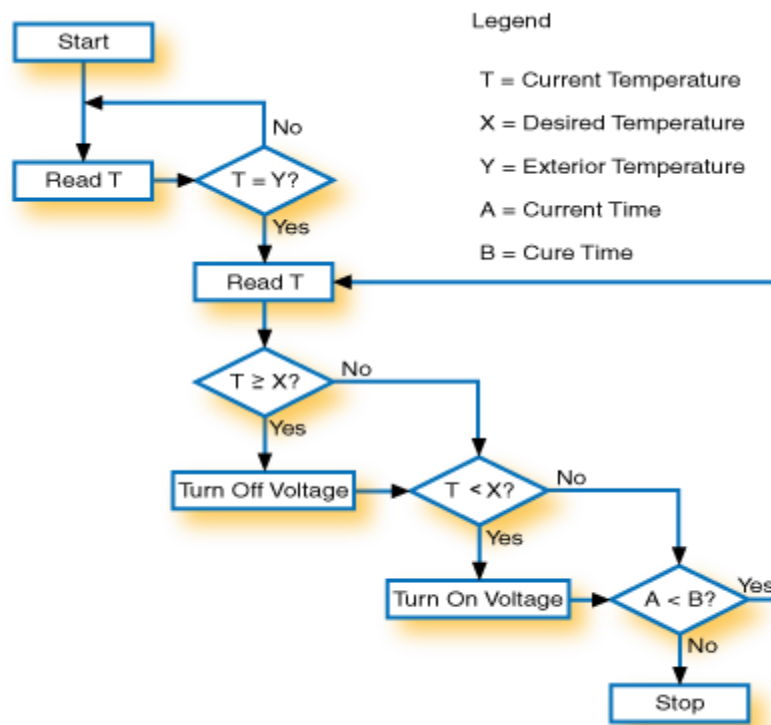
PRINT *, "1 + ... + ", Number, "=", Sum, ">", Limit

END PROGRAM

Sample Flowchart



Sample Flowchart



Programming Algorithms

General form of a computer algorithm:

INPUT → PROCESSING → OUTPUT

Algorithm types: pseudo code & flowchart.

Pseudo code is the steps of an algorithm written in concise natural language for problem solving.

Example: **Ohms law**.

1. Start.
2. Input resistor R & voltage V values.
3. Compute current $I = V/R$.
4. Output I (Current).
5. Stop.

Visual Basic	https://en.wikipedia.org/wiki/Visual_Basic
Statistics	http://www.langpop.com

Resources

- Google
- MOOCs (eg Udacity, Coursera)
- <https://www.thenewboston.com/videos.php?cat=39>
- https://www.youtube.com/watch?v=kKimJGA2grI&list=PL2Rdg_g4Jx7hdAFPtEC1p9CDIB4-PCDMM

Free eBooks and Additional Resources

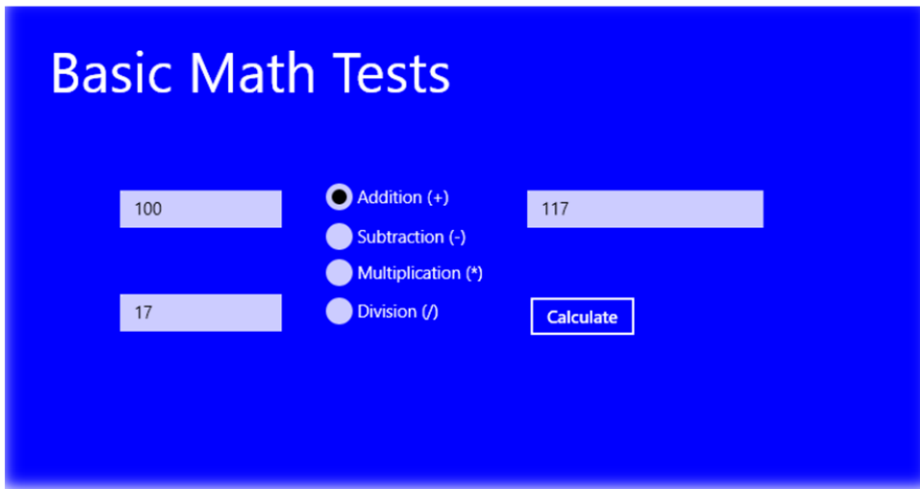
- <https://mitseu.files.wordpress.com/2014/08/ms-microsoft-visual-basic-2013-step-by-step.pdf>
- <http://www.onlineprogrammingbooks.com/>

Getting started

- Download Visual Studio – An Integrated Development Environment (IDE)
- Install Visual Studio – when you run Visual Studio for the first time be sure you select Visual Basic settings

Beginning New Project

- Sketch your screen layout (planning and design phase is important)
- Begin Visual Studio



```
Private Sub Calculate_Click(sender As Object, e As RoutedEventArgs) Handles Calculate.Click
```

```
'Assign textbox values to variables
```

```
firstNum = FirstTextBox.Text
```

```
secondNum = SecondTextBox.Text
```

```
'Determine checked button and calculate
```

```
If Addition.IsChecked Then
```

```
Result.Text = firstNum + secondNum
```

```
End If
```

```
If Subtraction.IsChecked Then
```

```
Result.Text = firstNum - secondNum
```

```
End If
```

```
If Multiplication.IsChecked Then
```

```
Result.Text = firstNum * secondNum
```

```
End If
```

```
If Division.IsChecked Then
```

```
Result.Text = firstNum / secondNum
```

```
End If
```

```
End Sub
```


Useful Tutorials
