158.212
Application Software Development

Suriadi Suriadi

Lecture 2

Basic Programming:

- Flow Control
- Conditions
- Comparisons
- If/else statements
- While loops
- For loops

Recap

- We have looked at basic instructions used in a program:
  - data types and variables,
  - assignments,
  - calculations,
  - and conversions.

- Instructions need to be `orchestrated' to form a logical sequence of actions depending on some conditions
  - Need `flow control'

Flow Control

- Flow control allows us to execute the desired set of instructions depending on some conditions.

- This condition may depend on user input or the result of some previous calculation.

Conditional/Comparison Operators

To check for conditions, any flow control statements need to use conditional operators. These operators compare the values of two variables. These operators are:

VB Operators	C# Operators	Meaning
=	==	Equal to
>	>	Greater than
<	<	Less than
>=	>=	Greater than or equal to
<=	<=	Less than or equal to
<>	!=	Not equal to

VB Logical Operators

We can make more complicated comparisons by using logical operators. These operators are:

VB Operators	C# Operators	Meaning
And	&&	Both must be true
Or	||	Either one or both must be true
Xor	^	One, but *not both*, must be true
Not	!	Reverses Condition

Example

VB	C#
`Dim i1 as Integer = 10`	`int i1 = 10;`
`Dim i2 as Integer = 5`	`int i2 = 5;`
`(i1 = i2)`	`(i1 == i2)`
`(i1 > i2)`	`(i1 > i2)`
`(i1 < i2)`	`(i1 < i2)`
`(i1 >= i2)`	`(i1 >= i2)`
`(i1 <= i2)`	`(i1 <= i2)`
`(i1 <> i2)`	`(i1 != i2)`

Example

VB

```
Dim i1 as Integer = 10
Dim i2 as Integer = 10

(i1 = i2)
(i1 > i2)
(i1 < i2)
(i1 >= i2)
(i1 <= i2)
(i1 <> i2)
```

C#

```
int i1 = 10;
int i2 = 10;

(i1 == i2)
(i1 > i2)
(i1 < i2)
(i1 >= i2)
(i1 <= i2)
(i1 != i2)
```

Example

VB

```
Dim i1 as Integer = 10
Dim i2 as Integer = 5
Dim i3 as Integer = 5

((i1 = i2) And (i1 = i3))
((i1 > i2) And (i1 > i3))
((i1 > i2) And (i2 = i3))
((i1 > i2) Or (i1 > i3))
((i1 > i2) Xor (i1 > i3))
Not((i1 > i2) And (i1 > i3))
```

C#

```
int i1 = 10;
int i2 = 5;
int i3 = 5;

((i1 == i2) && (i1 == i3))
((i1 > i2) &&(i1 > i3))
((i1 > i2) &&(i2 == i3))
((i1 > i2) || (i1 > i3))
((i1 > i2) ^ (i1 > i3))
!((i1 > i2) && (i1 > i3))
```

If statements

If statements allow a set of instructions to be executed depending on the condition. These statements follow the pattern:

VB

```
If condition1 Then
     statements
End if
```

C#

```
if(condition1) {
       statements
}
```

If statements

VB

```
Dim i1 As Integer = 10
Dim i2 As Integer = 5
If i1 = i2 Then
  Console.Write (“equal”)
End if

If i1 > i2 Then
  Console.Write (“greater”)
End if

If i1 < i2 Then
  Console.Write (“less”)
End if
```

C#

```
int i1 = 10;
int i2 = 5;
if(i1 == i2) {
  Console.Write (“equal”);
}

if(i1 > i2) {
 Console.Write (“greater”);
}

if(i1 < i2) {
 Console.Write (“less”);
}
```

If/Else statements

The if/else statement allows one of two instructions paths to be taken depending on the result of a condition. These statements follow the pattern:

VB

```
If condition1 Then
    statements
Else
    statements
End if
```

C#

```
if(condition1) {
        statements
} else {
        statements
}
```

If/Else statements

Example:

```
Dim i1 As Integer = 10
Dim i2 As Integer = 5

If i1 = i2 Then
  Console.Write("equal")
Else
  Console.Write("not equal")
End if
```

```
int i1 = 10;
int i2 = 5;

if(i1 == i2) {
  Console.Write("equal");
} else {
  Console.Write("not equal");
}
```

If/Elseif statements

The if/else statement allows one of two instructions paths to be taken depending on the result of a condition. These statements follow the pattern:

VB

```
If condition1 Then
     statements
ElseIf condition2 Then
     statements
Else
     statements
End If
```

C#

```
if(condition1) {
        statements
} elseif(condition2) {
        statements
} else {
        statements
}
```

If/Elseif statements

Example:

```
Dim i1 As Integer = 10
Dim i2 As Integer = 5

If i1 > i2 Then
  Console.Write (“greater”)
ElseIf i1 < i2 Then
  Console.Write (“less”)
Else
  Console.Write (“equal”)
End if
```

```
int i1 = 10;
int i2 = 5;

if(i1 > i2) {
  Console.Write (“greater”);
} else if(i1 < i2) {
  Console.Write (“less”);
} else {
  Console.Write (“equal”);
}
```

If/Elseif statements

Conventions and best-practices:

- Booleans: do not use ‘flag’ as a name
    - Boolean variables should always imply true/false
    - Eg. if(flag1) **BAD**
      if(isDataReady) **GOOD**
    - Bad names: status, flag, negative names!
    - Good names: done, error, found, success, ok

- When experiencing “trying to figure out” a section of code – consider renaming **all** variable names

- Favour “read-time” over “write-time” convenience

If/Elseif statements

DOs and DON'Ts:

- Write the nominal path through the code first, then code the unusual cases
- Put the normal/expected case immediately after the "If" and not after the "Else". Error conditions should always go into the "Else"
- Careful to branch correctly on equality to prevent "off-by-one" errors eg. Using > instead of >=
- Sometimes an empty "Else" block with comments is useful to indicate that the case has been considered and deemed unnecessary.
- With nested and complicated "If/Elseif" statements, make sure ALL possibilities are covered
- For clarity, place complicated conditional statements into functions

Flow Control - Loops

- Another important method of controlling the flow of a program is a **loop**.

- Loops **repeatedly** perform the same set of instructions until they **terminate**.

- Loops are differentiated by “flexibility” and “location” of tests.

- We will look at the two main types of loops:

  While and **For**

While Loops

A While loop will continue executing as long as a certain condition is evaluated to true. They follow this form:

VB

```
While condition
     statements
End While
```

C#

```
while(condition1) {
       statements
}
```

While Loops

Example:

VB

```vb
Dim i1 As Integer = 0
Dim i2 As Integer = 5

While i1 < i2
     i1 = i1 + 1
End While
```

C#

```csharp
int i1 = 0;
int i2 = 5;

while(i1 < i2) {
        i1 = i1 + 1;
}
```

A Warning

Be careful with loops.

If your loop condition always evaluates to true you have an infinite loop. This loop will continue executing forever and your program will never finish.

If your program isn’t responding, you may have an infinite loop.

For Loops

For loops are most commonly used for looping over a pre-defined range.

VB

```
For i = start To end (step)
     statements
Next
```

C#

```
for(i = start; i <= end; i += step)
     statements
}
```

For Loops

For loops are most commonly used for looping over a pre-defined range.

VB

```
For i As Integer = 1 To 10
    statements
Next
```

C#

```
for(int i = 0; i < 10; i++)
    statements
}
```

For Loops

For loops are most commonly used for looping over a pre-defined range.

VB

```
For i As Integer = 1 To 100 Step 10
    statements
Next
```

C#

```
for(int i = 0; i < 100; i += 10){
    statements
}
```

For Loops – Best Practice

- Consider a while loop if there is a condition in the loop that jumps out – terminates the loop.
- Customary to use *i,j,k* as loop control/index variable names.
- Rename looping variable if it is used outside the loop.
- Use very meaningful names in long loops.
- In nested For loops, carefully names variables prevent the “index cross talk”
- Don’t monkey with the loop index to make the loop terminate

Do While

VB

```
Do
    statements
Loop While condition
```

C#

```
do {
    statements
} while (condition);
```

```
------------------------------------------------------------
Do While condition
    statements
Loop
------------------------------------------------------------
Do Until condition
    statements
Loop
------------------------------------------------------------
Do
    statements
Loop Until condition
```

Do While

VB

```
Dim i As Integer = 10
Do
    i = i - 1
Loop While I > 0
```

C#

```
int i = 10;
do {
    i = i - 1;
} while (i > 0);
```

Loops

Language	Type	Location
VB	For-Next	beginning
	While-End	beginning
	Do-Loop-While	beginning/end
	For Each	beginning
C#	for	beginning
	while	beginning
	do-while	end
	foreach	beginning

Loops and Issues

What goes wrong….frequently:

- Incorrect or omitted loop initializations
- Improper nesting
- Incorrect termination condition
- Indexing array elements incorrectly
- Loop not even starting or infinite loops

for vs. while vs. do-while ?

Loops – Best Practice

Remedies and best-practice:

- Minimize the factors that affect the loop (simplify)
- Treat the loop body like a black-box – keep control of it outside
- Make your loops short enough to view all at once
- Limit nesting loops-within-loops to three levels
- Move sections of the loop-body into functions/routines when the complexity becomes too high

Loops – Best Practice

Remedies and best-practice:

- Make long loops especially clear (a single clear and unmistakable exit, comments)
- Enter loop in one location only!
- Initialize the loop immediately before it – proximity principle
    - Keep loop 'housekeeping' chores at either beginning or end
- Use *while(True)* for infinite loops
- Revert to For loops from While when you find yourself forgetting to modify loop control variables at the bottom
    - Conversely, don't cram the For header – revert to While

Strings

In VB, Strings can be compared using the same comparison operators as numeric types. An equal or not equal test is easy to understand (based on the sorting order):

“hello” = “hello”	true
“hello” = “world”	false
“hello” < “world”	true
“hello” > “world”	false

Strings

In C#, Strings can be compared by using CompareTo().

“hello”.CompareTo(“hello”) == 0	true
“hello”.CompareTo(“world”) == 0	false
“hello”.CompareTo(“world”) < 0	true
“hello”.CompareTo(“world”) > 0	false

Summary

Basic Programming:

- Flow Control
- Conditions
- Comparisons
- If/else statements
- While loops
- For loops

Assignment

Some important functions for the assignment:

Console.Write()

Console.WriteLine()

Console.ReadLine()

Console.ReadKey()

Assignment

These functions write output to the Console.

```
Console.Write(“hello ”)
Console.Write(“world”)
```

```
hello world
```

```
Console.WriteLine(“hello ”)
Console.WriteLine(“world”)
```

```
hello
world
```

Assignment

These functions write output to the Console.

VB

```
Dim  i As Integer
i = Console.ReadLine()
Console.WriteLine(i)
```

C#

```
int i;
i = int.Parse(Console.ReadLine());
Console.WriteLine(i);
```

Use an array to store input values

VB

```
Dim inputValues(100) As Integer
inputValues(index) = value
Console.Write(inputValues(index))
```

C#

```
int[] inputValues = new int[100];
inputValues[index] = value;
Console.Write(inputValues[index])
```