COMP2012H Honors Object-Oriented Programming and Data Structures
Syntax Comparison between VBA and C++: Basics and Program Flow Control

The purpose of this set of notes is to help you quickly transfer your basic knowledge of VBA to that of C++. Please note that it is not a complete summary of our lecture notes. For all the C++ features discussed in COMP2012H, you have to carefully study the lecture notes on our course website.

In VBA	In C++

Hello Word Program

```
Private Sub Workbook_Open()
    MsgBox "Hello World!" ' Showing a message
End Sub
```

```
/*
 * File: hello_world.cpp
 * A common program used to demo a new language
 */
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello world" << endl;
    return 0;
}
```

Note: Every C++ program must have exactly one main() function which is the entry point of the program.

Executing a VBA code

- The event handler will be executed automatically when an event occurs.

- Or, run (apply) a macro

Executing a C++ program

1. compile the program:
   g++ -o hello_world.out hello_world.cpp

2. execute the program:
   hello_world.out

Basic Output

To show a message "abc" with a newline character:

```
MsgBox "abc"
```

To show a message "abc" with a newline character:

```
cout << "abc" << endl;
```

where endl means "end of the line".
Or,

```
cout << "abc\n";
```

Comments

- for one line of comment only:

  ```
  ' ...
  ```

- for one or more lines of comments:

  ```
  /* ... */
  ```

- for one line of comment only:

  ```
  // ...
  ```

Statements

- A statement is a line of code.

- Only those extra blanks and tabs are ignored.

- If the line of the statement is too long, one may break it into several lines using "_".

For example:

```
MsgBox "Hello" &  _
" world"
MsgBox "!"
```

- Each statement ends in a semicolon ";"

- Extra blanks, tabs, lines are ignored.

- More than one statement can be on one line.

- A statement may be spread over several lines.

For example:

```
cout << "Hello" <<
" world"; cout << "!" << endl;
```

Variables

- Basic Data Types:

  - Integer: `Integer`, `Long`
    Examples of values: `0`, `1`, `100`, `-101`, ...

  - Floating point: `Single`, `Double`
    Examples of values: `0.5`, `-123.908232`

  - String: `String`
    Examples of values: `"A"`, `"abc"`, `"comp 2012h"`, ...

  - Boolean: `Boolean`
    Examples of values: `True`, `False`

  - Variant: `Variant`

- Variables can be created for different types by using the keyword `Dim`.
  For examples:

  ```
  Dim num1 As Integer
  num1 =  100  ' Integer data type
  Dim num2 As Double
  num2 =  0.05 ' Double data type
  ```

- Basic Data Types:

  - Integer: `short`, `int`, `long`, `long long`, etc.
    Examples of values: `0`, `1`, `100`, `-101`, ...

  - Floating point: `float`, `double`, `long double`, etc.
    Examples of values: `0.5`, `-123.908232`

  - Character: `char`
    Examples of values: `'A'`, `'a'`, `'B'`, `'b'`, ...

  - Boolean: `bool`
    Examples of values: `true`, `false`

- Variables have to be declared and defined.
  For examples:

  ```
  int num1;
  num1 = 100;
  double num2 = 0.05;
  ```

if Statement

`If (<bool-expr>) Then`
`    <stmt>`
`End If`
	`if (<bool-expr>) <stmt>`

`If (<bool-expr>) Then`
`    <stmt(s)>`
`End If`
	`if (<bool-expr>) { <stmt(s)> }`

`If (<bool-expr>) Then`
`    <stmt>`
`Else`
`    <stmt>`
`End If`
	`if (<bool-expr>) <stmt> else <stmt>`

`If (<bool-expr>) Then`
`    <stmt(s)>`
`Else`
`    <stmt(s)>`
`End If`
	`if (<bool-expr>) { <stmt(s)> } else { <stmt(s)> }`

`if (<bool-expr>) Then`
`    <stmt(s)>`
`ElseIf (<bool-expr>) Then`
`    <stmt(s)>`
`Else`
`    <stmt(s)>`
`End If`
	`if (<bool-expr>)`
	`{`
	`    <stmt(s)>`
	`} else if (<bool-expr>) {`
	`    <stmt(s)>`
	`} else {`
	`    <stmt(s)>`
	`}`

Note: Blocks are identified by `End If`, `ElseIf`, `Else`.
For example:

```vb
Dim x As Integer
x = -5
Dim Result As String
If x > 0 Then
    Result = "x is positive"
    If x Mod 2 Then
        Result = Result + " and odd."
    Else
        Result = Result + " and even."
    End If
ElseIf (x < 0) And (x Mod 2) Then
    Result = "x is negative and odd."
ElseIf (x < 0) And (Not (x Mod 2)) Then
    Result = "x is negative and even."
Else
    Result = "x is zero."
End If
MsgBox Result
```

Note: Blocks are identified by pairs of braces (`{}`).
For example:

```cpp
int x = -5;
if (x > 0)
{
    cout << "x is positive";
    if (x % 2)
        cout << " and odd." << endl;
    else
        cout << " and even." << endl;
} else if ((x < 0) && (x % 2)) {
    cout << "x is negative and odd." << endl;
} else if ((x < 0) && !(x % 2)) {
    cout << "x is negative and even." << endl;
} else {
    cout << "x is zero." << endl;
}
```

if-else Operator

In C++, there are the if-else expressions. The syntax is:
    `<condition> ? <result1> : <result2>`
It means that if `<condition>` is true, the expression's value will be `<result1>`, otherwise it will be `<result2>`.
For example:

```cpp
int x = 2, y = 3;
int z = (x > y) ? x : y;
cout << z << endl;
// the output will be 3
```

while Loop

```vb
While (<bool-expr>)
    <stmt(s)>
Wend
```

Note: Blocks are identified by the closest `Wend`.

```vb
Do While (<bool-expr>)
    <stmt(s)>
Loop
```

Note: Blocks are identified by the closest `Loop`.

```cpp
while (<bool-expr>)
    <stmt>

while (<bool-expr>)
{
    <stmt(s)>
}
```

Note: Blocks are identified by pairs of braces (`{}`).

```vb
Do
    <stmt(s)>
while (<bool-expr>)
```

Note: Blocks are identified by having the same indentation.

```cpp
do
    <stmt>
while (<bool-expr>);

do
{
    <stmt(s)>
} while (<bool-expr>);
```

Note: Blocks are identified by pairs of braces (`{}`).

For example:

```vba
Dim i As Integer
i = 10
Do While i > 0
    i = i - 2
    MsgBox i
Loop
```

For example:

```cpp
int i = 10;
while (i > 0)
{
    i -= 2;
    cout << i << endl;
}
```

for Loop

```vba
For <counter> = <start> To <end> Step <step>
    <stmt(s)>
Next
```

For example:

```vba
Dim i As Integer
For i = 0 To 9 Step 1
    MsgBox i
Next
```

```cpp
for (<for-initialization>; <bool-exp>;
<post-processing>) { <stmt(s)> }
```

For example:

```cpp
for (int i = 0; i < 10; i++)
    cout << i << endl;
```

Finishing a Loop Early

In a `For` loop, the statement
`Exit For`
means to stop the whole `For` loop.
In a `Do While ... Loop` and `Do ... Loop While` loop, the statement
`Exit Do`
means to stop the whole loop.

In a `for` loop or a `while` loop, `break` means to stop the whole loop; while `continue` means to skip the current execution.

Functions and Subroutines

- A VBA function runs some code and returns something after the code is finished.

- A VBA subroutine runs some code, but does not return anything.

For example,

```vba
' A subroutine with no return value
Sub PrintNum(ByVal num As Integer)
    MsgBox "The number is" + Str(num)
End Sub

' A function with return value
Function AddOne(ByVal num As Integer)
    AddOne = num + 1
End Function

' An event handler in calling
' the subroutine PrintNum() and
' the function AddOne()
Private Sub Workbook_Open()
    PrintNum 10
    PrintNum AddOne(10)
End Sub
```

- A C++ function may or may not return a value.

For example,

```cpp
/* File: function_example.cpp
   A C++ program with two functions:
   PrintNum() and AddOne()
 */
#include <iostream>
using namespace std;
// A function with no return value
void PrintNum(int num)
{
    cout << "The number is " << num << endl;
}
// A function with return value  of integer type
int AddOne(int num)
{
    return (num + 1);
}
// A main function in calling
// the two functions PrintNum() and AddOne()
int main()
{
    PrintNum(10);
    PrintNum(AddOne(10));
    return 0;
}
```

Some Operators in VBA and C++

		VBA			C++		
		Symbol	Example	Output	Symbol	Example	Output
Arithmetic Operators	Addition	+	1 + 2	3	Same		
	Subtraction	−	1 − 2	-1	Same		
	Multiplication	*	1 * 2	2	Same		
	Division	/	1 / 2	0.5	/	1.0 / 2	0.5
					/	1 / 2	0
	Modulus (Remainder)	Mod	9 Mod 4	1	Same		
	Power	^	2 ^ 3	8	Nil		
Assignment Operators	Assignment	=	x = y		Same		
	Addition Assignment	+=	x += y		Same		
	Subtraction Assignment	-=	x -= y		Same		
	Multiplication Assignment	*=	x *= y		Same		
	Division Assignment	/=	x /= y		Same		
Relational Operators	And	And	x And y		&&	x && y	
	Or	Or	True Or False	True	||	true || false	true
	Not	Not	Not False	True	!	!false	true
Comparison Operators	Larger than	>	20 > 10	True	Same		
	Larger than or equal to	>=	20 >= 10	True	Same		
	Smaller than	<	20 < 10	False	Same		
	Smaller than or equal to	<=	20 <= 10	False	Same		
	Equal to	=	20 = 10	False	==	20 == 10	false
	Not equal to	<>	20 <> 10	True	!=	20 != 10	true
Increment Operators	Post-increment	Nil			++	x = 1; y = 2; y = x++; cout << x << " " << y;	2 1
	Pre-increment	Nil			++	x = 1; y = 2; y = ++x; cout << x << " " << y;	2 2
Decrement Operators	Post-decrement	Nil			--	x = 1; y = 2; y = x--; cout << x << " " << y;	0 1
	Pre-decrement	Nil			--	x = 1; y = 2; y = --x; cout << x << " " << y;	0 0

References:

1. David Rossiter and Gibson Lam. (2015). Excel and Excel VBA Programming for Beginners. Third Edition. McGraw Hill Education.

2. Cay Horstmann. (2012). C++ For Everyone. Second Edition. Wiley.