College of Engineering and Computer Science
Mechanical Engineering Department

California State University Northridge

Mechanical Engineering 309
Numerical Analysis of Engineering Systems

Fall 2014 Number: 15237 Instructor: Larry Caretto

Moving from C++ to VBA

Introduction

These notes are intended to ease the transition between C++ and VBA for students who are moving between these two languages. The next page shows the same program in both languages. You may want to look at those listings to get an overview of what the differences are before reading the detailed description of the differences below.

Entering the program

Both languages allow you to (a) enter a statement on one line, (b) to continue a statement over several lines, and (c) to enter more than one statement on a line. The different ways in which this is done are summarized in the table below. (<Enter> denotes the Enter key.)

	C++	VBA
End of statement at end of line	**;<Enter>**	**<Enter>**
End of line for statement continued to next line (see note below)	**<Enter>**	**_<Enter>**
End of statement before another statement on the same line	**;**	**:**
Note: To continue to another line in VBA enter a space then the underscore before <Enter>.		

C++ commands are all typed in lower case and the language is case sensitive. That is, you can have test and Test as different variables in C++. VBA is not case sensitive. Test and test are the same variable. You can enter variables and keywords in all lowercase, all uppercase or any combination. The Visual Basic Editor (VBE) will set keywords into particular cases and will give the variables you declare the same case structure as they have in your Dim statements.

Multiple Statements in a Control Structure

In C++ braces ({}) are used to define the start and end of a group of statements in a particular control structure. This includes the complete set of all statements in a function, statements in various parts of an If-Else If block, statements in a for loop and statements in while loops. (Braces are not needed if there is only one statement in the structure.)

In VBA the multiple statements do not use braces. Instead, there is an additional statement to terminate the control structure: End Function, End If, Next (to end a For loop), and Loop to end a conditional loop that starts with Do.

Comments

In C++, single line comments start with a double forward slash (//). In VBA single line comments start with an apostrophe (‘). There is no VBA equivalent to C++’s multiline comment that starts with a /* and end with an */. In C++ a single line comment may be placed at the end of a line that is not the end of a statement (a line that ends without a semicolon.) This cannot be done in VBA; a VBA statement that is continued to the next line (with the space+underscore) cannot have a comment following the underscore.

Data types

VBA and C++ have similar data types, but they do not all have the same names. The table below shows some equivalent data types in the two languages.

C++	int	double	string	bool	No Equivalent
VBA	Long	Double	String	Boolean	Date

Jacaranda (Engineering) 3314 Mail Code Phone: N/A
Email: lcaretto@csun.edu 8348 Fax: 818.677.7062

```
#include <iostream>
#include <cmath>
using namespace std;

int main()
{
  cout << "Enter the argument, x, to compute exp(x): ";
  double x;
  cin >> x;
  const int maxN = 100;          // avoid infinite loop
  const double maxError = 1e-12;  // desired error
  bool converged = false;        // loop control var
  double term = 1;              // initial exp(x) term
  double sum = term;             // initialize sum
  int n = 0;                    // initialize counter
  while ( !converged && n < maxN) // start loop
  {
    n++;                       // increment counter
    term *= x / n;             // get new term
    sum += term;               // update partial sum
    converged = fabs(term)      // convergence cond
      <= maxError * fabs( sum );
  }
  if (converged)                // output value
  {     cout << "sin(" << x <<   //if  solution is
              ") = " << sum << "\n";   // converged or
  }                                    // error message
  else
        cout << "No convergence\n";
  return EXIT_SUCCESS;
}
```

```
Option Explicit
Sub getExpX()
   Dim x As Double
   x = InputBox("Enter x to compute exp(x)")
   Const maxN As Integer = 100    'Avoid infinite loop
   Const maxError As Double = 1e-12    'desired error
   Dim converged As Boolean          'loop control variable
   converged = False
   Dim term As Double      'General series term
   term = 1
   Dim sum As Double       'Running series sum
   sum = term
   Dim n As Integer
   n = 0
   Do While (Not converged And n < maxN)
      n = n + 1
      term = term * x / n
      sum = sum + term
      converged = Abs(term) <= _
         maxError * Abs(sum)         'Convergence condition
   Loop
   If converged Then
     MsgBox "exp(" & x & ") = " & sum
   Else
      MsgBox "No convergence in getExpX"
   End If
End Sub
```

Declaring variables

Declaration statements in C++ start with the variable type and several variables can be assigned the same type in a single statement; *e.g.*, `double x, y, z`

Declaration statements in VBA start with the word `Dim` followed by the variable name followed by the keyword As followed by the data type. Although several variables can be declared in one statement, the data types must be repeated for each variable. Two examples are shown below.

```
Dim x As Double, y as double, z As Double
Dim s As String, k As Integer, R As Double
```

Operators

The following arithmetic and relational operations are the same in C++ and VBA: + – * < <= => and >. Also both languages use the equal sign as the assignment operator (x = y + z). In VBA the forward slash division operator (/) always produces a real result, even if the two operands are integers. To get integer division in VBA one has to use the backslash operator (\). In C++ there is only one division operator, the forward slash. It produces a real result if either the numerator or denominator is real. If both the numerator and denominator are integer, the result is an integer.

The following operators are different in the two languages. The third entry (fourth column) shows the string concatenation operator, an & in VBA and a + in C++. (The + can also be used in VBA.)

	Compute x^y	Misc		Relational equal/not equal		Logical Operators		
C++	`pow(x,y)`	`&`	`+`	`==`	`!=`	`&&`	`||`	`!`
VBA	`x^y`	`Mod`	`&`	`=`	`<>`	`And`	`Or`	`Not`

The C++ combination operators (++, --, +=, -=, /=, *=, *etc*.) do not exist in VBA.

Procedures

In C++, there are only function procedures. In VBA there are two types of procedures: functions and subs. A sub procedure in VBA is like a void function in C++; neither returns a value to the calling program through the procedure name. VBA functions return a value to the program that calls the function by setting the function name equal to a value. C++ functions use a return statement in the function to return a value to the program that calls them. Because all C++ procedures are functions, there is no need to use the word "function" in the function header. The pattern for a C++ function header `<function type> <function name> ( <argument list> )`

The argument list for a C++ function consist of zero or more entries like the following

`<data type> <variable name>`

When there is more than one variable in the argument list, all variables, except the last one, are followed by commas.

Examples of C++ function headers are shown below:

```
bool leap( int year )
int getValidInt( int xMin, int xMax, string name )
```

The header for a VBA function is shown below:

`Function <function name> ( <argument list> ) As <function type>`

The argument list for a VBA function consist of zero or more entries like the following

`<variable name> As <data type>`

Like the argument list in C++, when there is more than one variable in the argument list, all variables, except the last one, are followed by commas.

The VBA function headers that would be used in place of the C++ function headers shown above are given below:

```
Function leap( year As Long ) As Boolean
Function getValidInt(xMin As Long, xMax As Long, name As String ) As Long
```

The full code for a function that inputs a year and returns true if the year is a leap year (or false if it is not) is shown below for C++ (on the left) and VBA.

```
bool leap( int year)
{
    return ( year % 4 == 0 ) &&
           ( year % 100 != 0 ) ||
           ( year % 400 == 0 );
}
```

```
Function leap( year As Long ) As _
                                 Boolean
    leap = ( year Mod 4 = 0 ) And
           ( year Mod 100 <> 0) Or
           ( year Mod 400 = 0 )
End Function
```

Passing function parameters by value or by reference

There are two ways to pass a variable to a function (or sub in VBA). The first, known as pass-by-reference, is to pass the memory location of the variable. In this case any changes to the corresponding variable in the function (or sub) take place in the corresponding variable in the calling program. In the alternative, pass-by-value, only the value of the variable in the calling program is passed to the function (or sub). Any changes to the variable in the function (or sub) do not affect the corresponding variable in the calling program.

In C++ the default is pass-by- value; in VBA the default is pass-by-reference. These defaults and the changes required to obtain the alternative as compared in the table below.

	C++	VBA
Pass-by-value	bool f(double x)	Function bool (ByVal x as Double) as Boolean
Pass-by-reference	bool f (double& x)	Function bool (x as Double) as Boolean

If – else if statements

There are several minor differences between these choice statements in C++ and VBA as shown in the comparison below. In both languages the final else block is optional.

C++	VBA

if (<condition1>)
{ <statements done if condition1 is true> }
else if (<condition2>)
{ <statements done if condition2 is true> }
else if (<condition3>)
{ <statements done if condition3 is true> }
<other possible else if statements>
else
{ <statements done if all conditions false> }
	If <condition1> Then
	<statements done if condition1 is true>
	ElseIf <condition2> Then
	<statements done if condition2 is true>
	ElseIf <condition3> Then
	<statements done if condition3 is true>
	<other possible ElseIf statements>
	Else
	<statements done if all conditions false>
	End If

The conditions **are** placed in parentheses.	The conditions **are not** placed in parentheses.
The word then **is not** used after the conditions.	The word Then **is** used after the conditions.
Statements **are** placed in braces.	Statements **are not** placed in braces.
Use two separate words: else if	ElseIf is one word.
There is no end if.	An End If is required.
The words if and else must be lower case.	The words else, if elseif and end can be typed in any case; the editor will format them as shown.

Count-controlled looping

Both C++ and VBA use a “for” loop, but there is no direct comparison because the for loop in C++ is much more powerful (and more complex) than the one in VBA. The comparison below shows the VBA for loop and its equivalent in C++. The general structures are shown in the first row and the two code versions for a specific example are shown in the second row.

C++	VBA
for (<index> = <start>; <index> <= <end>; <index>+=<increment>) { <loop statements that can use index> }	For <index> = <start> To <end> Step <increment> <loop statements that can use index> Next <index>
for (k = 1; k <= 10; k += 2) { cout >> k >> endl; cout >> 2 * k >> endl; }.	For k = 1 to 10 Step 2 msgBox k msgBox 2 * k Next k

Conditional looping

VBA has more forms of conditional looping than C++. Both have loops on "while" conditions where the loop continues so long as the condition is true. VBA also has a loop on an "until" condition in which the loop continues while the condition is false. Each language has two while loops. One starts with a test and will not be executed if the condition is false at the start of the loop. The other has its test at the end of the loop so that the loop code will always execute at least once. These two loops are compared below.

C++	VBA
while (<condition>) { <loop statements > }	Do While <condition> < loop statements > Loop
do { <loop statements > } while (<condition>);	Do < loop statements > Loop While <condition>

Arrays

The basic notational differences between different array types are shown in the table below.

	C++	VBA
Notation for one-dimensional arrays	A[k]	A(k)
Notation for two-dimensional arrays	A[k][m]	A(k,m)
Default lowest subscript	0	0 (or Option Base 1)
Can redefine lowest subscript for each array	No	Yes

The size of an array is declared in the same statements used to declare a variable. An example of this in C++ is: `double x(10), b(10), A(10,10)`. Note that the number 10 in this example represents the **number of elements** in the array. The array index runs from 0 to 9. The same statement in VBA, assuming the default minimum subscript of zero, is: `Dim x(9) as Double, b(9) as Double, A(9,9) as Double`. Note that in VBA the dimension information represents the **maximum subscript**. VBA can redefine the default minimum subscript as one by placing the statement, `Option Base 1`, in the declarations section of the module. VBA can also specify both the minimum and maximum subscript for each array by a statement of the following form: `Dim y(2 To 7) as Double, B(3 to 8, -9 To -4) As Double`.

Passing arrays to functions is compared in the table below. The use of the global variable, LIM, is required in C++ to maintain consistency in array dimensions between the calling program and the function. It is not required in VBA. In both the C++ and VBA statements below the array index can run from 0 to 9. This is the difference between the C++ practice of setting the number of array component in the dimension information and the VBA practice of setting the maximum subscript in the dimension information.

	C++	VBA
Global variable	const int LIM = 10	const LIM as Long = 9
Calling program	int a[LIM]; int b[LIM][LIM]; f = fun(a, b);	Dim a(0 to LIM) As Double Dim b(0 To LIM,0To LIM) As Double f = fun(a,b)
Function header	Int f(int a[], int b[][LIM])	Function f (A() As Long, B()) As Long