## MATLAB or Octave?

Matlab is a proprietary product, and you must pay a license fee to use it. If you are logged onto a University of Waterloo computer, you can use Matlab with the university license. There is a free application that is very similar to Matlab, and that is GNU Octave. You can download GNU Octave here:

https://www.gnu.org/software/octave/

It is a free application, as are all products from GNU. For the purpose of this course, anything you do in Octave should work in Matlab and vice versa.

## Introduction to MATLAB

Matlab is a interpreted language, similar to Python and unlike compiled languages such as C++ and Java. What this means is that rather than a program being compiled into an executable, instead, each statement is parsed in real time by the interpreter and the desired executions are performed.

We will step through all of the structures in C++ and look at the equivalent variations in Matlab.

### Fucntions

In C++, functions have parameters with types, and a function can return one object, be that object a primitive data type or an instance of a class.

In Matlab, arguments are not typed: by default, the function assumes it is being called with the correct arguments, and it is up to the programmer to check those arguments if that individual wishes.

```
unsigned int factorial( unsigned int n ) {
    if ( n <= 1 ) {
        return 1;
    } else {
        return n*factorial( n - 1 );
    }
}
```

The corresponding code in Matlab is:

```
function [result] = factorial( n )
    if n <= 1
        result = 1;
    else
        result = n*factorial( n - 1 );
    end
end
```

In Octave, you can enter the function right at the command line; however, in Matlab, all functions must be stored in the working directory, and the name of the file must be the function name appended with a `.m`. Thus, the above function must be saved to a file `factorial.m` .

Looking at this function, you will see there is no return statement. Instead, the return variable is described in the signature:

```
function [result] = factorial( n )
```

When the function finishes, the returned value is the last value assigned to `result`:

```
>> value = factorial( 10 )
     value =  3628800
```

You may note that Matlab always prints out every single statement unless you put a semi-colon at the end of the statement. This is a convenient way of doing simple debugging (that is, not using the debugger).

The following function is more difficult to implement in C++:

```
function [min, max] = minmax( a, b )
    if a <= b
        min = a;
        max = b;
    else
        min = b;
        max = a;
    end
end
```

To get two return values, you use the following statement:

```
>> [v1, v2] = minmax( 5.32532, -2.55324 )
v1 = -2.5532
v2 =  5.3253
>>
```

Thus, as you may guess, a function can have an arbitrary number of parameters, but also an arbitrary number of return values.

| Why does C++ only allow one return value? Recall that C++ is compiled into machine instructions, and all instruction sets allow for function calls that have one return value, and one return value only. If you want more than one return value, it is up to you to determine how to do this (either passing by reference, or returning a data structure). |
|---|

## Conditional statements

As you can see, the conditional statement in Matlab is also quite straight-forward, with the only difference being the existence of an elseif statement:

```
function [result] = unit_step( x )
    if  x < 0.0
        result = 0.0;
    elseif x == 0.0
        result = 0.5;
    else
        result = 1.0;
    end
end
```

You can also see here that comparison operators are similar to that in C++. The only difference here is that `!=` uses the format `~=`. You should, however, be comfortable with this, as you already know that `!` is the unary logical not operator, while `~` is the unary bitwise complement operator. Thus, the authors of Matlab chose the other format.

## Comments

In Matlab, end-of-line comment symbol is the percent symbol:

```
% This calculates the greatest common divisor
function [result] = gcd( a, b )
    if b = 0
        result = a;
    else
        result = gcd( b, a % b );
    end
end
```

## While loops

In Matlab, while loops are implemented in a similar manner to for loops:

```
function [result] = factorial( n )
    result = 1;

    while n > 1
        result = n*result;
        n = n - 1;
    end
end
```

You can also use a while loop to create an infinite loop:

```
function [result] = exp_approx( x )
    old_result = 1.0;
    n = 0;

    while 1
        n = n + 1;
        result = old_result + x^n/factorial( n );

        if abs( result - old_result ) < 1e-10
            return; % whatever 'result' is assigned is what is returned
        end

        old_result = result;
    end
end
```

This is useful for calculating $e^x$ for small values of $x$, but less useful for larger values.

We can also use fixed-point iteration:

```
function [result] = fixed( f, x0, max_iterations, epsilon )
    n = 0;

    while ( n < max_iterations )
        n = n + 1;
        result = f( x0 );

        if abs( x0 - result ) < epsilon
            return;
        end

        x0 = result;
    end

    error( 'fixed-point iteration did not converge' );
end

>> % approximate x = cos(x)
>> %   correct answer to 50 decimal digits of precision
>> %        0.73908513321516064165531208767387340401341175890757
>> fixed( @cos, 0.3, 10, 1e-10 ) % the '@cos' is a function pointer
error: fixed-point iteration did not converge
error: called from
    fixed at line 11 column 5
>> fixed( @cos, 0.3, 100, 1e-10 )
ans =    7.390851331781744e-01
```

## For loops

A for loop now diverges from C++:

```
function [result] = factorial( n )
    result = 1.0;

    for k = 2:n
        result = result*k;
    end
end
```

## Auto-assignment, auto-increment and auto-decrement operators

As the previous example suggests, Matlab does not have the auto-assignment operators such as `+=`, `-=`, `*=`, `/=` and `%=`. Also, it does not have `++` or `--`; consequently, you must use the long versions as shown in the previous example.

> Be warned: Octave allows auto-assignment, auto-increment and auto-decrement operators. In general, however, it is best to restrict your choice of operators to be compatible with both systems. If you submit code with one of these automatic operators, you will be given one warning without penalty.

## Going beyond the end of a line

If you want to continue a statement on the next line, in C++, you just hit 'Enter' and continue the statement. In Matlab, remember that a statement without a semi-colon is printed, while a statement with a semi-colon is not printed, and thus, we need some other mechanism to determine the end of a statement.

In Matlab, if a statement is terminated without a semicolon, the interpreter assumes it is the end of a statement and attempts to execute it. However, if you append the line with `...` (that is, three periods), the interpreter assumes the statement continues on the next line.

For example, if you were authoring a long polynomial, you may try breaking it up:

```
result = 6.9415674220e-02*x^4 + ...
         1.4027600414e-01*x^3 + ...
         5.0978713801e-01*x^2 + ...
         9.9880301207e-01*x +   ...
         1.0;
```

## Matrices and vectors

Unlike C++, matrices and vectors are easy to create in Matlab:

```
>> format long
>> b = [5.3 0.2 0.1]'    % a 3-dimensional column vector
   v =
       5.300000000000000e+00
       2.000000000000000e-01
       1.000000000000000e-01
>> A = [5.3 0.3 -0.2; 0.3 4.9 0.1; -0.2 0.1 6.7]
    A =
       5.30000000000e+00   3.00000000000e-01  -2.00000000000e-01
       3.00000000000e-01   4.90000000000e+00   1.00000000000e-01
      -2.00000000000e-01   1.00000000000e-01   6.70000000000e+00
```

We can ask Matlab to solve a system of linear equations, and we can multiply a matrix and a vector:

```
>> v = A \ b      % Solve Av = b for 'v'
    v =
       1.002922574869322e+00
      -2.150922690386116e-02
       4.518439368123143e-02
>> A*v           % Note that Av is very close to 'b'
     ans =
        5.300000000000001e+00
        2.000000000000000e-01
        1.000000000000001e-01
```

We can have Matlab do vector and matrix addition, and scalar multiplication:

```
>> v + [1 2 3]'
     ans =
        6.300000000000000
        2.200000000000000
        3.100000000000000

>> A + [1 0 0; 0 1 0; 0 0 1]    % A plus the identity matrix
    ans =

        6.300000000e+00   3.000000000e-01  -2.000000000e-01
        3.000000000e-01   5.900000000e+00   1.000000000e-01
       -2.000000000e-01   1.000000000e-01   7.700000000e+00

>> A/6.7     % each entry of A divided by 6.7
     ans =
         7.910447761…9e-01   4.477611940…7e-02  -2.985074626…2e-02
         4.477611940…7e-02   7.313432835…6e-01   1.492537313…6e-02
        -2.985074626…2e-02   1.492537313…6e-02   1.000000000…0e+00
```

Other useful commands: we can extract the diagonal entries of a matrix using the `diag` command:

```
>> diag( A )
ans =

   5.300000000000000
   4.900000000000000
   6.700000000000000
```

If we pass the diag command a vector, it creates a matrix with those entries on the diagonal:

```
>> diag( [1 2 3 4 5] )
     ans =
          1   0   0   0   0
          0   2   0   0   0
          0   0   3   0   0
          0   0   0   4   0
          0   0   0   0   5
```

If we pass the result of the diagonal of a matrix to `diag`, it creates matrix with all off-diagonal entries equal to zero:

```
>> diag( diag( A ) ) % a matrix with only the diagonal entries of 'A'
     ans =
          5.300000000000000                   0                   0
                          0   4.900000000000000                   0
                          0                   0   6.700000000000000
```

We can calculate the norm of a vector with the norm command:

```
>> norm( [0.5 -0.5 0.5 0.5] )
     ans =  1

>> norm( [3/5 4/5] )
     ans =  1

>> norm( [1 2 3 4 5] )
     ans =  7.416198487095663
```

One nice feature of Matlab is that you can request Matlab to display doubles in their hexadecimal format:

```
>> format hex
>> 1
     ans = 3ff0000000000000

>> -1                        % Note the only difference is the leading 1
     ans = bff0000000000000

>> pi
     ans = 400921fb54442d18

>> -pi
     ans = c00921fb54442d18

>> 2.125         % 1.0001 x 2^1
     ans = 4001000000000000

>> 4.125         % 1.00001 x 2^2
     ans = 4010800000000000
```