Lab 1: Getting started with SimTK

Developing physics-based simulations of biological structures with SimTK requires a rudimentary understanding of biology and physics and a background in C or C^{++}. The point of this first lab is to:

- Create a SimTKproject to submit your computer projects
- Install the Microsoft Visual Studio C^{++} compiler
- Read and understand the C^{++} coding standards
- Submit the "Hello Math" exercises (basic mathematical calculations, input, and output)
- Learn about creating, compiling, running, and debugging C^{++} programs

1.1 Create a SimTKProject at www.simtk.org

All computer/electronic assignments are passed in via **your project** on the Simtk.org website. To become a member of Simtk.org (required to set-up a project), go to www.simtk.org, click on **register** and follow the on-screen information. Your SimTK login information is set to the e-mail you provide during registration. After receiving your login information, go back to www.simtk.org, click on **Log In** and enter your Login Name and Password. Next, create a project, e.g., by clicking on **Register Project** and following the on-screen instructions. Use a **project identifier** with your last name (all in lower case) and a **project title** of "2007BioE215 YourLastName" .

The labs in this course are distributed via the training project on the Simtk.org website. To get labs, go to www.simtk.org/home/training, and click on the Documents link on the left-hand side.

Enabling groundbreaking biomedical research by providing open access to high-quality simulation tools, accurate models and the people behind them.

1.2 Install a C^{++} compiler

There are many excellent C^{++} compilers, including Code Warrior, Borland, g^{++}, etc. Instructions for programming with Microsoft Visual C^{++} are provided in Section 1.8. Programmers who are proficient with another compiler can translate these directions to work with their preferred compiler.[1]

One way to install the **free** Microsoft Visual C^{++} Express compiler, is to downloaded and install it from http://www.microsoft.com/vstudio/express/support/install/. To use this compiler, you need Microsoft Windows XP with Service Pack 2 (or later) and you need to install the compiler on the `C:` drive.

Alternately, request an easy-to-use installation CD from your SimTK instructor.

---

[0] Last updated April 13, 2007 by Paul Mitiguy.

[1] For example, to use the GNU C^{++} compiler, type `g++ HelloMath.cpp -o HelloMath`

1.3 Read the short C^{++} coding standards

Coding standards increase the efficiency and enjoyment of team-based C^{++} development and reflect the significant value of uniform coding practices. To get coding standards, visit www.simtk.org/home/training, click on the Documents link on the left-hand side, and download **CodingStandardsCpp.pdf**.

1.4 Hello Math Exercises

The point of these exercises is to ensure that C and C^{++} programmers have sufficient programming and mathematical knowledge to use C^{++} SimTK source code.[2] In addition, programmers will build highly modular code that is reused in later lab examples, e.g., code to read input and/or write output with files, the keyboard, or screen.

To start, go to www.simtk.org/home/training, click on the Documents link on the left-hand side, download the file LabGettingStartedWithSimTK.zip, and unzip the file `HelloMath.cpp`. Ensure that you can compile, link, and run this file (e.g., as described in Section 1.8).

Note: `HelloMath.cpp` is a helpful starting point for the exercises that follow.

1.5 Hello Math Exercise 1

Submit a C^{++} program to your SimTK project called `HelloMath1.cpp` that does the following:

- Prompts the user to enter an angle in degrees (from the keyboard)
  Use `GetStringFromKeyboard(...)` to call `GetStringFromFile(..., stdin)` which itself uses the fast, efficient, standard C input function `fgets`.
- Verifies that the quantity entered can be unambiguously interpreted as a real number. Otherwise, inform the user that 45° will be used. Hint: See the example in **CodingStandardsCpp.pdf**.
- Converts the angle to radians
- Calculates the sine, cosine, and tangent of the angle
- Writes information to the screen which clearly communicates the angle in degrees, the angle in radians, and the sine, cosine, and tangent of the angle.
  Note: Use the standard C output function `fprintf` instead of the C^{++} overloaded operators `>>` and `<<`.
  For example, `printf( "%g", someNumber )` will write the double-precision number `someNumber` to the screen.

The prototypes for the functions that you need to create and use are:[3]

```
bool   GetStringFromKeyboard( char inputString[], unsigned long sizeOfString );
bool   GetStringFromFile(     char inputString[], unsigned long sizeOfString, FILE *fptr );
bool   WriteStringToScreen(   const char outputString[] );
bool   WriteStringToFile(     const char outputString[], FILE *fptr );

const char*  ConvertStringToDouble( const char *s, double &returnValue, double defaultValue );

double   ConvertFromRadiansToDegrees( double angleInRadians );
double   ConvertFromDegreesToRadians( double angleInDegrees );
```

---

[2] Programmers who know C (instead of C^{++}) need to learn how to use C^{++} comments, the better ways of writing for-loops in C^{++}, how to declare variables where they are needed (instead of at the top of a function), how to use references (&), and how to call object-oriented methods. Conversely, C programmers do ***not*** have to learn how to create their own classes or methods as they are provided for you.

[3] Note: A simple modular way to implement `GetStringFromKeyboard` is to call `GetStringFromFile( inputString, sizeOfString, stdin );` This uses the fact that `stdin` is the file pointer associated with the keyboard. Similarly, a simple modular way to implement `WriteStringToScreen` is to call `WriteStringToFile( outputString, stdout );` This uses the fact that `stdout` is the file pointer associated with the screen.

1.6 Hello Math Exercise 2

The first exercise used the screen for output. This second exercise writes numbers to a file for subsequent plotting and **requires a function that opens a file or issues a message if it does not open**.

Note: To turn off the Microsoft Visual C^{++} warning about `fopen` being deprecated (it prefers a safer, non-ANSI, non-portable, function), add the line `#pragma warning(disable:4996)` to the top of your code.

Submit a C^{++} program to your SimTK project called `HelloMath2.cpp` that does the following:

- Prompts the user to enter an integer between 180 and 720 that represents an angle in degrees.
- Verifies that `angleInDegrees` is an integer and $180 \leq$ `angleInDegrees` ≤ 720.
  Otherwise, informs the user that `angleInDegrees = 360` will be used.
- Prompts the user to enter the **precision** for writing numbers.
  Precision is the number of digits in the mantissa after the decimal point, e.g., `precision`=5 for `0.12345E78`.
- Verifies that `precision` is an integer and $1 \leq$ `precision` ≤ 17.
  Otherwise, informs the user that `precision = 5` will be used. **Hint: See strtol**.
- Creates a for-loop starting at `i`$=0$, ending at `i`$=$`angleInDegrees`, and incrementing by 1
- Creates a double precision number called `angleInRadians` equal to the radian measure of `i`
- Calculates the sine, cosine, and tangent of `angleInRadians`
- Writes each value of the angle in degrees, `angleInRadians`, and the sine, cosine, and tangent of the angle to a file `HelloMath2.txt`. Write output values in the format specified by `precision`. Use `fclose` to close the file when done writing. The file `HelloMath2.txt` should look something like:

```
 0.000000000E+000   0.000000000E+000   0.000000000E+000   1.000000000E+000   0.000000000E+000
 1.000000000E+000   1.745329252E-002   1.745240644E-002   9.998476952E-001   1.745506493E-002
 2.000000000E+000   3.490658504E-002   3.489949670E-002   9.993908270E-001   3.492076949E-002
 3.000000000E+000   5.235987756E-002   5.233595624E-002   9.986295348E-001   5.240777928E-002
 4.000000000E+000   6.981317008E-002   6.975647374E-002   9.975640503E-001   6.992681194E-002
```

- Submit a plot of the sine, cosine, and tangent vs. the angle in degrees. (Plot with Matlab, Excel, . . .)

The prototypes for the additional functions that you need to create and use are shown below. The first function should use the C `fprintf` function with a format specifier that:[4]

- Uses the − flag after the % symbol so numbers are left-aligned within their field width.
- Uses a blank space after the − flag to ensure that positive numbers start with a blank space.
  (Note: The default format for printing a positive number is without a leading space or + sign, hence positive numbers are not vertically aligned with negative numbers in subsequent rows.)
- Uses a positive integer (called the field-width) after the blank space that is 8 more than the precision. This provides sufficient room for printing a double-precision number in the form `0.12345E78` with:
  - Leading blank or negative sign (1 character)
  - One digit before the decimal point (1 character)
  - The decimal point (1 character)
  - The specified precision (number of digits in the mantissa after the decimal point) (`precision` characters)
  - The letter E denoting exponential notation (1 character)
  - Three digits for the exponent (3 characters)
  - One extra spaces after the number (1 character)
- Uses a decimal point after the field-width integer
- Uses a positive integer number (called the precision) after the decimal point
- Uses the letter E to denote exponential notation

```
FILE*  FileOpenWithMessageIfCannotOpen( const char *filename, const char *attribute );
bool  WriteDoubleToFile( double x, int precision, FILE *fptr );
const char*  ConvertStringToLong(   const char *s, long   &returnValue,   long defaultValue );
```

---

[4] For example, the format specifier %̈- 13.5E¨ prints a double precision number in the form `0.12345E78` and ensures it starts with an extra space on the left, is left aligned, leads with a blank or a negative sign, is 13 characters wide, and has 5 digits in the mantissa after the decimal point. **Hint: See the example in HelloMath.pdf**.

1.7 Hello Math Exercise 3

This coding exercise reads an array of numbers from a file, does matrix operations, and writes the results to another file. Submit a C^{++} program to your SimTK project called `HelloMath3.cpp` that does the following:
(Note: This should be a stand-alone program, i.e., no SimTK or other 3rd-party software.)

- Creates a 5×7 matrix M whose elements are read from the file `HelloMath3In.txt` (**given**)
- Calculates $M + M$ and writes the results to the file `HelloMath3Out.txt`

The prototypes for the additional functions that you need to create and use are shown below.

```
bool  GetDoubleRowElementsFromFile( double *array, unsigned int numberOfCols, FILE *fptr );
bool  GetDoubleMatrixElementsFromFile( double *array, unsigned int numberOfRows, unsigned int numberOfCols, FILE *fptr );
bool  WriteDoubleRowElementsToFile( const double *array, unsigned int numberOfCols, int precision, FILE *fptr );
bool  WriteDoubleMatrixElementsToFile( const double *array, unsigned int numberOfRows, unsigned int numberOfCols, int precision, FILE *
void  AddMatrices( const double *arrayA, const double *arrayB, double *arraySum, unsigned int numberOfElements );
```

The contents of the file HelloMath3In.txt are:

```
  1.0    2.0    3.0    4.0    5.0    6.0    7.0
  2.1    2.2    2.3    2.4    2.5    2.6    2.7
  3.7    3.6    3.5    3.4    3.3    3.2    3.1
  4.2    4.4    4.6    4.8    4.3    4.5    4.7
  5.0    5.2    5.4    5.6    5.8    5.9    5.7
```

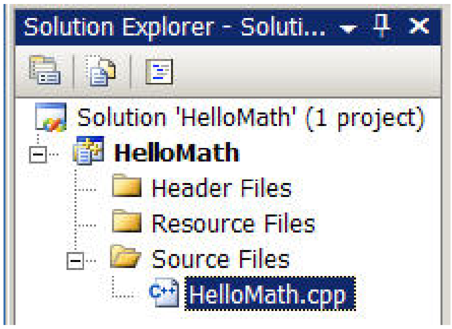
1.8 Compiling and linking C^{++} programs with Microsoft Visual C^{++}

1.8.1 Creating an empty Win32 Console Application in the SimTK folder

- Ensure Microsoft Visual C^{++} Express Edition is installed on your computer
- Click on the Windows Start menu, select **Programs**, then slide-the-mouse to the **Visual C^{++} 2005 Express Edition** folder, then slide-right-and-down to the **Microsoft Visual C^{++} 2005 Express Edition** executable
- From within the Visual C^{++} 2005 Express Edition, click on the **File** menu, click on the **New** menu item, and the slide-right to click on the **Project...** menu item.
  - Under **Project types:**, select **Win32**
  - Give the project a sensible **name**, e.g., `HelloMath`
  - Make the project's **location** `C:\Program Files\SimTK` (If necessary, create the folder SimTK)
  - Click the OK button, then click the Next > button **(not the Finish button)**
  - Ensure the **Application Settings** are set to create a **Console Application**, an **Empty Project**, with **NO precompiled header**.
  - Click the Finish button
- This process creates a file `HelloMath.sln` in the folder `C:\Program Files\SimTk\HelloMath`
- From within Visual C^{++}, click on the **File** menu and then click on the **Exit** menu item.

1.8.2 Adding files to the "HelloMath" Win32 Console Application

- Double-click on the file `HelloMath.sln` in the folder `C:\Program Files\SimTk\HelloMath`
  (This should invoke Microsoft Visual C^{++} and open the "HelloMath" project.)
- Click on the **Project** menu and slide-down and click on **Add Existing Item...**.
- Browse to the directory containing the files to add, select the files, and click Add.
  (For example, to add the file `C:\Program Files\SimTK\HelloMath\HelloMath.cpp`, browse to the directory `C:\Program Files\SimTK\HelloMath`, select the file `HelloMath.cpp`, then click Add.
  Note: A commented sample `HelloMath.cpp` file is available by visiting www.simtk.org/home/training, clicking on the Documents link on the left-hand side, downloading **LabGettingStartedWithSimTK.zip**, and extracting the file **HelloMath.cpp**. Alternately, type the file in Section 1.9.
- To ensure the file was properly added, click on the **Solution Explorer** tab on the left panel in Visual C^{++} (if necessary, click on the + sign in front of **HelloMath** and click on the + sign in front of ***Source files***), then double-click on the file `HelloMath.cpp`


Solution Explorer - Soluti...
Solution 'HelloMath' (1 project)
HelloMath
Header Files
Resource Files
Source Files
HelloMath.cpp

- From within Visual C^{++}, click on the **File** menu and then click on the **Save all** menu item.
- Click on the **File** menu and then click on the **Exit** menu item.

1.8.3 Building and running the "HelloMath" Win32 Console Application

- If the `HelloMath` project is not already open, double-click on the file `HelloMath.sln`
- Click on the **Build** menu and then slide-down and click on **Build Solution**
- The compiler will attempt to ***compile*** the relevant C^{++} ***source files***.
  For example, the source file `HelloMath.cpp` will compile to the ***object file*** named `HelloMath.obj`
- You must fix all compiler errors (bugs in your program) before you can ***link***.
  It is highly advisable to also fix all your warnings as many of them will show up later as run-time errors.
  Note: To turn off the Microsoft Visual C^{++} warning about `sprintf` being deprecated (it prefers a safer, non-ANSI, non-portable, function), add the line `#pragma warning(disable:4996)` to the top of your code.
- If compiling all the source files is successful, Visual C^{++} will attempt to link your object files to each other and to the standard C^{++} libraries.[5]
- If all goes well, you may see a message such as

```
------ Build started: Project: HelloMath, Configuration: Debug Win32 ------
Compiling...
HelloMath.cpp
Compiling manifest to resources...
Linking...
Embedding manifest...
Build log was saved at "file://c:\Program Files\SimTK\HelloMath\Debug\BuildLog.htm"
HelloMath - 0 error(s), 0 warning(s)
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

- To run the program from within the Visual C^{++} environment, click on the **Build** menu (or **Debug** menu) and select **Start Debugging**
- If you encounter a run-time debug error (the program runs but it produces incorrect results), try the Visual C^{++} ***debugger***. To start the debug process, do the following:
  - Get help (preferably human) to understand the debugging process
  - Ensure that the build is set to **Win32 Debug** (not **Win32 Release**)
  - Insert a ***breakpoint*** in your code by right-mouse clicking on a suspect line in your code.
    If in doubt, set the breakpoint at the first line of code in **main** (the line that appears after the first { in **main**).
  - Click on the **Debug** menu and then slide-down and click on **Start Debugging**.
  - Press function key **F10** to step over a function/method
  - Press function key **F11** to step into a function/method
  - Press function key **F5** to go to the next breakpoint

---

[5] The standard C^{++} libraries include the standard math library that calculates cos(0.2), the standard input/output library that prints characters to the screen, and the standard time library that gets time and date information from your computer.

1.9 Sample HelloMath.cpp

```
//-----------------------------------------------------------------------------
// File:     HelloMath.cpp
// Class:    None
// Parent:   None
// Children: None
// Purpose:  Tests out various mathematical functions
//-----------------------------------------------------------------------------
// The following are standard C/C++ header files.
// If a filename is enclosed inside < >  it means the header file is in the Include directory.
// If a filename is enclosed inside " "  it means the header file is in the current directory.
#include <ctype.h>      // Character Types
#include <math.h>       // Mathematical Constants
#include <stdarg.h>     // Variable Argument Lists
#include <stdio.h>      // Standard Input/Output Functions
#include <stdlib.h>     // Utility Functions
#include <string.h>     // String Operations
#include <signal.h>     // Signals (Contol-C + Unix System Calls)
#include <setjmp.h>     // Nonlocal Goto (For Control-C)
#include <time.h>       // Time and Date information
#include <assert.h>     // Verify Program Assertion
#include <errno.h>      // Error Codes (Used in Unix system())
#include <float.h>      // Floating Point Constants
#include <limits.h>     // Implementation Constants
#include <stddef.h>     // Standard Definitions
#include <exception>    // Exception handling (e.g., try, catch throw)
//-----------------------------------------------------------------------------

//-----------------------------------------------------------------------------
// Prototypes for local functions (functions not called by code in other files)
//-----------------------------------------------------------------------------
bool  WriteDoubleToFile( double x, int precision, FILE *fptr );


//-----------------------------------------------------------------------------
int  main( int numberOfCommandLineArguments, char *arrayOfCommandLineArguments[] )
{
   // Write  " Hello math!"  to the screen and then put a newline
   printf( " Hello math!\n" );

   // Write the number of command line arguments to the screen.
   // Write each of the command line arguments on a separate line.
   // Note: The first command line argument may be the name of the program
   // In Microsoft Windows, dragging and dropping files onto the executable
   // usually results in multiple command line arguments (the names of the files)
   printf( "\n The number of command line arguments is %d\n", numberOfCommandLineArguments );
   for( int i = 0;  i < numberOfCommandLineArguments;  i++ )
      printf( "\n Command line argument %d is:\n %s", i, arrayOfCommandLineArguments[i] );

   // Calculate the sine of 1.0 radian (1 radians is approximately 57.3 degrees) plus other stuff
   double x = sin(1.0) + cos(1.0) + tan(1.0) +  asin(0.7) + sqrt(4.2) + pow(3.3,0.8)
            + log(1.5) + log10(4.2) + exp(1.4) + sinh(0.3) + 2*3.2 + rand();

   // Writes the result to the screen (stdout)
   printf( "\n\n The computed number is: " );
   WriteDoubleToFile( x, 7, stdout );

   // Keep the screen displayed until the user presses the Enter key
   printf( "\n\n Press  Enter  to terminate the program: " );
   int key = getchar();

   // A normal program exit returns 0 (other return values usually signal an error)
   return 0;
}


//-----------------------------------------------------------------------------
bool  WriteDoubleToFile( double x, int precision, FILE *fptr )
{
   // Ensure the precision (number of digits in the mantissa after the decimal point) makes sense.
   // Next, calculate the field width so it includes one extra space to the right of the number.
   if( precision < 0 || precision > 17 ) precision = 15;
   int fieldWidth = precision + 8;

   // Create the format specifier and print the number
   char format[20];
   sprintf( format, " %%- %d.%dE", fieldWidth, precision );
   return fprintf( fptr, format, x ) > 0;
}
```