# STAT40780 Data Programming with C (online)

## Lab Sheet 3 (Solutions)

Lecturer: Marie Galligan

This week's lab requires you to modify the cube() function you wrote last week to make it callable from R through the .C interface.

## 1 The .C interface

In last week's lab, the first task was to write a function in C++ that would accept as input a variable of type double, and return its value cubed. A possible solution is shown below.

**cube() function**

```
1
2 double cube( double x )
3 {
4   return( x * x * x );
5 }
```

**Task 1:** Modify the cube() function above to a form that can be called from R, through the .C interface. Save the cube() function in a .cpp file, (as cube.cpp).

**Solution to Task 1**

To make a C++ function callable from R through the .C interface, it should:

- have return type void (i.e. function does not directly return a values - can only return values through its input arguments)
- its input arguments should be pointers
- be wrapped inside the extern "C" statement

Therefore, the following modification to the cube() function makes it callable from R through the .C interface.

**cube.cpp**

```
1
2
3  extern "C" {
4
5  void cube( double * x  )
6   {
7    *x =  (*x) * (*x) * (*x)  ;
8   }
9
10 }
```

This function takes a single input argument x, a pointer to a variable of type double (corresponding to the numeric type scalar passed from R), and then modifies the value x points to, setting it equal to its cube. This function returns the cube to R through the modified input.

Note that the parentheses around `*x` are unnecessary, as the dereference operator * for pointers has higher precedence than the multiplication operator * and hence *x will be evaluated first as the value pointed to by x. Therefore, the function could be re-written as follows:

**cube.cpp**

```
1
2
3  extern "C" {
4
5  void cube( double * x  )
6   {
7    *x =  *x * *x * *x  ;
8   }
9
10 }
```

**Task 2:** Compile the .cpp function from the command line to produce a .dll (on Windows) or .so (on MAC).

**Solution to Task 2**

To compile the source code file cube.cpp, open the command prompt.

- To open the command prompt in Windows, click 'Start', and in the search box, type 'cmd'. This should provide a search result 'Command Prompt' - click on this and the command prompt should open. The following web link provides more instructions on this:
  `www.7tutorials.com/7-ways-launch-command-prompt-windows-7-windows-8`
- On OSX systems, open Terminal.app by first opening 'Finder', then searching for Terminal.app (usually stored in the Utilities folder, within the Applications folder). Double click to open.

To compile the cube.cpp file, first change the current working directory to the location where file cube.cpp is stored, using the `cd` command. To find the directly, locate the folder it's stored in through My Computer (on Windows) or Finder (on Mac), and click and drag this folder to the command prompt

window. Then compile the function using `R CMD SHLIB`.

```
cd path/to/file
R CMD SHLIB cube.cpp
```

Check output for errors - the output should look something like this:

```
My-MacBook-Pro:Lab3 myusername$ R CMD SHLIB cube.cpp
clang++ -I/Library/Frameworks/R.framework/Resources/include -DNDEBUG
 -I/usr/local/include -I/usr/local/include/freetype2 -I/opt/X11/include
  -fPIC  -Wall -mtune=core2 -g -O2  -c cube.cpp -o cube.o
clang++ -dynamiclib -Wl,-headerpad_max_install_names
-undefined dynamic_lookup -single_module -multiply_defined
 suppress -L/Library/Frameworks/R.framework/Resources/lib
 -L/usr/local/lib -o cube.so cube.o -F/Library/Frameworks/R.framework/..
 -framework R -Wl,-framework -Wl,CoreFoundation
```

The above output was generated from shows two calls to the C++ compiler (`clang++`) - the first converts the .cpp (C++ source code) file to .o (object file), while the second converts cube.o to a shared object file cube.so. No error messages follow these calls, and hence the file seems to have compiled successfully. Checking the folder containing cube.cpp should show the two new files - cube.o and cube.so. On a Windows machine, cube.dll should be created rather than cube.so.

**Task 3:** Load the .dll or .so file into R. Call the cube() function from R through the .C() interface. Experiment with different input arguments. Before passing an argument to the function, be sure that it is of the correct type! Note: since the cube function expects input of type double, you should make sure the argument passed from R is a numeric vector of length 1 (corresponding to C++ type double).

## Solution to Task 3

Open an R script file. Change the working directory in R to the location where the compiled cube function is stored. Load the C++ cube function into R using the `dyn.load()` function.

load cube() into R

```
1 setwd("path/to/file/")
2
3 dyn.load("cube.so") #load the compiled function  (MAC)
4
5 #or
6
7 dyn.load("cube.dll") #load compiled function (Windows)
```

Call the C++ cube() function through the .C interface. Pass in different numeric type values. Ensure the type of the input is `numeric` using the `as.numeric` function.

calls to cube()

```
#examples of calls to cube()
.C("cube", as.numeric(0.3))
.C("cube", as.numeric(2))
.C("cube", as.numeric(-2))
```

**Task 4:** Unload the cube() function from R

**Solution to Task 4**

Unload the function from R using the `dyn.unload()` function.

unload cube() into R

```

dyn.unload("cube.so") #unload the compiled function  (MAC)

#or

dyn.unload("cube.dll") #unload compiled function (Windows)
```