Graduate School Class Reminders

- Maintain six feet of distancing
- Please sit in the same chair each class time
- Observe entry/exit doors as marked
- Use hand sanitizer when you enter/exit the classroom
- Use a disinfectant wipe/spray to wipe down your learning space before and after class
- Media Services: 414 955-4357 option 2

Documentation on the web

- CRAN: `http://cran.r-project.org`
- R manuals: `https://cran.r-project.org/manuals.html`
- SAS: `http://support.sas.com/documentation`
- Step-by-Step Programming with Base SAS 9.4 (SbS): `https://documentation.sas.com/api/docsets/basess/9.4/content/basess.pdf`
- SAS 9.4 Programmer s Guide: Essentials (PGE): `https://documentation.sas.com/api/docsets/lepg/9.4/content/lepg.pdf`
- Wiki: `https://wiki.biostat.mcw.edu` (MCW/VPN)

C and C++

- On the TIOBE Index of Programming Language Popularity: currently, C is first and C++ is fourth
- C is a mid-level language designed for hardware portability
- C developed by Bell Labs in 1972
- Recent C standards (reaching maturity): C99 in 1999; C11 in 2011; and C18 in 2018
- C++ is backwards compatible with C
- Recent C++ standards (still evolving): C++11 in 2011; C++14 in 2014; C++17 in 2017 and C++20 in 2020
- C++ developed by Bell Labs in 1982
- C++ a high-level multi-paradigm language with four flavors: C, Object-oriented C++, Template C++ and the Standard Template Library (STL)
- C/C++ are compiled languages: The GNU Compiler Collection (GCC) provides open source compilers
- GCC docs: `https://gcc.gnu.org/onlinedocs/gcc-9.2.0`
- Excellent C++ documentation: `https://cppreference.com`

C and C++

- R relies on the system's C/C++ compiler setup via `R CMD ...`
- R requires the following: `.c` for C and `.cpp` for C++
- Statically typed languages that require variable definitions `double` for double-precision numbers and `int` for integers `double x, y=0.; int i, j=0;`
- Unlike R, C/C++ have *block* comments
  RED is commented out
  ```
  /* double x, y=0.;
    integer i, j=0; */
  ```
- C++ also has single line comments: `// double x, y=0.;`
- Additional functionality provided by C pre-processor header files: don't confuse the C pre-processor command, cpp, with C++, `.cpp` filename extension
- C++ expressions are followed by semi-colons
  but C pre-processor commands do NOT end in semi-colons

C and C++ header files

- Added functionality like R add-on packages
- C: header files and compiled libraries
- C++: uncompiled source header-only class libraries
  the R/Rcpp recognizes the following extensions:
  `.h` or none at all (we will talk about build
  ```
  #include <Rcpp.h> // Rcpp header
  #include <complex> // complex number header
  ```
- `#include <FILENAME>` used for headers in the compiler's path of header files: typically, within the `/usr/include` directories/sub-directories or within R's package library
- `#include "FILENAME"` used for headers that are elsewhere
- The C/C++ headers are documentation all by themselves since they define the application programming interface (API) you can find them at `/usr/include/c++/9.2.0`
- For R packages, you can find their headers with the `system.file` function
  ```
  > system.file("include", package="Rcpp")
  ```

C/C++ expressions

- R and C/C++ expressions are similar: R is written in C
- Each expression returns a value and ends in a semi-colon
- You can group several expressions in curly brackets:
  `{ expr`$_1$`; ...; expr`$_m$`; }`
  where the return value comes from the last: `expr`$_m$`;`

C/C++ `if` command syntax: RED and BLUE lines optional

R and C/C++ if commands are similar

if(cond$_1$) expr$_1$;
else if(cond$_2$) expr$_2$;
$\vdots$
else if(cond$_m$) expr$_m$;
else expr$_{m+1}$;

- you can combine conditions with || for OR and/or && for AND

C/C++ `for` command syntax

- `for(INITIALIZATION; CONDITION; ITERATOR) expr;`
- There is also a `break` statement like R
- However, there is NO `next` statement
  rather it is `continue`
- But, conveniently, the `CONDITION` can be more complex
- Example:
  `for(int k=0; k<n; ++k) expr;`
- This example iterates `k=0, ..., n-1`
- These are C/C++ vector indices as opposed to
  R indices like `1, ..., n`
- In my experience, the 0-based indices are the biggest transitional challenge to learning C/C++ vs. other languages
- It is based on memory pointer address arithmetic
  an area where C++ is much more user-friendly
  but it inherits the 0-basis and other C-isms

Writing your own C++ functions

TYPE NAME1($\texttt{TYPE}_1$ $\texttt{name}_1$, ..., $\texttt{TYPE}_m$ $\texttt{name}_m$) { expr }

- the value of expr is NOT automatically returned like R
- You return a value at any place within expr via the return command: return $\texttt{expr}_r$;
- Call this function via NAME1($\texttt{expr}_1$, ..., $\texttt{expr}_m$);
  you must supply all of the arguments unless they have defaults
- Some, or all, arguments can have default values so that you do not have to supply every single argument

TYPE NAME2($\texttt{TYPE}_1$ $\texttt{name}_1$=$\texttt{expr}_1$,...,$\texttt{TYPE}_m$ $\texttt{name}_m$=$\texttt{expr}_m$)

But you still have to call NAME2 with the arguments in order:

NAME2 ($\texttt{expr}_1$, ..., $\texttt{expr}_{m-n}$)

Rcpp and R

- The C++ interface to R is seamlessly provided by the Rcpp package which efficiently passes object references from R to C++ (and vice versa) as well as providing direct access to the R random number generator
- Rcpp is mainly an Object-oriented C++ flavor
  but behind the scenes it utilizes Template C++ and
  it was heavily influenced by the STL
- see EddeFran11 in the lit directory for an intro
- The Rcpp Gallery has lots of examples
  `https://gallery.rcpp.org`

An example of the C interface to R

```
SEXP a;
PROTECT(a = allocVector(REALSXP, 2));
REAL(a)[0] = 123.45;
REAL(a)[1] = 67.89;
UNPROTECT(1);
```

- The C interface is NOT user-friendly unless you are intimately familiar with the R source code
- There is documentation, but it is written *by programmers for other programmers* to read
- `SEXP` is a C pointer, or memory address, corresponding to an R object which can also be useful with Rcpp
- the `Rcpp::wrap` function returns a `SEXP` generally back to R as a return value from a function call

Same example of the C++ interface to R provided by Rcpp

```
Rcpp::NumericVector a(2); // a new length 2 numeric vector
                          // created in R's memory space
a[0] = 123.45;
a[1] = 67.89;
```

- The C++ interface is very user-friendly
- Notice that C/C++ vector indices range from 0 to n-1
  as opposed to R vector indices that range from 1 to n
- `Rcpp::` is a C++ namespace and you reference Rcpp classes
  by `Rcpp::CLASS`
  similar to the way you can reference *visible* R functions by
  specifying their packages, i.e., `parallel::detectCores`
- there is also an `R::` C++ namespace that you can use to
  reference R-like C functions such as `R::dxxx`, `R::pxxx`,
  `R::qxxx` and `R::rxxx` from what is known as
  the Standalone Rmath Library which is part of the R project
  we will see the `Rcpp_rnorm` example later

R and Rcpp: vectors, matrices, lists and random numbers

R code	Rcpp code
`a = numeric(n)`	`Rcpp::NumericVector a(n);`
`a[i]`	`a[i-1]` is a `double`
`a = rnorm(n)`	`Rcpp::RNGScope state;`
`for(i in 1:n)`	`for(int k=0; k<n; ++k)`
`  a[i]=rnorm(1)`	`  a[k]=R::rnorm(0., 1.);`
`b = integer(n)`	`Rcpp::IntegerVector b(n);`
`c = logical(n)`	`Rcpp::LogicalVector c(n);`
`d = character(n)`	`Rcpp::CharacterVector d(n);`
`A = matrix(nrow=m, ncol=n)`	`Rcpp::NumericMatrix A(m, n);`
`A[i, j]`	`A(i-1, j-1)` is a `double`
`B = matrix(nrow=m, ncol=n)`	`Rcpp::IntegerMatrix B(m, n);`
`x = list()`	`Rcpp::List x;`
`x$NAME`	`x["NAME"]`

R `numeric` vs. Rcpp `Numeric`

- R's `numeric` is the default atomic and matrix type
  but it is a hybrid of `integer` and `double` (for floating point)
- If the object is equally well represented by an integer expression, then an `integer` object is created
  a literal ending in `L` is automatically an `integer`: `0L`
- However, if any operations performed on the object require floating point operations, then it is automatically changed to `double` which is `numeric`'s alter-ego
- There are conversion functions `as.numeric`, `as.double` and `as.integer`
- `as.integer` truncates the decimal portion TOWARDS ZERO
  like `floor(object)` for positive or zero values
  with `-floor(-object)` for negative values
- Rcpp's `NumericVector` and `NumericMatrix` are not ambidextrous: they are made of `double` elements
- Use `IntegerVector` and `IntegerMatrix` as needed
  they are made of `int` elements

Simple Rcpp examples

- `fibonacci.R` and `fibonacci.cpp`
- `Rcpp_rnorm.R` and `Rcpp_rnorm.cpp`
- These use the `sourceCpp` function
- Common debugging options are `verbose=TRUE` and `rebuild=TRUE`

HW hands-on: Rcpp and the Mandelbrot set

- With Rcpp, we can make an extremely fast version of the `mandelbrot` function: far faster than multi-threading see `RcppMandelbrot.R`
- You need to create `RcppMandelbrot.cpp`
- Hints: R and C++ have similar structure so it is convenient to start with `mandelbrot.R` as a guide
- The `complex` header is required to create complex numbers so you need to `#include <complex>`
- The class/type and constructor line looks like this `std::complex<double> c(x[i], y[j]), d(0., 0.);`
- The `std::pow` function is used for exponentiation and it works with complex numbers as well as other types (C/C++ has no exponentiation operator like the caret in R)
- There is no modulus/magnitude function: instead they provide the `std::norm` function which is the square of the modulus/magnitude
- email me two files: your C++ program and your PDF file