Self study note

Steven Chiou

Chapter 1

- `.Call()`
- `SEXP` pointer to S expression type
- `Rcpp::as<int>` converts the incoming argument from `SEXP` to integer.
- `Rcpp::wrap()` converts the result to `SEXP` type returned by a function used with `.Call()`.
- `inline` package provides complete wrappers for *compilation*, *linking*, and *loading* steps.
  - `cxxfunction()`
    - `sig` signature (input variables) of the function
    - `body` C++ codes to include/compiled; wrap with "`...`"
    - `plugin` Rcpp or others
    - `include` pure C++ function to pass through
- `compiler` package compile regular `R` codes with `cmpfun` function.

Chapter 2

- To compile, link, and load:
  - `R CMD SHLIB` triggers `g++`
  - `dyn.load` to load the generated `.so` file
  - `.Call()` to call the `C/C++` file that's made available
- `cxxfunction` with `rcpp` plugin reduce to `rcpp` function
- Add `LinkingTo: Rcpp` in `DESCRIPTION` when including it in a package.
- Usage of `Rcpp` via `inline` is portable as `R` itself.
- Adding `verbose=TRUE` in `cxxfunction` or `rcpp` shows both the temporary file created by `cxxfunction()` and the invocations by `R CMD SHLIB`.
- Use `include=` in `cxxfunction` to reduce the number of operator needed.
- Plugins provide a general mechanism to supply additional information which may be needed to compile and link the particular package.
- `try throw` from `std::exception` work similar to the `tryCatch` function in `R`; what happen after the `throw` is that a suitable `catch()` segment is identified.
- `Rcpp::cppFunction` vs `inline::cxxFunction`

Chapter 3

- R object itself is internally represented by a `SEXP`, a pointer to a *S expression object*
- Users of **Rcpp** API never need to manually allocate memory, or free it after use.
- User visible classes derive from the `RObject` class:
  - `IntegerVector`/`IntegerMatrix` for vectors/matrices of type `integer`.
  - `NumericVector`/`NumericMatrix` for vectors/matrices of type `numeric`.
  - `LogicalVector`/`LogicalMatrix` for vectors/matrices of type `logical`.
  - `CharacterVector` for vectors of type `character`.
  - `GenericVector` for generic vectors which implement `List` types (equivalent to `List`).
  - `ExpressionVector` for vectors of `expression` types.
  - `RawVector` for vectors of type `raw`.
- `as<>()` function for converting from R to C++ and the `wrap()` function for the inverse direction.
- R integer vectors can be converted into `std::vector<int>`

- Reading a vector: `Rcpp::NumericVector a(b);`, where `a` is the name used in the chunk and `b` is the input specified by the signature in `cxxfunction`.
- Reading a scalar and store it under a name: `double a = Rcpp::as<double>(b)`.
- If more than one vectors are constructed from the input, the code will try to modify copy along with the original vector; need a separate name.
- `clone` is a generic feature of vectors derived from `RObject` object; `a = Rcpp::clone(b)`.
- When calling from the same algorithm, e.g., STL algorithm, `std::` only need to be specified in the first appearance, and `::` afterward.
- `return Rcpp::wrap(xxx);` can be replaced with `return xxx;` when `xxx` is declared through `Rcpp::`.
- Why `clone` in example 3.3.3?

Chapter 4

- The `Named` class promits the usage of named vector.
- `Rcpp::NumericVector::create(` or `NumericVector::create(` to create a named numerical vector.
- `Rcpp::Named("key")` can be replaced with `_["key"]`.
- `Rcpp::List` to create a list.
- `Rcpp::as<double>(list["key"])` to extract components from list `key`.
- `return Rcpp:List::create(Rcpp::Named("key") = xxx, ...` to return a created list.
- An alternative way to create a list is to use `Rcpp::List ll(4)` up front. This requires prior knowledge of the list length. When running out, one can use `push_back()` or `push_front()` for insertion.
- Data frame can be seen as a specialization of a list, with the added restrictions of excluding nesting types and of imposing common length.
- Creating a data frame is similar to creating a list with `Rcpp::DataFrame::create(` and `Rcpp::Named("key") = x` to name.
- `cxxfunction` can pass function, when this happens extern variables that are dumpped into this supplied function are not instantiated and not checked for type matching.
- In the `C` envirment, one can assess `R` function via `Rcpp::Function`
- Prefix `Rf_` is required when calling from `Rcpp` mathematical library.
- Need the head file `#include <Rcpp>` when using `Rcpp` to access R functions in C.
- `S4` object can be created in the `C++` level; `{S4 foo(x);  foo.slot(.Data")= "some data"}`

Chapter 5

- Using `Rcpp` in package
  - Use `.Call()`
  - Add `xxx.h` header file
  - Add `.cpp` file that with the `#include xxx.h` header
  - Add `Depends: Rcpp` and `LinkingTo: Rcpp`
  - Manually add the Rcpp library to `PKG_LIBS` variable in the `Makevars` and `Makevars.win` files.
  - Add `useDynLib()` and `exportPattern()` in `NAMESPACE`.

Chapter 6

- This chapter provides an overview of the steps to extend Rcpp for use with customized classes and class libraries.
- Customized `.h` header should be included before `#include <Rcpp.h>`.

Chapter 7

- Talks about how to expose C++ functions, classes, and modules to R
- When `void` is used as a function return type, it indicates that the function does not return a value.
- When `void` appears in a pointer declaration, it specifies that the pointer is universal.
- When `void` is used in a function's parameter list, void indicates that the function takes no parameters.

- Use `RCPP_MODULE()` to export C++ functions.
- Possible to add documentation in `RRCPP_MODULE()` function. The comments can be printed with `show` in R.
- The exported function does not have formal function arguments. To add funcation arguments (and default values), add `List::create(_["key1"] = 0, _["key2"], _["..."])` in `RCPP_MODULE()`, after the function pointer, and before the documentation entry. In this example, `key2` does not have a default value.
- Do we need to expose classes and modules?

Chapter 8

- `[[Rcpp::export]]` is called an Rcpp attribute, and tells Rcpp to make hello available for use from R.
  - Rcpp sugar takes advantage of C++ operator overloading.
  - Vector operation in Rcpp sugar requires two operants to be of the same length, or one has to be a single primitive C++ type such as `double`.
  - §8.3.2 Functinos producing sugar expressions
    * `is_na` works like `is.na` in R
    * `seq_along(x)` works like `1:length(x)` in R. It only requires `size` of the input.
    * `seq_len(x)` works like `1:x` in R.
    * `pmin` and `pmax` same as these in R
    * `ifelse` requires either 1) two compatible suger expressions or 2) one sugar expressoin and one compatible primitive. Otherwise, works like the `ifelse` in R.
    * `sapply`, `lapply`, `mapply` work like their R counterparts.
    * `sign`, `diff`, `setdiff`, `union_`, `intersect`, `unique`, `sort_unique`, `table`, `duplicated` all work like the R counterparts.
    * `clamp(a, x, b)` returns `pmax(a, pmin(x, b))`
    * Most math/stat functions remain the same

Chapter 10

- `#include <iostream>` and `#include <armadillo>` are included to provide the required declarations.