

Calling C from R

Stat 580: Statistical Computing

- Theme: [Black - White](#)
- [Printable version](#)

References

- "Advanced R", by Hadley Wickham.
- "Writing R Extensions", by R Core Team.

R's C interface

- `.C()`, `.Call()` `.External()` (support variable number of arguments)
- `.Call()` allows
 - inputting R objects to C
 - creation of R objects in C
 - manipulation of R objects in C
 - usage of R functions in C
 - returning R objects
- variables are passed by reference by `.Call()`
- "Writing R Extensions" is a good reference of `.Call()`

R's C interface

- Some setup required:
 - Windows: [Rtools](#)
 - Mac: Xcode command line tools
 - (Most) Linux distributions: usually comes with the required compilers

Hello world (again?!)

- C code:

```
/* Rhello.c */  
  
#include <R.h>  
  
#include <Rinternals.h>  
/* alternative: Rdefines.h, which includes Rinternals */  
  
SEXP sayhello(){  
    Rprintf("Hello world\n");  
    return R_NilValue;  
}
```

- in command line:

```
R CMD SHLIB Rhello.c
```

- you will get a shared object (.so) (.dll in Windows)

Hello world (again?!)

- in R: (change to the right directory)

```
dyn.load("Rhello.so")  
out <- .Call("sayhello")
```

- we usually write a wrapper function for ease of calling:

```
hello <- function(){  
  .Call("sayhello")  
}
```

Hello world (again?!)

```
/* Rhello.c */

#include <R.h>

#include <Rinternals.h>
/* alternative: Rdefines.h, which includes Rinternals */

SEXP sayhello(){
    Rprintf("Hello world\n");
    return R_NilValue;
}
```

- commonly used header files: R.h, Rinternals.h, Rdefines.h, Rmath.h
 - check [here](#)
- function input and output are both of type SEXP
 - SEXP standards for S Expression
 - SEXP is actually a pointer to SEXPREC
 - this can be treated as the R object in C

Hello world (again?!)

```
/* Rhello.c */

#include <R.h>

#include <Rinternals.h>
/* alternative: Rdefines.h, which includes Rinternals */

SEXP sayhello(){
  Rprintf("Hello world\n");
  return R_NilValue;
}
```

- `Rprintf()` is defined in `R_ext/Print.h` which is included in `R.h`
 - customized `printf()` for R output
- `R_NilValue` is the `NULL` in R
 - `R_NilValue` is of type `NILSXP`, a subtype of `SEXP`

SEXP

In C, R objects are stored in a common data type, `SEXP`:

- `SEXP` is a variant type, with subtypes for all R's data structures
 - `REALSXP`: numeric vector
 - `INTSXP`: integer vector
 - `LGLSXP`: logical vector
 - `STRSXP`: character vector
 - `VECSXP`: list
 - `CLOSXP`: function (closure)
 - `ENVSXP`: environment
 - [more](#)

Example

```
/* vecprod.c */
#include <R.h>
#include <Rinternals.h>

SEXP vecprod(SEXP Rx){
    double *x, temp=1;
    int i, n;

    x = REAL(Rx);
    n = length(Rx);

    for (i=0; i<n; i++){
        temp *= x[i];
    }

    Rprintf("The product is %f.\n", temp);
    return R_NilValue;
}
```

Example

Compile with R CMD SHLIB. In R, we write

```
dyn.load("vecprod.so")  
.
```

- Why not `.Call("vecprod", 1:3)`?
 - wrapper function:

```
vecsum <- function(x) .Call("vecprod", as.double(x))
```

- To coerce objects at the C level, use `PROTECT(coerceVector(old, SEXPTYPE))`. (we will talk about `PROTECT()` later.)

Another example

```
x = REAL(Rx);  
n = length(Rx);
```

- `REAL()` returns a pointer to the C array inside numeric vector `Rx` (R object)
 - if `Rx` is a vector, you can use `REAL(Rx)[1]` to refer to the second element of `Rx`.
 - other similar (commonly used) functions: `CHAR()`, `INTEGER()`, `LOGICAL()`
- `length()` returns the length of a vector

Garbage collection

- the memory allocated for R objects is not freed by the user
- the memory is freed from time to time by a process called garbage collection
- If you create R object (to be precise, `SEXP`) in C, you must tell R that the object is in use
 - otherwise R will destroy it during garbage collection
 - you can use the `PROTECT` macro (on the pointer `SEXP`) to achieve this
- you don't have to protect/unprotect R objects passed from R

PROTECT ()

- note that it is the object (`SEXPREC`) is protected, not the pointer `SEXPR`
 - if you invoke `PROTECT (p)`, it is the object that `p` is pointing to at this time get protected
 - if you change where `p` is pointing to later, this new object may not be protected
- the programmer is responsible for un-protecting the protected object by `UNPROTECT (n)`:
 - it unprotects the last `n` objects which were protected.

Example

```
/* vecprod2.c */
#include <R.h>
#include <Rinternals.h>

SEXP vecprod(SEXP Rx){
    double *x, temp=1;
    int i, n;
    SEXP Rout = PROTECT(allocVector(REALSXP, 1));

    x = REAL(Rx);
    n = length(Rx);
    for (i=0; i<n; i++){
        temp *= x[i];
    }
    REAL(Rout)[0] = temp;
    UNPROTECT(1);
    return Rout;
}
```

- `allocVector(REALSXP, n)` creates an SEXP object corresponding to a numeric vector of length `n`
 - replace `REALSXP` by `INTSXP`, `VECSXP` to create integer vector and list

Lists

- accessing vector data: `REAL()`, `INTEGER()`, ... (use them to get a pointer; work as if dealing with a C array)
- for lists, each element is `SEXP`
- `allocVector(VECSXP, n)`: allocate a list of length `n`
- `SET_VECTOR_ELT(x, i, value)`: set the `i`-th (C indexing) of `x` to `value`
- `VECTOR_ELT(x, i)`: access the value of the `i`-th (C indexing) of `x`

Lists

```
double xsum=0, xmean, *x;
int n, i;
SEXP Rout;

/* codes for computation */

Rout = PROTECT(allocVector(VECSXP, 2));
SET_VECTOR_ELT(Rout, 0, ScalarReal(xsum));
SET_VECTOR_ELT(Rout, 1, ScalarReal(xmean));

UNPROTECT(1);
return Rout;
```

Useful functions for coercing scalars

- `asLogical(x): LGLSXP to int`
- `asInteger(x): INTSXP to int`
- `asReal(x): REALSXP to double`
- `CHAR(asChar(x)): STRSXP to const char*`
- `ScalarLogical(x): int to LGLSXP`
- `ScalarInteger(x): int to INTSXP`
- `ScalarReal(x): double to REALSXP`
- `mkString(x): const char* to STRSXP`

Example

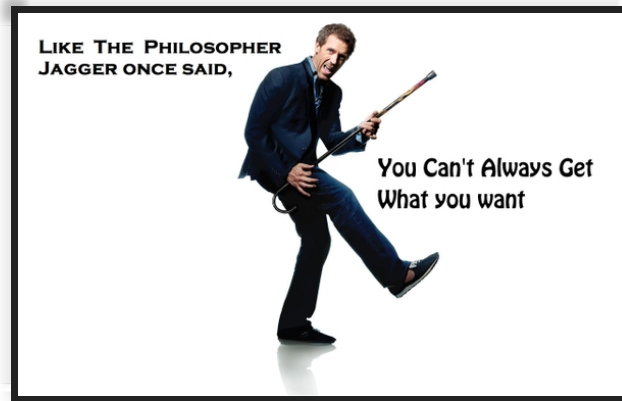
```
#include <R.h>
#include <Rinternals.h>

SEXP myrowsum(SEXP Rmat){
    double *mat, *out;
    int n, m, i, j;
    SEXP Rout;

    /* alternative:
    n = INTEGER(getAttrib(Rmat, R_DimSymbol))[0];
    m = INTEGER(getAttrib(Rmat, R_DimSymbol))[1];
    */
    n = nrows(Rmat);
    m = ncols(Rmat);
    mat = REAL(Rmat);
    Rout = PROTECT(allocVector(REALSXP, n));
    out = REAL(Rout);
    for (i=0; i<n; i++){
        out[i] = 0;
        for (j=0; j<m; j++){
            out[i] += mat[i + n * j];
        }
    }
    UNPROTECT(1);
    return Rout;
}
```

Example

```
y <- matrix(rnorm(100), nr=10)
microbenchmark(.Call("myrowsum", y), apply(y, 1, sum), rowSums(y))
```



- "But if you try sometimes, well you might find... You get what you need!"

```
y <- matrix(rnorm(100000), nr=1000)
microbenchmark(.Call("myrowsum", y), apply(y, 1, sum), rowSums(y))
```

Picture credit: www.quotesaga.com

Example

```
#include <R.h>
#include <Rinternals.h>

SEXP myasvec(SEXP x){
    SEXP y;
    y = PROTECT(duplicate(x));
    setAttrib(y, R_DimSymbol, ScalarReal(length(y)));
    UNPROTECT(1);
    return y;
}
```

```
#include <R.h>
#include <Rinternals.h>

SEXP myasvec(SEXP x){
    setAttrib(x, R_DimSymbol, ScalarReal(length(x)));
    return R_NilValue;
}
```

Randomness

```
#include <R.h>
#include <Rinternals.h>
#include <Rmath.h>

SEXP myrnorm(SEXP Rn){

    int i, n = asInteger(Rn);
    SEXP Rout = PROTECT(allocVector(REALSXP, n));
    double * out = REAL(Rout);

    GetRNGstate();

    for (i=0; i<n; i++){
        out[i] = norm_rand();
    }

    PutRNGstate();

    UNPROTECT(1);
    return Rout;
}
```

- use `GetRNGstate()` and `PutRNGstate()` to get/return the R RNG state from/to R.

Randomness

```
set.seed(1234)  
.Call("myrnorm", 10)
```

```
set.seed(1234)  
rnorm(10)
```

LAPACK

```
/* vecprod.c */
#include <R.h>
#include <Rinternals.h>
#include <R_ext/Lapack.h>

SEXP mysolve(SEXP RA, SEXP RB){
    double *A, *B, *A1, *B1;
    int i, j, n1, n2, info, *ipiv;
    SEXP Rout;

    A = REAL(RA);
    B = REAL(RB);
    n1 = nrows(RA);
    n2 = ncols(RB);

    Rout = PROTECT(allocMatrix(REALSXP, n1, n2));
    A1 = (double*)malloc(sizeof(double)*(n1*n1));
    ipiv = (int*)malloc(sizeof(int)*n1);
    B1 = REAL(Rout);
```


LAPACK

```
for (i=0; i<(n1*n1); i++)  
    A1[i] = A[i];  
  
for (i=0; i<(n1*n2); i++)  
    B1[i] = B[i];  
  
dgesv_(&n1, &n2, A1, &n1, ipiv, B1, &n1, &info);  
  
free(A1);  
free(ipiv);  
UNPROTECT(1);  
return Rout;  
}
```