

Topics

- we are going to see:
 - how to pass a list to a C function and use it within the function
 - how to create a new list from within a C function and return it

Return A List I

- allocate a new list: suppose our returned list is going to have two named elements called `samples` and `n_iters`
- in R, a list is a generic vector
- its elements are of type `VECSXP` as opposed to `REALSXP` or `INTSXP`
- to allocate the list we do:

```
SEXP retList;  
PROTECT(retList = allocVector(VECSXP, 2));  
++nProtected;
```

- to attach the element names we should do:

```
SEXP names;  
PROTECT(names = allocVector(STRSXP, 2));  
++nProtected;  
SET_STRING_ELT(names, 0, mkChar("samples"));  
SET_STRING_ELT(names, 1, mkChar("n_iters"));  
  
setAttrib(retList, R_NamesSymbol, names);
```

Return A List II

- to fill up the elements we do:

- for samples, say its vector of length `n_iters`:

```
SEXP samples_vec;  
PROTECT(samples_vec = allocVector(REALSXP, n_iters));  
++nProtected;  
GetRNGstate();  
for (ii = 0; ii < n_iters; ++ii)  
    REAL(samples_vec)[ii] = rnorm(0, 1.0);  
PutRNGstate();
```

- now say `n_iters` has been passed as an argument but we want to return it as well just to report back:

```
SEXP n_iters_vec;  
PROTECT(n_iters_vec = allocVector(INTSXP, 1));  
++nProtected;  
INTEGER(n_iters_vec)[0] = n_iters;
```

- now to add these elements to the list we do:

```
SET_VECTOR_ELT(retList, 0, samples_vec);  
SET_VECTOR_ELT(retList, 1, n_iters_vec);
```

Return A List III

- we are going to see a rather silly but illustrative example of a C function which returns a `list` (type: `VECSXP`) which has the following components:
 1. an integer vector (type: `INTSXP`)
 2. a numeric vector (type: `REALSXP`)
 3. a character vector (type: `STRSXP`)
 4. a matrix of given dimension (type: `REALSXP`)
 5. another matrix of given dimension (type: `REALSXP`)
 6. a `list` (type: `VECSXP`)

Return A List IV

- the R side:

```
doStuff <-  
  function (matDim)  
  {  
    .Call("do_stuff",  
          as.integer(matDim))  
  }
```

Return A List V

- the C side (part I):

```
#define MAX_LINE_LENGTH 100
SEXP
do_stuff (SEXP mat_dim)
{
    int ii, jj, nProtected = 0, n_comps = 0, len, nrows, ncols;
    char tmp[MAX_LINE_LENGTH];
    SEXP ivals_vec, dvals_vec, cvals_vec, mat1, mat2, dim;
    SEXP list_ivals_vec, list;
    SEXP names, ret_list;

    /*
     * "create" an integer vector
     */
    len = 3;
    PROTECT(ivals_vec = allocVector(INTSXP, len));
    ++nProtected;
    for (ii = 0; ii < len; ++ii) {
        INTEGER(ivals_vec)[ii] = ii;
    }
    ++n_comps;
}
```

Return A List VI

- the C side (part II):

```
/*
 * "create" a numeric vector
 */
len = 5;
PROTECT(dvals_vec = allocVector(REALSXP, len));
++nProtected;
for (ii = 0; ii < len; ++ii) {
    REAL(dvals_vec)[ii] = rnorm(0, 1.0);
}
++n_comps;

/*
 * "create" a character/string vector
 */
len = 5;
PROTECT(cvals_vec = allocVector(STRSXP, len));
++nProtected;
for (ii = 0; ii < len; ++ii) {
    sprintf(tmp, "Hello: %d", ii);
    SET_STRING_ELT(cvals_vec, ii, mkChar(tmp));
}
++n_comps;
```

Return A List VII

- the C side (part III):

```
/*
 * "create" a matrix
 */
nrows = INTEGER(mat_dim)[0];
ncols = INTEGER(mat_dim)[1];
PROTECT(mat1 = allocVector(REALSXP, nrows * ncols));
++nProtected;
for (jj = 0; jj < ncols; ++jj) {
    for (ii = 0; ii < nrows; ++ii) {
        REAL(mat1)[jj * nrows + ii] = rnorm(0, 1.0);
    }
}
PROTECT(dim = allocVector(INTSXP, 2));
++nProtected;
INTEGER(dim)[0] = nrows;
INTEGER(dim)[1] = ncols;
setAttrib(mat1, R_DimSymbol, dim);
++n_comps;
```

Return A List VIII

- the C side (part IV):

```
/*
 * "create" another matrix
 */
nrows = INTEGER(mat_dim)[0];
ncols = INTEGER(mat_dim)[1];
PROTECT(mat2 = allocVector(REALSXP, nrows * ncols));
++nProtected;
for (jj = 0; jj < ncols; ++jj) {
    for (ii = 0; ii < nrows; ++ii) {
        REAL(mat2)[jj * nrows + ii] = rnorm(0, 1.0);
    }
}
setAttrib(mat2, R_DimSymbol, mat_dim);
++n_comps;
```

Return A List IX

- the C side (part V):

```
/*
 * "create" an unnamed list
 */
len = 2;
PROTECT(list = allocVector(VECSXP, len));
++nProtected;
len = 3;
PROTECT(list_ivals_vec = allocVector(INTSXP, len));
++nProtected;
for (ii = 0; ii < len; ++ii) {
    INTEGER(list_ivals_vec)[ii] = ii;
}
SET_VECTOR_ELT(list, 0, list_ivals_vec);
SET_VECTOR_ELT(list, 1, ival_vec);
++n_comps;
```

Return A List X

- the C side (part VI):

```
/*
 * "create" the final list
 */
PROTECT(ret_list = allocVector(VECSXP, n_comps));
++nProtected;
SET_VECTOR_ELT(ret_list, 0, ival_vec);
SET_VECTOR_ELT(ret_list, 1, dval_vec);
SET_VECTOR_ELT(ret_list, 2, cval_vec);
SET_VECTOR_ELT(ret_list, 3, mat1);
SET_VECTOR_ELT(ret_list, 4, mat2);
SET_VECTOR_ELT(ret_list, 5, list);
PROTECT(names = allocVector(STRSXP, n_comps));
++nProtected;
SET_STRING_ELT(names, 0, mkChar("ival_vec"));
SET_STRING_ELT(names, 1, mkChar("dval_vec"));
SET_STRING_ELT(names, 2, mkChar("cval_vec"));
SET_STRING_ELT(names, 3, mkChar("mat1"));
SET_STRING_ELT(names, 4, mkChar("mat2"));
SET_STRING_ELT(names, 5, mkChar("list"));
setAttrib(ret_list, R_NamesSymbol, names);

UNPROTECT(nProtected);
return ret_list;
}
```

Pass A List I

- you could take a list as an argument to your R function and pass it to the underlying C function
- you could create a list within your R function and then pass it to the underlying C function
- as an illustrative application of this technique let's consider the following:
 - pass all the arguments to a R function as a list to the underlying C function: `SEXP foo(SEXP argslst)`

Pass A List II

- passing all the arguments to a R as a list to SEXP `foo(SEXP argslst)`
 - we can use either the `.Call` or the `.External` interface because either would support one arguments for sure
 - to extract the individual parameters from this list we use the following function

```
SEXP
getListElement (SEXP list, char *str)
{
    SEXP elmt = R_NilValue, names = getAttrib(list, R_NamesSymbol);
    int i;

    for (i = 0; i < length(list); i++)
        if(strcmp(CHAR(STRING_ELT(names, i)), str) == 0) {
            elmt = VECTOR_ELT(list, i);
            break;
        }
    return elmt;
}
```

Pass A List III

- suppose our R side of things look like:

```
doStuff <-  
  function (nIters, timeInSecs, propBurnIn)  
  {  
    args <- list("n_iters" = as.integer(nIters),  
                "time_in_secs" = as.numeric(timeInSecs),  
                "prop_burn_in" = as.numeric(propBurnIn))  
    .External("do_stuff",  
              args)  
  }
```

- in the C function SEXP do_stuff (SEXP args) we should then do the following to extract the arguments:

```
SEXP list;  
args = CDR(args);  
list = CAR(args);  
n_iters = INTEGER(getListElement(list, "n_iters"))[0];  
time_in_secs = REAL(getListElement(list, "time_in_secs"))[0];  
prop_burn_in = REAL(getListElement(list, "prop_burn_in"))[0];
```

Pass A List IV

- note, if we had used the `.Call` interface above then we could have just said:

```
SEXP list = args;
n_iters = INTEGER(getListElement(list, "n_iters"))[0];
time_in_secs = REAL(getListElement(list, "time_in_secs"))[0];
prop_burn_in = REAL(getListElement(list, "prop_burn_in"))[0];
```

- note after extracting the arguments as shown before one can do whatever the C was originally written for
- this use of a list to pass all the potential arguments to a function and extract the individual arguments with the `getListElement()` function is very convenient
 - it does not require you to make passing a ton of arguments in their correct positions in a `.Call` interface
 - it eliminates the need of correct positional use of complicated combinations of `CAR()`s and `CDR()`s in the `.External` interface

Code Files

prog7.c

prog7.R

prog8.c

prog8.R