

Using C and C++ with R

Laurent Gatto lg390@cam.ac.uk

University of Cambridge

May 15, 2013

Calling foreign languages

Calling foreign languages

Foreign languages

- ▶ C, C++¹,
- ▶ Fortran
- ▶ Java².

Other scripting languages

- ▶ R/Perl³ and R/Python⁴ bidirectional interfaces.
- ▶ There is also the `system()` function for direct access to OS functions.

¹<http://dirk.eddelbuettel.com/code/rcpp.html>

²<http://www.rforge.net/rJava/>

³<http://www.omegahat.org/RSPerl/>

⁴<http://www.omegahat.org/RSPython/>

- ▶ **Why** R is getting slow or is not doing well in terms of memory management.
- ▶ **When** R can't do better **and** the slow code has been identified → code proiling.

- ▶ **Why** R is getting slow or is not doing well in terms of memory management.
- ▶ **When** R can't do better **and** the slow code has been identified → code proiling.
- ▶ **Why** Re-using existing infrastructure

R's build-in C interfaces

- ▶ Better know how to program in C.
- ▶ Documentation is not always easy to follow: R-Ext, R Internals as well as R and other package's code.

.C

- ▶ *Easy* way
- ▶ Arguments and return values must be *primitive* (vectors of doubles or integers)

.Call

- ▶ Accepts R data structures as arguments and return values (SEXP and friends) (no type checking is done though).
- ▶ Memory management: memory allocated for R objects is garbage collected. Thus R objects in C code, you must be explicitly `PROTECT`ed to avoid being `gc()`ed and

.Call example

Example

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

SEXP gccount(SEXP inseq) {
    int i, l;
    SEXP ans, dnaseq;
    PROTECT(dnaseq = STRING_ELT(inseq, 0));
    l = LENGTH(dnaseq);
    printf("length %d\n", l);
    PROTECT(ans = NEW_NUMERIC(4));

    for (i = 0; i < 4; i++)
        REAL(ans)[i] = 0;

    for (i = 0; i < l; i++) {
        char p = CHAR(dnaseq)[i];
        if (p=='A')
            REAL(ans)[0]++;
        else if (p=='C')
            REAL(ans)[1]++;
        else if (p=='G')
            REAL(ans)[2]++;
        else if (p=='T')
            REAL(ans)[3]++;
        else
            error("Wrong alphabet");
    }
    UNPROTECT(2);
    return(ans);
}
```


Using your C code

Directly

1. Create a shared library: `R CMD SHLIB gccount.c`
2. Load the shared object: `dyn.load("gccount.so")`
3. Create an R function that uses it: `gccount <- function(inseq) .Call("gccount",inseq)`
4. Use you C code: `gccount("GACAGCATCA")`

In a package

- ▶ Document you function.
- ▶ Export the shared objects with `useDynLib` in your `NAMESPACE`.
- ▶ (Without `NAMESPACE`, overwrite `.First.lib` to `dyn.load` you shared object.)

Example

In *sequences*, we have

- ▶ The gccount.c code in src.
- ▶ Defined a R function in R/functions.R

```
gccount <- function(inseq) {  
  .Call("gccount",  
        inseq,  
        PACKAGE="sequences")  
}
```
- ▶ Written the man/gccount.Rd man page.
- ▶ Exported the function in NAMESPACE using `export(gccount)` and the shared library with `useDynLib(sequences)`

Example

```
## Loading required package: Rcpp  
## This is package 'sequences'
```

```
s <- "GACTACGA"  
gccount  
  
## function (inseq)  
## {  
##     .Call("gccount", inseq, PACKAGE = "sequences")  
## }  
## <environment: namespace:sequences>  
  
gccount(s)  
  
## [1] 3 2 2 1
```

Using C/C++ code with Rcpp

Rcpp package

Rcpp is a great package for writing both C and C++ code:

- ▶ It comes with loads of documentation and examples.
- ▶ All basic R types are implemented as C++ classes.
- ▶ No need to worry about garbage collection.
- ▶ With package inline code can be easily compiled in R

```
library(Rcpp)
library(inline)

##
## Attaching package: 'inline'
## The following object is masked from 'package:Rcpp':
##
##      registerPlugin

cppCode <- "\nRcpp::NumericVector cx(x);\nRcpp::NumericVector ret(1);\nret[0] =
squareOne <- cxxfunction(signature(x = \"numeric\"), plugin = \"Rcpp\", body = cppC
squareOne(10)
```

Example

```
#include <Rcpp.h>

using namespace Rcpp;

RcppExport SEXP gccount2(SEXP inseq){
  Rcpp::CharacterVector dnaseq(inseq);
  Rcpp::NumericVector ans(4);
  std::string s = Rcpp::as<std::string>(dnaseq);

  for (int i = 0; i < 4; i++)
    ans[i] = 0;

  for (int i = 0; i < s.size(); i++) {
    char p = s[i];
    if (p=='A')
      ans[0]++;
    else if (p=='C')
      ans[1]++;
    else if (p=='G')
      ans[2]++;
    else if (p=='T')
      ans[3]++;
    else
      Rf_error("Wrong alphabet");
  }

  return(ans);
}
```

Using in a package

1. You will need a Makevars file in the src directory
2. Modify DESCRIPTION file:
Depends: Rcpp
LinkingTo: Rcpp
3. Create an R function that uses it: gccount2 <-
function(inseq) .Call("gccount2",inseq)
4. In NAMESPACE file export the shared objects with useDynLib.

Example

See package sequences for a working example.

Further reading

- ▶ R Installation and Administration, R Core team
- ▶ Writing R Extensions, R Core team
- ▶ R Programming for Bioinformatics, Robert Gentleman
- ▶ *Rcpp* documentation
- ▶ Relevant devtools sections: *C interface* and *Rcpp*.

- ▶ This work is licensed under a CC BY-SA 3.0 License
- ▶ Course web page:
<https://github.com/lgatto/TeachingMaterial>

Thank you for your attention.