Intro to C++ and Rcpp

Colin Rundel

2019-02-26

Types

Type	Size (bits*)	Description	Value Range
`bool`	*	Logical value: `true` or `false`	
`char`	8	Character (ASCII or UTF8)	± 127
`short int`	16	Small integers	$\pm 3.27 \cdot 10^4$
`int`	32	Medium integers	$\pm 2.14 \cdot 10^9$
`long int`	64	Large integers	$\pm 9.22 \cdot 10^18$
`float`	32	Small floating point value	$\pm 10^{-38}$ to $\pm 10^{38}$
`double`	64	Large floating point value	$\pm 10^{-308}$ to $\pm 10^{308}$

R types vs C++ types

All of the basic types in R are vectors by default, in C++ the types we just discussed are all scalar. So it is necessary to have one more level of abstraction to translate between the two. Rcpp provides for this with several built in classes:

C++ type (scalar)	Rcpp Class	R type (typeof)
int	Rcpp::IntegerVector	integer
double	Rcpp::NumericVector	numeric
bool	Rcpp::LogicalVector	logical
std::string	Rcpp::CharacterVector	character
char	Rcpp::RawVector	raw
std::complex<double>	Rcpp::ComplexVector	complex
	Rcpp::List	list
	Rcpp::Environment	environment
	Rcpp::Function	function
	Rcpp::XPtr	externalptr
	Rcpp::S4	S4

Trying things out

Rcpp provides some helpful functions for trying out simple C++ expressions (`evalCpp`), functions (`cppFunction`), or cpp files (`sourceCpp`). It is even possible to include C++ code in RMarkdown documents using the Rcpp engine.

```
evalCpp("2+2")
```

```
## [1] 4
```

```
evalCpp("2+2") %>% typeof()
```

```
## [1] "integer"
```

```
evalCpp("2+2.") %>% typeof()
```

```
## [1] "double"
```

What's happening?

```
evalCpp("2+2", verbose = TRUE, rebuild = TRUE)
```

```
##
## Generated code for function definition:
## --------------------------------------------------------
##
## #include <Rcpp.h>
##
## using namespace Rcpp;
##
## // [[Rcpp::export]]
## SEXP get_value(){ return wrap( 2+2 ) ; }
##
## Generated extern "C" functions
## --------------------------------------------------------
##
##
## #include <Rcpp.h>
## // get_value
## SEXP get_value();
## RcppExport SEXP sourceCpp_1_get_value() {
## BEGIN_RCPP
##     Rcpp::RObject rcpp_result_gen;
##     Rcpp::RNGScope rcpp_rngScope_gen;
##     rcpp_result_gen = Rcpp::wrap(get_value());
##     return rcpp_result_gen;
## END_RCPP
## }
##
## Generated R functions
## --------------------------------------------------------
##
## `.sourceCpp_1_DLLInfo` <- dyn.load('/private/var/folders/k8/4z3ndjqd0nj1xczfp4ktnnzc0000gp/T/RtmpLYvWAG/sourceCpp-x86_64-apple-darwin18.2.0-1.0.1/sourcecpp_29a
##
## get_value <- Rcpp:::sourceCppFunction(function() {}, FALSE, `.sourceCpp_1_DLLInfo`, 'sourceCpp_1_get_value')
##
## rm(`.sourceCpp_1_DLLInfo`)
##
## Building shared library
## --------------------------------------------------------
##
## DIR: /private/var/folders/k8/4z3ndjqd0nj1xczfp4ktnnzc0000gp/T/RtmpLYvWAG/sourceCpp-x86_64-apple-darwin18.2.0-1.0.1/sourcecpp_29a327d895fb
##
## /usr/local/Cellar/r/3.5.3/lib/R/bin/R CMD SHLIB -o 'sourceCpp_5.so' --preclean  'file29a3784f724a.cpp'

## [1] 4
```

C++ functions as R functions

```
cppFunction('
double cpp_mean(double x, double y) {
  return (x+y)/2;
}
')
cpp_mean
```

```
## function (x, y)
## .Call(<pointer: 0x10f5ccc40>, x, y)
```

```
cpp_mean(1,2)
```

```
## [1] 1.5
```

```
cpp_mean(TRUE,2L)
```

```
## [1] 1.5
```

```
cpp_mean(1,"A")
```

```
## Error in cpp_mean(1, "A"): Not compatible with requested type: [type=character; target=double].
```

```
cpp_mean(c(1,2), c(1,2))
```

```
## Error in cpp_mean(c(1, 2), c(1, 2)): Expecting a single value: [extent=2].
```

Using sourceCpp

Generally this is the prefered way of working with C++ code and is well supported by RStudio.

- Make sure to include the Rcpp header

```
#include <Rcpp.h>
```

- If you hate typing `Rcpp::` everywhere, include the namespace

```
using namespace Rcpp;
```

- Specify any desired plugins with

```
// [[Rcpp::plugins(cpp11)]]
```

- Prefix any functions that will be exported with R with

```
// [[Rcpp::export]]
```

- Testing code can be included using an R code block:

```
/*** R
# This R code will be run automatically
*/
```

Example

The following would be included in a file called `mean.cpp` or something similar.

```
#include <Rcpp.h>

//[[Rcpp::plugins(cpp11)]]

//[[Rcpp::export]]
double cpp_mean(double x, double y) {
  return (x+y)/2;
}

/*** R
x <- runif(1e5)
bench::mark(
  cpp_mean(1, 2),
  mean(c(1, 2))
)
*/
```

`for` loops

In C & C++ `for` loops are traditionally constructed as,

```
for(initialization; end condition; increment) {
  //...loop code ..
}
```

```
#include <Rcpp.h>
//[[Rcpp::export]]
double cpp_mean(Rcpp::NumericVector x) {
  double sum = 0.0;
  for(int i=0; i != x.size(); i++) {
    sum += x[i];
  }
  return sum/x.size();
}
```

```
cpp_mean(1:10)
```

```
## [1] 5.5
```

Range based for loops (C++11)

Since the adoption of the C++11 standard there is an alternative for loop syntax,

```
#include <Rcpp.h>
//[[Rcpp::plugins(cpp11)]]

//[[Rcpp::export]]
double cpp11_mean(Rcpp::NumericVector x) {
  double sum = 0.0;
  for(auto v : x) {
    sum += v;
  }

  return sum/x.size();
}
```

```
cpp11_mean(1:10)
```

```
## [1] 5.5
```

```
ls(envir = Rcpp:::.plugins)
```

```
##  [1] "cpp0x"         "cpp11"         "cpp14"         "cpp17"
##  [5] "cpp1y"         "cpp1z"         "cpp2a"         "cpp98"
##  [9] "openmp"        "unwindProtect"
```

Rcpp Sugar

Rcpp also attempts to provide many of the base R functions within the C++ scope, generally there are refered to as Rcpp Sugar, more can be found here or by examining the Rcpp source.

```
#include <Rcpp.h>
//[[Rcpp::plugins(cpp11)]]

//[[Rcpp::export]]
double rcpp_mean(Rcpp::NumericVector x) {
  return Rcpp::mean(x);
}
```

```
rcpp_mean(1:10)
```

```
## [1] 5.5
```

Edge cases

```
x = c(1:10,NA)
typeof(x)
```

```
## [1] "integer"
```

```
mean(x)
```

```
## [1] NA
```

```
cpp_mean(x)
```

```
## [1] NA
```

```
cpp11_mean(x)
```

```
## [1] NA
```

```
rcpp_mean(x)
```

```
## [1] NA
```

```
y = c(1:10,Inf)
typeof(y)
```

```
## [1] "double"
```

```
mean(y)
```

```
## [1] Inf
```

```
cpp_mean(y)
```

```
## [1] Inf
```

```
cpp11_mean(y)
```

```
## [1] Inf
```

```
rcpp_mean(y)
```

```
## [1] Inf
```

Integer mean

```cpp
#include <Rcpp.h>
//[[Rcpp::plugins(cpp11)]]

//[[Rcpp::export]]
double cpp_imean(Rcpp::IntegerVector x) {
  double sum = 0.0;
  for(int i=0; i != x.size(); i++) {
    sum += x[i];
  }

  return sum/x.size();
}

//[[Rcpp::export]]
double cpp11_imean(Rcpp::IntegerVector x) {
  double sum = 0.0;
  for(auto v : x) {
    sum += v;
  }

  return sum/x.size();
}

//[[Rcpp::export]]
double rcpp_imean(Rcpp::IntegerVector x) {
  return Rcpp::mean(x);
}
```

Integer edge cases

```
x = c(1:10,NA)
typeof(x)
```

```
## [1] "integer"
```

```
mean(x)
```

```
## [1] NA
```

```
cpp_imean(x)
```

```
## [1] -195225781
```

```
cpp11_imean(x)
```

```
## [1] -195225781
```

```
rcpp_imean(x)
```

```
## [1] NA
```

```
y = c(1:10,Inf)
typeof(y)
```

```
## [1] "double"
```

```
mean(y)
```

```
## [1] Inf
```

```
cpp_imean(y)
```

```
## Warning in cpp_imean(y): NAs introduced by
```

```
## [1] -195225781
```

```
cpp11_imean(y)
```

```
## Warning in cpp11_imean(y): NAs introduced b
```

```
## [1] -195225781
```

```
rcpp_imean(y)
```

```
## Warning in rcpp_imean(y): NAs introduced by
```

```
## [1] NA
```

Missing values - C++ Scalars

From Hadley's Adv-R Rcpp chapter,

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::List scalar_missings() {
  int int_s          = NA_INTEGER;
  Rcpp::String chr_s = NA_STRING;
  bool lgl_s         = NA_LOGICAL;
  double num_s       = NA_REAL;

  return Rcpp::List::create(int_s, chr_s, lgl_s, num_s);
}
```

```
scalar_missings()
```

```
## [[1]]
## [1] NA
##
## [[2]]
## [1] NA
##
## [[3]]
## [1] TRUE
##
## [[4]]
## [1] NA
```

Missing values - Rcpp Vectors

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::List vector_missing() {
  return Rcpp::List::create(
    Rcpp::NumericVector::create(NA_REAL),
    Rcpp::IntegerVector::create(NA_INTEGER),
    Rcpp::LogicalVector::create(NA_LOGICAL),
    Rcpp::CharacterVector::create(NA_STRING)
  );
}
```

```
vector_missing()
```

```
## [[1]]
## [1] NA
##
## [[2]]
## [1] NA
##
## [[3]]
## [1] NA
##
## [[4]]
## [1] NA
```

Performance

```
r_mean = function(x) {
  sum = 0
  for(v in x) {
    sum = sum + v
  }
  sum / length(x)
}
```

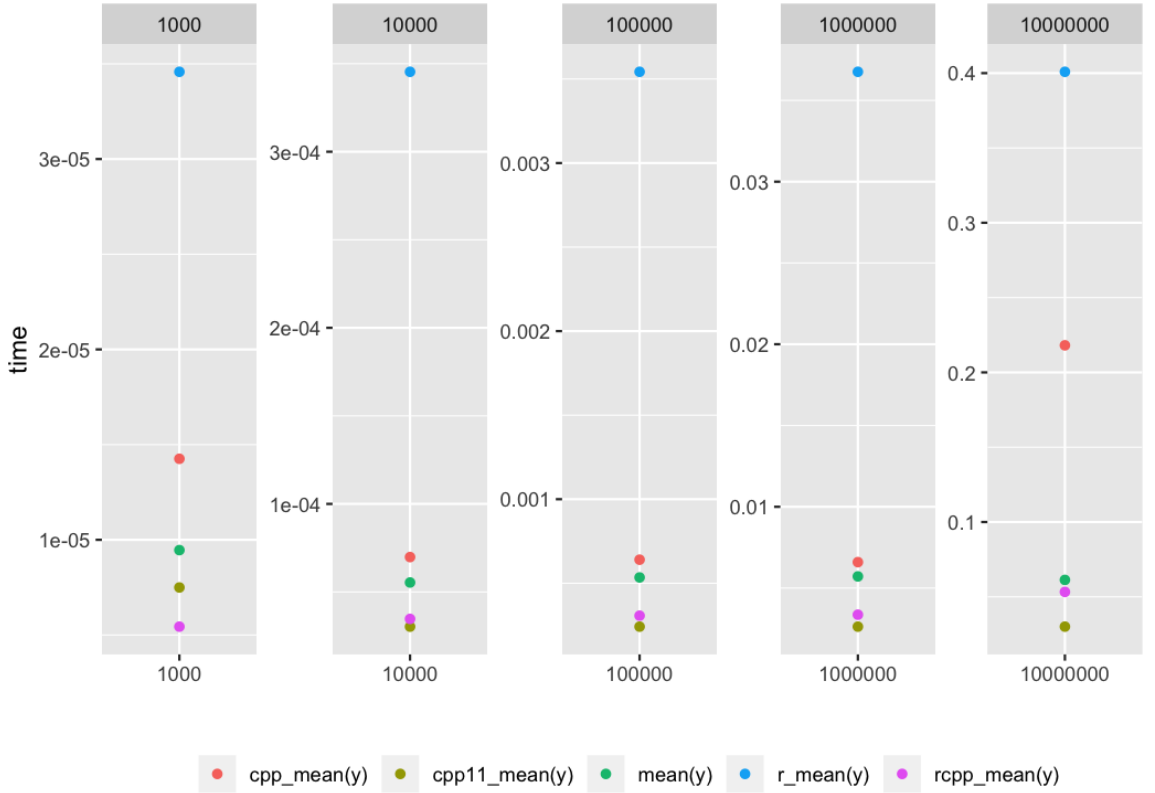
```
y = seq_len(1e6)
bench::mark(
  mean(y),
  cpp_mean(y),
  cpp11_mean(y),
  rcpp_mean(y),
  r_mean(y)
)
```

```
## # A tibble: 5 x 10
##   expression     min    mean  median     max `itr/sec` mem_alloc  n_gc
##   <chr>      <bch:t> <bch:t> <bch:t> <bch:t>     <dbl> <bch:byt> <dbl>
## 1 mean(y)     5.81ms  6.58ms  6.42ms  8.17ms     152.         0B     0
## 2 cpp_mean(…  5.49ms  7.99ms  7.48ms 16.75ms     125.     7.63MB    11
## 3 cpp11_mea…  2.12ms   2.9ms  2.83ms   4.8ms     344.     7.63MB    20
## 4 rcpp_mean…  2.81ms  3.93ms  3.83ms   7.3ms     255.     7.63MB    14
## 5 r_mean(y)   31.9ms 35.78ms 35.94ms 39.87ms      28.0  100.19KB     0
## # … with 2 more variables: n_itr <int>, total_time <bch:tm>
```

bench::press

```
b = bench::press(
  n = 10^c(3:7),
  {
    y = seq_len(n)
    bench::mark(
      mean(y),
      cpp_mean(y),
      cpp11_mean(y),
      rcpp_mean(y),
      r_mean(y)
    )
  }
)
```

```
## Running with:
##          n

## 1     1000

## 2    10000

## 3   100000

## 4  1000000

## 5 10000000
```


1000
10000
100000
1000000
10000000
time
3e-05
2e-05
1e-05
3e-04
2e-04
1e-04
0.003
0.002
0.001
0.03
0.02
0.01
0.4
0.3
0.2
0.1
1000
10000
100000
1000000
10000000
cpp_mean(y)
cpp11_mean(y)
mean(y)
r_mean(y)
rcpp_mean(y)

Creating a `list`

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::List make_list(int n) {
  return Rcpp::List::create(
    Rcpp::Named("norm") = Rcpp::rnorm(n, 0, 1),
    Rcpp::Named("beta") = Rcpp::rbeta(n, 1, 1),
    Rcpp::IntegerVector::create(1,2,3,4,5, NA_INTEGER)
  );
}
```

```
make_list(10)
```

```
## $norm
##  [1] -1.4571593 -0.5890910 -0.3553319 -1.8817167 -1.0387151 -0.9843522
##  [7]  1.7666458 -1.6976358  0.9262356  0.4120744
##
## $beta
##  [1] 0.5718434 0.3707151 0.4487789 0.7104076 0.5235477 0.6740764 0.9356531
##  [8] 0.2474596 0.8789587 0.8653678
##
## [[3]]
## [1]  1  2  3  4  5 NA
```

Creating a `data.frame`

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::DataFrame make_df(int n) {
  return Rcpp::DataFrame::create(
    Rcpp::Named("norm") = Rcpp::rnorm(n, 0, 1),
    Rcpp::Named("beta") = Rcpp::rbeta(n, 1, 1)
  );
}
```

```
make_df(10)
```

```
##          norm       beta
## 1   1.3253697 0.50683178
## 2   0.6427004 0.01043097
## 3   0.9097914 0.76920733
## 4   0.6597730 0.05330513
## 5  -0.1640834 0.56876709
## 6  -0.7922707 0.26508582
## 7  -0.6660425 0.34764589
## 8  -0.5000887 0.43290857
## 9  -0.5694419 0.78583149
## 10  1.4558494 0.66066330
```

Creating a tbl

```
#include <Rcpp.h>

// [[Rcpp::export]]
Rcpp::DataFrame make_tbl(int n) {
  Rcpp::DataFrame df = Rcpp::DataFrame::create(
    Rcpp::Named("norm") = Rcpp::rnorm(n, 0, 1),
    Rcpp::Named("beta") = Rcpp::rbeta(n, 1, 1)
  );
  df.attr("class") = Rcpp::CharacterVector::create("tbl_df", "tbl", "data.frame");

  return df;
}
```

```
make_tbl(10)
```

```
## # A tibble: 10 x 2
##      norm  beta
##     <dbl> <dbl>
##  1  0.613 0.551
##  2 -1.04  0.792
##  3 -0.385 0.380
##  4 -0.500 0.462
##  5  1.61  0.473
##  6  0.782 0.154
##  7  0.964 0.697
##  8 -1.23  0.450
##  9  1.61  0.986
## 10  0.299 0.179
```

Printing

R has some weird behavior when it comes to printing text from C++, Rcpp has function that resolves this, `Rcout`

```
#include <Rcpp.h>

// [[Rcpp::export]]
void n_hello(int n) {
  for(int i=0; i!=n; ++i) {
    Rcpp::Rcout << i+1 << ". Hello world!\n";
  }
}
```

```
n_hello(5)
```

```
## 1. Hello world!
## 2. Hello world!
## 3. Hello world!
## 4. Hello world!
## 5. Hello world!
```

Printing NAs

```
#include <Rcpp.h>

// [[Rcpp::export]]
void print_na() {
  Rcpp::Rcout << "NA_INTEGER : " << NA_INTEGER << "\n";
  Rcpp::Rcout << "NA_STRING  : " << NA_STRING  << "\n";
  Rcpp::Rcout << "NA_LOGICAL : " << NA_LOGICAL << "\n";
  Rcpp::Rcout << "NA_REAL    : " << NA_REAL    << "\n";
}
```

```
print_na()
```

```
## NA_INTEGER : -2147483648
## NA_STRING  : 0x7fa860003e00
## NA_LOGICAL : -2147483648
## NA_REAL    : nan
```

SEXP Conversion

Rcpp attributes provides a bunch of convenience tools that handle much of the conversion from R SEXP's to C++ / Rcpp types and back. Some times it is necessary to handle this directly.

```
#include <Rcpp.h>

// [[Rcpp::export]]
SEXP as_wrap(SEXP input) {
  Rcpp::NumericVector r = Rcpp::as<Rcpp::NumericVector>(input);
  Rcpp::NumericVector rev_r = Rcpp::rev(r);

  return Rcpp::wrap(rev_r);
}
```

```
as_wrap(1:10)
```

```
##  [1] 10  9  8  7  6  5  4  3  2  1
```