



Entuzjastów R
Krakowska Alternatywa

Nie taki C++ straszny jak go
połączysz z R

Zygmunt Zawadzki

Szybki plan

- R – dlaczego potrzebujemy wsparcia C++?
- C++ - (subiektywne) fakty i mity.
- Rcpp – jak łatwo i bezpiecznie połączyć R z C++?
- FSelectorRcpp

Dlaczego potrzebujemy wsparcia C++?

	Fortran	Julia	Python	R	Matlab	Octave
	gcc 5.1.1	0.4.0	3.4.3	3.2.2	R2015b	4.2.0
fib	0.70	2.11	77.76	533.52	26.89	9324
parse_int	5.05	1.45	17.02	45.73	802.52	9581
quicksort	1.31	1.15	32.89	264.54	4.92	1866
mandel	0.81	0.79	15.32	53.16	7.58	451
pi_sum	1.00	1.00	21.99	9.56	1.00	299
rand_mat_stat	1.45	1.66	17.93	14.56	14.52	9
rand_mat_mul	3.48	1.02	1.14	1.57	1.12	1

R

3.2.2

533.52
45.73
264.54
53.16
9.56
14.56
1.57

Script
8
71.19
3.36
6.06
2.70
0.66
1.01
2.30
15.07

Go
go1.5
1.86
1.20
1.29
1.11
1.00
2.96
1.42

LuaJIT
gsl-shell
2.3.1
1.71
5.77
2.03
0.67
1.00
3.27
1.16

Java
1.8.0_45
1.21
3.35
2.60
1.35
1.00
3.92
2.36

R bywa wolny...

Ale to nie jedyny powód...

W innych językach **też** powstają ciekawe rzeczy.

Dlaczego mielibyśmy z nich nie skorzystać?

A C++ może służyć za most.

*Jeden, by wszystkimi rządzić, Jeden, by wszystkie odnaleźć,
Jeden, by wszystkie zgromadzić i w ciemności związać...*

z powieści [Władca Pierścieni](#) autorstwa [J.R.R. Tolkiena](#).



I wiele innych!

C++ ma dostęp do bogatego zbioru struktur danych:

Wektory, zbiory, mapy, stosy...

Poza tym C++ ma



a w nim jeszcze więcej wszystkiego...



“...one of the most highly regarded and expertly designed C++ library projects in the world.”
— Herb Sutter and Andrei Alexandrescu, C++ Coding Standards

C++ - (subiektywne) fakty i
mity

C++ jest za blisko surowej pamięci dla zwykłego użytkownika. Wycieki pamięci to codzienność...

... o ile zejdziesz z prawej ścieżki STL.

Alokacja n-elementowego wektora w C++:

```
double *vec = new double[n];
```

NIE!

NIE!

Użycie **new** może być bardzo niebezpieczne.

Zarządzanie pamięcią staje się skomplikowane, a brak wywołania **delete** prowadzi będzie do wycieku pamięci.

Na początek najbezpieczniej omijać **wskaźniki**, **new** i **delete**.

Wbrew pozorom można bardzo dużo napisać w C++ bez nich!!!



Samouczki często wprowadzają wskaźniki wcześniej niż konieczne. Dlatego początkujący może odnieść wrażenie* że trzeba pisać kod z ich wykorzystaniem.

* - tzn. ja kiedyś miałem takie wrażenie?

Prawa droga STL:

```
std::vector<double> vec(n);
```

- Automatyczne zarządzanie pamięcią (trudno spowodować wyciek pamięci).
- (Dosyć)* Efektywne implementacje.
- Bogata baza dostępnych algorytmów.

* - będąc początkującym adeptem C++ i tak pewnie nie napisze się nic szybszego i jednocześnie tak wygodnego...

STL

Strony pomocne w nauce C++:

- <http://www.cplusplus.com/>
- <http://en.cppreference.com/w/>

http://www.cplusplus.com/reference/vector/vector/push_back/

Example

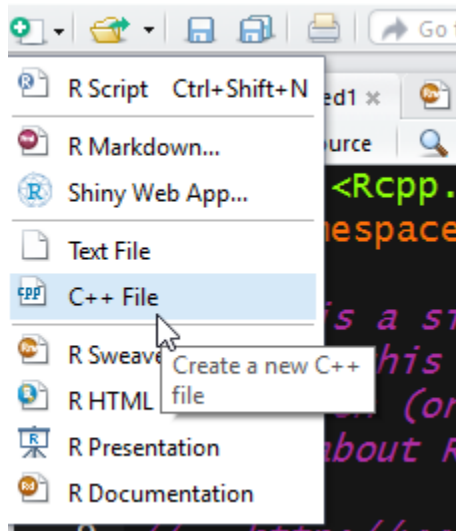
```
1 // vector::push_back
2 #include <iostream>
3 #include <vector>
4
5 int main ()
6 {
7     std::vector<int> myvector;
8     int myint;
9
10    std::cout << "Please enter some integers (enter 0 to end):\n";
11
12    do {
13        std::cin >> myint;
14        myvector.push_back (myint);
15    } while (myint);
16
17    std::cout << "myvector stores " << int(myvector.size()) << " numbers.\n";
18
19    return 0;
20 }
```

 Edit & Run

Możliwość własnoręcznego
przetestowania kodu w
przeglądarce

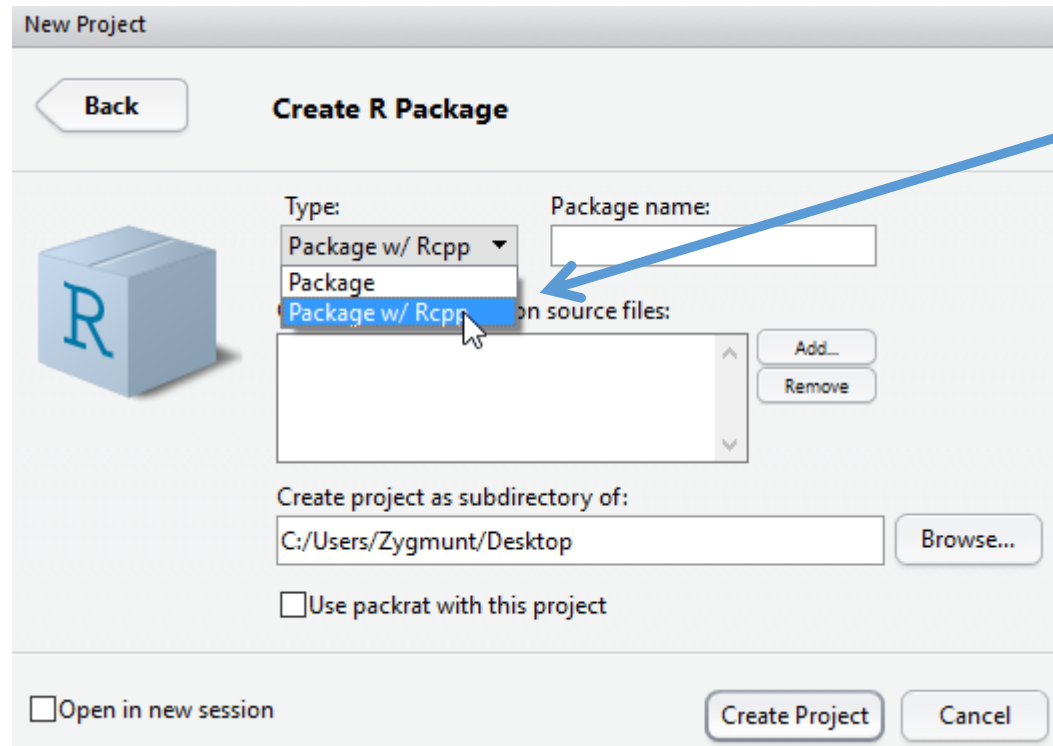
Rcpp – jak łatwo i bezpiecznie
połączyć R z C++?

Wszystkie funkcje zawierające taki wpis nad swoją definicją zostaną wyeksportowane do R.



```
14 // [[Rcpp::export]]  
15 NumericVector timesTwo(NumericVector x) {  
16     return x * 2;  
17 }
```

Pakiet zawierający kod C++:



The screenshot shows the 'New Project' dialog box in R Studio, specifically the 'Create R Package' step. The 'Type' dropdown menu is open, and 'Package w/ Rcpp' is selected. A blue arrow points from the text on the right to this selection. The 'Package name' field is empty. The 'Create project as subdirectory of:' field contains 'C:/Users/Zygmunt/Desktop'. The 'Use packrat with this project' checkbox is unchecked. The 'Open in new session' checkbox is also unchecked. The 'Create Project' and 'Cancel' buttons are at the bottom right.

Na etapie tworzenia pakietu wybieramy **Package w/ Rcpp** i gotowe.

Struktury z R w Rcpp

numeric	↔	NumericVector
integer	↔	IntegerVector
character	↔	CharacterVector
matrix	↔	NumericMatrix
list	↔	List

```
14 // [[Rcpp::export]]  
15 NumericVector timesTwo(NumericVector x) {  
16     return x * 2;  
17 }
```

WARNING!



UWAGA!

Rcpp domyślnie przekazuje obiekty przez referencję (nie jest wykonywana żadna kopia). Dlatego zmiany w obiektach po stronie C++ spowodują zmiany w obiektach w R. Może to prowadzić do strasznych konsekwencji!

Funkcja nic nie zwraca.

```
28 // [Rcpp::export]  
29 void mod(NumericVector x, double b)  
30 {  
31     x[0] = b;  
32 }
```

Obiekt
zmodyfikowany w
c++ pozostaje
zmodyfikowany w R!

Powinno być...

```
> x = c(5.0, 3.0)  
> mod(x, 2000)  
> x  
[1] 2000 3
```

... a jest...

```
> x = c(5.0, 3.0)
> y = x
> mod(x, 2000)
> x
[1] 2000      3
> y
[1] 2000      3
```

R nie zawsze kopiuje obiekty...

... co w tym przypadku prowadzi do tragedii...

pryr pozwala lepiej poznać w jaki sposób R zarządza pamięcią.

```
> library(pryr)
> address(x)
[1] "0x161b5cc0"
> address(y)
[1] "0x161b5cc0"
```

```
// [[Rcpp::export]]
NumericVector mod_z(NumericVector x, double b)
{
  NumericVector z = x;
  z[0] = b;
  return z;
}
```

Domyślnie przy przypisaniu
obiekty nie są kopiowane!

Z stworzony w c++ ma ten
sam adres co **X**.

```
> x = c(5.0, 3.0)
> z = mod_z(x, 2000)
> x
[1] 2000      3
> z
[1] 2000      3
> address(x)
[1] "0x2466c0d8"
> address(z)
[1] "0x2466c0d8"
```

```
// [[Rcpp::export]]  
NumericVector mod_clone_z(NumericVector x, double b)  
{  
  NumericVector z = Rcpp::clone(x);  
  z[0] = b;  
  return z;  
}
```

Kopiowanie obiektu przy
pomocy **Rcpp::clone**

```
> x = c(5.0, 3.0)  
> z = mod_clone_z(x, 2000)
```

```
> x  
[1] 5 3  
> z  
[1] 2000
```


x pozostał
taki sam!


Stała referencja

Pisząc w C++ dobrym przyzwyczajeniem jest przesyłanie każdego argumentu przez stałą referencję.

- Referencja sprawia, iż obiekty nie są niepotrzebnie kopiowane.
- `const` zabezpiecza przed przypadkową modyfikacją obiektu.

```
56 // [[Rcpp::export]]
57 void mod_const(const NumericVector& x, double b)
58 {
59     x[0] = b;
60 }
61
```

 **const** nas zabezpiecza przed przypadkową modyfikacją obiektu.

 Nie skompiluje się!


```
// [[Rcpp::export]]  
void mod_const_copy(const NumericVector& x, double b)  
{  
  NumericVector z = x;  
  z[0] = b;  
}
```

Stała referencja – x nie wolno modyfikować!

Kompiluje się!!!

Zmodyfikowaliśmy stałą
wartość!



```
> x = c(5.0, 3.0)  
> mod_const_copy(x, 2000)  
> x  
[1] 2000      3
```

Automatyczne konwersje w Rcpp

```
> x = 1:2  
> mod(x, 2000)  
> x  
[1] 1 2
```



W pewnych przypadkach
modyfikacja po stronie C++ nie
zadziała!

```
// [[Rcpp::export]]  
void mod(NumericVector x, double b)  
{  
  x[0] = b;  
}
```



Oczekiwany NumericVector

x jest IntegerVector w R




```
> class(x)  
[1] "integer"
```

Jeżeli oczekiwany typ po stronie C++ nie zgadza się z przesyłanym
typem Rcpp spróbuje konwersji. Obiekt jest wtedy kopiowany.

By zawsze skopiować obiekt użyj STL'a.

```
// [[Rcpp::export]]  
void mod_stl(std::vector<double> x, double b)  
{  
  x[0] = b;  
}
```



W Rcpp możemy w argumentach użyć wektora z STL!

x nie został
zmodyfikowany!

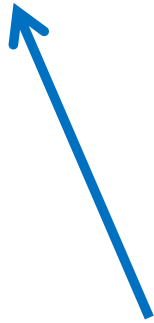


```
> x = c(5.0, 3.0)  
> mod_stl(x, 2000)  
> x  
[1] 5 3
```

FselectorRcpp w końcu!

Krótki opis motywacji:

<https://github.com/mi2-warsaw/FSelectorRcpp/issues/7>



Koniecznie trzeba przeczytać!

Jeszcze jeden problem który napotkaliśmy tworząc FselectorRcpp.

R jest dynamicznie typowany.

```
fnc = function( x ) { ... }
```

Nie musimy deklarować
czym jest x.



C++ jest statycznie typowany.

```
double fnc(double x)
```

Musimy określić czym jest
x i co zwróci funkcja.



A co jeśli nasza funkcja napisana z pomocą Rcpp ma
obsługiwać character, numeric, integer?

Jedno z możliwych rozwiązań:

<http://stackoverflow.com/questions/19823915/how-can-i-handle-vectors-without-knowing-the-type-in-rcpp>

My zrobiliśmy inaczej:)

Zależało nam byśmy mogli wykorzystać łatwo STL-a i w tylko minimalnym stopniu uzależniać się od struktur Rcpp.

Najlepiej by było, żeby kod C++ móc wykorzystać w ogóle bez zależności Rcpp.

Skorzystaliśmy z **templatów**!

Templaty pozwalając zmniejszyć niedogodności związane ze statycznym typowaniem.

T w trakcie kompilacji zostanie zamieniony na odpowiedni typ.

```
template <class T> const T& max (const T& a, const T& b) {  
    return (a<b)?b:a;    // or: return comp(a,b)?b:a; for version (2)  
}
```

Źródło: <http://www.cplusplus.com/reference/algorithm/max/>

```
// max example  
#include <iostream>    // std::cout  
#include <algorithm>    // std::max  
  
int main () {  
    std::cout << "max(1,2)==" << std::max(1,2) << '\n';  
    std::cout << "max(2,1)==" << std::max(2,1) << '\n';  
    std::cout << "max('a','z')==" << std::max('a','z') << '\n';  
    std::cout << "max(3.14,2.73)==" << std::max(3.14,2.73) << '\n';  
    return 0;  
}
```

Dzięki templatom
nie trzeba
definiować **max** dla
każdego typu.

Iteratory.

W STL rzadko kiedy przesyła się całe obiekty do funkcji. Dużo częściej korzysta się z iteratorów.

W R by posortować wektor wykorzystamy:

```
sort(x)
```

W c++ (korzystając z STL) zrobimy:

```
std::sort(x.begin(), x.end());
```

Podejście STLowe pozwala na trochę więcej swobody – np. by posortować tylko pierwszą połowę wektora zrobimy:

```
std::sort(x.begin(), x.begin() + x.size() / 2);
```


Template oparty na iteratorze



Więcej na temat iteratorów:

<http://en.cppreference.com/w/cpp/concept/Iterator>

```
13
14 template<class InputIterator>
15 size_t count_levels(InputIterator first, InputIterator last)
16 {
17     std::set<typename std::iterator_traits<InputIterator>::value_type> set(first, last);
18     return set.size();
19 }
20
```



Dzięki temu kawałkowi dowiadujemy się na jaki typ wskazuje iterator.

Jeżeli stworzymy wektor intów, a później wywołamy na nim naszą funkcję to C++ będzie wiedział, że w tym przypadku powinna wyglądać tak:

```
std::vector<int> x;
count_levels(x.begin(), x.end());
```

```
size_t count_levels(InputIterator first, InputIterator last)
{
    std::set<int> set(first, last);
    return set.size();
}
```



```
8 // [[Rcpp::export]]
9 double fs_entropy1d(SEXP x)
10 {
```

Przekazujemy goły obiekt Rowy (bez określania czym ma być).

- REALSXP: numeric vector
- INTSXP: integer vector
- LGLSXP: logical vector
- STRSXP: character vector
- VECSXP: list
- CLOSXP: function (closure)
- ENVSXP: environment

Źródło: <http://adv-r.had.co.nz/C-interface.html>

```
switch(TYPEOF(x))
{
  case REALSXP:
  {
    ... // kod dla numeric
    break;
  }

  case STRSXP:
  {
    ... // kod dla character
    break;
  }
}
```

TYPEOF i switch
załatwia resztę.

Wskazówka w nauce C++:

Czas przeznaczony na naukę
wskaźników lepiej poświęcić
na naukę **iteratorów**!

A czas przeznaczony na naukę mowy lepiej poświęcić na naukę walki!

Dziękuję za uwagę!

Dołącz do nas na:

<https://github.com/mi2-warsaw/FSelectorRcpp>

> eRka()

Entuzjastów R
Krakowska Alternatywa

STL

...tak na odchodne:)