

# On the Way to Perl 6++

or

# How to speed up your Perl 6 app

How to speed up  
your Perl 6 app  
and make it  
10 000 times faster

# Preface

YAPC::Europe 2009

# Translating human language with computer grammar

# YAPC::Europe 2009

```
sentence : statement eofile
statement : subject verb object fullstop
subject : adjective noun
          | article noun
          | noun
object : noun
noun : 'Name'
      | 'Andreas'
adjective : 'Mein'
           | 'Sein'
verb : 'ist'
article : 'Die'
         | 'Der'
         | 'Das'
fullstop : '.'
eofile : /^Z/
```

German Perl Workshop 2008

# Syntaxanalyse von URL mit die Grammatik

# German Perl Workshop 2008

```
home      : S
division  : S 'worldservice' S
           'languages' S(?)
           | S word S(?)
section   : division word S(?)
article   : section word ext
```

# Personal Perl 6 compiler

Let's take  
simple  
programme

Let's take  
simple  
CPU consuming  
programme

```
say is_prime(@*ARGS[0]);
```

```
sub is_prime($n) {  
  for 2 .. $n - 1 -> $d {  
    return 0 unless $n % $d;  
  }  
  
  return 1;  
}
```

\$ perl6

```
$ perl6 is_prime.p6
```

```
$ perl6 is_prime.p6 37
```

```
$ perl6 is_prime.p6 37
```

```
1
```

```
$ perl6 is_prime.p6 38
```

```
0
```

```
$ perl6 is_prime.p6 15511
```

```
$ perl6 is_prime.p6 15511  
1
```

Runs 20 seconds under Rakudo :-(

```
$ perl6 is_prime.p6 15511  
1
```

Runs 20 seconds under Rakudo\* :-)

\* previous version :-)

Perl 6  
can be fast!

**Perl 6  
can be fast!**

Perl 6  
can be fast!

**Boost it!**

OK, let's visit [boost.org](https://boost.org) :-)

```
struct p6grammar: public grammar<p6grammar> {  
    template<typename ScannerT>  
    struct definition {  
        definition(p6grammar const& self) {
```

```
struct p6grammar: public grammar<p6grammar> {
    template<typename ScannerT>
    struct definition {
        definition(p6grammar const& self) {
            program
                = statement_list >> end_p;
        }
    };
};
```

```
struct p6grammar: public grammar<p6grammar> {
    template<typename ScannerT>
    struct definition {
        definition(p6grammar const& self) {
            program
                = statement_list >> end_p;
            statement_list
                = *statement;
```

```
struct p6grammar: public grammar<p6grammar> {
    template<typename ScannerT>
    struct definition {
        definition(p6grammar const& self) {
            program
                = statement_list >> end_p;
            statement_list
                = *statement;
            statement
                = statement_code
                >> !statement_separator;
```

```
statement_code
    = comment
    | statement_control
    | sub_def
    | sub_call
    | expression;
```

```
statement_code
    = comment
    | statement_control
    | sub_def
    | sub_call
    | expression;
```

```
comment
```

```
    = ch_p( '#' )
      >> lexeme_d[ *~ch_p( '\n' ) ];
```

```
statement_code
    = comment
    | statement_control
    | sub_def
    | sub_call
    | expression;
```

```
comment
```

```
    = ch_p( '#' )
```

```
        >> lexeme_d[ *~ch_p( '\n' ) ];
```

```
sub_call
```

```
    = sub_name
```

```
        >> !sub_arguments;
```

```

struct p6grammar: public grammar<p6grammar> {
    template<typename ScannerT> struct
    definition {
        definition(p6grammar const& self) {
            program
                = statement_list >> end_p;

            statement_list
                = *statement;

            statement
                = statement_code >> !
statement_separator[&a_statement];

            statement_code
                = comment
                | statement_control
                | sub_def
                //| (sub_call[&a_sub_call_start]
>> !statement_modifier[&a_sub_call_end]
                | sub_call
                | expression;

            comment
                = ch_p('#') >>
lexeme_d[*~ch_p('\n')];

            statement_separator
                = ch_p(';');

            sub_call
                = (sub_name >> !sub_arguments)
[&a_sub_called];

            sub_name
                = builtin_sub_name
                | identifier[&a_sub_name]
[&a_sub_calling];

            builtin_sub_name
                = str_p("say")[&a_builtin_say]
                | str_p("print")
[&a_builtin_print]
                | str_p("return")
[&a_builtin_return];

            identifier
                = lexeme_d[(alpha_p | '_' ) >>
*(alnum_p | '_')];

            sub_arguments
                = ch_p('(') >> !argument_list >>
ch_p(')')
                | argument_list;

            argument_list
                = list_p(argument, ch_p(', '))
[&a_argument_list_sep]);

```

```

        argument
            = expression
            | variable;

        variable
            = array
            | scalar;

        scalar
            = constant
            | ch_p('$') >>
identifier[&a_scalar]
            | array_element;

        array
            = simple_array
            | global_array;

        simple_array
            = ch_p('@') >> (identifier >>
~eps_p('[')][&a_array];

        global_array
            = str_p("@*") >> (identifier >>
~eps_p('[')][&a_global_array];

        array_element
            = simple_array_element
            | global_array_element;

        simple_array_element
            = (ch_p('@') >>
array_element_name_and_index)[&a_array_element];

        global_array_element
            = (str_p("@*") >>
array_element_name_and_index)
[&a_global_array_element];

        array_element_name_and_index
            =
identifier[assign_a(array_element_name)]
                >> ch_p('[') >> array_index
                >> ch_p(')');

        array_index
            =
int_p[assign_a(array_element_index)];

        constant
            = real_p[&a_constant];

        sub_def
            = str_p("sub")[&a_sub_def_start]
>> sub_name
                >> !
sub_arguments[&a_sub_def_arguments_end]
                >> sub_body[&a_sub_def_end];

```

```

            sub_body
                = ch_p('{')
[&a_sub_def_body_start] >> !statement_list >>
ch_p(')');

            statement_control
                = for_cycle
                | conditional_block;

            conditional_block
                = (str_p("if") |
str_p("unless"))[&a_condition_start] >>
expression[&a_condition_end] >>
code_block[&a_conditional_block_end];

            code_block
                = ch_p('{') >> statement_list >>
ch_p(')');

            for_cycle
                = str_p("for")
[&a_for_cycle_start]
                >> (range >> !(str_p("->") >>
ch_p('$') >> identifier[&a_for_cycle_variable]))
[&a_for_cycle_range]
                >>
for_cycle_body[&a_for_cycle_body];

            range
                = ch_p('(') >> range_content >>
ch_p(')')
                | range_content;

            range_content
                = expression >> str_p("..")
[&a_range_sep] >> expression;

            expression
                = ch_p('(')[&a_paren] >>
expression_content >> ch_p(')')[&a_paren]
                | expression_content;

            expression_content
                = scalar >>
*(math_op[&a_math_op] >> expression)
                | sub_call;

            math_op
                = ch_p('+')
                | ch_p('-') >> eps_p(~ch_p('>'))
                | ch_p('*')
                | ch_p('/')
                | ch_p('%');

            for_cycle_body
                = sub_body;

            statement_modifier
                = (str_p("unless") |
str_p("if")) >> expression;

```

```
say is_prime(@*ARGS[0]);

sub is_prime($n) {
    for 2 .. $n - 1 -> $d {
        return 0 unless $n % $d;
    }

    return 1;
}
```

```
say is_prime(@*ARGS[0]);

sub is_prime($n) {
  for 2 .. $n - 1 -> $d {
    return 0 unless $n % $d;
  }
  return 1;
}
```

```
say is_prime(@*ARGS[0]);

sub is_prime($n) {
    for 2 .. $n - 1 -> $d {
        unless $n % $d {
            return 0;
        }
    }

    return 1;
}
```

# Programme 1

Says if the argument is a prime

```
say is_prime(@*ARGS[0]);

sub is_prime($n) {
    for 2 .. $n - 1 -> $d {
        unless $n % $d {
            return 0;
        }
    }

    return 1;
}
```

```
./p++ < t/is_prime.p6
```

```
#include<iostream>
#include<string>

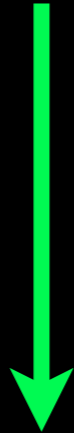
using namespace std;

typedef int variable_t;
typedef int scalar_t;
typedef int range_boundary_t;

variable_t sub_is_prime(scalar_t var_scalar_n) {
    for (range_boundary_t var_scalar_d = 2; var_scalar_d <=
var_scalar_n - 1; var_scalar_d++) {
        if(!(var_scalar_n % var_scalar_d)) {
            return 0;
        }
    }
    return 1;
}

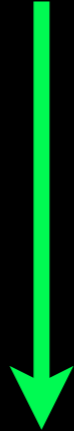
int main(int argn, char** var_global_array_ARGS) {
    cout << sub_is_prime(atoi(var_global_array_ARGS[1 + 0])) << "\n";
    return 0;
}
```

```
say is_prime(@*ARGS[0]);
```



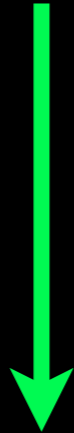
```
cout << sub_is_prime(  
    atoi(var_global_array_ARGS[1 + 0])  
)  
<< "\n";
```

```
sub is_prime($n) {
```



```
variable_t sub_is_prime(scalar_t var_scalar_n)  
{
```

```
for 2 .. $n - 1 -> $d {
```



```
for (range_boundary_t var_scalar_d = 2;  
     var_scalar_d <= var_scalar_n - 1;  
     var_scalar_d++) {
```

```
$ time ./is_prime 15511  
1
```

```
$ time ./is_prime 15511
```

```
1
```

```
real 0m0.002s
```

# Programme 2

Checks if the number is prime  
and prints its first diviser

```
say is_prime(@*ARGS[0]);

sub is_prime($n) {
    for 2 .. $n - 1 -> $d {
        unless $n % $d
            say $d;
            return 0;
        }
    }

    return 1;
}
```

# Programme 3

Finds prime numbers below 1 000 000

```
for 2 .. 1000000 -> $n {  
  if is_prime($n) {  
    say $n;  
  }  
}
```

```
sub is_prime($n) {  
  for 2 .. $n - 1 -> $d {  
    unless $n % $d {  
      return 0;  
    }  
  }  
  return 1;  
}
```

Random  
background thoughts

# 1. C++'s type casts

```
my $var = 100;  
say $var;
```

```
$var = "string";  
say $var;
```

```
my $var = 100;  
say $var;
```

```
$var = "string";  
say $var;
```

```
variable_var.set(100);  
cout << variable_var;
```

```
variable_var.set("string");  
cout << variable_var;
```

```
class PerlVariable {  
    . . .  
public:  
    operator int();  
    operator string();  
}
```

2. Leave difficulties  
to C++ compiler

RIGA CITY:  
EASY TO GO,  
HARD TO LIVE!



```
class MyClass {  
    has $member;  
    method meth() {...}  
}
```

```
class MyClass {  
    has $member;  
    method meth() {...}  
}
```

```
class MyClass {  
public:  
    PerlVariable member_member;  
    void meth_method();  
}
```

```
class MyClass {  
    has $.member;  
    method meth() {...}  
}
```

```
class MyClass {  
    PerlVariable member_member;  
public:  
    void meth_method();  
}
```

# 3. Where to start (prime principle)

a) Write a Perl 6 programme  
that I'd like to run in  
production

a) Write a Perl 6 programme  
that I'd like to run in  
production

b) Implement the compiler  
for that Perl 6 code

a) Write a Perl 6 programme that I'd like to run in production

b) Implement the compiler for that Perl 6 code (following STD.pm)

```
say is_prime(@*ARGS[0]);
```

```
sub is_prime($n) {  
  for 2 .. $n - 1 -> $d {  
    return 0 unless $n % $d;  
  }  
  return 1;  
}
```

```
say is_prime(@*ARGS[0]);
```

```
sub is_prime($n) {  
  for 2 .. $n - 1 -> $d {  
    unless $n % $d {  
      return 0;  
    }  
  }  
  
  return 1;  
}
```

Test suite

**t/expr.p6**

1 + 2;

\$x + 3;

\$x \* 10;

\$x - 5;

7 / 8;

10 % 20;

\$x + \$y;

10 % \$d;

1 + 2 + 3;

\$x \* \$y / \$z;

(1 + 2);

\$x + (\$y - \$z);

5.6 + (\$x % 5 / (4 - \$z));

t/say.p6

```
# One-line comment
say(1);
say 3.14;
say 1, 2;
say(3, 14);
say($x);
say($x, 3.14);
say(@arr);
say(@arr1_2);
say(@arr[23]);
say(@arr[2], 3.14, $x, @y);
say(@*ARGV);
say(@*ARGV[4]);
```

```
say(2, 3.4, $x, @y, @*G, @y[5], @*G[2]);  
say @z[4];  
say(is_prime);  
say(is_prime(37));  
say is_prime;  
say is_prime();  
say is_prime(7);  
say is_prime($x);  
say is_prime(7, $x);  
say is_prime(@*ARGS[0]);  
say 42;  
print $x;  
say(1 + 2);
```

t/sub.p6

```
sub x() {  
    print 1;  
}  
sub y($x) {  
    say $x;  
}
```

```
x(1);
```

```
x();
```

```
x(1, 2);
```

t/for.p6

```
for 1 .. 10 {  
}
```

```
for 1 .. 5 -> $x {  
}
```

```
for 1 .. 10 {  
    say $_;  
}
```

```
for (3 .. 20) {  
    say $_;  
}
```

```
for $a .. $b {  
    for $x .. $y {  
        say $x;  
    }  
}
```

<svn://svn.shitov.ru/p6c>

<http://perl6.ru/p6c>

<svn://svn.shitov.ru/p6c>

<http://perl6.ru/p6c>

END

Andrew Shitov | [mail@andy.sh](mailto:mail@andy.sh)