On top of the Elephpant

Using PHP as a compiler

Structure of this talk

1.	**Motivation**	*Why would you build a programming language?*
2.	**History**	*What is Syntax and how has it evolved?*
3.	**Design**	*What is operator precedence?*
4.	**Parsing**	*How to process a file into a syntax tree*
5.	**Implementation**	*What is top down operator precedence?*
6.	**Show and tell**	*Putting what we know in practice*
7.	**Compiling**	*How to convert the AST into instructions*
8.	**Running**	*How to run the created instructions*
9.	**Optimization**	*How to reduce the amount of required code to run*
10.	**Conclusion**	*Did it work?*

Motivation

Why build a custom language?

To implement domain specific language (DSL)

```
@products
    .ensure('tnt')
    .branch('media_gallery_entries.*')
    .filter(
        $image => /portrait\.(jpg|png)$/.test($image.file);
    );
```

Symbiont

Why build a custom language?

Describe domain logic in language constructs

```
<?php

// Get the TNT product
/** @var Acme\CollectionManagerInterface $manager */
$manager = require __DIR__ . ‘/collection-manager.php’;
$catalog = $manager->get(‘products’);
$product = $catalog->get(‘tnt’);

$gallery = $product->get(‘media_gallery_entries’);
$changed = false;

// Treat each media gallery entry individually
foreach ($gallery as $idx => $entry) {
    // Only proceed when `media_type` is set to `image`
    if ($entry->get(‘media_type’) !== ‘image’) {
        continue;
    }

    // Remove all images that do not end in portrait and
    // do not have either .png or .jpg as their extension.
    if (preg_match(‘/portrait\.(png|jpg)$/’, $entry->get(‘file’)) < 1) {
        unset($gallery[$idx]);
        $changed = true;
    }
}
```

```
@products
    # Get the TNT product
    .get(‘tnt’)

    # Treat each media gallery entry individually
    .forEach($_.media_gallery_entries)

    # Only proceed when `media_type` is set to `image`
    .where($_.media_type = ‘image’)

    # Remove all images that do not end in portrait and
    # do not have either .png or .jpg as their extension.
    .filter($_.file matches pcre2:/portrait\.(png|jpg)$/);
```

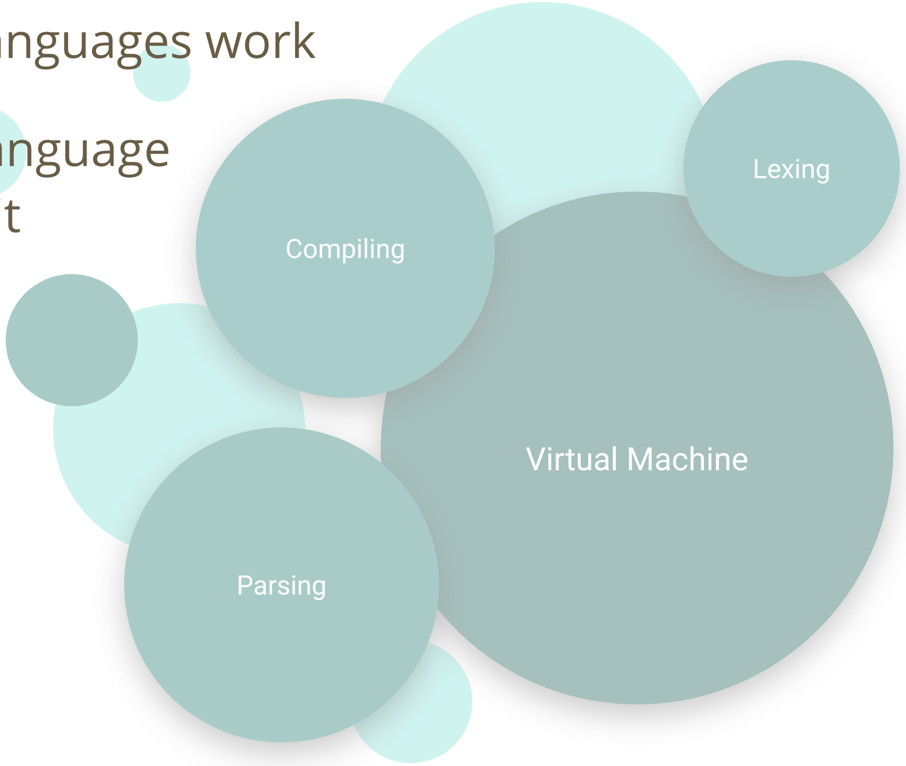
```
// Explicitly store the product when the media gallery entries are updated
if ($changed) {
    $product->set(‘media_gallery_entries’, $gallery);
    $catalog->update($product);
}
```

```
# The language automatically detects and persists changes to @products
```

Why build a custom language?

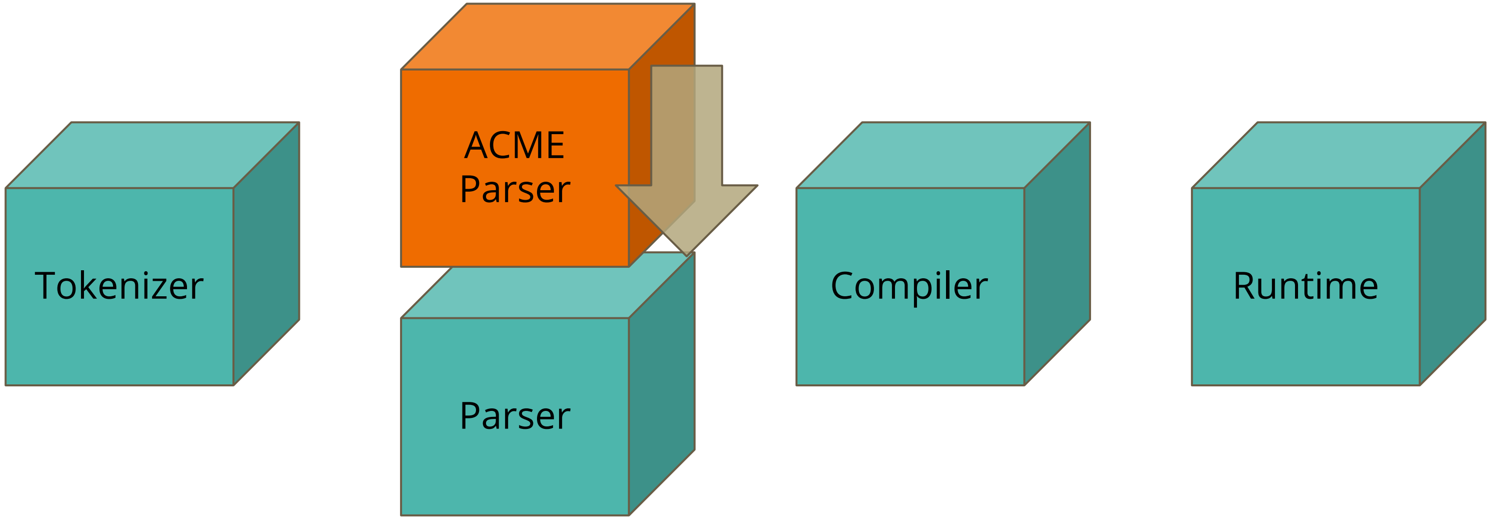
To learn more about how existing languages work

- What components make up a language
- How can I use that to my benefit



Why build a custom language?

- Let the language specification be runtime configuration
- Allow any part of the language to be swapped out

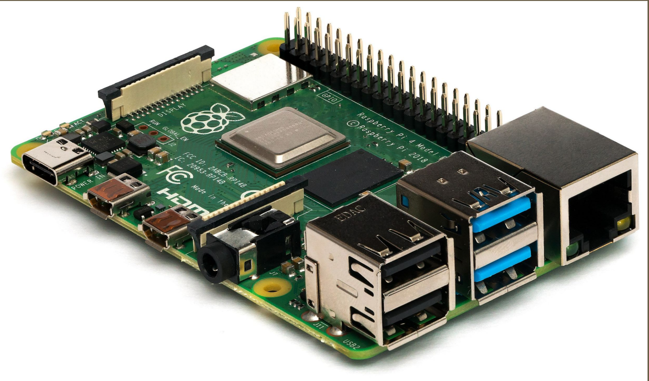


Why build a custom language?

To challenge oneself

- Because it is loads of fun and particularly nerdy
- It encourages to learn about languages (you would otherwise not use)
- Improved understanding of the relationship between programming languages and hardware
  - Personal computers
  - Emulators
  - FPGA (Field programmable gate arrays)
  - PLC (Programmable logic controllers)
  - Embedded devices

Raspberry Pi 4 Model B



By Miiicihiaieil Hieinizilieir / Wikimedia Commons, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=80140656

History

Syntax

```
function () {
}
```

vs

```
func begin
end
```

By Robert Claypool - Own work, CC0, https://commons.wikimedia.org/w/index.php?curid=24517464

"Syntax is the least important part of programming language design."

"Fashion is the least important part of clothing design."

- Douglas Crockford

Evolution of if-statement: Fortran

```
C     FORTRAN
      IF(A-B)20,20,10
   10 A=B
   20 CONTINUE
```

Evolution of if-statement: Fortran IV

```
C     FORTRAN IV

      IF(A.LE.B)GO TO 30

      A=B

   30 CONTINUE
```

```
C     FORTRAN IV

      IF(A.LE.B)GOTO 30

      A=B

   30 CONTINUE
```

Evolution of if-statement: ALGOL 60

```
comment ALGOL 60;


if a>b then begin
    a:=b
end;
```

Evolution of if-statement: BCPL

```
// BCPL



IF A > B {

    A := B

}
```

Evolution of if-statement: B

```
/* B */


if (a > b) {
    a = b;
}
```

Evolution of if-statement: Ada

```
-- Ada


if a > b then
    a := b;
end if;
```

Evolution of if-statement: Algol 68

```
¢ Algol 68 ¢


if a > b then
    a := b
fi
```

Evolution of if-statement: PHP 7

```
<?php

// PHP 7


if ($a > $b) {

    $a = $b;

}
```

Evolution of if-statement: Symbiont

```
# Symbiont


if $a > $b {
    $a : $b;
};
```

Design

Operator precedence

a + b * c

a	2
b	3
c	4

= ?

Operator precedence: PHP 7

$a + $b * $c

$a	2
$b	3
$c	4

= 14

Associativity	Operators
Right	**
Left	* / %
Left	+ - .

Overloading of words

Variable	`a, b, c`
Statement keyword	`function, while, for`
Operator	`$foo & $bar, $foo & &$baz, $foo && $baz`

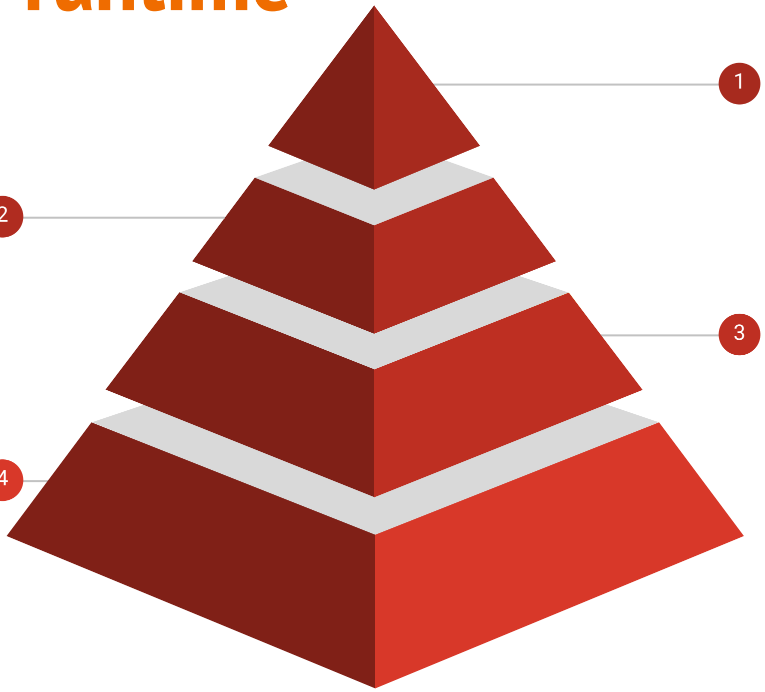
Most languages maintain a list of reserved keywords, to prevent overloading of both existing words, and words that may or may not be used in the future.

Operator overloading: PHP 7 print

```php
<?php


print("10") && print("20");

print "30" && print "40";

(print "50") && (print "60");
```

```
201

401

5060
```

From source to runtime



Lexing / tokenization

1 The source file is read as byte chunks, which are formed into “words” the language can understand.

Parsing

2 The words are formed into a tree, as their relationship is grammatically checked against the language specification.

Compiling

3 The AST, created by the parser, is converted into opcodes understood by the runtime.

Execution

4 The opcodes are fed into a virtual machine, which executes them one-by-one.

Parsing

Identifying code

Code point	`$`
Token	`$foo`
Expression	`$foo : 12`
Statement	`$foo : 12;`

Statement list	`$foo : 12;`
	`$bar : 24;`

Block	`{`
	    `$foo : 12;`
	    `$bar : 24;`
	`}`

Code point iterator

- Define atoms of source file
  - Support multi-byte characters?
  - What locale to use?
- Act as a cursor for the tokenizer
  - Store line and column numbers

`$foo;`					
	1	2	3	4	5
1	$	f	o	o	;

Tokenizing: Function

```
function () {

    # no-op

};



# Whitespace and comments

# are stripped out
```

T_FUNCTION	'function'
T_PAREN_OPEN	'('
T_PAREN_CLOSE	')'
T_CURLY_OPEN	'{'
T_CURLY_CLOSE	'}'
T_END_STATEMENT	';'
T_END_PROGRAM	NULL

Tokenizing: Numbers

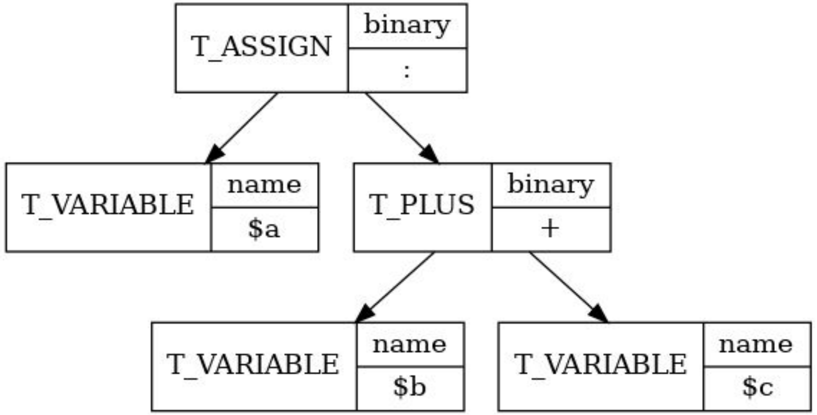
```
# Integers
1;

# Negative integers
-12;

# Floats
1.0;
1.;
.0;

# Negative floats
-33.0;

# Negative number, positive exponent.
-1e2;

# Negative number, negative exponent.
-1e-2;

# Octal numbers
0644;

# Binary numbers
0b1011;

# Hex numbers
0x09afAF;
```

```
T_NUMBER                    '1'
T_END_STATEMENT             ';'
T_NUMBER                    '-12'
T_END_STATEMENT             ';'
T_NUMBER                    '1.0'
T_END_STATEMENT             ';'
T_NUMBER                    '1.'
T_END_STATEMENT             ';'
T_NUMBER                    '.0'
T_END_STATEMENT             ';'
T_NUMBER                    '-33.0'
T_END_STATEMENT             ';'
T_NUMBER                    '-1e2'
T_END_STATEMENT             ';'
T_NUMBER                    '-1e-2'
T_END_STATEMENT             ';'
T_NUMBER                    '0644'
T_END_STATEMENT             ';'
T_NUMBER                    '0b1011'
T_END_STATEMENT             ';'
T_NUMBER                    '0x09afAF'
T_END_STATEMENT             ';'
T_END_PROGRAM               NULL
```

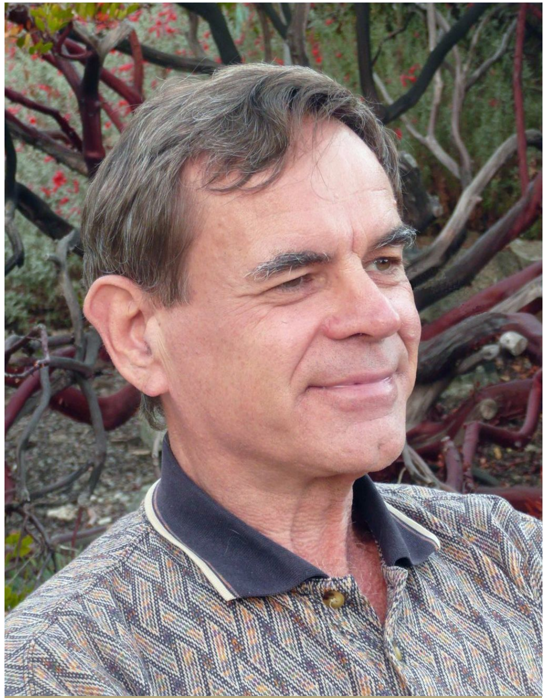
Parsing tokens

```
$a : $b + $c;
```

```
T_VARIABLE              '$a'
T_ASSIGN                ':'
T_VARIABLE              '$b'
T_PLUS                  '+'
T_VARIABLE              '$c'
T_END_STATEMENT         ';'
T_END_PROGRAM           NULL
```



Implementation


By Vaughan Pratt - Photograph taken and owned by Vaughan Pratt, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=5771111

Top down operator precedence

- Invented by *Vaughan Pratt*
- Requires a *functional* programming language
- Designed for *expression languages*
- Parses using the *binding power* of operators
- Divides expressions in *left denotation* (*led*) and *null denotation* (*nud*)

Examples of nud and led

```
$b : 12;

return $a + 12;
```

From token to node

```
$a : $b + $c;
```

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

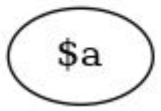
$current   Name

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$current	Name
$left	NamedNode

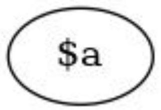


```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$current	Name
$left	NamedNode

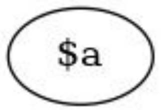


```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
$left	NamedNode

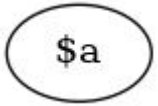


```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
bindingPower	10
$left	NamedNode



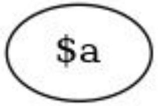
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
bindingPower	10
$left	NamedNode



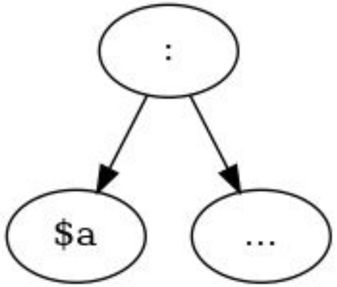
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
bindingPower	10
$left	AssignmentNode



$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$current	Name
bindingPower	0

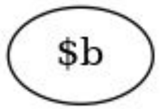
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$current	Name
bindingPower	0
$left	NamedNode



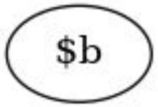
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$current	Name
bindingPower	0
$left	NamedNode



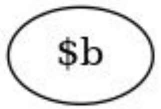
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	NamedNode



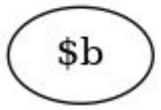
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	NamedNode



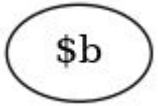
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	NamedNode



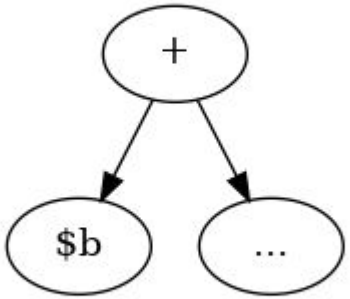
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	AdditionNode



$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$c;
```

$current	Name
bindingPower	0

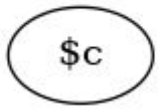
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

$c ;

$current	Name
bindingPower	0
$left	NamedNode



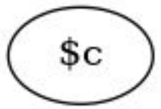
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

$c ;

$subject	Symbol
bindingPower	0
$left	NamedNode



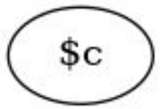
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

$c ;

$symbol	Symbol
bindingPower	0
$left	NamedNode



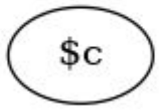
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$c ;
```

$symbol	Symbol
bindingPower	0
$left	NamedNode



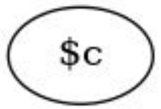
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

$c ;

$symbol	Symbol
bindingPower	0
$left	NamedNode



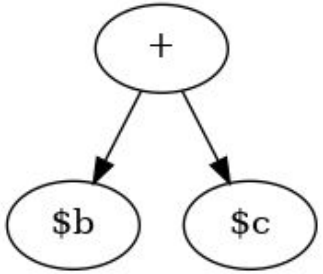
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	AdditionNode



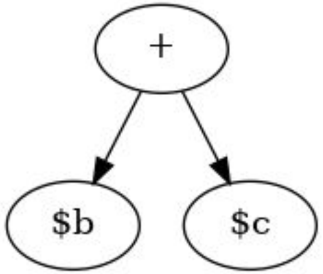
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Addition
bindingPower	10
$left	AdditionNode



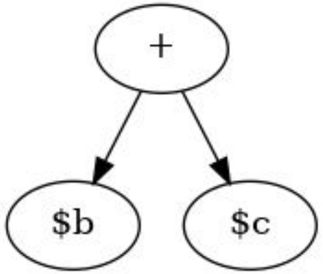
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AdditionNode



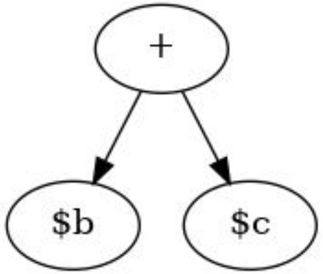
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AdditionNode



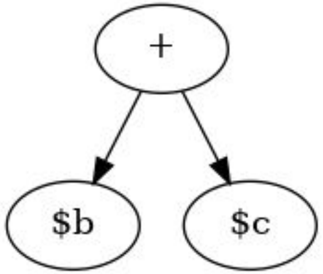
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AdditionNode



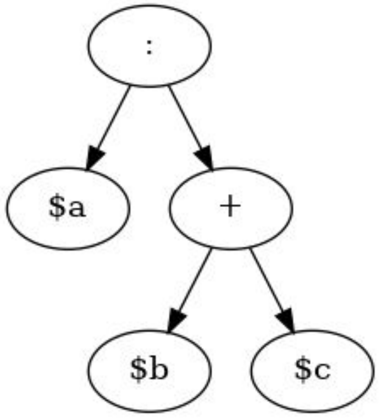
$bindingPower = 9

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
bindingPower	10
$left	AssignmentNode



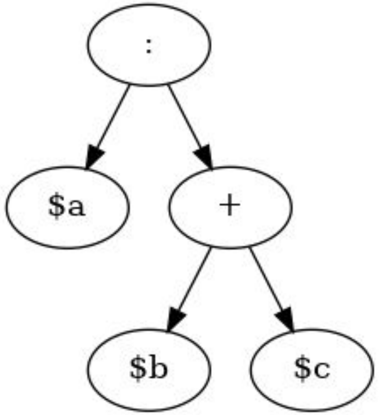
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Assignment
bindingPower	10
$left	AssignmentNode



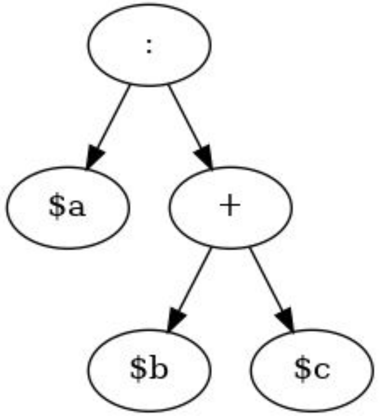
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AssignmentNode



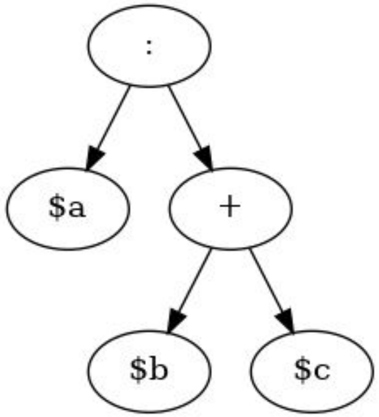
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AssignmentNode



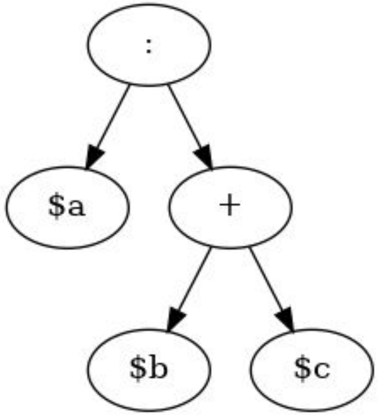
$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

From token to node

```
$a : $b + $c;
```

$symbol	Symbol
bindingPower	0
$left	AssignmentNode

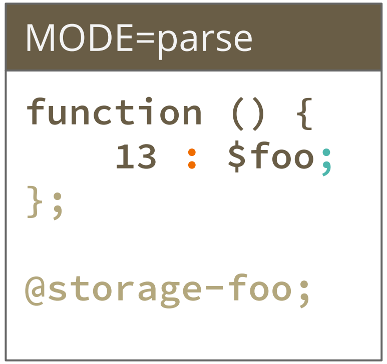


$bindingPower = 0

```
$current = $this->symbols->getSymbol($context->current());
$left    = $current->nud($context);
$subject = $context->advance();
$symbol  = $this->symbols->getSymbol($subject);

while ($symbol !== null
    && $bindingPower < $symbol->getBindingPower()
) {
    $context->advance();
    $left    = $symbol->led($context, $subject, $left);
    $subject = $context->current();
    $symbol  = $this->symbols->getSymbol($subject);
}

return $left;
```

Show and tell

Just in Time



```
MODE=parse

function () {
    13 : $foo;
};

@storage-foo;
```

- 🟫 Unread source code
- 🟦 Current code point
- ⬛ Tokenized
- 🟧 Current symbol

```
Invalid Assignment(":") encountered. Unexpected literal. Can not assign to literal
"13" - In tests/syntax/error/unknown_token_after_syntax_error.sym on line 2 column 8

1: function () {
──────────────┐
2:      13 : $foo;
──────────────┘
3: };
4:
5: @storage-foo;
6:
```

Just in Time

MODE=tokenize

```
function () {
    13 : $foo;
};

@storage-foo;
```

🟨 Unread source code

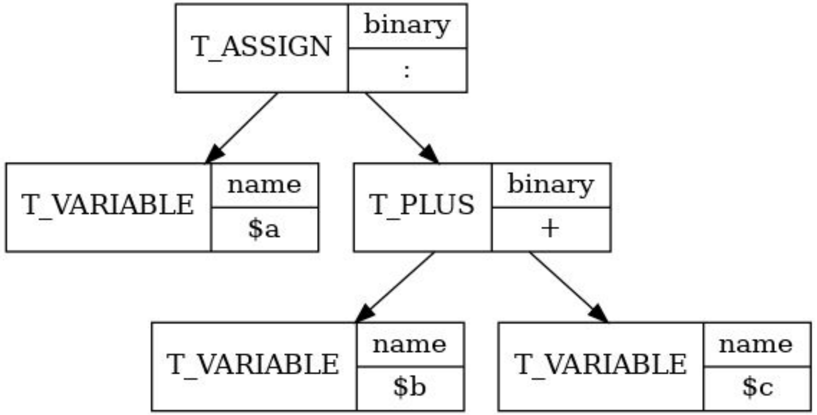
🟦 Current code point

🟫 Tokenized

```
Unexpected token sequence "@" in file
tests/syntax/error/unknown_token_after_syntax_error.sym at line 5 column 1.

1:  function () {
2:      13 : $foo;
3:  };
4:
────┐
5:  @storage-foo;
────┘
6:
```

Compiling

Compiling



Variables	
$a	!0
$b	!1
$c	!2

Line	#	Op	Ext	Return	Operands
1	0	ADD		~3	!1, !2
	1	ASSIGN			!0, ~3

Running

Runtime

- Virtual Machine
  - Processor (CPU)
    - Instruction set
      - Instruction ↔ Opcode
    - Registries (Internal memory)
      - A, B, X, Y, Z
  - Object storage (RAM)
  - Filesystem layer (Persistent storage)
  - Input layer - Program arguments (Keyboard)
  - Output layer - CLI / FPM / CGI (Video monitor)

Optimizations

Optimizations

Tokens

- Remove whitespace tokens
- Remove comment tokens

AST nodes / statements

- Remove unused variables
- Remove unreachable code
- Convert single-use variables in literals

Opcodes

- Remove opcodes that are negated down the line
- Combine opcodes in specialized instructions

Opcache

- Cache opcodes against filename and modification timestamp

Conclusion

Conclusion

- Were all goals met?
- What were challenges so far?
  - Functional vs OOP
  - Expression language *with* function statements on top
  - Learning to split up the language in components that make sense
  - Identifying tokens that share characters (*T_NUMBER -12* vs *T_MINUS -*)
  - Determining how to enforce the grammar of the language
  - Making the specification and language components interchangeable
  - PHP Specifically:
    - having to represent objects through a class based system is really tedious and made otherwise simple value objects / structs into objects that require class methods (PHP 7.4 and 8.0 address this partially).
    - Each layer of the JIT implementation is currently blocked by the next.
- What challenges remain?
  - Implementing a compiler
  - Adding a disassembler, for debugging purposes
  - Figuring out stack traces
  - Creating a runtime that understands the compiled code
  - Write up documentation for implementing Symbiont within application frameworks or bespoke software

Questions?

Where you can find me:

🌐 janmarten.name

🦜 johmanx10

Where are the slides?

📄 janmarten.name/talks

Resources

Top down operator precedence (paper)	https://dl.acm.org/doi/10.1145/512927.512931
Parser for simplified JavaScript	http://crockford.com/javascript/tdop/
PHP Language specifications	https://github.com/php/php-langspec/
PHP Operator precedence	https://www.php.net/manual/en/language.operators.precedence.php
GOTO 2013 - Syntaxation	https://youtu.be/Nlqv6NtBXcA
Symbiont	https://janmarten.name/symbiont/
Vulcan Logic Dumper	https://derickrethans.nl/projects.html#vld