A `#lang` for data structures students

Jesse Tov, RacketCon.eighth()

Welcome to DSSL2

```
#lang dssl2

struct nil: pass

struct cons:
    let car
    let cdr
```

```
class ListBuilder:
    let head
    let tail

    def __init__(self):
        self.head = nil()
        self.tail = nil()

    def snoc(self, x):
        let old_tail = self.tail
        self.tail = cons(x, nil())
        if nil?(old_tail):
            self.head = self.tail
        else:
            old_tail.cdr = self.tail

    def build(self):
        let result = self.head
        self.__init__()
        result
```

Road map

- What’s DSSL2 for?
- What’s it like?
- How does it work?
- Was it worth it?

What’s it for?

Why would I want to program in it?

Why would I want to program in it?

You wouldn’t!

A story

A story

- C

A story

- C
- Java

A story

- C
- Java
- Python

A story

- C
- Java
- Python
- Ruby

A story

- C
- Java
- Python
- Ruby
- C++

Disaster!

```
Node* node;
node->data = data;
```

Disaster!

```
Node* node;
node->data = data;

Node* current = new Node;
current = this->root;
```

Prerequisites for data structures at Northwestern

- 10 weeks of BSL/ISL
- 10 weeks of C++

What now?

What now?

- Teach C++ with a side of data structures?

What now?

- Teach C++ with a side of data structures?
- Teach Java with a side of data structures?

What now?

- Teach C++ with a side of data structures?
- Teach Java with a side of data structures?
- Teach data structures in a language the students already know?

What now?

- Teach C++ with a side of data structures?
- Teach Java with a side of data structures?
- Teach data structures in a language the students already know?
- Teach data structures in a language the students can easily learn?

What now?

- Teach C++ with a side of data structures?
- Teach Java with a side of data structures?
- Teach data structures in a language the students already know?
- Teach data structures in a language the students can easily learn?

What now?

- Teach C++ with a side of data structures?
- Teach Java with a side of data structures?
- Teach data structures in a language the students already know?
- Teach data structures in a language the students can easily learn?

What about a teaching language?

From Advanced Student Language…

```
(define (insert! t k)
  (cond
    [(tree-empty? t) (new-node k)]
    [(zero? (random (+ 1 (size t))))
     (root-insert! t k)]
    [(< k (node-key t))
     (begin
       (set-node-left! t (insert! (node-left t) k))
       (fix-size! t)
       t)]
    [(> k (node-key t))
     (begin
       (set-node-right! t (insert! (node-right t) k))
       (fix-size! t)
       t)]
    [else t]))
```

…to DSSL…

```
(define (insert! t k)
  (cond
    [(empty? t) (new-node k)]
    [(zero? (random (+ 1 (size t))))
     (root-insert! t k)]
    [(< k (node-key t))
     (set-node-left! t (insert! (node-left t) k))
     (fix-size! t)
     t]
    [(> k (node-key t))
     (set-node-right! t (insert! (node-right t) k))
     (fix-size! t)
     t]
    [else t]))
```

…to DSSL2

```
def insert!(t, k):
    if empty?(t): new_node(k)
    elif random(size(t) + 1) == 0:
        root_insert!(t, k)
    elif k < t.key:
        t.left = insert!(t.left, k)
        fix_size!(t)
        t
    elif k > t.key:
        t.right = insert!(t.right, k)
        fix_size!(t)
        t
    else: t
```

What’s it like?

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked
- structs (not dictionaries)

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked
- structs (not dictionaries)
- fixed-sized arrays

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked
- structs (not dictionaries)
- fixed-sized arrays
- classes and interfaces

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked
- structs (not dictionaries)
- fixed-sized arrays
- classes and interfaces
- built-in support for unit tests

DSSL2 in a nutshell

- strict/eager, untyped, dynamically checked
- structs (not dictionaries)
- fixed-sized arrays
- classes and interfaces
- built-in support for unit tests
- contracts

A DSSL2 example (1/3)

```
interface STACK:
    def push(self, element)
    def pop(self)
    def empty?(self)
    def full?(self)
```

A DSSL2 example (2/3)

```
class VecStack (STACK):
    let _data
    let _len

    def __init__(self, capacity):
        self._data = [False; capacity]
        self._len  = 0

    def empty?(self):
        self._len == 0

    def full?(self):
        self._len == self._data.len()
```

A DSSL2 example (3/3)

```
    def push(self, element):
        if self.full?(): error('VecStack.push: full')
        self._data[self._len] = element
        self._len = self._len + 1

    def pop(self):
        if self.empty?(): error('VecStack.pop: empty')
        self._len = self._len - 1
        let result = self._data[self._len]
        self._data[self._len] = False
        result

test 'VecStack':
    let s = VecStack(8)
    s.push('hello')
    assert_eq s.pop(), 'hello'
```

How does it work?

Implementation of DSSL2

It’s a Racket `#lang`:

- Run-time system for free
- IDE with syntax coloring and renaming for “free”
- A nice documentation system

Custom reader

It has a custom reader, written using `parser-tools/lex`:

```
[#\λ                              (token-LAMBDA)]
["True"                           (token-LITERAL #t)]
["False"                          (token-LITERAL #f)]
["def"                            (token-DEF)]
```

Custom reader

It has a custom reader, written using `parser-tools/lex`:

```
[#\λ                                (token-LAMBDA)]
["True"                             (token-LITERAL #t)]
["False"                            (token-LITERAL #f)]
["def"                              (token-DEF)]
```

And `parser-tools/yacc`:

```
(<expr6>
  [(<expr6> OP6 <expr7>)            (loc/2 (list $2 $1 $3))]
  [(<expr7>)                        $1])
```

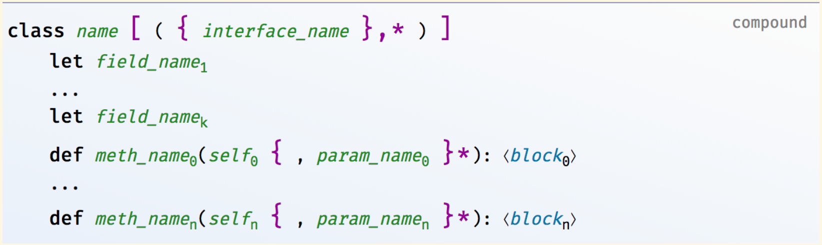
A bunch of macros

```
(define-syntax-rule (dssl-while test expr ...)
  (let/ec break-f
    (let loop ()
      (define (continue-f) (loop) (break-f (void)))
      (syntax-parameterize
        ([dssl-break
          (syntax-rules () [(_) (break-f (void))])]
         [dssl-continue
          (syntax-rules () [(_) (continue-f)])])
        (when test
          (dssl-begin expr ...)
          (loop))))))
```

Implementation difficulties: symbols

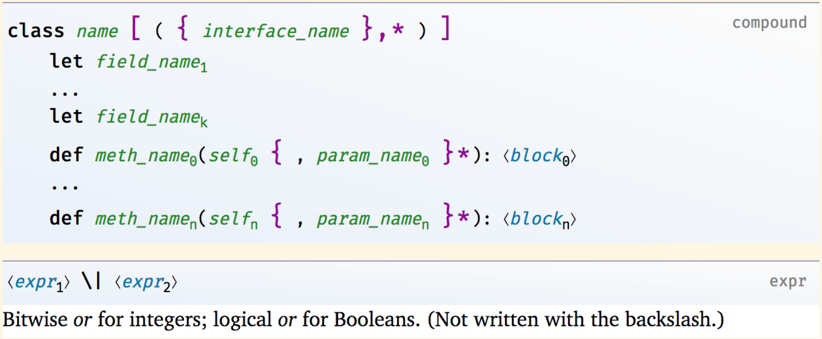
```
(define (locate/symbol sym pos)
  (let ([port (open-input-string (format "~s" sym))])
    (port-count-lines! port)
    (set-port-next-location! port
                             (position-line pos)
                             (position-col pos)
                             (position-offset pos))
    (read-syntax src port)))
```

Implementation difficulties: documentation



```
class name [ ( { interface_name },* ) ]                    compound
    let field_name$_1$
    ...
    let field_name$_k$
    def meth_name$_0$(self$_0$ { , param_name$_0$ }*): ⟨block$_0$⟩
    ...
    def meth_name$_n$(self$_n$ { , param_name$_n$ }*): ⟨block$_n$⟩
```

Implementation difficulties: documentation

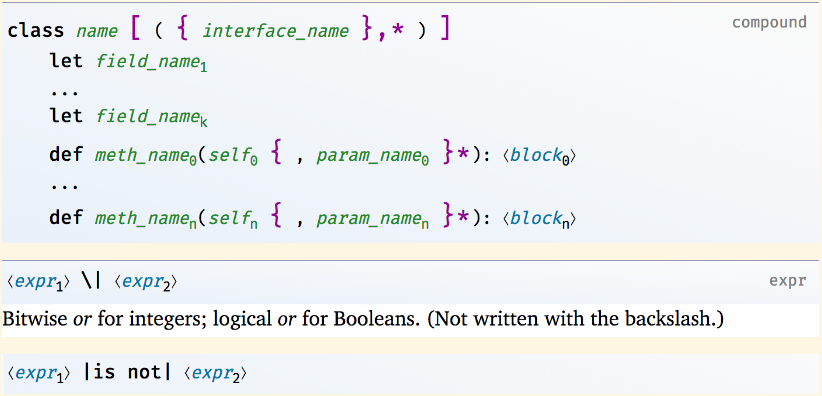


```
class name [ ( { interface_name },* ) ]                          compound
    let field_name1
    ...
    let field_namek
    def meth_name0(self0 { , param_name0 }*): ⟨block0⟩
    ...
    def meth_namen(selfn { , param_namen }*): ⟨blockn⟩
```

```
⟨expr1⟩ \| ⟨expr2⟩                                               expr
```

Bitwise *or* for integers; logical *or* for Booleans. (Not written with the backslash.)

Implementation difficulties: documentation



```
class name [ ( { interface_name },* ) ]                    compound
    let field_name₁
    ...
    let field_nameₖ
    def meth_name₀(self₀ { , param_name₀ }*): ⟨block₀⟩
    ...
    def meth_nameₙ(selfₙ { , param_nameₙ }*): ⟨blockₙ⟩
```

```
⟨expr₁⟩ \| ⟨expr₂⟩                                          expr
```

Bitwise *or* for integers; logical *or* for Booleans. (Not written with the backslash.)

```
⟨expr₁⟩ |is not| ⟨expr₂⟩
```

Was it worth it?

Pros

Pros

- Students seem to pick it up easily—no complaints

Pros

- Students seem to pick it up easily—no complaints
- I don’t get badly indented code

Pros

- Students seem to pick it up easily—no complaints
- I don’t get badly indented code
- They can’t copy code off the internet

Cons

Cons

- I have a language to maintain

Cons

- I have a language to maintain
- It might be better for them to get better at C++ or learn Java

Thank you