

## Lecture 4. DefineLang – Global Variables

September 22, 2019

# Local and Global Variables

- ▶ Local variable: available based on the scope of the let expression
- ▶ Global variable: available during the entire iteration with the interpreter
  - ▶ (define Sun 0)
  - ▶ (define a 97)
- ▶ DefineLang: new feature added to VarLang, called *define declaration*
- ▶ Syntax: keyword define, name, initial value
- ▶ Example:  
(define i 1)  
(define ii 2)  
(\* i ii)

# Examples

```
$ (define R 8.3145) // The gas constant R
$ (define n 2) // 2 moles of gas
$ (define V 0.0224) // Volume of gas 0.0224 m^2
$ (define T 273) // Temperature of gas 273 K
$ (define P (/ (* n R T) V)) // Using Boyles law to compute pressure
$ P //What is the pressure?
202665.93750000003
```

```
$ (define F 96454.56) (define R 10973731.6)
```

The Definelang language also permits defining one or more constants and then computing the value of an expression.

```
$ (define R 8.3145) (/ (* 2 R 273) 0.0224)
202665.93750000003
```

# DefineLang Demo

Type a program to evaluate and press the enter key, e.g. `(let ((a 3) (b 100) (c 84) (d 279) (e 277)) (+ (* a b) (/ c (- d e))))`

Press Ctrl + C to exit.

```
(define P 3.1415926)
```

```
$
```

```
P
```

```
$ 3.1415926
```

```
(let ((P 3)) (* 2 2 P))
```

```
$ 12
```

```
P
```

```
$ 3.1415926
```

```
(define P (* 3.14 2 2))
```

```
$
```

```
P
```

```
$ 12.56
```

```
(define P (let ((r 2)) (* 3.14 r r)))
```

```
$
```

```
P
```

```
$ 12.56
```

```
(define P 3.14) (define r 2)
```

```
$
```

```
P
```

```
$ 3.14
```

```
r
```

```
$ 2
```

# Grammar

Program	::=	DefineDecl* Exp?	<i>Program</i>
DefineDecl	::=	(define Identifier Exp)	<i>Define</i>
Exp	::=	Number	<i>Expressions</i>
		(+ Exp Exp <sup>+</sup> )	<i>NumExp</i>
		(- Exp Exp <sup>+</sup> )	<i>AddExp</i>
		(* Exp Exp <sup>+</sup> )	<i>SubExp</i>
		(/ Exp Exp <sup>+</sup> )	<i>MultExp</i>
		Identifier	<i>DivExp</i>
		(let ((Identifier Exp) <sup>+</sup> ) Exp)	<i>VarExp</i>
Number	::=	Digit	<i>LetExp</i>
		DigitNotZero Digit <sup>+</sup>	<i>Number</i>
Digit	::=	[0-9]	<i>Digits</i>
DigitNotZero	::=	[1-9]	<i>Non-zero Digits</i>
Identifier	::=	Letter LetterOrDigit*	<i>Identifier</i>
Letter	::=	[a-zA-Z\$_]	<i>Letter</i>
LetterOrDigit	::=	[a-zA-Z0-9\$_]	<i>LetterOrDigit</i>

# Extending AST (syntax): Read Phase

1. New AST node: DefineDecl
2. Modify program to store DefineDecl
3. Modify visitor interface to support DefineDecl
4. Modify formatter regrading print DefineDecl

# Extending Semantics: Eval

- ▶ Varlang: evaluate a program starting in an empty environment; DefineLang: when a program starts running, the declared global variables are defined; the program can have free variables
- ▶ Unitval: A UnitVal is like a void type in Java. It allows programming language definitions and implementations to uniformly treat programs and expressions as evaluating to 'a value' – e.g., a define declaration

Value	::=		<i>Values</i>
		NumVal	<i>Numeric Values</i>
		UnitVal	<i>Unit Values</i>
NumVal	::=	(NumVal n), where $n \in$ the set of doubles	<i>NumVal</i>
UnitVal	::=	(UnitVal)	<i>NumVal</i>

- ▶ each definition changes the global initEnv to add a new binding from name to value.

# Review and Further Reading

Definelang: support globals

- ▶ Syntax: definition declaration (AST node, visitor interface)
- ▶ Semantics: modify environment for each run; unitval;

Let's walk through the code in the interpreter

Further reading:

- ▶ Rajan: CH 4, Sebesta Ch 7, 8