

Interpreters

Dr. Mattox Beckman

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
DEPARTMENT OF COMPUTER SCIENCE

Objectives

You should be able to ...

- ▶ Enumerate and explain the different parts of an interpreter.
- ▶ Explain what an abstract syntax tree is.
- ▶ Explain the difference between an interpreter and a compiler.
- ▶ Explain what REPL means and what it does.
- ▶ Show how to define types in `HASKELL` to represent expressions, values, and statements.

What Is an Interpreter?

- ▶ There are two ways to execute code on a computer:
 - ▶ Convert the code to machine code and run it directly.
 - ▶ Have another program read the code and “do what it says.”
- ▶ The second method is what we will do in this course.

Parts of an Interpreter

- ▶ The *parser*
 - ▶ Converts your ASCII input into an *abstract syntax tree*
- ▶ The *evaluator*
 - ▶ Processes the abstract syntax tree to yield a result
 - ▶ A type to represent values
 - ▶ A function to evaluate the expressions into values (`eval`),
- ▶ An *environment* to keep track of the values of variables
- ▶ A top-level function to tie all this together: the *REPL*
 - ▶ Read
 - ▶ Eval
 - ▶ Print
 - ▶ Loop

Our Language

Let's write an interpreter for a simple functional language. We want the language to have:

- ▶ Integers
- ▶ Variables
- ▶ Arithmetic (+, -, *, /)
- ▶ Comparisons (<, <=, >, >=, =, ≠)
- ▶ Booleans (true, false, and, or, not)
- ▶ Local variables (let)
- ▶ Conditionals
- ▶ Functions

File Structure

- ▶ A reference version has been provided for you.
- ▶ `stack build` will compile them for you.
- ▶ `stack exec i1` will run the first interpreter.
- ▶ `stack repl i1/Main.hs` will load the interpreter but give you a `HASKELL` prompt.

```
first/
|---- i1/
|      |--- Main.hs
|      |--- Types.hs
|      |--- Parser.hs
|---- i2/
|      |--- Main.hs
|      |--- Types.hs
|      |--- Parser.hs
|---- ...
```

Define the Types – Types.hs

```
1 data Exp = IntExp Integer
2   deriving (Show,Eq)
3
4 data Val = IntVal Integer
5   deriving (Show,Eq)
6
7 type Env = [(String,Val)]
```

Eval – I1.hs

```
1 eval :: Exp -> Env -> Val
2 eval (IntExp i) _ = IntVal i

% stack repl i1/Main.hs
*Main Lib Parser Types> :t eval
eval :: Exp -> Env -> Val
*Main Lib Parser Types> eval (IntExp 123) []
IntVal 123
*Main Lib Parser Types> main
Welcome to your interpreter!
i1> 23
IntVal 23
i1> quit
```

► Or use `stack exec i1` to run it directly.

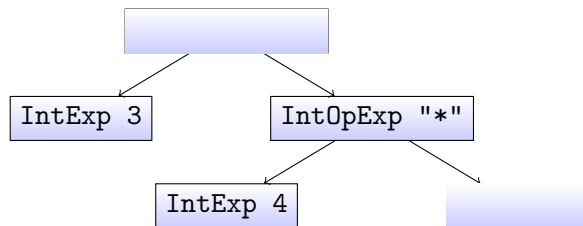
Adding Arithmetic and Abstract Syntax Trees

- Add the following to the Exp type.

```
1 data Exp = IntOpExp String Exp Exp
2         | ...
```

- Represent $3 + 4 * 5$ with HASKELL code.

```
1 IntOpExp "+" (IntExp 3)
2   (IntOpExp "*" (IntExp 4) (IntExp 5))
```



- Note that this is a tree!

Eval – i2.hs

```
1 eval (IntOpExp "+" e1 e2) env =
2   let v1 = eval e1 env
3       v2 = eval e2 env
4   in IntVal (getInt v1 + getInt v2)
5 eval (IntOpExp "*" e1 e2) env =
6   let v1 = eval e1 env
7       v2 = eval e2 env
8   in IntVal (getInt v1 * getInt v2)
9 eval (IntOpExp "-" e1 e2) env =
10  let v1 = eval e1 env
11      v2 = eval e2 env
12  in IntVal (getInt v1 - getInt v2)
13 getInt (IntVal i) = i
14 getInt _          = 0
```

Making a Dictionary

```
1 intOps = [ ("+", (+))  
2           , ("-", (-))  
3           , ("*", (*))  
4           , ("/", div)]  
5  
6 liftIntOp f (IntVal i1) (IntVal i2) = IntVal (f i1 i2)  
7 liftIntOp f _ _ = IntVal 0
```

The compiler will give you a warning about liftIntOp.

```
Main> liftIntOp (*) (IntVal 10) (IntVal 20)  
IntVal 200
```

Our New Eval – I2

```
Main> let Just f = lookup "*" intOps
```

```
Main> f 10 20
```

```
200
```

```
1 eval (IntOpExp op e1 e2) env =  
2   let v1 = eval e1 env  
3       v2 = eval e2 env  
4       Just f = lookup op intOps  
5   in liftIntOp f v1 v2
```

You try!

- ▶ The interpreter `i3` is a copy of `i2` with some extra constructors added:
 - ▶ `RelOpExp` – for integer comparisons like \geq
 - ▶ `BoolOpExp` – for `&&` and `||`
 - ▶ `BoolExp` – for `True` and `False`
 - ▶ `BoolVal` – the corresponding value
- ▶ The parser has also been updated to return these expressions.
- ▶ See if you can update `eval` to work with these new things.
- ▶ The next video will go over the solutions, plus show you how to add variables to the language. (Or you can peek at `i4` ...)