CSE 12 – Basic Data Structures

Prof. Christine Alvarado
[Slides borrowed/adapted from slides by Cynthia Lee, Rakesh Varna, & Roshni Chandrashekhar]
Announcements

1. HW8
   1. Released Wednesday
   2. Due Tuesday week 10 (2 week assignment)
   3. Worth 400 points.
   4. It will be LONG. If you wait to start you will not succeed and you will significantly drop your grade in the class!

2. HW8 is PARTNER ENCOURAGED
   1. You must sign up via the Google form no later than next FRIDAY at 5pm
   2. Once you sign up you are “married” to your partner. Divorces must be done in person by me or Dr. P
Announcements

1. I am traveling this afternoon-Thursday
   1. No office hours tomorrow. If you want to meet me send me an email.
   2. No class Wednesday. Watch the video. Really, I mean it. (Did I mention HW8 is LONG?)
Today’s topics

- Language Specification
- BNF Grammars
- Derivations, parse trees and abstract syntax trees
Welcome to DrJava. Working directory is ...

> int x = 7;
> int y = 19;
> x + y
26
> String s = "a string";
> s + y
"a string19"
> s / y
Static Error: Bad type in numeric expression
The Unicalc language

> java Unicalc

input> 14 m + 9 m
Result: 23.0 meter

input> 60 Hz * 30 s
Result: 1800.0 ~ 60.0 Hz s

input> # 364.4 smoot
Result: 364.4 ~ 0.0 smoot

input> def smoot 67 in
Result: 67.0 ~ 0.0 in

input> # 364.4 smoot
Result: 620.13592 ~ 0.0 meter
Compilers and interpreters
Compilers and Interpreters

- One of the most important kinds of programs are programs that read other programs and translate them into action
  - Compilers and interpreters
- They both have two main steps in their processing
  - Syntax analyzer
  - Semantic evaluator
Compilers and Interpreters

- Syntax analyzer:
  - Recursively builds up an AST

- Semantic evaluator:
  - Takes an AST and does the calculation by traversing the tree
Stages of an Interpreter

- Lexical Analyzer
  - AKA Tokenizer
- Parser
- Evaluator
  - Environment
- Printer
Stages of an Interpreter: Minimath

Tokenizer

"5 + 3 * 2"

Parser

5 + 3 * 2

Evaluator

symbol + number 5 subexp

Environment

symbol * number 3 number 2

Printer

number 11.0

"11.0"
Your own language...

Tokenizer

Parser

Evaluator

Printer

String input

List<String> tokens

AST parseTree

Quantity result

Printer

result.toString

Printed result
Getting the Symbols

Tokenizer

String input

List<String> tokens

"5+3*2"  \[\rightarrow\]  ["5", "+", "3", "+", "2"]

"5plus3times2"  \[\rightarrow\]  ["5", "plus", "3", "times", "2"]

"fiveplusthreetimestwo"  \[\rightarrow\]  ["five", "plus", "three", "times", "two"]
public static void main(String[] arg)
{
    int x = 32;
    int y = 9;
    int z;

    z = x+++y;  \textit{Is this even legal?!}

    System.out.println(“x is “ + x);
    System.out.println(“y is “ + y);
    System.out.println(“z is “ + z);
}
Syntax Analyzer (Parser)

Building Abstract Syntax Trees
Understanding a sentence

He gave her food
He gave her cat food

Understanding a sentence

He | gave | food
---|------|------
her | cat

He | gave | food
---|------|------
cat | her
Understanding a sentence?

Food cat gave he.
Understanding a sentence

"5 + 3 * 2"

? 

(5 + 3) * 2  

5 + (3 * 2) 

"5 2 + * 3"

error
A sample grammar

\begin{align*}
\langle S \rangle & := \langle S \rangle + \langle P \rangle \mid \langle S \rangle - \langle P \rangle \mid \langle P \rangle \\
\langle P \rangle & := \langle P \rangle \ast \langle M \rangle \mid \langle P \rangle / \langle M \rangle \mid \langle M \rangle \\
\langle M \rangle & := \langle \text{const} \rangle \mid (\langle S \rangle) \\
\langle \text{const} \rangle & := 0 \mid 1 \mid 2 \mid 3 \mid 4
\end{align*}
Sample derivation

“2”

\(<S> \Rightarrow <P> \Rightarrow <M> \Rightarrow \text{<const>} \Rightarrow 2\)

\(<S> := <S> + <P> | <S> - <P> | <P>\)

\(<P> := <P> * <M> | <P> / <M> | <M>\)

\(<M> := \text{<const>} | (<S>)\)

\(<\text{const}> := 0 | 1 | 2 | 3 | 4\)
You turn!

“2 + 5 + 3”

Which of the following steps can NOT follow immediately after the other in a derivation?

A. \( <S> \rightarrow <S> + <P> \rightarrow \)  
B. \( <S> + <P> + 3 \rightarrow <P> + <P> + 3 \rightarrow \)  
C. \( <M> + <P> + 3 \rightarrow <\text{const}> + <M> + 3 \rightarrow \)  
D. \( 2 + <\text{const}> + 3 \rightarrow 2 + 5 + 3 \)
E. Other/none/more
Solution

"2 + w + 3"

\(<S> \Rightarrow \langle S\rangle + \langle P\rangle \Rightarrow \langle S\rangle + \langle M\rangle \Rightarrow \langle S\rangle + \langle \text{const}\rangle \Rightarrow \langle S\rangle + 3 \Rightarrow \langle P\rangle + \langle P\rangle + 3 \Rightarrow \langle M\rangle + \langle M\rangle + 3 \Rightarrow \langle \text{const}\rangle + \langle M\rangle + 3 \Rightarrow \langle \text{const}\rangle + \langle \text{const}\rangle + 3 \Rightarrow 2 + \langle \text{const}\rangle + 3 \Rightarrow 2 + 5 + 3\)

\(<S> := <S> + <P> | <S> - <P> | <P> \\
<P> := <P> * <M> | <P> / <M> | <M> \\
<M> := <\text{const}> | (<S>) \\
<\text{const}> := 0 | 1 | 2 | 3 | 4 \)
Your turn!

“2 + 5 * 3”
From derivations to trees

“2 + 5 * 3”

\[ \langle S \rangle := \langle S \rangle + \langle P \rangle | \langle S \rangle - \langle P \rangle | \langle P \rangle \]
\[ \langle P \rangle := \langle P \rangle \times \langle M \rangle | \langle P \rangle / \langle M \rangle | \langle M \rangle \]
\[ \langle M \rangle := \langle \text{const} \rangle | (\langle S \rangle) \]
\[ \langle \text{const} \rangle := 0 | 1 | 2 | 3 | 4 \]

\[ \begin{align*}
\langle S \rangle & \Rightarrow \langle S \rangle + \langle P \rangle \\
& \Rightarrow \langle S \rangle + \langle P \rangle \times \langle M \rangle \\
& \Rightarrow \langle S \rangle + \langle P \rangle \times \langle \text{const} \rangle \\
& \Rightarrow \langle S \rangle + \langle M \rangle \times 3 \\
& \Rightarrow \langle S \rangle + \langle \text{const} \rangle \times 3 \\
& \Rightarrow \langle S \rangle + 5 \times 3 \\
& \Rightarrow 2 + 5 \times 3
\end{align*} \]
Your turn!

Draw the parse tree for “(2 + 5) * 3”

\[
\begin{align*}
<S> & ::= <S> + <P> | <S> - <P> | <P> \\
<P> & ::= <P> * <M> | <P> / <M> | <M> \\
<M> & ::= <\text{const}> | (<S>) \\
<\text{const}> & ::= 0 | 1 | 2 | 3 | 4
\end{align*}
\]
Abstract Syntax Trees

```
( (const 2) + (const 5) )
```

Parse Tree

```
<SP> <PM> <SM> <SS> + <SP> <MM> <SC> <SC> 5
  |  |  |
2  S  M  S
```

AST

```
<SP> <PM> <MM> <SC> <SC> 5
  |  |  |
2  M  S  5
```
OK, so how does the computer create the tree?

\[ \langle S \rangle := \langle P \rangle + \langle S \rangle \mid \langle P \rangle - \langle S \rangle \mid \langle P \rangle \]
\[ \langle P \rangle := \langle M \rangle \times \langle P \rangle \mid \langle M \rangle / \langle P \rangle \mid \langle M \rangle \]
\[ \langle M \rangle := \langle \text{const} \rangle \mid (\langle S \rangle) \]
\[ \langle \text{const} \rangle := 0 \mid 1 \mid 2 \mid 3 \mid 4 \]

A slight modification to the grammar (no longer follows proper associativity)
What is the parse tree for “( 2 + 5 )”? 

Recursive Decent Parsing

```
< S > := < P > + < S > | < P > - < S > | < P >
< P > := < M > * < P > | < M > / < P > | < M >
< M > := < const > | (< S >)
< const > := 0 | 1 | 2 | 3 | 4
```

tokens: ["(" , "2" , "+" , "5" , ")"]
S: "I start with a P, so ask P to parse"

tokens: ["(" , "2" , "+" , "5" , ")"]
P: "I start with a M, so ask M to parse"

tokens: ["(" , "2" , "+" , "5" , ")"]
M: "I could start with < const > so let’s try it"

tokens: ["(" , "2" , "+" , "5" , ")"]
const: "I don’t match ")"! Failure!

tokens: ["(" , "2" , "+" , "5" , ")"]
M: "OK, I could also start with ".
M: "Ah ha! I see a ")". Now I’ll ask S to parse"

tokens: ["2" , "+" , "5" , ")"]
S: "I start with a P, so ask P to parse"

tokens: ["2" , "+" , "5" , ")"]
P: "I start with a M, so ask M to parse"

tokens: ["2" , "+" , "5" , ")"]
M: "I could start with < const > so let’s try it"

tokens: ["2" , "+" , "5" , ")"]
const: "Oh good, a 2"

tokens: ["+" , "5" , ")"]
S: "Now that I have seen a P, I could see a +" 
S: "Oh look. There’s one! I’m a + S"
S: Now I need another S so ask S to parse…

tokens: ["5" , ")"]
Become a believer

"The only way to be successful with Recursive Descent is to have complete faith in recursion"
-- A HMC CS 60 student

Recursive Descent will test your faith…
Behold! miniMath

To the code...
innumbrage

atonement

i only in graven

ravens peck

softhearts seek kind images in a memento
gathers scarlet note

stendangers slay her relapses

At one mention, lying ravens peck soft hearts.

Go dig at her scar.

See kind images in a memento.

Let no test end.

Anger slays her.

Ten dangers lay.

She relapses.

In dim ages, name men to God, I gather scarlet notes.

seek.

of the arts,