Announcements

1. HW8
   1. Posted! Due Tuesday week 10 (2 week assignment)
   2. Checkpoint due next Tuesday
   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 TODAY 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. Monday is a holiday
   1. No class
   2. No tutor hours UNLESS they are on the schedule
2. Next week: C!
RD Parsing: A little cheating

What is the parse tree for “( 2 + 5 )”?

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

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!

Look ahead to avoid failure
Recursive Decent Parsing

What is the parse tree for “( 2 + 5 )”?

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 have 2 choices. Let’s see if I 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 have 2 choices. Let’s see if I start with (“

M: No, I don’t. So I must be <const>

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" , ")"]
Semantic Evaluator
Traversing Abstract Syntax Trees
Semantic Evaluator

- This is really just a **traversal** of the tree
- We have been talking about the “visit” operation as being something like “print the this node’s data variable”
- But here it will be to **evaluate the expression**
Which kind of traversal? (and WHY?)

A. Pre-order:
   1. "visit" self
   2. Traverse left
   3. Traverse right

B. In-order:
   1. Traverse left
   2. "visit" self
   3. Traverse right

C. Post-order:
   1. Traverse left
   2. Traverse right
   3. "visit" self

evaluate (tree) = -4
Ok so we can do eval() on paper…

- ...how do we do it in code??
public AST S()
{
    // S := <P>+<S> | <P>-<S> | <P>

    AST p = P();
    String next = toks.peek();
    if (next == null) {
        return p;
    }
    else if (next.equals("+")) {
        toks.pop();
        return new Sum(p, S());
    }
    else if (next.equals("-")) {
        toks.pop();
        return new Difference(p, S());
    }
    else {
        return p;
    }
}
interface AST {
    public String toString();
    public double eval(Map<String, Double> env);
}

A different class for each “action”.

class Sum implements AST {
    private AST left;
    private AST right;

    public Sum(AST l, AST r) {
        this.left = l;
        this.right = r;
    }

    public double eval(Map<String, Double> env) {
        return _____________________;
    }

    public String toString() {
        return ("Sum(" + left + "," + right + ")");
    }
}
What code goes in the blank?

A. left + right
B. left
C. left.eval()
D. left.eval() + right.eval()
E. left.eval(env) + right.eval(env)
interface AST {
    public String toString();
    public double eval(Map<String,Double> env);
}

class Constant implements AST {
    private double quant;

    public Constant(double d) {
        this.quant = d;
    }

    public double eval(Map<String,Double> env) {
        return quant;  // Nothing to eval: the leaf of the tree
    }

    public String toString() {
        return "Const(" + quant + ")";
    }
}

What do we use env for anyway??
But what about variables?

\[ \begin{align*}
\langle A \rangle & := \langle B \rangle \mid \langle S \rangle \\
\langle B \rangle & := \langle \text{ident} \rangle = \langle A \rangle \\
\langle S \rangle & := \langle P \rangle + \langle S \rangle \mid \langle P \rangle - \langle S \rangle \mid \langle P \rangle \\
\langle P \rangle & := \langle M \rangle \ast \langle P \rangle \mid \langle M \rangle / \langle P \rangle \mid \langle M \rangle \\
\langle M \rangle & := \langle \text{ident} \rangle \mid \langle \text{const} \rangle \mid (\langle S \rangle) \\
\langle \text{ident} \rangle & := w \mid x \mid y \mid z \\
\langle \text{const} \rangle & := 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid \ldots \mid 9
\end{align*} \]
But what about variables?

public AST parse()
{
    // Begin parsing with the start symbol
    AST answer = this.S();
    // more code here...
    return answer;
}

Which line should replace the line in bold above?
A. AST answer = this.S(); (i.e. stays the same)
B. AST answer = this.B();
C. AST answer = this.A();
D. Variable answer = this.S();
Adding variables

public AST A()
{
    if ( toks.size() > 1 && "=".equals( toks.get(1) ) ) {
        return ___(1)___;
    }
    else {
        return ___(2)___;
    }
}

What should go in blank 1?
A. A()
B. B()
C. S()
D. ident()
Adding variables

public AST A()
{
    if ( toks.size() > 1 && "=".equals( toks.get(1) ) ) {
        return ___(1)___;
    }
    else {
        return ___(2)___;
    }
}

What should go in blank 2?
A. A()
B. B()
C. S()
D. ident()
Adding variables

public AST B()
{
    String name = ident();
    String next = toks.peek();
    if (!_____ .equals( next )) {
        throw new ParseError( ... );
    }
}

Assume ident() finds a variable, removes it from toks, and returns it. What should go in the blank?
A. “+”
B. “=”
C. “-”
D. toks.peek()
E. null
public AST B()
{
    //<B> := <ident> = <A>
    String name = ident();
    String next = toks.peek();
    if ( !_equals( next ) ) {
        throw new ParseError( ... );
    }
}

What should B return?
A. A Sum (AST) object
B. A Product (AST) object
C. A Constant (AST) object
D. A different kind of AST object
class Define implements AST {
    private AST value;
    private String name;

    public Define(String n, AST v) {
        this.name = n;
        this.value = v;
    }

    public double eval(Map<String, Double> env) {
        double val = value.eval(env);
        env.put(name, val);
        return val;
    }

    public String toString() {
        return "Define(" + name + ", " + value + ")";
    }
}
class Variable implements AST
{
    private String name;

    public Variable(String n)
    {
        this.name = n;
    }

    public double eval(Map<String, Double> env)
    {
        return env.get(name);
    }

    public String toString()
    {
        return ("Variable(" + name + ")");
    }
}

Check out Minimath’s main for where the env comes from
Hash Tables (HashMaps)
Implementing the Map interface with Hash Tables
Imagine you want to look up your neighbors’ names, based on their house number.

- House numbers: 2555 through 10567 (roughly 4000 houses)
- Names: one last name per house
Array vs Tree

- You could store them in a balanced TreeMap of some kind
  - Log(n) to do get, put, delete
- Or you could store them in an array
  - Array is really fast lookup! O(1)
  - Just look in myarray[housenumber] to get the name
Hash Table is just a modified, more flexible array

- Keys don’t have to be integers 0-(size-1)
- (Ideally) avoids big gaps like our gap from 0 to 2555 in the house numbers

- Hash function is what at makes this all work:
Hash key collisions

- Hash function takes key and maps it to an integer
- Sometimes will map two DIFFERENT keys to the same integer
  - “Collision”
- We can NOT overwrite the value the way we would if it really were the same key
- Need a way of storing multiple values in a given “place” in the hash table
Closed Addressing with Linear Probing

- Where does “Annie” go if hashkey(“Annie”) = 3?
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. Other

<table>
<thead>
<tr>
<th>Array index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Closed Addressing with Linear Probing

Where does “Juan” go if hashkey(“Juan”) = 4?

A. 1
B. 2
C. 3
D. 4
E. Other

<table>
<thead>
<tr>
<th>Array index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Annie</td>
</tr>
<tr>
<td>4</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Closed Addressing with Linear Probing

- Where does “Julian” go if hashkey(“Julian”) = 3?

  A. 1
  B. 2
  C. 3
  D. 4
  E. Other

<table>
<thead>
<tr>
<th>Array index</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Annie</td>
</tr>
<tr>
<td>4</td>
<td>Juan</td>
</tr>
<tr>
<td>5</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
</tr>
</tbody>
</table>
Closed Addressing with Linear Probing

- Where does “Solange” go if hashkey(“Solange”) = 5?
  
  A. 3  
  B. 4  
  C. 5  
  D. 6  
  E. Other