Describing and Interpreting Strings: Regular Expressions

March 21–22, 2011

CS 60: Principles of Computer Science

Can you parse these "garden path" sentences?

The horse raced past the barn fell.

The young horse around.

The woman that whistles tunes pianos.

The man who hunts ducks out on weekends.

Those who admire a man that paints like Monet.

Sets. What about union? Intersection?

Method Summary	
boolean	**add**(E e) Adds the specified element to this set if it is not already present (optional operation).
boolean	**addAll**(Collection<? extends E> c) Adds all of the elements in the specified collection to this set if they're not already present (optional operation).
void	**clear**() Removes all of the elements from this set (optional operation).
boolean	**contains**(Object o) Returns `true` if this set contains the specified element.
boolean	**containsAll**(Collection<?> c) Returns `true` if this set contains all of the elements of the specified collection.
boolean	**equals**(Object o) Compares the specified object with this set for equality.
int	**hashCode**() Returns the hash code value for this set.
boolean	**isEmpty**() Returns `true` if this set contains no elements.
Iterator<E>	**iterator**() Returns an iterator over the elements in this set.
boolean	**remove**(Object o) Removes the specified element from this set if it is present (optional operation).
boolean	**removeAll**(Collection<?> c) Removes from this set all of its elements that are contained in the specified collection (optional operation).
boolean	**retainAll**(Collection<?> c) Retains only the elements in this set that are contained in the specified collection (optional operation).
int	**size**() Returns the number of elements in this set (its cardinality).
Object[]	**toArray**() Returns an array containing all of the elements in this set.

Hash Tables vs. Trees

Classes `HashSet<T>` *and* `TreeSet<T>` *both implement the interface* `Set<T>`. *How should we choose?*

`TreeSet` *uses self-balancing binary search trees:*

- ✓ *Worst-case* $O(\log n)$ *add, remove, contains*
- ✓ *Iteration gives you the elements in order*
- ✓ *Has extra functionality (e.g., find* x*-or-bigger)*

`HashSet` *uses hash tables*

- ✓ *"Expected Amortized Worst-case"* $O(1)$ *add, remove, contains*
- ✓ *Iteration goes bucket-by-bucket*
- ✓ *More opportunities for fiddly tuning (number of buckets, etc.)*

When Hash Tables Go Bad...

Huge portions of the Web vulnerable to hashing denial-of-service attack

By Jon Brodkin | Published 2 months ago

Researchers have shown how a flaw that is common to most popular Web programming languages can be used to launch denial-of-service attacks by exploiting hash tables. Announced publicly on Wednesday at the Chaos Communication Congress event in Germany, the flaw affects a long list of technologies, including PHP, ASP.NET, Java, Python, Ruby, Apache Tomcat, Apache Geronimo, Jetty, and Glassfish, as well as Google's open source JavaScript engine V8. The vendors and developers behind these technologies are working to close the vulnerability, with Microsoft warning of "imminent public release of exploit code" for what is known as a hash collision attack.

"Hash tables are a commonly used data structure in most programming languages," they explained. "Web application servers or platforms commonly parse attacker-controlled POST form data into hash tables automatically, so that they can be accessed by application developers. If the language does not provide a randomized hash function or the application server does not recognize attacks using multi-collisions, an attacker can degenerate the hash table by sending lots of colliding keys. The algorithmic complexity of inserting n elements into the table then goes to O(n**2), making it possible to exhaust hours of CPU time using a single HTTP request."

http://arstechnica.com/business/news/2011/12/huge-portions-of-web-vulnerable-to-hashing-denial-of-service-attack.ars

Describing and Interpreting Strings: Regular Expressions

March 21–22, 2011

CS 60: Principles of Computer Science

Syntax vs. Semantics

- ✓ Syntax: Formation Rules
  - *Colorless green ideas sleep* furiously.
  - `(+ 3 true)`
- ✓ Semantics: Meaning

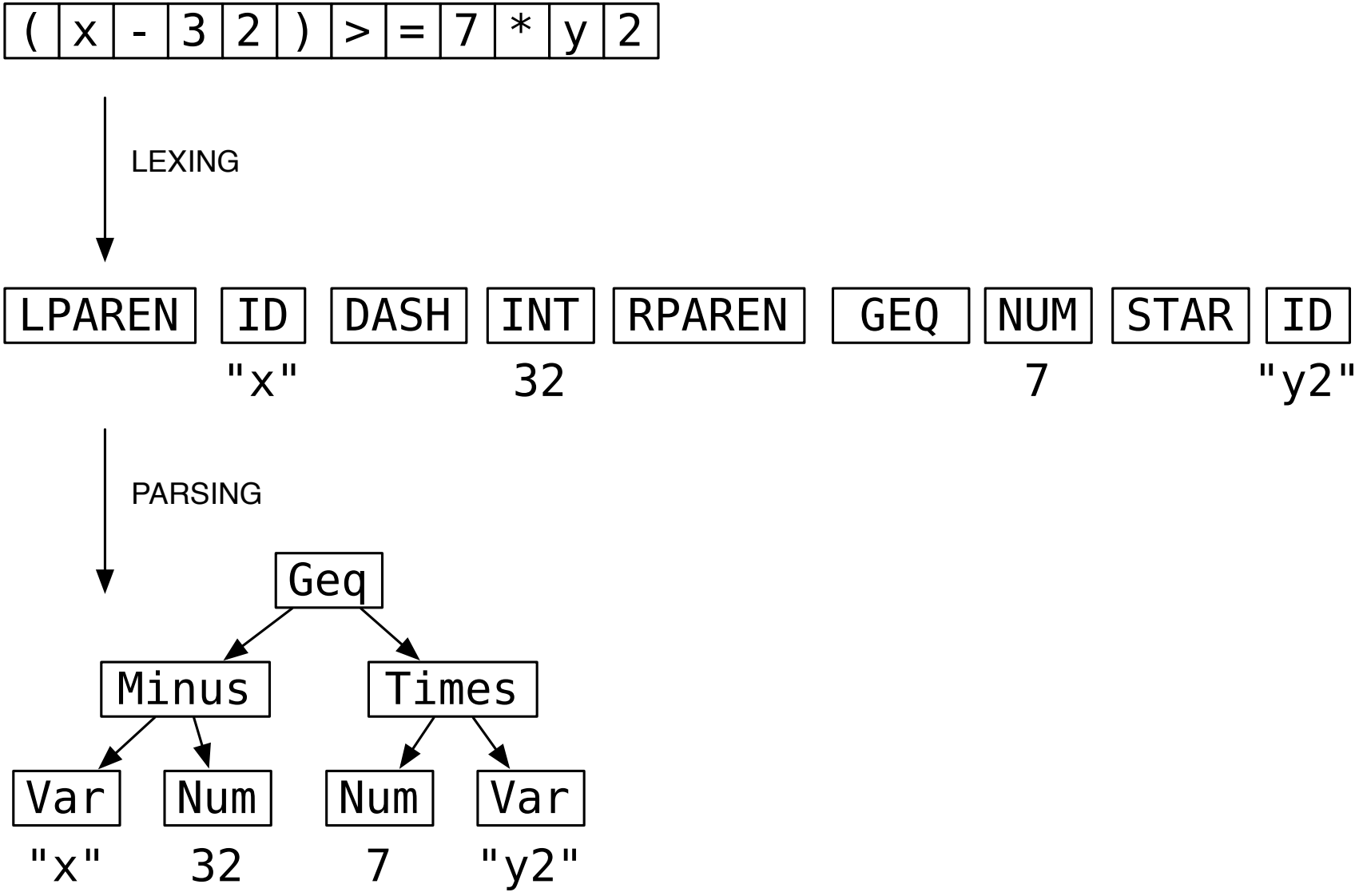
"Understanding" Strings

Strings (and computer files!) are just linear sequences of characters.

```
(x-32>=7*y2

)x-32(>=7*y2

(x-32>=7*y)2

(x-32)>=7*y2
```

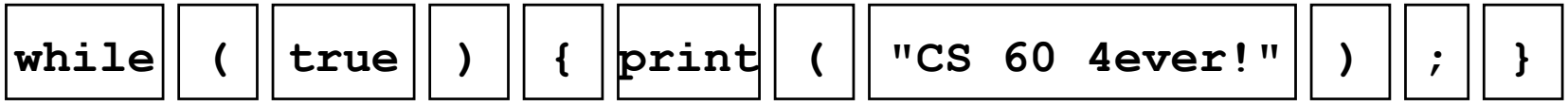
How can we figure out whether an input makes sense?
If it does, what are we supposed to do with it?

Traditional Lexing and Parsing



Lexing: Java vs. Python

```
while ( true )
{
   print( "CS 60 4ever!" );
}
```



```
while true:
   print "CS 60 4ever!"
```

Thinking Like Java

```
public static void main(String[] arg)
{
   int x = 32;
   int y =   9;
   int z;

   z = x+++y; // Legal? Meaning?

   System.out.println("x is "  +  x);
   System.out.println("y is "  +  y);
   System.out.println("z is "  +  z);
}
```

Thinking Like Java

```
public static void main(String[] arg)
{
   int x = 32;
   int y =   9;
   int z;

   z = x++++y; // Legal? Meaning?

   System.out.println("x is "  +  x);
   System.out.println("y is "  +  y);
   System.out.println("z is "  +  z);
}
```

Thinking Like Java

```
public static void main(String[] arg)
{
   int x = 32;
   int y =   9;
   int z;

   z = x++-+y; // This is OK!

   System.out.println("x is "  +  x);
   System.out.println("y is "  +  y);
   System.out.println("z is "  +  z);
}
```

This is too!

```
public static void main(String[] arg)
{
   int x = 32;
   int y =   9;
   int z;

   z = x++-++y;

   System.out.println("x is "  +  x);
   System.out.println("y is "  +  y);
   System.out.println("z is "  +  z);
}
```

Why Stop There?

```
public static void main(String[] arg)
{
   int x = 32;
   int y =   9;
   int z;

   z = x++-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-++y;

   System.out.println("x is "  +  x);
   System.out.println("y is "  +  y);
   System.out.println("z is "  +  z);
}
```

Describing Tokens

- ✓ - is a token in Java.
- ✓ -- is a token in Java.
- ✓ An decimal integer constant is a token in Java
- ✓ A variable is a token in Java

Can a human understand these?
Can a computer understand these?

Describing Tokens

- ✓ - is a token in Java.
- ✓ -- is a token in Java.
- ✓ Any digit, followed immediately by zero or more digits, is a token in Java.
- ✓ A letter, dollar sign, or underscore, followed immediately by zero or more letters, dollar signs, underscores, and digits, is a token in Java.

Can a human understand these?
Can a computer understand these?

Regular Expression Ingredients

Regular expressions are a formal way of describing simple patterns.
A regular expression can be:

- ✓ The empty string (sometimes written ϵ or λ)
- ✓ A single character (e.g., $\mathtt{a}$ or $\mathtt{0}$)
- ✓ Concatenation: $r_1\, r_2$
- ✓ Alternative: $r_1 \mid r_2$
- ✓ Repetition ("Kleene Star"): r_1^*.

In practice we use lots of abbreviations, e.g.,

$$\begin{array}{lcl} \texttt{[a-e]} & := & \texttt{(a|b|c|d|e)} \\ r? & := & r \mid \epsilon \\ r^+ & := & r\, r^* \end{array}$$

Describing Tokens with Regular Expressions

- ✓ `-` is a token in Java.
- ✓ `--` is a token in Java.
- ✓ `[0-9]`$^+$ is a token in Java.
- ✓ `[A-Za-z_$][A-Za-z_0-9]`* is a token in Java

Can a human understand these?
Can a computer understand these?

Application: Regular Expressions and Search

Unix's **egrep** command does line-by-line search for text matching a regular expression.

```
egrep   'hh'        /usr/share/dict/words
egrep   'y.*y'      /usr/share/dict/words
egrep   '(xq|hq)'   /usr/share/dict/words
egrep   '^y.*y$'    /usr/share/dict/words
```

(You can try these in a **Terminal** window on a Mac, or at any command-line on a Linux machine.)