Lexing

CS 132: Compiler Design

January 24, 2011

Definitions

- ✓ What is lexing (a.k.a. tokenizing)?

Definitions

- ✓ What is lexing (a.k.a. tokenizing)?
- ✓ What is parsing?

Definitions

- ✓ What is lexing (a.k.a. tokenizing)?
- ✓ What is parsing?
- ✓ What is abstract syntax?

Concrete Syntaxes

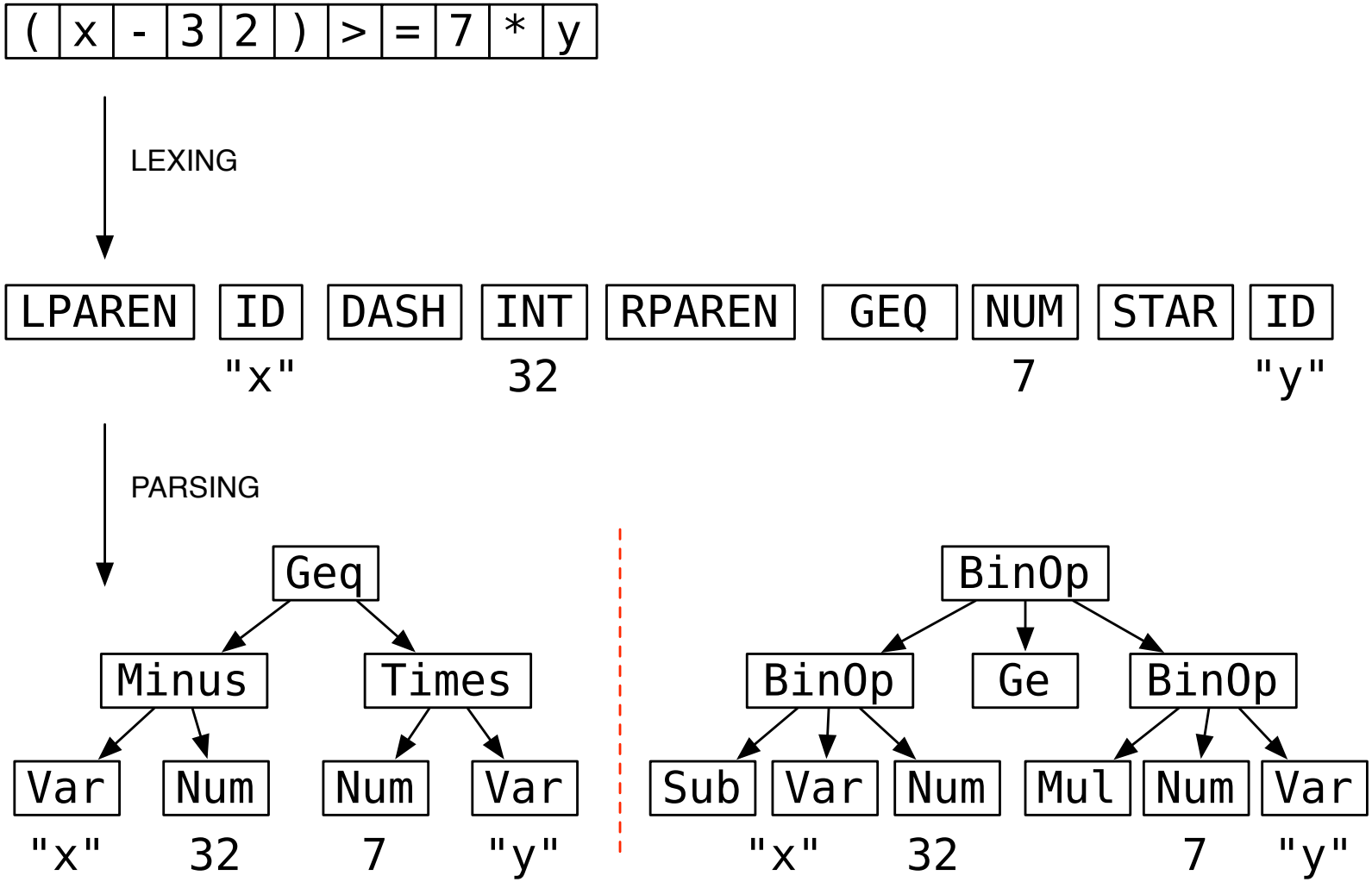
Pascal vs. C vs. Python

```
if space_left <> 0 then
  begin
    processLine(line);
    getNextLine
  end
else
  begin
    getNextLine;
    skipped := 1
  end
```

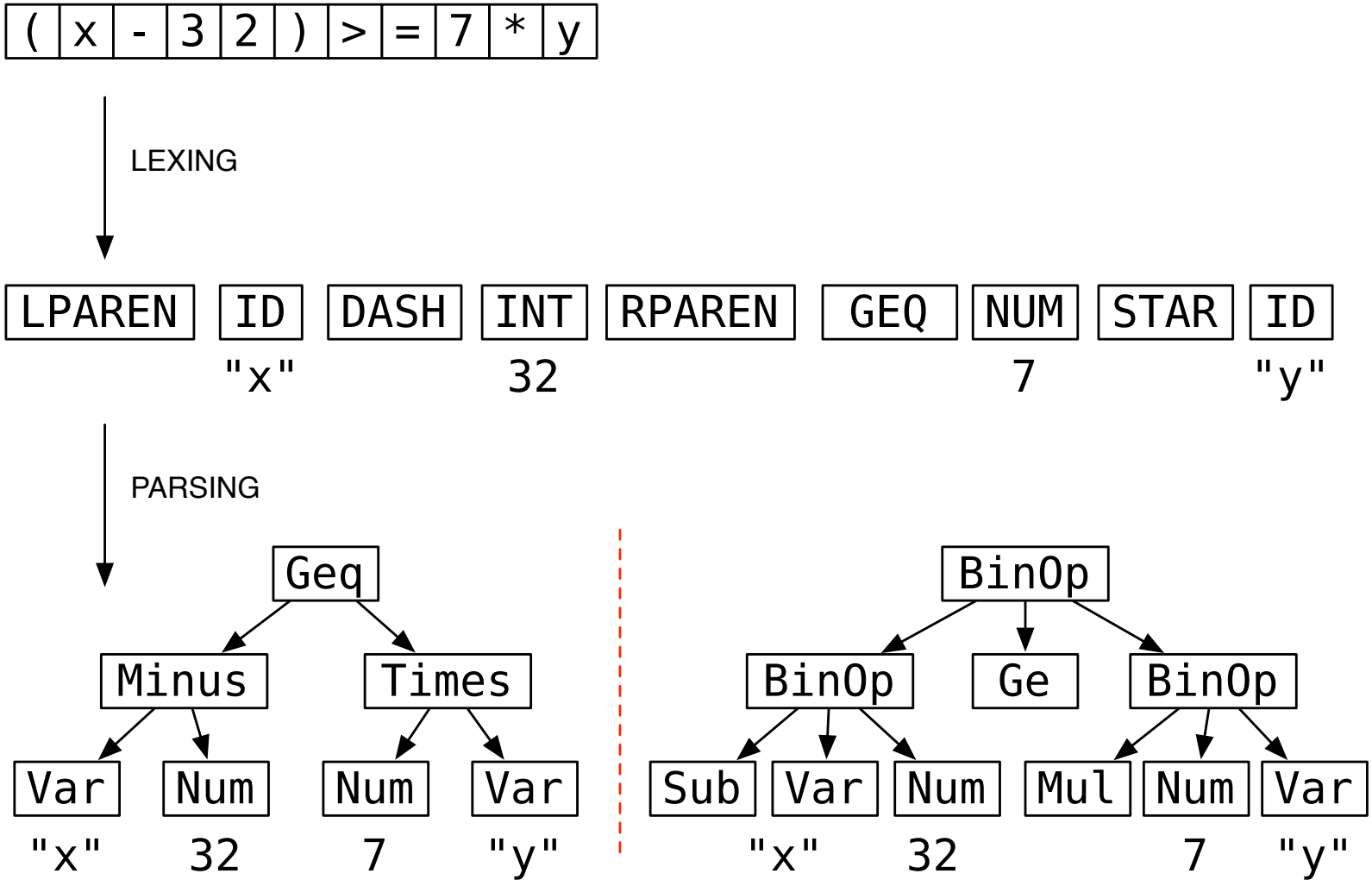
```
if (space_left != 0)
  {
    processLine(line);
    getNextLine();
  }
else
  {
    getNextLine();
    skipped = 1;
  }
```

```
if (space_left != 0):
   processLine(line)
   getNextLine()
else:
   getNextLine()
   skipped = 1
```

Traditional Parsing



Traditional Parsing



Question: Why bother with this two-step process?

Regular Expression Review

- ✓ What are the usual "primitive" regular expressions?

Regular Expression Review

- ✓ What are the usual "primitive" regular expressions?

- ✓ What other regular expressions are you familiar with?

```
M[ou]'?am+[ae]r .*([AEae]l[- ])?[GKQ] h?[aeu]+([dtz][dhz]?)+af[iy]
```

```
M[ou]'?am+[ae]r .*([AEae]l[- ])?[GKQ] h?[aeu]+([dtz][dhz]?)+af[iy]
```

```
Muammar Qaddafi
Mo'ammar Gadhafi
Muammar Kaddafi
Muammar Qadhafi
Moammar El Kadhafi
Muammar Gadafi
Mu'ammar al-Qadafi
Moamer El Kazzafi
Moamar al-Gaddafi
Mu'ammar Al Qathafi
Muammar Al Qathafi
Mo'ammar el-Gadhafi
Moamar El Kadhafi
Muammar al-Qadhafi
Mu'ammar al-Qadhdhafi
Mu'ammar Qadafi
Moamar Gaddafi
Mu'ammar Qadhdhafi
Muammar al-Khaddafi
Mu'amar al-Kadafi
Muammar Ghaddafy
Muammar Ghadafi
Muammar Ghaddafi
Muamar Kaddafi
Muammar Quathafi
Muammar Gheddafi
Muamar Al-Kaddafi
Moammar Khadafy
Moammar Qudhafi
Mu'ammar al-Qaddafi
Mu'ammar Muhammad Abu Minyar al-Qadhafi
```

From the RX library

Applying Regular Expressions

Give a regular expression for Ada identifiers, which

- ✓ Can contain letters, digits and underscores
- ✓ Begin with a letter
- ✓ Have no consecutive underscores or an underscore at the end

Give a regular expression for dollar amounts on US checks. (Note: many printed checks contain extra asterisks to prevent fraud, i.e., `$****1.00`.)

Finite Automata

Draw NFAs and DFAs for the following regular expressions:

- ✓ If keyword: `if`
- ✓ Case-insensitive if keyword: `[Ii][Ff]`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

Practical issues

- ✓ Keywords: `if` `int` `end`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

- ✓ Should `intend` be one token or two?

Practical issues

- ✓ Keywords: `if` `int` `end`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

- ✓ Should `intend` be one token or two?
- ✓ Is `if` a keyword or an identifier?

Practical issues

- ✓ Keywords: `if` `int` `end`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

- ✓ Should `intend` be one token or two?
- ✓ Is `if` a keyword or an identifier?
- ✓ How many tokens are there in `3-2` ?

Practical issues

- ✓ Keywords: `if` `int` `end`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

- ✓ Should `intend` be one token or two?
- ✓ Is `if` a keyword or an identifier?
- ✓ How many tokens are there in `3-2` ?
- ✓ How many tokens are there in `3 + -2` ?

Practical issues

- ✓ Keywords: `if` `int` `end`
- ✓ Identifiers: `[a-zA-z_][a-zA-Z_0-9]*`

- ✓ Should `intend` be one token or two?
- ✓ Is `if` a keyword or an identifier?
- ✓ How many tokens are there in `3-2` ?
- ✓ How many tokens are there in `3 + -2` ?
- ✓ What about `3-2147483648` ?

Note on Reserved Words

Many languages have reserved words (*keywords*) that cannot be used as identifiers (`if`, `return`, *etc.*)

Note on Reserved Words

Many languages have reserved words (*keywords*) that cannot be used as identifiers (`if`, `return`, *etc.*)

On the other hand, there are languages like PL/I:

```
IF THEN THEN THEN = ELSE; ELSE ELSE = THEN;
```

Building a Lexer

Pascal included tokens of the following form:

- ✓ Integers: `[0-9]+`
- ✓ Real numbers: `[0-9]+"."[0-9]+`
- ✓ Range marker: `".."`

How do we combine them into a lexer?

Building a Lexer

Pascal included tokens of the following form:

- ✓ Integers: `[0-9]+`
- ✓ Real numbers: `[0-9]+"."[0-9]+`
- ✓ Range marker: `".."`

How do we combine them into a lexer?

What is the worst-case running time for a DFA-based lexer?

Lexing Challenges: Fortran

- ✓ Pre-1977 Hollerith Constants

```
15HCOMPILER DESIGN
```

Lexing Challenges: Fortran

- ✓ Pre-1977 Hollerith Constants

```
15HCOMPILER DESIGN
```

- ✓ Fixed-Column Layouts

```
CALL DOIT(X,Y,Z)
      CALL DOIT(X,Y,Z)
```

Lexing Challenges: Fortran

- ✓ Pre-1977 Hollerith Constants

```
15HCOMPILER DESIGN
```

- ✓ Fixed-Column Layouts

```
CALL DOIT(X,Y,Z)
      CALL DOIT(X,Y,Z)
```

- ✓ Whitespace is *not* significant.

```
DO 10 I = 1,15
DO 10 I = 1.15
REAL X
REAL X = 3.5
INTEGER FUNCTION A(I)
```

Parsing Challenges: C, C++, C#, Java, ...

- ✓ Often can be resolved by extra information from the lexer.

```
X * Y;
X (Y);
X < Y > Z;
X ? Y : Z;
X ? Y = Z;
(T)-(U);
```

Parsing Challenges: C, C++, C#, Java, ...

- ✓ Often can be resolved by extra information from the lexer.

```
X * Y;
X (Y);
X < Y > Z;
X ? Y : Z;
X ? Y = Z;
(T)-(U);
```

- ✓ Not all challenges are this easy, though

```
int(x), y, *const z;   // variable declarations
int(x), y, new int;    // list of expressions
int(x), y, z = 0;      // variable declarations
```

Implementation

Once we have a DFA, how do we turn it into code?