Announcements

• First quiz has been posted on sakai. You have two tries. Last one counts. Quiz is 50 minutes long. Open book, open notes. Deadline: 02/19 @ 11:55pm

• Second homework deadline extension: Thursday, 02/18

• First project due next Tuesday (02/23) for code, and Friday (02/26) for report.
The purpose of the front end is to deal with the input language

- Perform a membership test: \( \text{code} \in \text{source language} \)?
- Is the program well-formed (semantically)?
- Build an IR version of the code for the rest of the compiler

*The front end is not monolithic*
The Front End

Scanner

- Maps stream of characters into words/tokens
  - Basic unit of syntax
  - $x = x + y$; becomes
    $$<\text{id}, x> <\text{eq},=><\text{id}, x> <\text{pl},+><\text{id}, y> <\text{sc},;>$$
- Characters that form a word/token are its **lexeme**
- Its **part of speech** (or **syntactic category**) is called its **token type**
- Scanner discards white space & (often) comments

Source code → Scanner → tokens → Parser → IR

Errors

Speed is an issue in scanning
⇒ use a specialized recognizer
Parser

• Checks stream of classified words (*tokens*) for grammatical correctness
• Determines if code is syntactically well-formed
• Guides checking at deeper levels than syntax (static semantics)
• Builds an IR representation of the code

*We’ll get to parsing in the next lectures*
• Language syntax is specified over *parts of speech* (tokens)
• Syntax checking matches *sequence of tokens* against a grammar
• Here is an example context free grammar (CFG) $G$:

\[
\begin{align*}
1. \quad goal & \rightarrow expr \\
2. \quad expr & \rightarrow expr \ op \ term \\
3. \quad \text{term} & \rightarrow \text{term} \\
4. \quad term & \rightarrow \text{number} \\
5. \quad \text{id} & \\
6. \quad op & \rightarrow + \\
7. \quad \text{=} &
\end{align*}
\]

$G$ in BNF form

$S = goal$

$T = \{ \text{number, id, +, -} \}$

$N = \{ goal, expr, term, op \}$

$P = \{ 1, 2, 3, 4, 5, 6, 7 \}$

$G = (S, T, N, P)$
Why study lexical analysis?

- We want to avoid writing scanners by hand

Goals:

- To simplify specification & implementation of scanners
- To understand the underlying techniques and technologies
Why study lexical analysis?

- **We want to avoid writing scanners by hand**

**Goals:**
- To simplify specification & implementation of scanners
- To understand the underlying techniques and technologies
Why study lexical analysis?

- We want to avoid writing scanners by hand

Goals:

→ To simplify specification & implementation of scanners
→ To understand the underlying techniques and technologies
Lexical patterns form a **regular language**

***any finite language is regular  ***

Regular expressions (REs) describe regular languages

Regular Expression (over an alphabet $\Sigma$, a finite set of symbols):

- $\epsilon$ is a RE denoting the set $\{\epsilon\}$
- If “a” is in $\Sigma$, then $a$ is a RE denoting $\{a\}$
- If $x$ and $y$ are REs denoting $L(x)$ and $L(y)$ then
  - $x \mid y$ is an RE denoting $L(x) \cup L(y)$
  - $xy$ is an RE denoting $L(x)L(y)$
  - $x^*$ is an RE denoting $L(x)^*$
  - $(x)$ is an RE denoting $L(x)$

Ever type “rm *.o a.out”?
### Set Operations

<table>
<thead>
<tr>
<th>Operation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Union of $L$ and $M$</td>
<td>$L \cup M = {s \mid s \in L \text{ or } s \in M}$</td>
</tr>
<tr>
<td>Written $L \cup M$</td>
<td></td>
</tr>
<tr>
<td>Concatenation of $L$ and $M$</td>
<td>$LM = {st \mid s \in L \text{ and } t \in M}$</td>
</tr>
<tr>
<td>Written $LM$</td>
<td></td>
</tr>
<tr>
<td>Kleene closure of $L$</td>
<td>$L^* = \bigcup_{0 \leq i \leq \infty} L^i$</td>
</tr>
<tr>
<td>Written $L^*$</td>
<td></td>
</tr>
<tr>
<td>Positive Closure of $L$</td>
<td>$L^+ = \bigcup_{1 \leq i \leq \infty} L^i$</td>
</tr>
<tr>
<td>Written $L^+$</td>
<td></td>
</tr>
</tbody>
</table>

These definitions should be well known
Examples of Regular Expressions

Identifiers:

\[
\begin{align*}
\text{Letter} & \rightarrow (a|b|c| \ldots |z|A|B|C| \ldots |Z) \\
\text{Digit} & \rightarrow (0|1|2| \ldots |9)
\end{align*}
\]

\[
\text{Identifier} \rightarrow \text{Letter} \ (\text{Letter} \mid \text{Digit})^*
\]

Numbers:

\[
\begin{align*}
\text{Integer} & \rightarrow (+|-|\varepsilon) (0| (1|2|3| \ldots |9)(\text{Digit}^*)) \\
\text{Decimal} & \rightarrow \text{Integer} \cdot \text{Digit}^*
\end{align*}
\]

\[
\begin{align*}
\text{Real} & \rightarrow (\text{Integer} \mid \text{Decimal}) \ E \ (\pm|\varepsilon) \ \text{Digit}^* \\
\text{Complex} & \rightarrow (\text{Real} \mid \text{Real})
\end{align*}
\]

Numbers can get much more complicated!
Regular expressions can be used to specify the words to be translated to parts of speech (tokens) by a lexical analyzer.

Using results from automata theory and theory of algorithms, we can automatically build recognizers from regular expressions.

→ We study REs and associated theory to automate scanner construction!
More Lexical Analysis

Read EaC: Chapters 2.1 – 2.5;