Lexical Analysis - An Introduction
The Front End

The front end is not monolithic
The Front End

Scanner

• Maps stream of characters into words
  → Basic unit of syntax
  → $x = x + y;$ becomes set of tokens $\langle \text{type, lexeme} \rangle$
    $\langle \text{id,x} \rangle \langle \text{eq,=} \rangle \langle \text{id,x} \rangle \langle \text{pl,}+\rangle \langle \text{id,y} \rangle \langle \text{sc,;} \rangle$
Where is Lexical Analysis Used?

For traditional languages but where else...

- Web page “compilation”
  - Lexical Analysis of HTML, XML, etc.
- Natural Language Processing
- Game Scripting Engines
- OS Shell Command Line
- GREP
- Prototyping high-level languages
- JavaScript, Perl, Python
The Big Picture

Why study lexical analysis?
• We want to avoid writing scanners by hand
• We want to harness the theory from classes like CISC 303

Goals:
→ To simplify specification & implementation of scanners
→ To understand the underlying techniques and technologies

Scanner Generator

Scanner

source code

parts of speech & words

specifications

Regular Expressions

tables or code
Regular Expressions

Powerful notation to specify lexical rules

- Simplifies scanner construction
- Notation describes set of strings over some alphabet
- Entire set of strings called the language
- If r is an RE, L(r) is the language it specifies
Regular Expressions (more formally)

- Over some alphabet $\Sigma$
- $\varepsilon$ is a RE denoting the empty set
- If $a$ is in $\Sigma$, then $a$ is a RE denoting $\{a\}$
Regular Expressions (more formally)

Given sets \( R \) and \( S \)

- **Closure**: \( R^* \) is an RE denoting
  \[
  \bigcup_{0 \leq i \leq \infty} R^i
  \]

- **Concatenation**: \( RS \) is an RE denoting
  \[
  \{st \mid s \in R \text{ and } t \in S\}
  \]

- **Alternation**: \( R \mid S \) is an RE denoting
  \[
  \{s \mid s \in R \text{ or } s \in S\}
  \]
  - Often written \( R \cup S \)

Note: Precedence is closure, then concatenation, then alternation
Examples of Regular Expressions

**Identifiers:**

- **Letter** → (a|b|c| ... |z|A|B|C| ... |Z)
- **Digit** → (0|1|2| ... |9)
- **Identifier** → Letter ( Letter | Digit )*  

**Numbers:**

- **Integer** → (+|-|ε) (0| (1|2|3| ... |9)(Digit *)) )
- **Decimal** → Integer . Digit *
- **Real** → ( Integer | Decimal ) E (+|-|ε) Digit *
- **Complex** → ( Real , Real )

*Numbers can get much more complicated!*
Regular Expressions  (the point)

REs are used to specify the words to be translated to parts of speech by a lexical analyzer.

Using results from automata theory and theory of algorithms, we can automatically build recognizers (i.e. scanners) from regular expressions.

You may have seen this construction in a Automata Course.

⇒ We study REs and associated theory to automate scanner construction!
Regular Expression Class Problem?

What is the regular expression for a register name?

Examples:  $r1, \ r25, \ r999$  $\leftrightarrow$ These are OK.

$r, \ s1, \ a25$  $\leftrightarrow$ These are **not** OK.
Consider the problem of recognizing register names

\[ \text{Register} \rightarrow r \ (0|1|2| \ldots \ | 9) \ (0|1|2| \ldots \ | 9)^* \]

- Allows registers of arbitrary number
- Requires at least one digit
Finite Automaton (FA)

- An abstract machine that corresponds to a particular RE
- Recognizers can scan a stream of symbols to find words

Transition Diagram for Number
Finite Automaton (FA)

An FA is a five-tuple \((S, \Sigma, \delta, s_0, S_F)\) where

- \(S\) is the set of states
- \(\Sigma\) is the alphabet
- \(\delta\) a set of transition functions
  - takes a state and a character and returns new state
- \(s_0\) is the start state
- \(S_F\) is the set of final states
Finite Automaton (FA)

Transition Diagram for Number

S₀ (0|1|2| ... 9)

S₁ 0

S₂ (0|1|2| ... 9)

accepting states