Course Administration

- Exercise 2 posted
- Project 1 Preliminary Exercise
// Accessors and Mutators
unsigned getDollars () const;
unsigned getCents () const;
void setDollars (unsigned D);
void setCents (unsigned C);
unsigned * getCurrency () const;
void setCurrency (unsigned * C) const;
unsigned & Dollars ();
unsigned & Cents ();
class money
{
    public:
        // Methods discussed on Monday

        // Accessors and Mutators
        int getDollars() const;
        int getCents() const;
        vector<unsigned> getCurrency() const;
        void setCurrency(vector<unsigned> & C);

    private:
        // Add attributes private member functions here.
        unsigned size;  // required
        unsigned * currency;  // required
};
I noticed that anything input which matches to the LISTOP_T category would also match for the IDKEY_T category. Can we assume that the order the regular expressions are listed are also a "precedence" order, so that it first checks if "car" matches the LISTOP_T category?
If we did go that route, all of the "intermediate" states in the DFA for the LISTOP_T regular expression would have to be accepting states for the IDKEY_T regular expression because "ca" matches IDKEY_T. Does this sound correct? Most of what we did in class only had 1 accepting state per category and while I know it is valid to have multiple, I just wanted to confirm that.
I wanted to confirm that DFAs shouldn't have any lambda transitions in them correct? as that wouldn't be "deterministic"?
DFAs as scanners (aka tokenizers)

- Alphabet = \{a, b, c, x, y, z, \text{\texttt{\textasciitilde}}\}
- Regular expression 1 (RE1)
  - \(a^* (ab | bc) a^+\)
- Regular expression 2 (RE2)
  - \(x^+ (xy | yz | xz) z^*\)
- Combined
  - \((a^* (ab | bc) a^+) | (x^+ (xy | yz | xz) z^*)\)
(a* (ab | bc) a+) | (x+ (xy | yz | xz) z*)
Programming a DFA

Table

<table>
<thead>
<tr>
<th>ws</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>Err1</td>
<td>7</td>
<td>Err1</td>
<td>Err1</td>
</tr>
<tr>
<td>2</td>
<td>Err2</td>
<td>2</td>
<td>4</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
</tr>
<tr>
<td>3</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>5</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
</tr>
<tr>
<td>4</td>
<td>Err2</td>
<td>6</td>
<td>Err2</td>
<td>5</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
</tr>
<tr>
<td>5</td>
<td>Err2</td>
<td>6</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
</tr>
<tr>
<td>6</td>
<td>RE1</td>
<td>6</td>
<td>RE1</td>
<td>RE1</td>
<td>RE1</td>
<td>RE1</td>
<td>RE1</td>
</tr>
<tr>
<td>7</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>8</td>
<td>9</td>
<td>Err2</td>
<td>Err2</td>
</tr>
<tr>
<td>8</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>8</td>
<td>10</td>
<td>10</td>
<td>Err2</td>
</tr>
<tr>
<td>9</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>Err2</td>
<td>11</td>
</tr>
<tr>
<td>10</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>RE2</td>
<td>11</td>
</tr>
</tbody>
</table>
Regular Expression for Numeric Literals

- Regular expression for general class of numeric literals signed/unsigned and integer/real
- Alphabet = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -, +, .\}
- Regular Expression

- How do you recognize the end of a numeric literal?
DFA for Numeric Literals
(+|-|λ)(0|1|2|3|4|5|6|7|8|9)+(. ( 0|1|2|3|4|5|6|7|8|9)+| λ)
DFA for Numeric Literals

\((+|-|\lambda)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+|\lambda)\)

a. 12
   a. 1 -> 3 -> 3 OK!

b. 1.2
   a. 1 -> 3 -> 4 -> 5 OK!

c. +12.34
   a. 1 -> 2 -> 3 -> 3 -> 4 -> 5 -> 5 OK!

d. 12.
   a. 1 -> 3 -> 3 -> 4 -> ends No!

e. .123
   a. 1 -> ends No!

f. 12.12.34
   a. 1 -> 3 -> 3 -> 4 -> 5 -> 5 see . error
DFA for Numeric Literals – with terminating states
(\+-\lambda)(0|1|2|3|4|5|6|7|8|9)+(.(0|1|2|3|4|5|6|7|8|9)+|\lambda)
DFA for Numeric Literals – with terminating states
(+|-|λ)(0|1|2|3|4|5|6|7|8|9)+.( 0|1|2|3|4|5|6|7|8|9)+| λ)

- _ → represents a space
- 12 (1 -> 3 -> 3 OK!)
- 1.2 (1-> 3 -> 4 -> 5 OK!)
- +12.34 (1 -> 2 -> 3-> 3 -> 4-> 5 -> 5)
- 12. (ends at 4)
- .123 (ends at 1)
- 12.12.34 (stops at 5 OK)
  - 12.12
- abcd (ends at -3)
- +abc (ends at -3)
- + (ends at -3)
- 4a (ends at -1)
  - 4
- 425 (1 -> 3 -> 3 -> 3 -> ends at -1)
  - 4
- -12.345 (1 -> 2 -> 3 -> 3 -> 4 -> 5 -> 5 -> 5 -> ends at -2)
  - -12.345

- What ends up at -1? integer
- What ends up at -2? double
- What ends up at -3? Non-numeric
How do the numeric literals for Project 1 differ from this example?
Next steps