|  |  |
| --- | --- |
| **CIS 675**  **Homework 3** | **Name:\_\_Nathan Hull\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** |

Write a parser for dot. You can find the grammar at <http://www.graphviz.org/content/dot-language> (their website can be unreliable, so here’s a [local copy](https://www.evernote.com/l/AECJykWlJhdN8J7dN4Ic1ZxF_5Rpe4nR9zw)).

As input, your parser should accept the following input (this is the same as the lexer input for homework 2). At this point in time, your parser should simply indicate if there is a syntax error or not. Your parser should indicate which lines contains errors (if any).

digraph G {

main [shape=box]; /\* this is a comment \*/

main -> parse [weight=8];

parse -> execute;

main -> init [style=dotted];

main -> cleanup;

execute -> make\_string;

init -> make\_string;

main -> printf [style=bold,label="100 times"];

make\_string [label="make a\nstring"];

node [shape=box,style=filled,color=".7 .3 1.0"];

execute -> compare;

}

Turn in the source code for your parser (any language you choose), and a screenshot showing you executing your parser on the input file above.

Then show your parser running on the following input. This file contains three syntax errors as indicated. You may run your parser three times (fixing the errors between each run) in order for all three errors to be caught and displayed.

digraph G {

main [shape=box; /\* missing the closing ] \*/

main -> parse [weight=8];

parse -> execute;

main -> [style=dotted]; /\* missing the target node ID \*/

main -> cleanup;

execute -> make\_string;

init -> make\_string;

main -> printf [style=bold, label=]; /\* missing value \*/

make\_string [label="make a\nstring"];

node [shape=box,style=filled,color=".7 .3 1.0"];

execute -> compare;

}