Brian Cullinan

Project 3: Parser

09/24/08

Before I started this project I needed to make some large changes to my scanner class. My approach to the problem was simple, I use regular expressions to check the content of the buffer (or soon to be token) and when it stops matching any type, I assume that everything in the buffer before must be a valid token. So now it will send a parsing error on a much wider range of possibilities. For example now any character that is not explicitly recognized such as a | (pipe) will throw a parsing error.

My node hierarchy has just as much abstraction as I need. All the literals are handled by 1 class; however there is an assignment node that can be called from multiple places, as shown in my EBNF. Some changes I made to the instructions include, no need for a parse\_command functions. The program node is responsible for looping through a number of commands. toString commands can be performed at any time after the parse function is called.

My parser class is just simply something that initializes the AST creation. You can see the results of the tree at:

<http://209.250.30.30/compiler/>

Scanner.php (Just because it changed a lot)

<?php

require\_once 'reader.php';

require\_once 'token.php';

require\_once 'error.php';

class Scanner

{

var $file;

var $tokens = array();

function Scanner($file)

{

$this->file = $file;

$reader = new SourceReader($this->file);

$buffer = '';

$start = 1; # starting column

while(true)

{

$int = $reader->peek(1);

// these are not regular expressions for reading in tokens, they are just used to verify token content

if($int == -1)

{

break;

}

elseif(preg\_match('/^[a-zA-Z][a-zA-Z0-9]\*$/', $buffer . chr($int)) != 0)

{

// always assume it is an identifier until proven otherwise

$type = T\_IDENTIFIER;

// check if it is a keyword

if(preg\_match('/^(v|c|p)$/', $buffer . chr($int)) != 0)

{

$type = T\_KEYWORD;

}

$buffer .= chr($reader->read());

}

// check if it is an operator

elseif(preg\_match('/^(\+|-|\\*|\/|\^|=)$/', $buffer . chr($int)) != 0)

{

$type = T\_OPERATOR;

$buffer .= chr($reader->read());

}

// check if it is a literal

elseif(preg\_match('/^([0-9]\*\.?[0-9]\*|\"[^\"]\*\"?)$/', $buffer . chr($int)) != 0)

{

$type = T\_LITERAL;

$buffer .= chr($reader->read());

}

// check if it is some kind of white space delimiter

elseif(preg\_match('/^( \*|\(|\))$/', $buffer . chr($int)) != 0)

{

$type = T\_DELIM;

$buffer .= chr($reader->read());

}

elseif(preg\_match('/^(\s\*|\w\*|\r\*|\n\*)$/', $buffer . chr($int)) != 0)

{

$type = T\_EOL;

$buffer .= chr($reader->read());

}

// check for comments

elseif(preg\_match('/^#.\*[\r|\n]?$/', $buffer . chr($int)) != 0)

{

$type = T\_COMMENT;

$buffer .= chr($reader->read());

}

// clear out buffer then set it to new character

else

{

if($buffer != '')

{

if($type != T\_COMMENT && $type != T\_DELIM)

{

// create token

$this->tokens[] = new Token($this->file, $reader->line, $start, $type, $buffer);

}

unset($type);

$buffer = '';

$start = $reader->col;

}

// must be an invalid character

else

{

// create error

$error = new CompileError($this->file, $reader->line, $reader->col, 'Invalid Character: ' . chr($int));

$error->error\_query();

}

}

}

}

}

?>

Parser.php

<?php

require\_once 'scanner.php';

require\_once 'nodes.php';

class Parser

{

var $file;

var $scanner;

var $tree;

function Parser($file)

{

$this->scanner = new Scanner($file);

//print\_r($this->scanner->tokens);

}

function parse()

{

$ast = new AST();

$this->tree = $ast->parseAST($this->scanner->tokens);

return $this->tree;

}

}

?>

Nodes.php

<?php

class AST

{

var $tree = array();

function accept(&$list)

{

//var\_dump(current($list)->type);

next($list);

}

function parseAST(&$list)

{

reset($list);

$program = new Program();

$this->tree['AST'] = $program->parseProgram($list);

return $this;

}

function toString()

{

$output = '';

if(is\_object($this->tree))

{

$output = $this->tree->toString();

}

else

{

foreach($this->tree as $key => $object)

{

$output .= $key . '(' . (is\_object($object)?$object->toString():$object) . '),';

}

$output = substr($output, 0, strlen($output) - 1);

}

return $output;

}

function make\_error(&$list)

{

// create error

$error = new CompileError(current($list)->file, current($list)->line, current($list)->col, 'Invalid Token (' . current($list)->type . '): ' . current($list)->content);

$error->error\_query();

}

}

class Literal extends AST

{

function parseLiteral(&$list)

{

if(current($list)->type == T\_LITERAL)

{

if(is\_numeric(current($list)->content) && intval(current($list)->content) == current($list)->content)

{

$this->parseIntLiteral($list);

}

elseif(is\_numeric(current($list)->content))

{

$this->parseDoubleLiteral($list);

}

else

{

$this->parseStringLiteral($list);

}

$this->accept($list);

}

else

{

$this->make\_error(&$list);

}

return $this;

}

function parseIntLiteral(&$list)

{

$this->tree['Literal']['IntLiteral'] = current($list)->content;

}

function parseDoubleLiteral(&$list)

{

$this->tree['Literal']['DoubleLiteral'] = current($list)->content;

}

function parseStringLiteral(&$list)

{

$this->tree['Literal']['StringLiteral'] = current($list)->content;

}

function toString()

{

$output = 'Literal(';

//if(is\_object($this->tree))

// $this->tree->toString();

foreach($this->tree['Literal'] as $key => $object)

{

$output .= $key . '(' . (is\_object($object)?$object->toString():$object) . '),';

}

$output = substr($output, 0, strlen($output) - 1) . ')';

return $output;

}

}

class Identifier extends AST

{

function parseIdentifier(&$list)

{

if(current($list)->type == T\_IDENTIFIER)

{

$this->tree['Identifier'] = current($list)->content;

$this->accept($list);

}

else

{

$this->make\_error(&$list);

}

return $this;

}

}

class Expression extends AST

{

function parseExpression(&$list)

{

if(current($list)->type == T\_LITERAL)

{

$literal = new Literal();

$this->tree = $literal->parseLiteral($list);

}

elseif(current($list)->type == T\_IDENTIFIER)

{

$identifier = new Identifier();

$this->tree['VariableName'] = $identifier->parseIdentifier($list);

if(current($list)->type == T\_OPERATOR)

{

$this->tree['Operator'] = current($list)->content;

$this->accept($list);

$expression = new Expression();

$this->tree['Expression'] = $expression->parseExpression($list);

}

else

{

prev($list);

}

}

elseif(current($list)->type == T\_OPERATOR)

{

$this->tree['Operator'] = current($list)->content;

$this->accept($list);

$expression = new Expression();

$this->tree['Expression'] = $expression->parseExpression($list);

}

else

{

$expression = new Expression();

$this->tree['Expression1'] = $expression->parseExpression($list);

if(current($list)->type == T\_OPERATOR)

{

$this->tree['Operator'] = current($list)->content;

$this->accept($list);

}

else

{

$this->make\_error($list);

}

$expression = new Expression();

$this->tree['Expression2'] = $expression->parseExpression($list);

}

return $this;

}

}

class Assignment extends Command

{

function parseAssignment(&$list)

{

$identifier = new Identifier();

$this->tree['Assignment']['VariableName'] = $identifier->parseIdentifier($list);

if(current($list)->type == T\_OPERATOR && current($list)->content == '=')

{

$this->tree['Assignment']['Operator'] = current($list)->content;

$this->accept($list);

}

else

{

$this->make\_error($list);

}

$expression = new Expression();

$this->tree['Assignment']['Expression'] = $expression->parseExpression($list);

return $this;

}

function toString()

{

$output = 'Assignment(';

//if(is\_object($this->tree))

// $this->tree->toString();

foreach($this->tree['Assignment'] as $key => $object)

{

$output .= $key . '(' . (is\_object($object)?$object->toString():$object) . '),';

}

$output = substr($output, 0, strlen($output) - 1) . ')';

return $output;

}

}

class Keyword extends AST

{

function parseKeyword(&$list)

{

switch(current($list)->content)

{

case 'v':

case 'c':

$this->tree['Keyword'] = current($list)->content;

$this->accept($list);

$assignment = new Assignment();

$this->tree['Declaration'] = $assignment->parseAssignment($list);

break;

case 'p':

$this->tree['Keyword'] = current($list)->content;

$this->accept($list);

$identifier = new Identifier();

$this->tree['Print'] = $identifier->parseIdentifier($list);

break;

default:

$this->make\_error(&$list);

}

return $this;

}

}

class Command extends AST

{

function parseCommand(&$list)

{

switch(current($list)->type)

{

case T\_KEYWORD:

$keyword = new Keyword();

$this->tree['Command'] = $keyword->parseKeyword($list);

if(current($list)->type == T\_EOL) $this->accept($list);

break;

case T\_IDENTIFIER:

$assignment = new Assignment();

$this->tree['Command'] = $assignment->parseAssignment($list);

if(current($list)->type == T\_EOL) $this->accept($list);

break;

default:

$this->make\_error(&$list);

}

return $this;

}

}

class Program extends AST

{

function parseProgram(&$list)

{

while(current($list) !== false)

{

$command = new Command();

$this->tree['Program'][] = $command->parseCommand($list);

}

return $this;

}

function toString()

{

$output = 'Program(';

//if(is\_object($this->tree))

// $this->tree->toString();

foreach($this->tree['Program'] as $key => $object)

{

$output .= $object->toString() . ',';

}

$output = substr($output, 0, strlen($output) - 1) . ')';

return $output;

}

}

?>

Index.php (Driver)

<code>

<pre>

<?php

// include some needed classes

require\_once 'parser.php';

// create a reader object

$parser = new Parser('code.txt');

$tree = $parser->parse();

print\_r($tree);

print\_r($tree->toString());

?>

</pre>

</code>