NAME

Parsers::Lexer

SYNOPSIS

use Parsers::Lexer;

use Parsers::Lexer qw(:all);

DESCRIPTION

Lexer class provides the following methods:

new, GetLex, Lex, Next, Peek, StringifyLexer

The object oriented chained Lexer is implemented based on examples available in Higher-order Perl [Ref 126] book by Mark J. Dominus. It is designed to be used both in standalone mode or as a base class for YYLexer.

A chained lexer is created by generating a lexer for for the first specified token specification using specified input and chaining it with other lexers generated for all subsequent token specifications. The lexer generated for the first token specification uses input iterator to retrieve any available input text; the subsequent chained lexeres for rest of the token specifications use lexers generated for previous token specifications to get next input, which might be unmatched input text or a reference to an array containing token and matched text pair.

METHODS

new

```
$Lexer = new Parsers::Lexer($Input, @TokensSpec);
```

Using specified *Input* and *TokensSpec*, new method generates a new lexer and returns a reference to newly created Lexer object.

Example:

```
# Tokens specifications supplied by the caller. It's an array containing references
# to arrays with each containing TokenLabel and TokenMatchRegex pair along with
# an option reference to code to be executed after a matched.
#
@LexerTokensSpec = (
    [ 'LETTER', qr/[a-zA-Z]/ ],
    [ 'NUMBER', qr/\d+/ ],
    [ 'SPACE', qr/[ ]*/,
        sub { my($This, $TokenLabel, $MatchedText) = @_; return ''; }
    ],
    [ 'NEWLINE', qr/(?:\r\n|\r|\n)/,
        sub { my($This, $TokenLabel, $MatchedText) = @_;  return "\n"; }
    ],
    [ 'CHAR', qr/./ ]
);

# Input string...
$InputText = 'y = 3 + 4';
$Lexer = new Parsers::Lexer($InputText, @LexerTokensSpec);

# Process input stream...
while (defined($Token = $Lexer->Lex())) {
    print "Token: " . ((ref $Token) ? "@{$Token}" : "$Token") . "\n";
}

# Input file...
$InputFile = "Input.txt";
open INPUTFILE, "$InputFile" or die "Couldn't open $InputFile: $!\n";
$Lexer = new Parsers::Lexer(\*INPUTFILE, @LexerTokensSpec);

# Input file iterator...
$InputFile = "TestSimpleCalcParser.txt";
open INPUTFILE, "$InputFile" or die "Couldn't open $InputFile: $!\n";
$InputIterator = sub { return <INPUTFILE>; };
$Lexer = new Parsers::Lexer($InputIterator, @LexerTokensSpec);

@LexerTokensSpec = (
    [ 'VAR', qr/[[:alpha:]]+/ ],
    [ 'NUM', qr/\d+/ ],
    [ 'OP', qr/[-+=\/]/,
        sub { my($This, $Label, $Value) = @_;
            $Value .= "; ord: " . ord $Value;
```

```
                    return [$Label, $Value];
                }
            ],
            [ 'NEWLINE', qr/(?:\r\n|\r|\n)/, sub { return [$_[1], 'NewLine']; } ],
            [ 'SPACE', qr/\s*/, sub { return [$_[1], 'Space']; } ],
        );

        # Look ahead without removing...
        $Token = $Lexer->Lex('Peek');
        if (defined $Token && ref $Token) {
            print "PEEK: Token: @{$Token}\n\n";
        }

        # Process input stream...
        while (defined($Token = $Lexer->Lex())) {
            print "Token: " . ((ref $Token) ? "@{$Token}" : "$Token") . "\n";
        }
```

GetLex

```
        $LexerRef = $Lexer->GetLex();
```

Returns a refernece to *Lexer* method to the caller for use in a specific YYLexer.

Lex

```
        $TokenRefOrText = $Lexer->Lex($Mode);
        if (ref $TokenRefOrText) {
            ($TokenLabel, $TokenValue) = @{$TokenRefOrText};
        }
        else {
            $TokenText = $TokenRefOrText;
        }
```

Get next available token label and value pair as an array reference or unrecognized text from input stream by either removing it from the input or simply peeking ahead and without removing it from the input stream.

Possible *Mode* values: *Peek*, *Next*. Default: *Next*.

Next

```
        $TokenRefOrText = $Lexer->Next();
```

Get next available token label and value pair as an array reference or unrecognized text from input stream by removing it from the input stream.

Peek

```
        $TokenRefOrText = $Lexer->Peek();
```

Get next available token label and value pair as an array reference or unrecognized text from input stream by by simply peeking ahead and without removing it from the input stream.

StringifyLexer

```
        $LexerString = $Lexer->StringifyLexer();
```

Returns a string containing information about *Lexer* object.

AUTHOR

Manish Sud <msud@san.rr.com>

SEE ALSO

YYLexer.pm, SimpleCalcYYLexer.pm, SimpleCalcParser.yy

COPYRIGHT