Phase 1. Scanner/Screener

Lawrence McKenny - 10034344 - 11lrm1
Zhiyuan Tong - 20043698 - 16zt9
Tara Carrete - 20021418 - 15tkc1
Jolene Lammers - 20029503 - 16jjl4

CISC/CMPE458
James Cordy
February 10, 2020

Introduction

The following are the changes made to the PT Pascal compiler in order to make the scanner able to process the Like language instead of PT Pascal.

When changes are made to the file, it is generally required to update the *parser.pt* in the scan.def section with the generated contents of the *scan.def* file. This is because as the other files are changed, the generated content can change, and it needs to remain in sync with the contents of *parser.pt*. This applies to many of the changes, so this warning here is used instead of noting the update every time.

1. String Literals

1.1

Location of Change

In the file parser.pt the charClassMap was updated.

What was Changed

Modified the charClassMap to recognize lQuote when a double (instead of single) quotation is encountered.

Original Line: `charClassMap[ord(quote)] := lQuote;`

Updated Line: `charClassMap[ord('"')] := lQuote;`

Reason

This was changed so that the scanning and screening of PT single-quoted char literals (e.g. 'hi there') would be replaced with the scanning and screening of Like double quoted string literals (e.g. "hi there"). The pStringLiteral is now used to represent the new Like double-quoted strings.

2. Comments

2.1. Update Comment Rule

Location of Change

At the end of the scan.ssl file

What was Changed

Updated rule *Comment* in scan.ssl to discards the contents of a comments until after `*/` instead of `}`.

Original Code:

```
Comment :
        % Discard the contents of a comment
        {[
           | '{':
                >
           | lNewLine:
                …
```

Updated code:

```
Comment :
        % Discard the contents of a comment
        {[
            | '*':
                [
                    | '/':
                        >
                    | *:
                ]
           | lNewLine:
                ...
```

Reason:
To discard the contents of the new /* */ comments instead of the old { } comments

2.2. Remove AlternativeComment Rule
Location of Change
At the end of the scan.ssl file

What was Changed
Remove rule *AlternativeComment*

Reason
The old pascal (* *) comments are no longer used

2.3. Add SingleLineComment Rule
Location of Change
At the end of the scan.ssl file

What was Changed
Created rule *SinlgeLineComment*

```
SingleLineComment:
        {[
            | lNewLine:
                .pNewLine
                >
            | lEndFile:
                #eCommentEOF
                .pEndFile
                >
            | *:
                ?
        ]};
```

Reason
To be called when there are // comments and discard contents of the comment and output the correct number of new line tokens

2.4. Update Scan Rule for Comments

Location of Change
In the file scan.ssl, in the scan rule

What was Changed
Removed the code calling @AlternativeComment and @Comment from `'{'` and `'('` options in the scan rule. Also added to the scan rule the `'/'` option and if the if next character is * then call @Comment and if next character is / then call @SingleLineComment

```
| '/':
    [
        | '*':   % Added to look for like comments of type /* */
            @Comment
        | '/':   % Added to look for like comments of type //
            @SingleLineComment
         ...
```

Reason
So that the correct rule will be called for the new Like /* */ and // comments and the old pascal { } and (* *) comments will not be used.

3. String Tokens

New tokens were added so that the relevant symbols used in Like can be detected in the PT Pascal compiler. There were 2 types of tokens: those with a single character, and those with multiple.

3.1 Single Character Tokens
The tokens to be added were: /, %, #, |, and !

To do this, a couple changes were made.

3.1.1
Location of Change
The charClassMap was updated in the *parser.pt* file.

What Was Changed
Lines were added so that new characters could be recognized by the compiler.

For example, to add ‘|’, this was added:
```
charClassMap[ord('|')] := lBar;
```

Reason
Mapping the symbol ex. ‘|’, to the character class defined in the *scan.ssl*. This allows this character to be recognized by the program, rather than being an illegal character.

3.1.2
Location of Change
In *parser.ssl* in the input section under non-compound tokens

What Was Changed
The name of the token was added to the list of tokens.
For example:
```
pBar                    '|'
```

Reason
The different types of possible tokens need to be known to the scanner.

3.1.3
Location of Change
The *scan.ssl* file, in the input section

What Was Changed
The tokens are added to the list of recognized input types.
For example:
```
lBar            '|'
```

Reason

This is where the program is broken into syntax tokens. This list corresponds to that found in the charClassMap. As updates were made there, this file needs to be updated as well to keep track of the new characters being processed.

3.1.4
<u>Location of Change</u>
The *scan.ssl* file, in the scan rule section.

<u>What Was Changed</u>
A rule was added for the added token.

```
| '!':
        .pBang
```

<u>Reason</u>
The rule needs to be updated so once the scanner successfully detects the token, it knows what to do. In this case, to emit the corresponding token.
Some of these rules become more complex if they share characters in common with multi character tokens, which will be dealt with in the next section.

3.2 Multi Character Tokens
The tokens added were ||, +=, -=, *=, /=, %=, and ==.

Once the single character tokens are added, all of the characters used are registered by the scanner, so all that needs to be done is making the compiler aware of the token types, and how to detect them.

Some files still need to be changed:

3.2.1
<u>Location of Change</u>
In *parser.ssl* in the input section under non-compound tokens

<u>What Was Changed</u>
The name of the token was added to the list of tokens.
For example:

```
pMinusEquals         '-='
```

<u>Reason</u>
The different types of possible tokens need to be known to the scanner.

3.2.2
Location of Change
The *scan.ssl* file, in the input section

What Was Changed
The tokens are added to the list of recognized input types.
For example:
```
pMinusEquals
```

Reason
This is where the program is broken into syntax tokens. This list corresponds to that found in the charClassMap. As updates were made there, this file needs to be updated as well to keep track of the new characters being processed.

3.2.3
Location of Change
The *scan.ssl* file, in the scan rule section.

What Was Changed
A rule was added for the added token. First the scanner looks for the first symbol in the token. Once found, it then checks whether or not the next symbol matches the multi-symbol token. If it is, the token is emitted. If any other character follows, then the token corresponding to the first symbol by itself is emitted.

```
| '-'
       [
              | '=':
                    .pMinusEquals
              | *:
                    .pMinus
       ]
```

Reason
The rule needs to be updated so once the scanner successfully detects the token, it knows what to do. In this case, to emit the corresponding token.

4. Keywords
The predefined keywords used by the PT Pascal compiler are different from the Like compiler. Some PT Pascal keywords were deleted, and new ones were added.

4.1 Remove Old PT Keywords:

The keywords array, begin, case, const, div, do, mod, procedure, program, type and until were removed**.**

4.1.1
<u>Location of Change</u>
In *scan.ssl* under the keyword tokens section

<u>What Was Changed</u>
Remove the keyword token from the list
For example:

```
pArray
```

This line was removed

<u>Reason</u>
So that the old PT keyword tokens are not longer emitted

4.1.2
<u>Location of Change</u>
In *stdIdentifiers* file, in the language keyword list

<u>What Was Changed</u>
The keyword was deleted from the keywords list
For example:

```
array
```

This line was removed

<u>Reason</u>
So that the old PT keywords that are no longer used are not recognized by Like

4.2 Add New Like Keywords
Add the new Like keywords choose, elseif, fun, is, like, pkg, public, using, val, when.

4.2.1
<u>Location of Change</u>
In *scan.ssl* file under the keyword tokens section

<u>What Was Changed</u>
Remove the keyword token from the list
For example:

```
pChoose
```

This line was added

<u>Reason</u>
So that the keyword tokens can be emitted for the new Like keywords

4.2.2
Location of Change
In *stdIdentifiers* file, in the language keyword list

What Was Changed
The identifier was added the list
For example:
```
using
```

Reason
So that the new keywords are recognized by Like

5. Predefined Identifiers
The predefined identifiers used by the PT Pascal compiler are different from the Like compiler. Some PT Pascal identifiers were deleted, and new ones were added.

5.1 Deleting Identifiers
The identifiers deleted are: read, readln, write, and writeln

5.1.1
Location of Change
The *stdIdentifiers* file, in the predeclared procedures and functions section

What Was Changed
The identifier was deleted from the list
For example:
```
write
```
This line was removed

Reason
These identifiers no longer need to be recognized

5.2 Adding Identifiers
The identifiers added are: get, getln, put, and putln

5.2.1
Location of Change
The *stdIdentifiers* file, in the predeclared procedures and functions section

What Was Changed
The identifier was added the list
For example:
`put`

Reason
These identifiers need to be recognized

Testing

A suite of test inputs (small Like programs) were designed to demonstrate the correctness of the scanner/screener by forcing it through every new or modified logic path in the S/SL source, checking each new path thoroughly.

Test Cases

These test cases are located in the source code in the ptsrc/testSuite/ directory. For each test case there are three files:

- The Like test source file (.pt)
- The ssltrace output using -e flag (.eOutput)
- Documentation for the test case containing the purpose and result of the test case (.txt)

The following is a list of our test case files broken down by category. Please see the documentation in the testSuite folder for more information on what each test case covers.

Category	**Test Cases**
Keywords	keywords.pt keywordsRemoved.pt keywordIntegration.pt
Predefined Identifiers	predefinedIdentifiers.pt predefinedIdentifiersRemoved.pt
Tokens	tokenBang.pt tokenBar.pt tokenDoubleBar.pt tokenEqualEquals.pt tokenHash.pt tokenMinusEquals.pt tokenPercent.pt tokenPercentEquals.pt

	toekenPlusEquals.pt tokenSlash.pt tokenSlashEquals.pt tokenStarEquals
Strings	stringEmpty.pt stringOld_fail.pt stringRegular.pt
Comments	commentMultiline.pt commentSingleline.pt commentWithSymbols.pt commentOld.pt

Auto Run Script

Inside the ptsrc directory we have also created a script called runTests.sh. This script can be used to autorun all of our test cases. It compares the output of the ssltrace (using the -e flag) for each file with its expected output and prints the results of which tests were passed. To run this script use the command ./runTest.sh in the ptsrc folder.