Testing Documentation

Phase 1 Testing Documentation

All files referred to are in ptsrc/test/phase-1.

The testing document also uses the term scantrace, this is a short form referring to using the SSL trace command on our built compiler for only the first phase, i.e. the scanning phase.

For example, running scantrace on test.pt is equivalent to:

```
ssltrace "ptc -o1 -t1 -L CMPE458/ptsrc/lib/pt test.pt"
CMPE458/ptsrc/lib/pt/scan.def -e
```

This was implemented with a shell script.

Testing new keywords

Quby adds the following keywords to the program:

- using
- val
- def
- break
- when
- module
- unless
- elsif

The test file new-keywords.pt was designed to test if the screener successfully screens the characters to their corresponding keyword tokens:

```
  .pIdentifier (screened to pUsing)
 .pNewLine
  .pIdentifier (screened to pVal)
 .pNewLine
  .pIdentifier (screened to pDef)
 .pNewLine
  .pIdentifier (screened to pBreak)
 .pNewLine
  .pIdentifier (screened to pWhen)
 .pNewLine
  .pIdentifier (screened to pModule)
 .pNewLine
  .pIdentifier (screened to pUnless)
 .pNewLine
  .pIdentifier (screened to pElsif)
 .pNewLine
 .pEndFile
```

As shown in the scantrace output, all keywords were appropriately screened to their assigned tokens (shown by the emitting of their given token e.g. `pWhen`, `pElsif` etc.)

It also removes the keywords:
not
until
program
const
procedure
begin
repeat

old-keywords.pt was written to verify that these are no longer scanned as keywords but instead treated as identifiers. As seen in the scantrace output (shown below), they are all scanned as identifiers rather than specific keywords:

```
   .pIdentifier
   % Output token text 'not'
  .pNewLine
   .pIdentifier
   % Output token text 'until'
  .pNewLine
   .pIdentifier
   % Output token text 'program'
  .pNewLine
   .pIdentifier
   % Output token text 'const'
  .pNewLine
   .pIdentifier
   % Output token text 'procedure'
```

```
.pNewLine
 .pIdentifier
 % Output token text 'begin'
.pNewLine
 .pIdentifier
 % Output token text 'repeat'
.pEndFile
```

Testing Assignment Operator and Equals Operator

Quby modifies the operators for assignment and equals comparison. In Quby the assignment operator is now `=` , and the comparison operator is now `==`.

These changes were implemented in the source files, and the test file equals.pt was designed to verify the scanning correctness.

equals.pt contains an assignment operator and the comparison operator on two separate lines. The following is the output of scantrace on equals.pt:

```
.pAssignEquals
.pNewLine
.pEquals
.pEndFile
```

In the above code, we emit the `pAssignEquals` token to match the `=` in the first line of equals.pt. We then emit a `pNewLine` token, and then a `pEquals` token which matches the `==` in equals.pt. Finally, we emmit a `pEndFile` token.

This indicates that the scanner recognizes the assignment and comparison operators, and is able to differentiate between the two.

If the old assignment operator is used (as shown in colon-equals.pt), it parsed to a colon token (`pColon`) and the assignment token (`pAssignEquals`):

```
.pColon
.pAssignEquals
.pEndFile
```

Testing Not Equals Operator

Quby also changes the inverse comparison operator from `<>` to `!=`. To test it, the script not-equals.pt was written to test this. Running scantrace on not-equals.pt gets the following output:

```
.pNotEqual
.pEndFile
```

As seen above, the string `!=` is properly parsed to the `pNotEquals` token. If the previous operator syntax is used, it is not recognized and instead emits a `pGreater` and `pLess` token (scantrace on old-not-equals.pt):

```
.pLess
.pGreater
.pEndFile
```

Testing New Quotation Character

Quby also replaces the string quotation character from single quotes to double quotes. To test this, the file string.pt was created and contains a single string surrounded in double quotes. The output of the scantrace on string.pt is shown below:

```
 .pStringLiteral
 % Output token text 'a dog'
.pEndFile
```

As seen above, the `pStringLiteral` token is emitted, along with the associated string `a dog`, which also includes the space between the characters.

If the old string format is used, (i.e. using single quotes as done old-string.pt), the scanner will not identify `a dog` as a string, but rather will read it as the two identifiers `a` and `dog`. It will also recognize the single quotes as an illegal character and throw a scan/parse error:

```
#eIllegalChar
 .pIdentifier
 % Output token text 'a'
 .pIdentifier
 % Output token text 'dog'
#eIllegalChar
.pEndFile
```

Testing Hash Character

The test file hash.pt is testing for hash in Quby.
The following is the output from running scantrace on hash.pt.

```
.pHash
.pEndFile
```

Testing New Not (`!`) Operator

The test file exclamation.pt is testing for not (!) in Quby.
The following is the output from running scantrace on exclamation.pt.

```
.pNot
.pEndFile
```

Testing Comments

The test file comments.pt is testing for comments in Quby.
The following is the output from running scantrace on comments.pt.

```
.pNewLine
 .pNewLine
.pNewLine
 .pEndFile
```

The following is the output run from bad_comments.pt that tests previous Pascal comment syntax that is now deprecated in Quby, and as expected, this fails.

```
.pNewLine
#eIllegalChar
 .pIdentifier
 % Output token text 'bad'
 .pIdentifier
 % Output token text 'comment'
 .pInteger
 % Output token text '1'
#eIllegalChar
.pNewLine
.pNewLine
.pLeftParen
.pStar
 .pIdentifier
 % Output token text 'bad'
 .pIdentifier
```

```
    % Output token text 'comment'
    .pInteger
    % Output token text '2'
   .pStar
   .pRightParen
   .pNewLine
   .pEndFile
```

Testing String Index Operator

The test file question.pt is testing for the recognition of the string index operator (?).
Below is the scantrace output for question.pt.

```
   .pQuestion
   .pNewLine
   .pEndFile
```

As seen above, the `pQuestion` token is emitted correctly.
If the scanning of ? symbols were implemented incorrectly,
the compiler would identify it as an illegal character.

Testing Substring Operator

The test file dollar.pt is testing for substring operator ($) recognition.
Below is the scantrace output for dollar.pt.

```
   .pDollar
   .pNewLine
   .pEndFile
```

As seen above, the `pDollar` token is emitted correctly.
If the scanning of $ symbols were implemented incorrectly,
the compiler would identify it as an illegal character.