Changelog Documentation

Keywords

Adding New Keywords - Iffy

Quby adds the following keywords to the language:

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

In order for these new keywords to be screened into their associated keyword tokens, the Scanner/Screener must know the string of characters that is associated to each keyword (for example, `break` is identified with the string "break").

These strings were added to parser/stdIdentifiers:

```
25  + using
26  + val
27  + def
28  + break
29  + when
30  + module
31  + unless
32  + elsif
```

Then, an output token was created for each new keyword and was put in the output section of parser/scan.ssl:

```
75  +         pUsing
76  +         pVal
77  +         pDef
78  +         pBreak
79  +         pWhen
80  +         pModule
81  +         pUnless
82  +         pElsif
83  +         lastKeywordToken = pElsif
```

The output tokens for the keywords were placed in the same order as their strings in stdIdentifiers in order for the tokens to be mapped to the appropriate strings.

Removing Old Keywords - Iffy

Quby also removes the following keywords from the program:

- not
- until
- program
- const
- procedure
- begin
- repeat

To remove these keywords, the opposite set of steps were taken for adding keywords.

First, the string value assigned to each keyword was removed from parser/stdIdentifiers:

```
11    and
12  - not
13    then
14    else
15    of
16    end
17  - until
18    do
19    array
20    file
21  - program
22  - const
23    var
24    type
25  - procedure
26  - begin
27    if
28    case
29    while
30  - repeat
```

Then, their associated tokens were removed from parser/scan.ssl:

```
57  -         pNot
58            pThen
59            pElse
60            pOf
61            pEnd
62  -         pUntil
63            pDo
64            pArray
65            pFile
66  -         pProgram
67  -         pConst
68            pVar
69            pType
70  -         pProcedure
71  -         pBegin
72            pIf
73            pCase
74            pWhile
75  -         pRepeat
76  -         lastKeywordToken = pRepeat
```

It is also important to note that since `pRepeat` was the last keyword token before the change, the new last keyword token is `pElsif` which was assigned to `lastKeywordToken`.

Updating parser.pt

The final step required is to update parser/parser.pt with the modified set of tokens. This is required because parser.pt contains the integer codes assigned to the scan.ssl output tokens. As we have both added and removed tokens, the defined tokens in parser.pt are out of date.

We update parser.pt by first running `make scanner` to generate parser/scan.def which contains the token integer code assignments written in Pascal, for example:

```
        lBackslash = 13;
        lLeftAngle = 14;
        lRightAngle = 15;
        lLeftParen = 16;
        lRightParen = 17;
        lLeftBracket = 18;
        lRightBracket = 19;
        lLeftBrace = 20;
        lRightBrace = 21;
        lExclamation = 22;
        lHash = 23;
```

We then copy the contents of scan.def and paste it into the specified locations at parser.pt. This essentially updates parser.pt with the new tokens and their associated integer codes. An excerpt of some the changes in parser.pt is shown below:

```
74  -       lQuote = 10;
75  -       lBackslash = 11;
76  -       lLeftAngle = 12;
77  -       lRightAngle = 13;
78  -       lLeftParen = 14;
79  -       lRightParen = 15;
80  -       lLeftBracket = 16;
81  -       lRightBracket = 17;
82  -       lLeftBrace = 18;
83  -       lRightBrace = 19;
84  -       lBlank = 20;
85  -       lIllegal = 21;
86  -       lNewLine = 22;
87  -       lCarriageReturn = 23;
88  -       lTab = 24;
89  -       lNewPage = 25;
90  -       lEndFile = 26;
91  -       lastCharClass = 26;
```

```
74  +       lQuestion = 10;
75  +       lDollar = 11;
76  +       lQuote = 12;
77  +       lBackslash = 13;
78  +       lLeftAngle = 14;
79  +       lRightAngle = 15;
80  +       lLeftParen = 16;
81  +       lRightParen = 17;
82  +       lLeftBracket = 18;
83  +       lRightBracket = 19;
84  +       lLeftBrace = 20;
85  +       lRightBrace = 21;
86  +       lExclamation = 22;
87  +       lHash = 23;
88  +       lPercent = 24;
89  +       lBlank = 25;
90  +       lIllegal = 26;
91  +       lNewLine = 27;
92  +       lCarriageReturn = 28;
93  +       lTab = 29;
94  +       lNewPage = 30;
95  +       lEndFile = 31;
96  +       lastCharClass = 31;
```

New String Literals - Iffy

In Quby, strings are defined with double quotes (`"`) rather than single quotes (`'`). To implement this specification, the defined quotation character would have to be changed.

The quotation character recognized by the compiler is defined in parser/parser.pt, assigned to the `quote` variable.

Therefore, the only change required is to update the value assigned to the quote variable from a single quote, to a double quote:

```
418           { misc. }
419           tab = ' ';
420           blank = ' ';
421   +       quote = '"';
422           null = 0;
```

New Operator Characters - (Iffy, Ethan, Liam and Noah)

Quby implements some new operators with the addition of the following characters: `%`, `#`, `?`, `$` and `!`.

These characters were not used in the Pascal program specification and are therefore not assigned any input tokens in scan.ssl. This was rectified by adding the input tokens along with the character mnemonic:

```
27            lSemicolon      ';'
28    +       lQuestion       '?'
29    +       lDollar         '$'
30            lQuote
31            lBackslash
32            lLeftAngle      '<'


37            lRightBracket   ']'
38            lLeftBrace      '{'
39            lRightBrace     '}'
40    +       lExclamation    '!'
41    +       lHash           '#'
42    +       lPercent        '%'
43            lBlank
```

Since the contents of scan.ssl were changed, we recompiled scan.def and copied it into parser.pt to add the new input tokens.

However, parser.pt does not get the string mnemonics from scan.def, only the integers assigned to each token. The `charClassMap` is used instead and maps tokens to their associated character. This was also modified to add the new operator characters. For each one, the character was assigned to their associated input token:

```
931 +           charClassMap[ord('!')] := lExclamation; { new addition to the table }
932 +           charClassMap[ord('$')] := lDollar; { new addition to the table }
933 +           charClassMap[ord('?')] := lQuestion; { new addition to the table }
934 +           charClassMap[ord('#')] := lHash; { new addition to the table }
935 +           charClassMap[ord('%')] := lPercent; { new addition to the table }
```

Scanning New Operators (`?`, `$` and `#`) - Liam & Ethan

The question mark, dollar and hash tag are new operators that have specific functionality in the Quby language. This means that they must have their own tokens like other operators and be emitted by the scanner when they are read.

Firstly, we implement their output tokens by adding them to the output section of scan.ssl:

```
108 +           pQuestion
109 +           pDollar
110             pColon
111             pEquals
112             pNotEqual

118             pRightParen
119             pLeftBracket
120             pRightBracket
121 +           pHash
```

As we have changed the set of defined tokens in scan.ssl, we updated parser.pt with the recompiled scan.def.

Next, we add new choices to the choice loop for the main Scan rule in scan.ssl. These choices match against our operator characters and emit their associated token:

```
159                 .pSemicolon
160 +           | '?':
161 +               .pQuestion
162 +           | '$':
163 +               .pDollar
```

```
237 +           | lHash:
238 +               .pHash %For hash string length operator
239 +
```

As seen above, when `?` is matched, it is consumed and emits `pQuestion`, the assigned `pQuestion` token.

Handling New Not (`!`) and Not Equals (`!=`) - Ethan & Iffy

Quby replaces Pascal's `not` and `<>` with `!` and `!=`. This was what led to the addition of the exclamation character in the program.

Since `not` is no longer a keyword, the `!` operator must be assigned its own token so that in later stages the parser knows how to handle it. This is done by adding its token to the list of output tokens in scan.ssl:

```
104 +       pNot % added as a non-compound token as it is just an operator
105         pDot
```

As we have changed the set of defined tokens in scan.ssl, we updated parser.pt with the recompiled scan.def.

Next we add a new choice option in the main Scan rule for the exclamation character and then handle the two different cases, for the not operation and the not equals:

```
194 +           | '!':
195 +               [
196 +                   | '=':
197 +                       .pNotEqual % if we see ! followed by =, means != which is our
    not equals operator
198 +                   | *:
199 +                       .pNot % if there is anything else, emit pNot token and then
    reloop to scan the value we are inverting
200 +               ]
```

Finally, we need to remove the support for the old not equals operator (`<>`), this is done by removing the second choice when the choice matches the `<` label:

```
214                 | '<':
215                     [
216 +                       % removed old not equals parsing since it is now invalid
```

Handling New Assignment (`=`) and Comparison `(==)` Operators - Iffy

Quby replaces Pascal's `:=` and `=` with `=` and `==` respectively. These changes would have to be made in the scanner in order to emit the correct tokens for the new characters.

Firstly, the `pColonEquals` token in scan.ssl was replaced with a more descriptive token name for the operation: `pAssignEquals`:

```
102             pStar
103 +           pAssignEquals % renamed from pColonEquals to pAssignEquals as it is more
    descriptive for the assignment operator
```

As we have changed the set of defined tokens in scan.ssl, we updated parser.pt with the recompiled scan.def.

Then the actions run when the `=` character is matched in the choice of the main Scan rule was modified to emit the expected tokens:

```
186                 | '=':
187 +                   [
188 +                       | '=':
189 +                           .pEquals % choice operator for this, if it sees = followed by
    =, means == which is comparison operator so emit pEquals
190 +
191 +                       | *: % if there is anything else, then reloop to handle it since
    it is the assignment operator
192 +                           .pAssignEquals
193 +                   ]
```

Finally, support for the old `:=` token was removed by modifying the choice of `:`:

```
174                 | ':':
175 +                   .pColon % not assignment operator anymore so just emit pColon
```

New Comments with `%` - Noah

Quby replaces Pascal's old version of comments with the simple use of the `%` symbol. Any characters after the `%` symbol are discarded.

This was done by changing the input token for the choice label that called the Comment rule in the main Scan rule to a `%`:

```
229 +           | '%':
230                 @Comment
```

The Comment Rule was modified to handle the new comment parsing, and the AlternateComment rule was removed to remove support for the other version of Pascal comments:

```
297     Comment :
298             % Discard the contents of a comment
299
300             {[
301                 | lNewLine:
302                     .pNewLine
303 +                   > % Terminate comment with new line
304 +                 | lEndFile:
305                     .pEndFile
306 +                   > % Terminate comment with end of file
307 +                 | *:
308 +                   ? % consume and discard input that is part of comment
309             ]};
```