Phase 2. Parser

Lawrence McKenny - 10034244 - 11lrm1
Zhiyuan Tong - 20043698 - 16zt9
Tara Carette - 20021418 - 15tkc1
Jolene Lammers - 20029503 - 16jjl4

CISC/CMPE458
James Cordy
March 3, 2020

Phase 1 Changes

In Phase 1, pDotDot, pDot, pBang and pColonEquals were missed, they have now been removed from the output list of scan.ssl and parser.pt has been updated with scan.def accordingly. Additionally the matching of '<>' and the emitting of a pBang was removed from the scan rule. All these changes have also been indicated with comments in scan.ssl.

1. Token Definitions

The keyword tokens and non-compound tokens were updated to correspond to the new set of output tokens emitted by the Like Scanner/Screener. Additionally, output tokens were added that are required for the like language.

1.1. Keyword tokens

<u>Location of Change</u>

The parser keyword input token list in parser.ssl. Additionally, when the list was updated, parser.pt was updated with the contents of parser.def.

<u>What was Changed</u>

Below are the keyword tokens that were added and removed. Note the exact location of the changes are also indicated clearly with comments in parser.ssl.

Removed:

pDiv	'div'	% Replaced by pSlash
pMod	'mod'	% Replaced by pPercent
pUntil	'until'	
pDo	'do'	
pArray	'array'	
pProgram	'program'	
pConst	'const'	
pType	'type'	
pProcedure	'procedure'	
pBegin	'begin'	
pCase	'case'	

Added:

pUsing	'using'
pVal	'val'
pFun	'fun'
pChoose	'choose'
pIs	'is'
pElseif	'elseif'
pWhen	'when'

pLike	'like'
pPkg	'pkg'
pPublic	'public'

Reason
The keyword parser token input list was modified so that it would correspond to the new set of keyword output tokens from Phase 1 emitted by the Like Scanner/Screener.

1.2. Non-Compound tokens
Location of Change
The parser non-compound input token list in parser.ssl. Additionally, when the list was updated, parser.pt was updated with the contents of parser.def.

What was Changed
Below are the non-compound tokens that were added, removed or changed. Note the exact location of the changes are also indicated clearly with comments in parser.ssl. Also note, that pDotDot, pDot and pColonEquals, were missed in Phase 1, they have now been removed from the output list of scan.ssl and accordingly were removed from the input list of parser.ssl as indicated below.

Removed:

pDot	'.'
pDotDot	'..'
pColonEquals	':='

Added:

pSlash	'/' % Replaced pDiv
pHash	'#'
pPercent	'%' % Replaced pMod
pBar	'|'
pBarBar	'||'
pPlusEquals	'+='
pMinusEquals	'-='
pStarEquals	'*='
pSlashEquals	'/='
pPercentEquals	'%='
pEqualEquals	'=='

Changed:

pNotEqual	'!=' % Symbol changed from '<>' to '!='

Reason
The non-compound parser input token list was modified so that it would correspond to the new set of non-compound output tokens from Phase 1 emitted by the Like Scanner/Screener.

1. 3. Output Tokens
Location of Change
The parser output token list in parser.ssl. Additionally, when the list was updated, parser.pt was updated with the contents of parser.def.

What was Changed
Below is a list of the output tokens that were added and changed. Note the exact location of the changes are also indicated clearly with comments in parser.ssl.
Added
sPackage
sPublic
sConcatenate
sRepeatString
sSubstring
sLength
sInitialValue
sCaseElse

Changed:
sLike   % Changed from sType to sLike

Reason
Added new parser output tokens as they are required by the like language. Added new parser output tokens for Like packages, sPackage and sPublic, and for the Like string operations sConcatenate, sRepeatString, sSubstring and sLength. Also, added a new sInitialValue output token for Like variable initial values, and an sCaseElse output token for Like choose statements. Finally, the old PT parser type declaration output token sType was replaced with sLike, since Like doesn't have type declarations and uses like for types instead.

2. Programs

Location of Change
In the program rule in parser.ssl

What was Changed
The keyword ‘program’ was changed to ‘using’. The pIndetifier program name identifier was removed and the brackets for around parameters were removed. The updated code for program is show below:

```
Program :
      'using'  .sProgram % changed using to program
       % removed program name as no name in Like
       % program parameters
       % removed brackets around parameters
```

```
        {
           pIdentifier  .sIdentifier
           [
              | ',':
              | *:
                 >
           ]
        }
       .sParmEnd
        ';'  @Block; % removed the need to end the program with a "."
```

The using clause was also added to the options in the block rule and calls program.

<u>Reason</u>
The program definition needed to be changed to match the Like Language specification of having using instead of program and no brackets around parameters.

The using clause was also added to the options in the block rule in case there is more than one program in a file.

3. Declarations

<u>Location of Change</u>
Removed Rules: the TypeDefinitions, TypeBody and SimpleType
Added Rules: PublicVariableDeclarations, ValueOrLike, LikeClause
Modified Rules: Block, VariableDeclarations, ConstantDefinitions

<u>What was Changed</u>

```
           | 'public': % Added to check for public
              [
                 | 'var':    % Added to check for var
                    @PublicVariableDeclarations
                    ';'
                 | 'val':    % Add to check for val
                    @PublicConstDefinitions
                 | 'fun':    % Added to check for fun
                    @PublicRoutines
                    ';'
                 | *:
              ]

           | 'val':    % Add to check for val
              @ConstantDefinitions % semicolon is part of definition
           | 'var':    % Added to check for var
```

```
                @VariableDeclarations
                ';'

VariableDeclarations :
        % Accept one or more constant, and like type declaration
        .sVar
        pIdentifier  .sIdentifier
        @ValueOrLike;

PublicVariableDeclarations :
        % a clone of variable declarations with a sPublic emit
        .sVar
        pIdentifier  .sIdentifier
        .sPublic
        @ValueOrLike;

% Added fuction for like declaration
ValueOrLike :
        % variable declarations uses either an inital value
        % or a like clause to specify the variable's type
        [
           | '=':
            @ValueClause
           | *:
            @LikeClause
        ];

ValueClause :
        % Accept one or more named constant definitions
        .sInitialValue
        @Expression

LikeClause :
        % allows for an array bound
        % followed by a colon, optionsal file keyword
        % and a variable or constant
        [
           | '[':
              .sArray @ConstantValue ']'
           | *:
        ]
        ':'
        %an optional file keyword
        [
            | 'file':
                .sFile
            | *:
        ]
        'like' .sLike
        @ConstantValue;

PublicConstDefinitions :
        % Accept one or more named constant definitions
        .sConst
        pIdentifier  .sIdentifier
        .sPublic
        '=' @ConstantValue
        {[
            | ',':
                pIdentifier .sIdentifier
                '=' @ConstantValue
            | ';':
            >
        ]};
```

<u>Reason</u>
modify the parsing of constant and variable declarations to meet Like specification

- modify the parsing of const declarations (ConstantDefinitions rule)
- modify parsing of variable declarations (VariableDeclarations rule)
- both constant and variable declarations can optionally be declared public

4. Routines

<u>Location of Change</u>
Modified Rules: Block, ProcedureHeading
Added Rules: Routines, PublicRoutines
Modify the parsing of routines

<u>What was Changed</u>
Block rule changes

```
            | 'public': % Added to check for public
                [
                    | 'var':    % Added to check for var
                        @PublicVariableDeclarations
                        ';'
                    | 'val':    % Add to check for val
                        @PublicConstDefinitions
                    | 'fun':    % Added to check for fun
                @PublicRoutines
                ';'
              | *:
            ]

          | 'fun':    % Added to check for fun
            @Routines
            ';'

Routines :
    % parsing of fun routine
    % end with 'end' ';'
    .sProcedure
    % procedure name
    pIdentifier  .sIdentifier
    @ProcedureHeading
    @Block
    'end';

PublicRoutines :
    % a clone of Routines rules, with a sPublic token emit
    .sProcedure
    % procedure name
    pIdentifier  .sIdentifier
    .sPublic
    @ProcedureHeading
    @Block
    'end';

ProcedureHeading :
    % Accept zero or more procedure formal parameter declarations.
    [
      | '(':
        {
          % formal parameter identifier
          [
            | 'var':
                pIdentifier  .sIdentifier
                .sVar
            | *:
                pIdentifier  .sIdentifier
          ]
          @LikeClause % use like clause instead of type names for parameter types
          [
                    | ',': % use commas sparate formal parameters
                    | *:
                        >
                  ]
              }
              ')'
          | *:
      ]
      .sParmEnd  % switched .sParmEnd to be emitted before reading the 'is' token else new line
characters are displayed before .sParmEnd instead of after
      'is';
```

Reason

- semicolons -> commas to separate formal parameters
- type names -> like clauses for parameter types
- PT procedure -> like function: minimize the difference of the output token stream
    - like function can be declared public

5. Packages

Location of Change

The changes were made in parser.ssl, a new rule called Package was added and changes were made in the block rule.

What was Changed

In the block rule an option as seen below was added for if the ‘pkg’ keyword occurs.

```
| 'pkg':
    @Package
```

From the package option in the block rule the Package rule is called. This new rule is shown below. It outputs the token sPackage to mark the beginning of the package and then outputs the body of the package, which is preceded by an sBegin token and ended by an sEnd token by using the Statement rule which in turn uses the Block rule.

```
Package :
        .sPackage
        pIdentifier
        .sIdentifier
        'is'
        @Statement;
```

Reason

This was added to allow for the parsing of packages as specified in the Like language specification.

7. Else If Clauses

Location of Change

In the IfStmt rule in parser.ssl

What was Changed

The ifStmt rule was changed to following, with a loop added checking for elseif statements:

```
IfStmt :
        .sIfStmt
        @Expression
        .sExpnEnd
        'then'  .sThen
        @Statement
        {[
            | 'else':
                .sElse
                @Statement
                >

            | 'elseif':
                .sIfStmt
                @Expression
                'then'    .sThen
                @Statement
            | *:
                >
        ]};
```

Reason

The changes were made to handle the elseif statements. We changed it how we did to simply output the token stream corresponding to the equivalent PT Pascal nested if statements, so that the semantic phase will not have to be modified to handle elseif at all.

8. Choose Statements

There are some token changes done as a part of the choose statement. The details are all covered in the Token section of the documentation. The relevant ones are: pChoose, pWhen, pCaseElse, and pCase.

8.1 CaseStmt

Location of Change

In the CaseStmt rule of parser.ssl

What was Changed

Add ‘when’ to be parsed before each call to CaseAlternative, as ‘when’ precedes every case in Like. It is also a choice once the CaseAlternative is processed, if there is another, ‘when’ will be the next thing encountered.
Add ‘else’ as a choice to the loop as that is a new functionality in Like.
‘;’ is no longer a choice as CaseAlternative is changed to parse the semicolon and therefore cannot be used here.

```
CaseStmt :
        .sCaseStmt
        @Expression
        .sExpnEnd
        'of'
            'when' % beginning of a case alternative, need it outside CaseAlternative to use in loop
        @CaseAlternative
        {[
                % else is not required
            | 'end':
                        .sCaseEnd
                >
                % add in else case
                | 'else':
                        .sCaseEnd
                        .sCaseElse
                        @Statement
                        >
                | 'when':
                        @CaseAlternative
        ]};
```

<u>Reason</u>
The parsing of the choose statement has to have the new else functionality. It also needs to adapt to the changes made to the other parts of the code while still working as intended.

8.2 CaseAlternative
<u>Location of Change</u>
In the CaseAlternative rule of parser.ssl

<u>What was Changed</u>
It parses ‘then’ before the body of the case alternative.
As the semicolon handling is now part of Statement, it no longer parsers the semicolon at the end.

```
CaseAlternative :
        % A case alternative is a statement labelled by
        % one or more optionally signed integer constants

        {
           @OptionallySignedIntegerConstant
           [
              | ',':
              | *:
                 >
           ]
        }
            'then' % add the then token to case statement
        .sLabelEnd
        @Statement;
```

Reason
The new ‘then’ token parsing is added. Also coping with the changes made to Statement while preserving functionality.

9. Repeat While Statements
The ‘do’, and ‘until’ tokens are no longer used here and so are deleted. See tokens section for more details.

9.1 While Statement
Location of Change
In the WhileStmt rule of parser.ssl

What was Changed
The body of the while loop was removed. It just calculates the expression of the while loop, no more ‘do’ and Statement being processed. Instead it reaches the end of the statement and checks for a semicolon.

```
WhileStmt :
   .sWhileStmt
   @Expression
   .sExpnEnd
   ';';  % ends with a semi colon
```

Reason
The while statement can be called in 2 types of loops. By restricting its functionality, the rule works for both kinds.

9.2 Repeat Statement

Location of Change
In the WhileStmt rule of parser.ssl

What was Changed
The repeat rule was changed to handle both types of loop. If ‘repeat’ is immediately followed by ‘while’, then we know it is functionally a while loop. So that is parsed and loops through Statements until ‘end’ is found.
If there is no ‘while’ following immediately, we know it is the repeat...while section. The body of this is parsed the same way as the previous RepeatStmt, though replacing the ‘until’ with a ‘while’. This is because while and until at the end are pretty much the same, except needing to add a .sNot to the expression.

```
RepeatStmt :
    [
      | 'while':   % Repeat while option
        @WhileStmt
        {
          @Statement
          [
            | ';':
            | 'end':
              >
          ]
        }
      | *:  % repeat ... while option
        .sRepeatStmt
        {
          @Statement
          [
            | ';':
            | 'while':
              .sRepeatEnd
              >
          ]
        }
        @Expression
        .sNot % reverse from previous "until" used in PT Pascal
        .sExpnEnd
    ];
```

Reason
Changed so it covers the while loop functionality no long in WhileStmt. Also adjusted the repeat part of the loop so that follows ‘while’ instead of old ‘until’ logic.

10. Block

Location

Updates were made in the Statement and Block rule of parser.ssl

What was Changed

- Migrated applicable terms from Statement to Block:
  - begin -> using
  - If
  - While
  - Repeat
  - Case -> choose
- Removed terms from Block that are no longer useful:
  - Const
  - Type
  - Procedure
- Added terms specific to Like:
  - Pkg
  - Public
  - Fun
  - val

Modified rule to emit .sBegin when called and .sEnd with exiting, changed the behaviour of semi-colon to match like specifications.

Reason

Updated as Like makes no distinction between declarations and statements. Additionally, the Block rule was changed to accept a sequence of any number of declarations or statements in any order.

11. Short Form Assignments

Location of Change

In the AssignmentOrCallStmt rule in parser.ssl

What was Changed

The following options were added to the AssignmentOrCallStmt rule for the parsing of like short form assignment statements:

```
| '+=':
    .sAssignmentStmt
    .sIdentifier
    .sIdentifier
    @Expression
    .sAdd
    .sExpnEnd
| '-=':
    .sIdentifier
    .sIdentifier
    @Expression
    .sSubtract
    .sExpnEnd
| '*=':
    .sIdentifier
    .sIdentifier
    @Expression
    .sMultiply
    .sExpnEnd
| '/=':
    .sIdentifier
    .sIdentifier
    @Expression
    .sDivide
    .sExpnEnd
| '%=':
    .sIdentifier
    .sIdentifier
    @Expression
    .sModulus
    .sExpnEnd
```

<u>Reason</u>
To allow for the parsing of the like short form assignment statements (+=, -=, *=, /=, %=). The added code above makes it so the parser outputs the same semantic team for short form assignments as their equivalent regular expression/non short form version. For example i += 10 will now have the same output as i = i + 10 does.

12. The String Type

<u>Location of Change</u>
Modified Rules: SimpleExpression, Term, Factor

<u>What was Changed</u>
SimpleExpression rule

```
        | '|':  % Added for string concatenate operation
           @Term   .sConcatenate
        | '||': % Added for string repeat operation
           @Term   .sRepeatString
```

```
Term rule
        | '/':
           @Factor
           [
              | ':':
              @Factor
              % Added for substring operation
              .sSubstring
              | *:
              .sDivide
           ]
Factor rule
        | '#':  % Added for string length operation
           @Factor
           .sLength
```

<u>Reason</u>

- char data type -> varying length string type
- add handling of the new Like string operators concatenate, repeat, substring and length

13. Operator Syntax

13.1 Token changes

This section has overlap with the changes discussed in the token changes section. These are just the changes needed for operator syntax to be changed.

<u>Location of Change</u>

In the Input section of parser.ssl

<u>What was Changed</u>

Tokens for each of the operator syntax were added. In the case of pDiv and pMod this meant deleting them and creating new tokens:

Delete:

```
pDiv     'div'
pMod   'mod'
```

Add:

```
pSlash  '/'
pPercent  '%'
```

For assignment, the required pEquals ('=') already existed. All that happened is the pColonEquals become unused and so it was deleted

Delete:

```
pColonEquals   ':='
```

For inequality, the token pNotEquals is a reasonable name, so all that needed changed is the symbol represented by the token.

Changed to:

pNotEquals 	‘!=’

For equality, pEqualEquals, already existed so nothing needed to be done.

Reason

The tokens are changed so that the correct symbols are matched up to the correct tokens. The tokens are used later in the code, so they can have the same functionality as before, simply with different symbols used in the Like code.

13.2 Assignment

Location of Change

In the AssignmentOrCallStmt rule of parser.ssl

What was Changed

In the choice, ‘:=’ was one of the options, which led it to assignment. That was switched to ‘=’, the new assignment operator.

It was also changed under the ‘[‘ choice, as an assignment could occur within that choice as well. (is assigning to subscripted variable)

Reason

When checking if something is an assignment, the updated assignment symbol needs to be used to parse correctly.

13.3 Equality

Location of Change

In the Expression rule of parser.ssl

What was Changed

The choice ‘=’ lead to processing of an equality statement. The symbol was updated to ‘==’

Reason

When checking if something is an equality, the updated equality symbol needs to be used to parse correctly.

13.4 Inequality

Location of Change

In the Expression rule of parser.ssl

What was Changed
The choice ‘<>’ lead to processing of an equality statement. The symbol was updated to ‘!=’

Reason
When checking if something is an inequality, the updated inequality symbol needs to be used to parse correctly.

13.5 Division
Location of Change
In the Term rule of parser.ssl

What was Changed
The choice ‘div’ leads to processing of an equality statement. The symbol was updated to ‘/’

Reason
When checking if something is a division, the updated division symbol needs to be used to parse correctly.

13.6 Modulus
Location of Change
In the Term rule of parser.ssl

What was Changed
The choice ‘mod’ lead to processing of an equality statement. The symbol was updated to ‘%’

Reason
When checking if something is a modulus, the updated modulus symbol needs to be used to parse correctly.

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 two files:

- The Like test source file (.pt)

- The ssltrace output using -e flag (.eOutput)

Additionally each test case includes documentation either in a .txt file with the same name or as a comment in the test case .pt file. (comment form for package and short form assignment tests)

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**
Programs	using.pt usingMulti.pt usingNoSemiColon.pt usingNone.pt usingMultiCases.pt
Declarations	declarationArrayLike.pt declarationFileLike.pt declarationLikeVar.pt declarationMultiVals.pt declarationPublicVar.pt declarationPublicVal.pt declarationSingleVal.pt declarationSingleVar.pt declarationValNoSemiColon.pt declarationVarNoSemiColon.pt declarationPublicVarNoSemiColon.pt
Routines	routineLikeParam.pt routineMultiParams.pt routinePublicFun.pt routineSimpleFun.pt routineVarLikeParam.pt routineFunNoSemiColon.pt routinePublicFunNoSemiColon.pt
Packages	packages.pt packagesBaseCase.pt packagesWithFunctions.pt packagesWithoutSemiColon.pt
Statements	nullStatement.pt
Else-if Clauses	elifNoElse.pt elseTest.pt IfElseElse.pt IfElseNoSemi.pt ifElseTest.pt ifTest.pt

Choose Statements	chooseValid.pt chooseNoElse.pt chooseNoSemiColon.pt
Repeat While Statements	repeatWhile_while.pt repeatWhile_whileNoEnd.pt repeatWhile_whileNoSemiColon.pt repeatWhile_whileNested.pt repeatWhile_whileBracketless.pt repeatWhile_until.pt repeatWhile_untilNoSemiColon.pt
Short Form Assignments	shortFormAssignmentsDivideEquals.pt shortFormAssignmentsMinusEquals.pt shortFormAssignmentsModuloEquals.pt shortFormAssignmentsMultipleEquals.pt shortFormAssignmentsPlusEquals.pt
The String Type	tringTypeConcatenate.pt stringTypeLength.pt stringTypeRepeat.pt stringTypeSubstring.pt
Operator Syntax	operatorAssigment.pt operatorDiv..pt operatorEquality.pt operatorMod.pt operatorNotEqual.pt operatorInvalidAssigment.pt operatorInvalidDiv..pt operatorInvalidEquality.pt operatorInvalidMod.pt operatorInvalidNotEqual.pt
Other Tests	baseCase.pt assignmentNoSemiColon.pt

Auto Run Script

Inside the ptsrc directory we have also created a script called runTestsPhase2.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 ./runTestsPhase2.sh in the ptsrc folder.