ProTem Implementation

Eric Hehner

The ProTem programming system is described at hehner.ca/PT.pdf. This is its implementation, written in ProTem. Still to do: data; assignment; \; || ; forward; predefined; arguments; operators; last-action Symbol level deleting and editing needs to be integrated with reading and scanning, with an opportunity to correct errors and continue programming. Bootstrap through Turing or C. Unused error numbers: 5, 6, 25,..∞; Unused apology numbers: 20,..∞.

input channel: keys for keying in a program
output channels: screen for echoing the program and msg for error and apology messages perhaps msg could be a popup box on top of screen indicating the location of the error

scanCode: 0,..100 terminals
parseCode: 100,..200 nonterminals
nameCode: 200,..300 name control
actionCode: 300,..999 object code generation
bottom = 999 of parse stack

SCANNER

` new scanCodeText := "". `so persistent definitions can be saved
new sourceCode: 0..100:= 99. `end
new sourceCodes: *nat:= nil. `sequence of scan codes.
After code 8 is an index into `sourceNumbers`;
after code 9 is an index into `sourceTexts`;
after code 10 is an index into `sourceNames`.

new `simpleName`: `text` := "".
new `sourceNames`: *[ `text` ] := nil. `sequence of source names
new `sourceTexts`: *[ `text` ] := nil. `sequence of source texts
new `sourceNumbers`: *[ `nat` ] := nil. `sequence of source numbers
new `error`: `bin` := ⊥. `Has an error been detected?
new `object`: *[ `nat` ] := nil. `the object code we are producing for execution

new `readChar` [[`?``"" (char)""`. !?. `source` := `source`; `?]].

new `scan`

[ use: bold end italic nl source tab
  `assign`: `error` `number` `simpleName` `source` `sourceCodes` `sourceNames` `sourceTexts`
  `call`: `readChar`
  `output`: `msg`
  `pre`: `?` has been output but not scanned
  `post`: `?=end

new `fancy` [ use: bold end italic nl source tab
  `assign`: `error` `number` `simpleName` `source` `sourceCodes` `sourceNames` `sourceTexts`
  `call`: `readChar`
  `output`: `msg`
  `pre`: `?` is within the name; it has been output but not scanned
  `post`: `?` = (first character after fancy name)
    `if` `?=""""` [[`simpleName`: `simpleName`; """"]
    `sourceCodes`: `sourceCodes`; 10; ↔`sourceNames`
    `sourceNames`: `sourceNames`; `[simpleName]. `readChar]
  else [[`simpleName`: `simpleName`; """"]
    `if` `?=""""`
      [[`readChar`
        `if` `?="""
          [[`delete`; `delete`; """"]
          `source`: `source` (0;.. ↔`source`–2); """"
          `simpleName`: `simpleName`; """
          `sourceCodes`: `sourceCodes`; 10; ↔`sourceNames`
          `sourceNames`: `sourceNames`; `[simpleName]. `readChar]
        else [[`simpleName`: `simpleName`; """"]
          `fancy]]
      else `if` `?="""
        [[`error`: T. `msg""""Error 13: unclosed fancy name""]
        else `[[`simpleName`: `simpleName`; `. `readChar. `fancy]]]]. `end of fancy

  for efficiency, the cases below should be in order of decreasing frequency

  `if` `?=end` [[`sourceCodes`: `sourceCodes`; 99]]

else `if` `(?=" "`) `v` `(?=tab)` `v` `(?=nl)` [[`readChar. `scan]]

else `if` `a` ≤ `?` ≤ "Z" `plain simple name or keyword
[[`new sx`: ↔`source`. `simpleName`: `. `nameOrKeyword]
  `if` ("a" ≤ `?` ≤ "Z") `v` ("0" ≤ `?` ≤ "9")
  [[`simpleName`: `simpleName`; `. `nameOrKeyword]]
else [[`see if it's a keyword or a name

`for efficiency, the cases below should be in order of decreasing frequency

`if` `?=end` [[`sourceCodes`: `sourceCodes`; 99]]

else `if` `(?=“ ”) `v` `(?=tab)` `v` `(?=nl)` [[`readChar. `scan]]

else `if` "a" ≤ `?` ≤ "Z" `plain simple name or keyword
[[`new sx`: ↔`source`. `simpleName`: `. `nameOrKeyword]
  `if` ("a" ≤ `?` ≤ "Z") `v` ("0" ≤ `?` ≤ "9")
  [[`simpleName`: `simpleName`; `. `nameOrKeyword]]
else [[`see if it's a keyword or a name

`for efficiency, the cases below should be in order of decreasing frequency

`if` `?=end` [[`sourceCodes`: `sourceCodes`; 99]]

else `if` `(?=“ ”) `v` `(?=tab)` `v` `(?=nl)` [[`readChar. `scan]]

else `if` "a" ≤ `?` ≤ "Z" `plain simple name or keyword
[[`new sx`: ↔`source`. `simpleName`: `. `nameOrKeyword]
  `if` ("a" ≤ `?` ≤ "Z") `v` ("0" ≤ `?` ≤ "9")
  [[`simpleName`: `simpleName`; `. `nameOrKeyword]]
else [[`see if it's a keyword or a name

`for efficiency, the cases below should be in order of decreasing frequency
` for efficiency, these should be in order of decreasing frequency

```prolog
if simpleName="case" [scanCode:= 0]
else if simpleName="else" [scanCode:= 1]
else if simpleName="for" [scanCode:= 2]
else if simpleName="if" [scanCode:= 3]
else if simpleName="new" [scanCode:= 4]
else if simpleName="old" [scanCode:= 5]
else if simpleName="plan" [scanCode:= 6]
else if simpleName="value" [scanCode:= 7]
else [scanCode:= 10].
```

```prolog
for n: 0;..simpleName + 1 ![delete].
if scanCode=10 ![italic simpleName; ?.
source:= source_(0;..sx); italic simpleName; ?.
sourceCodes:= sourceCodes; 10; ↔ sourceNames.
sourceNames:= sourceNames; [italic simpleName]]
else ![bold simpleName; ?.
source:= source_(0;..sx); bold simpleName; ?.
sourceCodes:= sourceCodes; scanCode].
```

else !["" ` fancy name

```prolog
[simpleName="". source:= source; "". readChar. fancy. scan]
```

else if “0” ≤ ? ≤ “9” ` number

```prolog
[new number: real:= ?.
moreNumber ![readChar. if “0” ≤ ? ≤ “9” [number:= number×10 + ?. moreNumber]].
if ?=".""
readChar.
if “0” ≤ ? ≤ “9”
moreFraction [number:= number + ?/denom. readChar.
if “0” ≤ ? ≤ “9” [denom:= denom×10. moreFraction]].
sourceCodes:= sourceCodes; 8; ↔ sourceNumbers.
sourceNumbers:= sourceNumbers; number. scan]]
```

else if ?==“” ` text using “ and ”

```prolog
[new txt: text:= "". moreText ![readChar.
if ?=""
[readChar. if ?=""
[!delete; delete; underline "". txt:= txt; "". moreText]
else [msg!“Error 2: lonely " within text”. error:= ⊤]]
else if ?=""
[readChar. if ?=""
[!delete; delete; underline "". txt:= txt; "". moreText]
else [sourceCodes:= sourceCodes; 9; ↔ sourceTexts.
sourceTexts:= sourceTexts; [txt]. scan]]
else if ?==end [error:= ⊤. msg!“Error 3: unclosed text”]]
else [txt:= txt; ?. moreText]]]]
```

else if ?==“” ` text using "

```prolog

```

else if ?==“” ` fancy name

```prolog
[!italic simpleName; ?.
source:= source; "". readChar. fancy. scan]
```

else if “0” ≤ ? ≤ “9” ` number

```prolog
[new number: real:= ?.
moreNumber ![readChar. if “0” ≤ ? ≤ “9” [number:= number×10 + ?. moreNumber]].
if ?=".""
readChar.
if “0” ≤ ? ≤ “9”
moreFraction [number:= number + ?/denom. readChar.
if “0” ≤ ? ≤ “9” [denom:= denom×10. moreFraction]].
sourceCodes:= sourceCodes; 8; ↔ sourceNumbers.
sourceNumbers:= sourceNumbers; number. scan]]
```

else if ?==“” ` text using “ and ”

```prolog
[new txt: text:= "". moreText ![readChar.
if ?=""
[readChar. if ?=""
[!delete; delete; underline "". txt:= txt; "". moreText]
else [msg!“Error 2: lonely " within text”. error:= ⊤]]
else if ?=""
[readChar. if ?=""
[!delete; delete; underline "". txt:= txt; "". moreText]
else [sourceCodes:= sourceCodes; 9; ↔ sourceTexts.
sourceTexts:= sourceTexts; [txt]. scan]]
else if ?==end [error:= ⊤. msg!“Error 3: unclosed text”]]
else [txt:= txt; ?. moreText]]]]
```

else if ?==“” ` text using "

```prolog

```
new txt: text:= "". source:= source; "_". 'delete; "_".
moreText [readChar.
    if ?="" [readChar.
        if ?="" [txt:= txt; "". moreText]]
    else ![delete; ""].
        sourceCodes:= sourceCodes; 9; ↔sourceTexts.
        sourceTexts:= sourceTexts; [txt]. scan]]
else [if ?=end [error:= ⊤. msg!"Error 14: unclosed text"]
    else [txt:= txt; ?. moreText]])]
else [if ?="" comment
    [moreComment [readChar. if ?=nl v ?=end [scan]
        else [moreComment]]]]
else [if ?="" ![delete; "". sourceCodes:= sourceCodes; 11. readChar. scan]
else [if ?="" .. or , or ,
    [readChar.
        if ?="" [readChar. if ?="" [sourceCodes:= sourceCodes; 13. readChar. scan]
            else [sourceCodes:= sourceCodes; 12; 17. scan]]]
    else [sourceCodes:= sourceCodes; 12. scan]]
else [if ?="" .. or ;; or ;
    [readChar.
        if ?="" [readChar. if ?="" [sourceCodes:= sourceCodes; 16. readChar. scan]
            else [sourceCodes:= sourceCodes; 14; 17. scan]]]
    else [sourceCodes:= sourceCodes; 14. scan]]]
else [if ?="" ;; or := or => or :- or :) or :
    [readChar.
        if ?="" [sourceCodes:= sourceCodes; 19. readChar. scan]
    else [if ?="=" [sourceCodes:= sourceCodes; 20. readChar. scan]
        else [if ?="">" ![delete; delete; """].
            sourceCodes:= sourceCodes; 38. readChar. scan]
    else [if ?="~" ![delete; delete; "∈"]. sourceCodes:= sourceCodes; 66.
            readChar. scan]
    else [if ?="~") ![delete; delete; "]]. sourceCodes:= sourceCodes; 76.
            readChar. scan]
    else [sourceCodes:= sourceCodes; 18. scan]]]]]
else [if ?="" =| or =
    [readChar. if ?="" [!delete; delete; "="]
        sourceCodes:= sourceCodes; 71.
        readChar. scan]
    else [sourceCodes:= sourceCodes; 21. scan]]
else [if ?="<" <> or <= or <! or <= or <= or < or <
    [readChar.
        if ?=""> ![delete; delete; "<>". sourceCodes:= sourceCodes; 53. readChar. scan]
    else [if ?="=" ![delete; delete; "≤". sourceCodes:= sourceCodes; 25. readChar. scan]
else [if ?="i" [!delete; delete; " «". sourceCodes:= sourceCodes; 69. readChar. scan]
  else [if ?="<" fancy name
    [simpleName:= "«". source:= source:= "«".
      !delete; delete; "«". readChar. fancy. scan]
  else [if ?="=" [!delete; delete; " «". sourceCodes:= sourceCodes; 37. readChar. scan]
    else [sourceCodes:= sourceCodes; 23. scan]]]]]]
else [if ?="" \> or \> or >
  [readChar.
    if ?="" [!delete; delete; " «". sourceCodes:= sourceCodes; 26. readChar. scan]
  else [if ?="<" [!delete; delete; " «". sourceCodes:= sourceCodes; 49. readChar. scan]
    else [sourceCodes:= sourceCodes; 24. scan]]]]]
else [if ?="[" \] or [l or [}
  [readChar.
    if ?="[" [!delete; delete; " «". sourceCodes:= sourceCodes; 67. readChar. scan]
  else [if ?="{" [!delete; delete; "[«". sourceCodes:= sourceCodes; 39. readChar. scan]
    else [sourceCodes:= sourceCodes; 35. scan]]]]]
else [if ?="|" \| or \| or \| or \| or \|]
  [readChar.
    if ?="|" [sourceCodes:= sourceCodes; 43. readChar. scan]
  else [if ?="\" [!delete; delete; " «". sourceCodes:= sourceCodes; 70. readChar. scan]
    else [if ?="=" [!delete; delete; " «". sourceCodes:= sourceCodes; 69. readChar. scan]
      else [if ?="(" [!delete; delete; " «". sourceCodes:= sourceCodes; 40. readChar. scan]
        else [if ?="( " [!delete; delete; " «". sourceCodes:= sourceCodes; 73. readChar. scan]
          else [sourceCodes:= sourceCodes; 42. scan]]]]]]]
else [if ?="( " \( or \( or \( or \( or \( or \)
  [readChar.
    if ?="( " [!delete; delete; " «". sourceCodes:= sourceCodes; 72. readChar. scan]
  else [if ?=":" [!delete; delete; " «". sourceCodes:= sourceCodes; 75. readChar. scan]
    else [sourceCodes:= sourceCodes; 31. scan]]]]]
else [if ?="\" \] or \] or \] or \] or \] or \]
  [readChar.
    if ?="\" [!delete; delete; " «". sourceCodes:= sourceCodes; 57. readChar. scan]
  else [sourceCodes:= sourceCodes; 41. scan]]]
else [if ?="-" \> or \> or \>
  [readChar.
    if ?="-" [!delete; delete; " «". sourceCodes:= sourceCodes; 52. readChar. scan]
  else [sourceCodes:= sourceCodes; 48. scan]]]
else [if input="/" \> or // or /= or /
[readChar.
  if ?="\"" [!delete; delete; "\"". sourceCodes:= sourceCodes; 56. readChar. scan]
  else [if ?="'" [!delete; delete; "'". sourceCodes:= sourceCodes; 63. readChar. scan]
    else [if ?="=" [!delete; delete; "=". sourceCodes:= sourceCodes; 22. 
      readChar. scan]
      else [sourceCodes:= sourceCodes; 50. scan]]]]]
else [if ?="^" \^ or ^
    [readChar. if ?="^" [sourceCodes:= sourceCodes; 59. readChar. scan]
      else [sourceCodes:= sourceCodes; 58. scan]]]
else [if ?="#" \#1 or #
    [readChar. if ?="1" [sourceCodes:= sourceCodes; 30. readChar. scan]
      else [sourceCodes:= sourceCodes; 29. scan]]]
else [if ?="?" \?? or ?
    [readChar. if ?="?" [sourceCodes:= sourceCodes; 74. readChar. scan]
      else [sourceCodes:= sourceCodes; 28. scan]]]
else [if ?="=" [sourceCodes:= sourceCodes; 11. readChar. scan]
else [if ?="." [sourceCodes:= sourceCodes; 17. readChar. scan]
else [if ?="=' [sourceCodes:= sourceCodes; 21. readChar. scan]
else [if ?="#' [sourceCodes:= sourceCodes; 22. readChar. scan]
else [if ?="<" [sourceCodes:= sourceCodes; 25. readChar. scan]
else [if ?=">" [sourceCodes:= sourceCodes; 26. readChar. scan]
else [if ?="!' [sourceCodes:= sourceCodes; 27. readChar. scan]
else [if ?="!" [sourceCodes:= sourceCodes; 32. readChar. scan]
else [if ?="{" [sourceCodes:= sourceCodes; 33. readChar. scan]
else [if ?="}" [sourceCodes:= sourceCodes; 34. readChar. scan]
else [if ?="[" [sourceCodes:= sourceCodes; 35. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 37. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 38. readChar. scan]
else [if ?="[" [sourceCodes:= sourceCodes; 39. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 40. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 41. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 44. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 45. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 46. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 47. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 48. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 49. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 51. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 52. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 53. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 54. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 55. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 56. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 57. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 60. readChar. scan]
else [if ?="\" [sourceCodes:= sourceCodes; 61. readChar. scan]
else [if ?="~" [sourceCodes:= sourceCodes; 62. readChar. scan]]
else [if ?="?" [sourceCodes:= sourceCodes; 63. readChar. scan]]
else [if ?="€" [sourceCodes:= sourceCodes; 64. readChar. scan]]
else [if ?="¢" [sourceCodes:= sourceCodes; 65. readChar. scan]]
else [if ?="$" [sourceCodes:= sourceCodes; 66. readChar. scan]]
else [if ?="\" [sourceCodes:= sourceCodes; 67. readChar. scan]]
else [if ?="$" [sourceCodes:= sourceCodes; 68. readChar. scan]]
else [if ?="≤" [sourceCodes:= sourceCodes; 69. readChar. scan]]
else [if ?="≠" [sourceCodes:= sourceCodes; 70. readChar. scan]]
else [if ?="∀" [sourceCodes:= sourceCodes; 71. readChar. scan]]
else [if ?="⊲" [sourceCodes:= sourceCodes; 72. readChar. scan]]
else [if ?="⊳" [sourceCodes:= sourceCodes; 73. readChar. scan]]
else [if ?="⊨" [sourceCodes:= sourceCodes; 74. readChar. scan]]
else [if ?="⫤" [sourceCodes:= sourceCodes; 75. readChar. scan]]
else [if ?=" homic" [sourceCodes:= sourceCodes; 76. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 77. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 78. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 79. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 80. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 81. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 82. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 83. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 84. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 85. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 86. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 87. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 88. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 89. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 90. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 91. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 92. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 93. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 94. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 95. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 96. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 97. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 98. readChar. scan]]
else [if ?="|" [sourceCodes:= sourceCodes; 99. readChar. scan]]

NAME CONTROLLER

new nameKind `names and their attributes
| "name" → text
| "kind" → ("variable", "constant", "data", "program", "channel", "input",
  "output", "unit", "dictionary", "synonym", "forward", "\"")
| "memo" → text
| "scope" → nat
| "relativeAddress" → nat `variable or constant
| "value" → all `variable or constant
| "source" → text `source text
| "codes" → *nat; 99 `scan codes; end
| "names" → *[text] `names mentioned in source
| "numbers" → *nat `numbers mentioned in source
| "texts" → *[text] `texts mentioned in source
| "object" → *nat `object code for data and program names

new nameDefault:= "name" → ""
  "kind" → ""
  "memo" → ""
  "scope" → 0
  "relativeAddress" → 0
  "value" → 0
  "source" → ""
  "codes" → 99
  "names" → nil
  "numbers" → nil
  "texts" → nil
  "object" → nil.

new nameStack: [*nameKind] `persistent names at scope 0, predefined names first..
  := [( "name" → "predefined" ` should be all predefined names; just 6 for now
    "kind" → "dictionary"
  ]
| “memo” → “the predefined dictionary”. |
| nameDefault); |

( “name” → “predefined\session” |
| “kind” → “data” |
| “memo” → “session: text data The join of all texts from channel keys ”; |
| “since the start of a session.” |
| nameDefault); |

( “name” → “predefined\keys” |
| “kind” → “input” |
| “memo” → “keys? text! “_ channel To the program that monitors key presses;”; |
| “it is an output channel; to all other programs, it is an input channel.” |
| nameDefault); |

( “name” → “predefined\screen” |
| “kind” → “output” |
| “memo” → “screen? text! “_ channel To the screen, it is an input channel;” |
| “to all other programs, it is an output channel.” |
| nameDefault); |

( “name” → “predefined\bin” |
| “kind” → “constant” |
| “memo” → “bin:= ⊤, ⊥ constant The binary values.” |
| nameDefault); |

( “name” → “predefined\char” |
| “kind” → “constant” |
| “memo” → “char data The characters.” |
| nameDefault); |

( “name” → “predefined\rand” |
| “kind” → “dictionary” |
| “memo” → “rand \ dictionary containing three definitions.” |
| nameDefault); |

( “name” → “predefined\var" `was predefined\var but it is now hidden |
| “kind” → “variable” |
| nameDefault); |

( “name” → “predefined\rand\next” |
| “kind” → “program” |
| “memo” → “next program Assigns a hidden variable to the next value ”; |
| “in a random sequence.” |
| nameDefault); |

( “name” → “predefined\rand\Int” |
| “kind” → “data” |
| “memo” → “Int: int→int→int data A function that is dependent on a hidden ”; |
| “variable, and is reasonably uniform over the interval from ”;
“(including) the first argument to (excluding) the second”; “argument.”

(nameDefault);

(“name” → “predefined\|rand\|Real”
| “kind” → “data”
| “memo” → “Real: real→real→real data A function that is dependent on a “;
| “hidden variable, and is reasonably uniform over the interval “;
| “between the arguments.”

(nameDefault)

new nSx: nat, –1:= –1. `nameStack index. –1 for not present
new scopeStack: *nat:= 0. `indexes into nameStack. 0 is start of persistent scope
new sourceStart: nat:= 0. `starting index for saving source of persistent definitions
new objectStart: nat:= 0. `starting index for saving object of persistent definitions
new nameCode: 200..300:= 299.
new name: text:= “”.
new newName: text:= “”.

new nameControl

["use: name nameCode nameStack nSx scopeStack
 `assign: error nameStack nSx scopeStack
 `output: msg

new localLookup `find name in current scope; if unfound, nSx:= –1
["use: name nameStack scopeStack
 `assign: nSx
 nSx:= #nameStack.
 loop [nSx:= nSx–1.
   if nSx ≥ scopeStack_ (≡ scopeStack–1)
   [if nameStack nSx “name” ≠ name [loop]]
   else [nSx:= –1]].`end of localLookup

new globalLookup `assign nSx to topmost name in nameStack; if unfound, nSx:= –1
["use: name nameStack scopeStack
 `assign: nSx
 nSx:= #nameStack.
 loop [nSx:= nSx–1.
   if nSx≥0 [if nameStack nSx “name” ≠ name [loop]]
   else [new pName:= “predefined”; name.
   nSx:= #nameStack.
   loop [nSx:= nSx–1.
     if nSx ≥ 0 [if nameStack nSx “name” ≠ pName [loop]]]]].
`end of globalLookup

case nameCode–200
 `nameCode 200: open scope
[scopeStack:= scopeStack; #nameStack]

 `nameCode 201: close scope
nameStack := nameStack (0..scopeStack_→scopeStack–1)).
scopeStack := scopeStack (0..→scopeStack–1)]

`nameCode 202: local lookup name to check that it is new in current scope
[localLookup.
  if nSx ≠ –1
    [msg!“Error 8: ”; name; “ is already defined in this scope”. error:= T ]
  new α. [new αβ:= 2. new α. new αβ:= 3] is legal, but the last definition is disallowed by 202

`nameCode 203: global lookup name to check that it is a dictionary
[globalLookup.
  if nSx = –1
    [msg!“Error 16: ”; name; “ is not defined”. error:= T ]
  else [if nameStack nSx “kind” ≠ “dictionary”
    [msg!“Error 17: ”; name; “ is not a dictionary” error:= T ]]
  in αβγδ 203 checks unnecessarily that a and αβ are dictionaries

`nameCode 204: save simpleName as name
[name:= simpleName]

`nameCode 205: save name as newName
[newName:= name]

`nameCode 206: compound name
[name:= name; “;” simpleName]

`nameCode 207: add name as data
[nameStack:= nameStack.; [“name” → name | “kind” → “data” | nameDefault]]

`nameCode 208: add name as dictionary
[nameStack:= nameStack.; [“name” → name | “kind” → “dictionary” | nameDefault]]

`nameCode 209: populate new dictionary newName from old dictionary name
[msg!“Apology 5: dictionary population is not yet implemented”. error:= T ]

`nameCode 210: add newName as synonym for name
[glogalLookup.
  nameStack:= nameStack.; [“name” → newName | nameStack nSx]]

`nameCode 211: forward definition
[msg!“Apology 3: forward definitions are not yet implemented”. error:= T ]

`nameCode 212: add name as variable
[nameStack:= nameStack.; [“name” → name | “kind” → “variable” | nameDefault]]

`nameCode 213: add name as constant
[nameStack:= nameStack.; [“name” → name | “kind” → “constant” | nameDefault]]

`nameCode 214: add name as channel
[nameStack:= nameStack.; [“name” → name | “kind” → “channel” | nameDefault]]
`nameCode 215: add name as program

\[
\text{nameStack} := \text{nameStack};; \left[ \text{name} \rightarrow \text{name} \mid \text{kind} \rightarrow \text{“program”} \mid \text{nameDefault} \right]
\]

`nameCode 216: add name as unit

\[
\text{nameStack} := \text{nameStack};; \left[ \text{name} \rightarrow \text{name} \mid \text{kind} \rightarrow \text{“unit”} \mid \text{nameDefault} \right]
\]

`nameCode 217: hide name at this nSx. If it's a dictionary, this hides all names within it too

\[
\text{nameStack} := (\text{nSx}; \left[ \text{name} \rightarrow \text{name} \mid \text{“*”} \mid \text{nameDefault} \right])
\]

`nameCode 218: should be concurrent composition, but apology for now

\[
\text{msg}!\text{“Apology 4: concurrent composition is not yet implemented”}. \text{error} := \top
\]

`nameCode 219: add name as input channel

\[
\text{nameStack} := \text{nameStack};; \left[ \text{name} \rightarrow \text{name} \mid \text{kind} \rightarrow \text{“input”} \mid \text{nameDefault} \right]
\]

`nameCode 220: add name as output channel

\[
\text{nameStack} := \text{nameStack};; \left[ \text{name} \rightarrow \text{name} \mid \text{kind} \rightarrow \text{“output”} \mid \text{nameDefault} \right]
\]

`nameCode 221: add name as dictionary

\[
\text{nameStack} := \text{nameStack};; \left[ \text{name} \rightarrow \text{name} \mid \text{kind} \rightarrow \text{“dictionary”} \mid \text{nameDefault} \right]
\]

`nameCode 222: implicit screen

\[
\text{name} := \text{“predefinedScreen”}. \text{globalLookup} \text{`once screen is predefined, replace globalLookup}
\]

`nameCode 223: implicit keys

\[
\text{name} := \text{“predefinedKeys”}. \text{globalLookup} \text{` once keys is predefined, replace globalLookup}
\]

`nameCode 224: global lookup name to check that it is a variable

\[
\text{globalLookup}. \begin{cases}
\text{if nSx} = -1 & \text{[msg}!\text{“Error 1: ”}; \text{name}; \text{“ is not defined.”}. \text{error} := \top \\
\text{else} & \begin{cases}
\text{[if nameStack nSx “kind”} \neq \text{“variable”} & \text{[msg}!\text{“Error 4: ”}; \text{name}; \text{“ is not a variable”}. \text{error} := \top \\
\text{]}\end{cases}
\end{cases}
\]

`nameCode 225: global lookup name to check that it is an (output) channel

\[
\text{globalLookup}. \begin{cases}
\text{if nSx} = -1 & \text{[msg}!\text{“Error 0: ”}; \text{name}; \text{“ is not defined”}. \text{error} := \top \\
\text{else} & \begin{cases}
\text{new kind} := \text{nameStack nSx “kind”}. & \text{if kind} \neq \text{“channel”} \land \text{kind} \neq \text{“output”} \\
\text{[msg}!\text{“Error 18: ”}; \text{name}; \text{“ is not an output channel”}. \text{error} := \top \\
\text{]}\end{cases}
\end{cases}
\]

`nameCode 226: global lookup name to check that it is an (input) channel

\[
\text{globalLookup}. \begin{cases}
\text{if nSx} = -1 & \text{[msg}!\text{“Error 19: ”}; \text{name}; \text{“ is not defined”}. \text{error} := \top \\
\text{else} & \begin{cases}
\text{new kind} := \text{nameStack nSx “kind”}. & \text{if kind} \neq \text{“channel”} \land \text{kind} \neq \text{“input”} \\
\text{[msg}!\text{“Error 20: ”}; \text{name}; \text{“ is not an input channel”}. \text{error} := \top \\
\text{]}\end{cases}
\end{cases}
\]
nameCode 227: global lookup name to check that it has a value
```
[globalLookup.
  if nSx = -1
    [msg!“Error 21: ”; name; “ is not defined”. error:= ⊤]
  else [new kind:= nameStack nSx “kind”.
    if kind ≠ “channel” ∧ kind ≠ “input” ∧ kind ≠ “constant” ∧ kind ≠ “variable”
      ∧ kind ≠ “data” ∧ kind ≠ “unit”
      [msg!“Error 22: ”; name; “ does not have a value”. error:= ⊤]]]
```

nameCode 228: end of a definition. If it's in the persistent scope, save its source
```
[if scopeStack_(↔ scopeStack–1) = 0 `it's in the persistent scope
 [nameStack:= (#nameStack – 1; “source”) → source_(sourceStart;.. ↔ source)
   ↓ nameStack]]
```

nameCode 229: local lookup name to check that it is defined in current scope
```
[localLookup.
  if nSx = -1 [msg!“Error 24: ”; name; “ is not defined in this scope”. error:= ⊤]]
```

nameCode 230: start of a definition. If it's in the persistent scope, save starting index of source
```
[if scopeStack_(↔ scopeStack–1) = 0 [sourceStart:= ↔ source]]
```

else [msg!“Apology 2: compiler error”. stop]]. `end of nameControl

` CODE GENERATOR

` instructions

new STOP:= 0. `STOP: Stop execution.
new GO:= 1. `GO a: Go to address a.
new IF:= 2. `IF a: Pop valueStack. If it's ⊥ go to address a.
new CASE:= 3. `CASE a: Look at top of valueStack. If it's 0, pop.
  ` If not, subtract 1 from it and go to address a.
new CALL:= 4. `CALL a: Push return address and go to address a.
new RETURN:= 5. `RETURN: Pop return address and go to it.
new PRINT:= 7. `PRINT: Pop valueStack and print it.

new actionCode: 300,..999:= 998.
new fixupStack: *nat:= nil. `forward branch address fixup stack
new caseCounterStack: *nat:= nil.
new argCounterStack: *nat:= nil. `counting arguments
`fixupStack and caseCounterStack and argCounterStack could all be one stack
new loaded: *[“nameStackIndex” → nat | “address” → nat]:= nil.

new codeGenerator
```
use: actionCode fixupStack nameStack nat nil nl object
  use: CALL CASE GO IF POP PRINT RETURN STOP
  assign: caseCounterStack error fixupStack object
  output: msg
case actionCode–300
`actionCode 300: after if data
[object:= object; IF; 0. fixupStack:= fixupStack; ↔object – 1]

`actionCode 301: fix up address; end of if-program or if-else-program or new name [ program ]
[object:= object<fixupStack_(↔fixupStack–1)> ↔object.
fixupStack:= fixupStack_(0;..↔fixupStack–1)]

`actionCode 302: after if data [ program ] else
[object:= object; GO; 0. object:= object<fixupStack_(↔fixupStack–1)> ↔object.
fixupStack:= fixupStack<↔fixupStack–1> ↔object–1]

`actionCode 303: Emit CASE and push its fixup address.
[object:= object; CASE; 0. fixupStack:= fixupStack; ↔object – 1]

`actionCode 304: Pop and hold earlier CASE fixup address.
` Emit GO and push its fixup address.
` Fixup held CASE address.
[new fxa:= fixupStack_(↔fixupStack–1). fixupStack:= fixupStack_(0;..↔fixupStack–1).
object:= object; GO; 0; fixupStack:= fixupStack; ↔object – 1.
object:= object<fxa> ↔object]

`actionCode 305: Push 0 onto case counter stack.
[caseCounterStack:= caseCounterStack; 0]

`actionCode 306: Increase top of case counter stack.
[caseCounterStack:= caseCounterStack<↔caseCounterStack–1>
 caseCounterStack_(↔caseCounterStack–1)+1]

`actionCode 307: Pop caseCounterStack and fixup and pop that many GO addresses from the
` fixupStack.
[new cc: nat:= caseCounterStack_(↔caseCounterStack–1).
cc:= cc–1. if cc>0 [loop]]

`actionCode 308: Emit POP.
[object:= object; POP]

`actionCode 309: Emit PRINT “Error 26: case index too large” and emit STOP.
[object:= object; PRINT; STOP] `print message must be added

`actionCode 310: Call program or data. Is object code loaded? If so, emit CALL. If not, emit
` GO around, load it, shift flow addresses, emit RETURN, fixup GO around, emit CALL.
[new i: nat:= ↔loaded.
loop [if i>0 [i:= i–1.
    if loaded_i nameStackIndex = nSx
        [object:= object; CALL; loaded_i “address”]
    else [loop]]
else [new shift:= ↔object+2.
fixupStack := fixupStack; ↔ object+1.
object := object; GO; 0; nameStack nSx “object”.
loaded := loaded; [“nameStackIndex” → nSx | “address” → shift].
\`shift all flow addresses up
new pc: nat := shift. \`program counter
loop [if pc ↔ object

  [case object_pc
    [pc := pc+1] \`0: STOP
    [object := object<pc+1> object_(pc+1) + shift. pc := pc+2] \`1: GO a
    [object := object<pc+1> object_(pc+1) + shift. pc := pc+2] \`2: IF a
    [object := object<pc+1> object_(pc+1) + shift. pc := pc+2] \`3: CASE a
    [object := object<pc+1> object_(pc+1) + shift. pc := pc+2] \`4: CALL a
    [pc := pc+1] \`5: RETURN
    [pc := pc+1] \`6: POP
    [pc := pc+1] \`7: PRINT
  else [msg!“Apology 7: compiler error”. stop].
  loop]]

object := object; RETURN.
object := object<fixupStack_(↔fixupStack–1)> ↔ object.
fixupStack := fixupStack_(0;↔fixupStack–1).
object := object; CALL; shift]]]

\`actionCode 311: emit forward GO
[object := object; GO; 0. fixupStack := fixupStack; ↔ object – 1]

\`actionCode 312: emit RETURN - end of program definition or named program
[object := object; RETURN]

\`actionCode 313: emit CALL to first name in topmost scope - end of named program
[object := object; CALL; nameStack (scopeStack_(↔scopeStack–1)) “objectStart”]

\`actionCode 314: end of a program or data definition. If it's in the persistent scope, save its
\` object, shifting the flow addresses back to 0 origin.
[if scopeStack_(↔scopeStack–1) = 0 \`it's in the persistent scope
  [nameStack := (nSx; “object”) → object_(objectStart..↔object) \| nameStack.
  new pc: nat := 0. \`program counter
  loop [if pc ↔ object – objectStart
      [case nameStack nSx “object” _ pc

      [pc := pc+1] \`0: STOP

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`1: GO a

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`2: IF a

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`3: CASE a

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`4: CALL a

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`5: RETURN

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`6: POP

      [nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1>
        nameStack nSx “object” _ (pc+1) – objectStart
        nameStack.
      pc := pc+2] \`7: PRINT

      else [msg!“Apology 7: compiler error”. stop].
      loop]]]
⟦nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1> nameStack nSx “object” _ (pc+1) – objectStart

| nameStack.
| pc := pc+2⟧ 3: CASE a

⟦nameStack := (nSx; “object”) → nameStack nSx “object” <pc+1> nameStack nSx “object” _ (pc+1) – objectStart

| nameStack.
| pc := pc+2⟧ 4: CALL a

⟦pc := pc+1⟧ 5: RETURN

⟦pc := pc+1⟧ 6: POP

⟦pc := pc+1⟧ 7: PRINT

else [msg!“Apology 8: compiler error”. stop]]

loop]]

```
`actionCode 315: start of a program or data definition.
` If it's in the persistent scope, save starting index of object
[if scopeStack_⟨⟩ scopeStack–1) = 0 [objectStart:= ↔ object]]

else [msg!“Apology 1: compiler error”. stop]]. `end of codeGenerator
```

`PARSER
```
` cheap LL(1) grammar -- no director sets. For efficiency, the productions (except possibly the 
` last) for each parse code (nonterminal) should be placed in order of decreasing frequency.

```
100 program 0 sequent moresequents
101 moresequents 1 . program
102 sequent 3 phrase parallelphrases
103 parallelphrases 4 || 218 sequent
5 empty
104 phrase 6 new 230 simplename 204 afternewname 228
7 old simplename 204 compounder 229 217
8 [ 200 program 201 ]
9 if data 300 [ 200 program 201 ] ifelse
10 case data 303 305 [ 200 program 201 ] morecases caseelse 307
11 for 200 simplename 204 213 : data [ program 201 ]
12 plan 200 simplename 204 parameterkind [ program 201 ] arguments
13 ! 222 data
14 ? 223 inputafterq
15 simplename 204 phraftsimname
105 afternewname 16 : 202 212 data := data
17 { 315 202 207 data 314 }
18 := 202 213 data
ProTem Implementation

```
\` 19 \ ? 202 214 data ! data
\` 20 [ [ 315 202 215 200 311 program 312 301 201 314 ] ]
\` 21 \ afterbackslash
\` 22 #1 202 216
\` 23 simplename 202 205 204 compounder 210
\` 24 empty 202 211
\` 106 afterbackslash 25 simplename 203 206 afternewname
\` 26 \ 202 208 simplename 204 compounder 203 209
\` 27 empty 202 208
\` 107 compounder 28 \ simplename 203 206 compounder
\` 29 empty
\` 108 elseif 30 else [ [ 200 302 program 301 201 ] ]
\` 31 empty 301
\` 109 case elseif 32 else [ [ 200 program 201 ] ]
\` 33 empty 309
\` 110 more cases 34 [ [ 304 303 306 200 program 201 ] ] more cases
\` 35 empty 304 308 306
\` 111 parameter kind 36 : 213 data
\` 37 := 212 data
\` 38 ! 220 data
\` 39 ? 219 data
\` 40 \ 221
\` 112 phrafsimname 41 [ [ 200 215 311 program 312 301 313 201 ] ]
\` 42 compounder aftername
\` 113 after name 43 := 224 data
\` 44 ! 225 data
\` 45 ? 226 inputafterq
\` 46 310 arguments
\` 114 input afterq 47 ! echo
\` 48 data ( data ) data after pattern
\` 115 after pattern 49 ! echo
\` 50 empty
\` 116 echo 51 simplename compounder 225
\` 52 empty 222
\` 117 arguments 53 number arguments
\` 54 \ arguments
\` 55 text arguments
\` 56 \ arguments
\` 57 \ arguments
\` 58 value 200 simplename : 204 212 data := data [ [ program 201 ] ] arguments
\` 59 { data } arguments
\` 60 [ data ] arguments
\` 61 ( data ) arguments
\` 62 ( 200 simplename : 204 213 data . data 201 ) arguments
\` 63 simplename 204 specific and arguments
\` 64 empty
\` 118 data 65 data6 more data
\` 119 more data 66 \ = data = data
\` 67 empty
\` 120 data6 68 data5 more data6
```
\begin{verbatim}
`121 moredata6 69 = data5 moredata6
`122 data5 70 ≠ data5 moredata6
`123 moredata5 71 < data5 moredata6
`124 data4 72 > data5 moredata6
`125 moredata4 73 ≤ data5 moredata6
`126 data3 74 ≥ data5 moredata6
`127 moredata3 75 : data5 moredata6
`128 data2 76 :: data5 moredata6
`129 moredata2 77 ∈ data5 moredata6
`130 data1 78 empty
`131 moredata1 79 data4 moredata5
`132 moredata5 80 , data4 moredata5
`133 moredata5 81 ... data4 moredata5
`134 empty 82 | data4 moredata5
`135 empty 83 < data4 moredata5
`136 empty 84 empty
`137 data4 85 data3 moredata4
`138 moredata4 86 + data3 moredata4
`139 moredata4 87 – data3 moredata4
`140 moredata4 88 ;; data3 moredata4
`141 ; data3 moredata4
`142 ;.. data3 moredata4
`143 ; empty
`144 empty 89 ' data3 moredata4
`145 empty 90 data3 moredata4
`146 empty 91 * data2 moredata3
`147 moredata3 92 empty
`148 moredata3 93 data2 moredata3
`149 moredata3 94 × data2 moredata3
`150 moredata3 95 / data2 moredata3
`151 moredata3 96 ∧ data2 moredata3
`152 moredata3 97 ∨ data2 moredata3
`153 empty
`154 data2 98 empty
`155 # data2
`156 – data2
`157 ∼ data2
`158 + data2
`159 □ data2
`160 ; data2
`161 * data2
`162 ≤ data2
`163 $ data2
`164 ↔ data2
`165 data1 moredata2
`166 moredata2 110 * data2 moredata2
`167 moredata2 111 – data2 moredata2
`168 moredata2 112 ∼ data2 moredata2
`169 moredata2 113 ∧ data2 moredata2
`170 moredata2 114 empty
`171 moredata2 115 data0 moredata1
`172 moredata2 116 % moredata1
`173 moredata2 117 ? moredata1
`174 moredata2 118 ?? moredata1
\end{verbatim}
```
119  _data0 moredata1
120  @ data0 moredata1
121  & data0 moredata1
122  310 arguments

132  data0
123  number
124  ∞
125  text
126  ⊤
127  ⊥
128  ?
129  ??
130  value 200 simplename : 204 212 data := data [] program 201 []
131  { data }
132  [ data ]
133  ( data )
134  { 200 simplename : 204 213 data . data 201 }
135  simplename 204 specificand

133  specificand
136  ¶ 200 207 data 201 ¶
137  compounder 227
```

**new productions:** `each production is in reverse order`

[101; 102]; `0 program 100
[100; 17]; `1 moresequents 101
[nil]; `2
[103; 104]; `3 sequent 102
[102; 218; 43]; `4 parallelphrases 103
[nil]; `5
[228; 105; 204; 10; 230; 4]; `6 phrase 104
[217; 229; 107; 204; 10; 5]; `7
[40; 201; 100; 200; 39]; `8
[108; 40; 201; 100; 200; 39; 300; 118; 3]; `9
[307; 109; 110; 40; 201; 100; 200; 39; 305; 303; 118; 0]; `10
[40; 201; 100; 39; 118; 18; 213; 204; 10; 200; 2]; `11
[117; 40; 201; 100; 39; 111; 204; 10; 200; 6]; `12
[118; 222; 27]; `13
[114; 223; 28]; `14
[112; 204; 10]; `15
[118; 20; 118; 212; 202; 18]; `16 afternewname 105
[73; 314; 118; 207; 202; 315; 72]; `17
[118; 213; 202; 20]; `18
[118; 27; 118; 214; 202; 28]; `19
[40; 314; 201; 301; 312; 100; 311; 200; 215; 202; 315; 39]; `20
[106; 41]; `21
[216; 202; 30]; `22
[210; 107; 204; 205; 202; 10]; `23
[211; 202]; `24
[105; 206; 203; 10]; `25 afterbackslash 106
[209; 203; 107; 204; 10; 208; 202; 41] `26
[208; 202]; `27
[107; 206; 203; 10; 41]; `28 compounder 107
new ntStart:= ` for each parse code (nonterminal), its first production number, plus one more
0; 1; 3; 4; 6; 16; 25; 28; 30; 32; 34; 36; 41; 43; 47; 49; 51; 53; 65; 66; 68; 69; 79; 80; 85;
86; 93; 94; 99; 110; 115; 116; 123; 136; 138.

new parseStack: *(0,..1000):= 999. `bottom; scan codes, parse codes, name codes, action codes
new top: nat:= 999.
new pop [parseStack:= parseStack_(0,..↔parseStack−1). top:= parseStack (↔parseStack − 1)].
new sCx: nat:= 0. `sourceCodes index
new nextScanCode [sCx:= sCx+1. scanCode:= sourceCodes_sCx].
new legals: text:= “”. `for good error messages

new parse ` expects a nonempty parseStack and scanCode
[ `use: nat nil ntStart productions scanCodeText sCx sourceCodes
 `assign: actionCode error legals nameCode parseStack sCx
 `call: codeGenerator nameControl pop
 `output: msg

if top<100 ` scan code (terminal)
 [if scanCode=top [pop. nextScanCode. legals:= “”. parse]
 else [if scanCode=99 [msg!”Error 11: input ended before program”]
 else [msg!”Error 12: wrong symbol “; ~scanCodeText_scanCode;
 “ Should be “; ~scanCodeText_top].
 error:= ⊤ ]]
else [if top<200 `parse code (nonterminal)
 [new p: nat:= ntStart_(top−100). ` start checking at production number p
 new q:= ntStart_(top−99). ` end checking before production number q
 loop [new rp:= productions_p. ` rp is the reversed production: a list of scan codes
 `(terminals), parse codes (nonterminals), name codes, and action codes
 new produce [parseStack:= parseStack_(0,..↔parseStack−1); ~rp.
 top:= parseStack (↔parseStack − 1)].
 if rp = [nil] [pop. parse]
 else [new prodHead:= rp (#rp − 1).
 if prodHead≥100 `parse code or name code or action code
 [produce. parse]
 else [ ` production starts with a scan code (terminal)
 if prodHead=scanCode [produce. parse]
 else [legals:= legals; “ “; scanCodeText prodHead.
 p:= p+1.
 if p < q [loop]
 else [if scanCode=99 ` end of input file
 [msg!”Error 9: input ended before program”]]]
else [msg:"Error 10: wrong symbol ";
   ~scanCodeText_scanCode;
   " Should be one of"; legals].

error:= ⊤⟧⟧⟧⟧
`5: RETURN
⟦pc:= pc+1. loop⟧

`6: POP
⟦pc:= pc+1. loop⟧

`7: PRINT
⟦pc:= pc+1. loop⟧

else [msg!“Apology 6: compiler error”. stop]]]. `end of optimize

` EXECUTER

new execute
[^
use: all nat nil object
`call: ok
`input: keys
`output: msg
new valueStack: *{all}= nil.
new scopeStack: *nat= 0. `scope numbers
new baseStack: *nat= 0. `indexes valueStack
new display: *nat= 0. `indexes valueStack
new returnAddressStack: *nat= nil. `valueStack and returnAddressStack could be one stack
new pc: nat= 0. `program counter
loop [if pc=⇒object
    [case object_pc
        `0: STOP - Stop execution.
        [ok]
        `1: GO a - Go to address a.
        [if pc+1=⇒object [pc:= object_(pc+1). loop]
        else [msg!“Apology 16: execution error”. stop]]
        `2: IF a - Pop valueStack. If it's ⊥ go to address a.
        [if pc+1=⇒object
            [new top:= ~valueStack_⇒valueStack–1).
            valueStack:= valueStack_(0;..⇒valueStack–1).
            if top=⊥ [pc:= object_(pc+1)] else [pc:= pc+2].
            loop]
        else [msg!“Apology 17: execution error”. stop]]
        `3: CASE a: Look at top of valueStack. If it's 0, pop.
        `             If not, subtract 1 from it and go to address a.
        [if pc+1=⇒object
            [if ~valueStack_⇒valueStack–1) = 0 [valueStack:= valueStack_(0;..⇒valueStack–1)]
            else [valueStack:= valueStack⇒valueStack–1⇒ ~valueStack_⇒valueStack–1) – 1.
            pc:= object_(pc+1)].
            loop]
        else [msg!“Apology 18: execution error”. stop]]

`4: CALL a: Push return address and go to address a.
   [if pc+1 <= object
   [returnAddressStack := returnAddressStack; pc+2. pc := object_(pc+1). loop]
   else [msg!“Apology 19: execution error”. stop]]

`5: RETURN: Pop return address and go to it.
   [pc := returnAddressStack_(<<returnAddressStack–1).
   returnAddressStack := returnAddressStack_(0..<<returnAddressStack–1). loop]

`6: POP: Pop valueStack.
   [valueStack := valueStack_(0..<<valueStack–1). loop]

`7: PRINT: Pop valueStack and print it. For now, print apology.
   [msg!“Apology 15: PRINT op-code not implemented”]
   else [msg!“Apology 15: execution error”. stop]]]. `end of execute

` MAIN - EXECUTION STARTS HERE

` get login name and password
new login: text: ““. new password: text: ““.
!“Please enter your login name followed by end: ”. ?!. login:= ?.
pswd [[“Please enter your password followed by end: ”.
   getChar [[? “” (char) “”.
     if ?=end [[password=“” [[“Empty password. Try again.”; nl. pswd]
       else [[’nl]]]
     else [[if ?=delete [[password=“”
       [[password:= password_(0..<<password–1). !delete]]]
     else [password:= password; ?. !“•”].
       getChar]].]
   login and password must be checked and used to connect to saved persistent scope

`repeatedly, forever, compile, optimize, and execute program from keys
loop [[`drain all persistent input channels. It should be
   `for i: 0;..#nameStack
     ` [if nameStack i “kind” = “channel” v nameStack i “kind” = “input”
     ` [drain SOMETHING]].
   `but for now,
   drain keys.
   sourceTexts:= nil. sourceNames:= nil.
   !nl; “⇒ ”. `the prompt
   readChar. scan. `reads and scans and prettifies and prints input until end
   `producing source and sourceCodes and sourceNumbers and sourceTexts and sourceNames
   if ¬error
     [scanCode:= sourceCodes_0. object:= nil. loaded:= nil.
      parseStack:= 999; 100. top:= 100. `bottom; program
      parse]. `parse calls nameControl and codeGenerator
   if ¬error [optimize].
if –error [execute].
loop]. `end of ProTem implementation

new printObject `for debugging and ctl d; not called from anywhere
[`use: nl object
`output: msg screen
new pc: nat:= 0. `program counter
loop [if pc<↔object
  [case object_pc
    [!pc; “: STOP”; nl. pc:= pc+1. loop]]
    [!pc; “: GO ”. if pc+1<↔object ![object_(pc+1); nl. pc:= pc+2. loop]
      else [msg!’Apology 11: compiler error”. stop]]
    [!pc; “: IF ”. if pc+1<↔object ![object_(pc+1); nl. pc:= pc+2. loop]
      else [msg!’Apology 12: compiler error”. stop]]
    [!pc; “: CASE ”. if pc+1<↔object ![object_(pc+1); nl. pc:= pc+2. loop]
      else [msg!’Apology 13: compiler error”. stop]]
    [!pc; “: CALL ”. if pc+1<↔object ![object_(pc+1); nl. pc:= pc+2. loop]
      else [msg!’Apology 14: compiler error”. stop]]
    [!pc; “: RETURN”; nl. pc:= pc+1. loop]
    [!pc; “: POP”; nl. pc:= pc+1. loop]
    [!pc; “: PRINT”; nl. pc:= pc+1. loop]
  else [msg!’Apology 10: compiler error”. stop]]] `end of printObject