ProTem Implementation

Eric Hehner

` The ProTem programming system is described at hehner.ca/PT.pdf.
` This is its implementation, written in ProTem.
` Here is the map of name definitions and uses.
` Still to do: data; assignment; \\ ; || ; forward; predefined; arguments; operators
` Symbol level deleting and editing needs to be integrated with reading and scanning.
` Bootstrap through Turing or C.
` Unused error numbers: 25,..∞ ; Unused apology numbers: 21,..∞ .

` 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

new *nameKind*:= “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; *esc*
| “names” → *[*text*] `names mentioned in source
| “numbers” → **nat* `numbers mentioned in source
| “texts” → *[*text*] `texts mentioned in source
| “object” → **nat*. `object code for data or program

new *nameDefault*:= “name” → “”
| “kind” → “”
| “memo” → “”
| “scope” → 0
| “relativeAddress” → 0
| “value” → 0
| “source” → “”
| “codes” → 99 `*esc*
| “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\rand\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 *error*: *bin*:= ⊥.` Has an error been detected?
new *object*: *nat*:= *nil*.`the object code we are producing for execution

`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 *CALL*:= 3.`CALL a: Push return address and go to address a.
new *RETURN*:= 4.`RETURN: Pop return address and go to it.
new *POP*:= 5.`POP: Pop *valueStack*.
new *PRINT*:= 6.`PRINT: Pop *valueStack* and print it.

new *scanCodeText*:=`for good error messages
[“**else**”]; [“**for**”]; [“**if**”]; [“**new**”]; [“**old**”]; [“**plan**”]; [“**value**”]; [“number”]; [“text”];
[“name”]; [“\\”];[“‘”]; [“,”]; [“,..”]; [“;”]; [“;;”]; [“;..”]; [“.”]; [“:”]; [“::”];[“:=”];[“=”];[“≠”];
[“<”]; [“>”]; [“≤”]; [“≥”]; [“!”]; [“?”]; [“#”]; [“#1”]; [“(”]; [“)”]; [“{”]; [“}”]; [“[”]; [“]”];
[“⟨”]; [“⟩”]; [“⟦”]; [“⟧”]; [“\”]; [“|”]; [“||”]; [“∞”]; [“%”]; [“&”]; [“+”]; [“–”]; [“×”]; [“/”];
[“_”]; [“→”]; [“↔”]; [“⊤”]; [“⊥”]; [“∧”]; [“∨”]; [“^”]; [“^^”]; [“@”]; [“*”]; [“~”]; [“↯”];
[“¢”]; [“$”]; [“∈”]; [“□”]; [“◅”]; [“▻”]; [“⊨”]; [“⫤”]; [“❪”]; [“❫”]; [“??”]; [“〈”];[“〉”];[“¸_”];
[“‾”].

new *compile*
⟦`assign: *error object*
`use: *bold CALL esc GO IF italic nat nil nl POP PRINT RETURN STOP tab*
`input: *keys*
`output: *msg screen*
`call: *stop*

`*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**

` scan codes (terminals)
` 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18
` **else for if new old plan value** number text simplename \\ ‘ , ,.. ; ;; ;.. . :

` 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
` :: := = ≠ < > ≤ ≥ ! ? # #1 ( ) { } [ ] ⟨ ⟩ ⟦ ⟧ \ | || ∞ % & + –

```
`      49 50 51 52 53 54 55 56 57 58  59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78
`       × / _ → ↔ ⊤ ⊥ ∧ ∨ ^ ^^ @ * ~ ≠ ¢ $ ∈ □ ◅ ▻ ⊨ ⫤ ⦇ ⦈ ?? ⟨ ⟩ ‚_ ¯
`
`      symbol alternatives
`      11 22  25  26  37 38 39 40 48 49  52  53 56 57 63 66 67 68  69 70  71  72 73 75  76  78
`       ' /= <= >= <: :> [| |]  - >< -> <> /\ \/  //  :~ [] <|  |> |=  =|  (|  |)  <. .>  -.

new source: text:= "". `so persistent definitions can be saved
new scanCode: 0,..100:= 99. `esc
new sourceCodes: *nat:= nil. `string of scan codes.
      `After source code 7 is an index into sourceNumbers;
      `after source code 8 is an index into sourceTexts;
      `after source code 9 is an index into sourceNames.
new simpleName: text:= "".
new sourceNames: *[text]:= nil. `sequence of source names
new txt: text:= "".
new sourceTexts: *[text]:= nil. `sequence of source texts
new sourceNumbers: *nat:= nil. `string of source numbers

new readChar [[? "" ⟨char⟩ "" !. source:= source; ?]].

new scan
 [[`use: bold esc italic nl source tab
  `assign: error number simpleName source sourceCodes sourceNames sourceTexts txt
  `call: readChar
  `output: msg
  `pre: ? has been output and joined to source but not scanned
  `post:  ?=esc

  new fancy [[`pre: ? is within the name;  it has been output but not scanned
              `post: ? = (first character after fancy name)
              if ?="»" [[simpleName:= simpleName; "».
                    sourceCodes:= sourceCodes; 9; ↔sourceNames.
                    sourceNames:= sourceNames; [simpleName]. readChar. scan]]
              else [[if ?=">"
                    [[readChar.
                     if ?=">" [[!delete; delete; "». source:= source_(0;..↔source–2); "».
                           simpleName:= simpleName; "».
                           sourceCodes:= sourceCodes; 9; ↔sourceNames.
                           sourceNames:= sourceNames; [simpleName]. readChar. scan]]
                      else [[simpleName:= simpleName; ">". fancy]]]]
                   else [[if ?="«" [[error:= ⊤. msg!"Error 5: bare « within fancy name"]]
                         else [[if ?="<"
                               [[readChar.
                                if ?="<" [[!delete; delete; "«".
                                      source:= source_(0;..↔source–2); "«".
                                      error:= ⊤. msg!"Error 6: bare ≪ within fancy name"]]
                                 else [[simpleName:= simpleName; "<". fancy]]]]
                              else [[if ?=esc [[error:= ⊤. msg!"Error 13: unclosed fancy name"]]
                                    else [[simpleName:= simpleName; ?. readChar. fancy]]]]]]]].
```

```
`end of fancy

new moreText ⟦`pre: input needed
                `post: ? = (first character after text)
                readChar.
                if ?="“" ⟦readChar.
                        if ?="“" ⟦!delete; delete; underline "“". txt:= txt; "“". moreText⟧
                        else ⟦msg!"Error 2: single bare “ within text". error:= ⊤⟧⟧
                else ⟦if ?="”" ⟦readChar.
                                if ?="”" ⟦!delete; delete; underline "”". txt:= txt; "”". moreText⟧
                                else ⟦sourceCodes:= sourceCodes; 8; ↔sourceTexts.
                                        sourceTexts:= sourceTexts; [txt]. scan⟧⟧
                        else ⟦if ?="'"
                                ⟦readChar.
                                  if ?="'" ⟦!delete; delete; "'". txt:= txt; "'". moreText⟧
                                  else ⟦!delete; delete; "”"; ?.
                                        sourceCodes:= sourceCodes; 8; ↔sourceTexts.
                                        sourceTexts:= sourceTexts; [txt]. scan⟧⟧
                                else ⟦if ?=esc ⟦error:= ⊤. msg!"Error 3: unclosed text"⟧
                                        else ⟦txt:= txt; ?. moreText⟧⟧⟧⟧⟧. `end of moreText

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

if ?=esc ⟦sourceCodes:= sourceCodes; 99⟧

else ⟦if (?=" ") ∨ (?=tab) ∨ (?=nl) ⟦readChar. scan⟧

else ⟦if "a" ≤ ? ≤ "Z" ` plain simple name or keyword
        ⟦new sx:= ↔source. simpleName:= ?.
          nameOrKeyword
                ⟦ readChar.
                  if ("a" ≤ ? ≤ "Z") ∨ ("0" ≤ ? ≤ "9")
                   ⟦simpleName:= simpleName; ?. nameOrKeyword⟧
                  else ⟦` see if it's a keyword or a name
                        ` for efficiency, these should be in order of decreasing frequency
                        if simpleName="if" ⟦scanCode:= 2⟧
                        else ⟦if simpleName="else" ⟦scanCode:= 0⟧
                        else ⟦if simpleName="new" ⟦scanCode:= 3⟧
                        else ⟦if simpleName="for" ⟦scanCode:= 1⟧
                        else ⟦if simpleName="plan" ⟦scanCode:= 5⟧
                        else ⟦if simpleName="old" ⟦scanCode:= 4⟧
                        else ⟦if simpleName="value" ⟦scanCode:= 6⟧
                        else ⟦scanCode:= 9⟧⟧⟧⟧⟧⟧⟧⟧⟧. `simplename
                  for n: 0;..↔simpleName + 1 ⟦!delete⟧.
                  if scanCode=9 ⟦! italic simpleName; ?.
                                source:= source_(0;..sx); italic simpleName; ?.
                                sourceCodes:= sourceCodes; 9; ↔sourceNames.
                                sourceNames:= sourceNames; [italic simpleName]⟧
                  else ⟦! bold simpleName; ?. source:= source_(0;..sx); bold simpleName; ?.
                        sourceCodes:= sourceCodes; scanCode⟧.
```

```
                    scan⟧⟧⟧⟧

else ⟦if ?="«" ` fancy name
        ⟦simpleName:= "«". source:= source; "«". readChar. fancy⟧

else ⟦if "0" ≤ ? ≤ "9" ` number
        ⟦new number: real:= ?.
          moreNumber ⟦readChar. if "0" ≤ ? ≤ "9" ⟦number:= number×10 + ?. moreNumber⟧⟧.
          if ?="."
          ⟦readChar.
            if "0" ≤ ? ≤ "9"
             ⟦new denom: nat:= 10.
               moreFraction ⟦number:= number + ?/denom. readChar.
                                 if "0" ≤ ? ≤ "9" ⟦denom:= denom×10. moreFraction⟧⟧⟧.
            sourceCodes:= sourceCodes; 7; ↔sourceNumbers.
            sourceNumbers:= sourceNumbers; number. scan⟧⟧⟧

else ⟦if ?="‾" ` text
        ⟦txt:= "". source:= source; "‾". moreText⟧

else ⟦if ?="'" ` text
        ⟦txt:= "". source:= source; "‾". !delete; "‾". moreText⟧

else ⟦if ?="`" ` comment
        ⟦moreComment ⟦readChar. if ?=esc ⟦scan⟧
                                        else ⟦if ?="`" ∨ ?=nl ⟦readChar. scan⟧
                                                else ⟦moreComment⟧⟧⟧⟧

else ⟦if ?="‘" ⟦!delete; "’". sourceCodes:= sourceCodes; 11. readChar. scan⟧

else ⟦if ?="," ` ,.. or , . or ,_ or ,
        ⟦readChar.
          if ?="." ⟦readChar. if ?="." ⟦sourceCodes:= sourceCodes; 13. readChar. scan⟧
                                  else ⟦sourceCodes:= sourceCodes; 12; 17. scan⟧⟧
          else ⟦if ?="_" ⟦sourceCodes:= sourceCodes; 77. readChar. 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 ⟦if ?=";" ⟦sourceCodes:= sourceCodes; 15. readChar. scan⟧
                  else ⟦sourceCodes:= sourceCodes; 14. scan⟧⟧⟧

else ⟦if ?=":" ` :: 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 [[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 ?="|" [[!delete; delete; "◁". sourceCodes:= sourceCodes; 69.
                          readChar. scan]]
                    else [[if ?="<" ` fancy name
                          [[simpleName:= "«". source:= source:= "«".
                           !delete; delete; "«". readChar. fancy]]
                          else [[if ?=":" [[!delete; delete; "⟨". sourceCodes:= sourceCodes; 37.
                                      readChar. scan]]
                                else [[if ?="." [[!delete; delete; "⟪".
                                            sourceCodes:= sourceCodes; 75. 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 [| 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 |) 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 (
      [[readChar.
```

```
        if ?="|" [[!delete; delete; "(". sourceCodes:= sourceCodes; 72. readChar. scan]]
        else [[sourceCodes:= sourceCodes; 31. scan]]]

else [[if ?="\" ` \/ or \\ or \
      [[readChar.
        if ?="/" [[!delete; delete; "∨". sourceCodes:= sourceCodes; 57. readChar. scan]]
        else [[if ?="\" [[sourceCodes:= sourceCodes; 10. readChar. scan]]
              else [[sourceCodes:= sourceCodes; 41. scan]]]]

else [[if ?="-" ` -> or -, or -
      [[readChar.
        if ?=">" [[!delete; delete; "→". sourceCodes:= sourceCodes; 52. readChar. scan]]
        else [[if ?="," [[!delete; delete; "¬". sourceCodes:= sourceCodes; 78. 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 ?="." `.> or .
      [[readChar. if ?=">" [[sourceCodes:= sourceCodes; 76. readChar. scan]]
                  else [[sourceCodes:= sourceCodes; 17. scan]]]

else [[if ?="'" [[sourceCodes:= sourceCodes; 11. 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** ?=“ϟ” ⟦*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*; 75. *readChar*. *scan*⟧
else ⟦**if** ?=“·⟩” ⟦*sourceCodes*:= *sourceCodes*; 76. *readChar*. *scan*⟧
else ⟦**if** ?=“;̄” ⟦*sourceCodes*:= *sourceCodes*; 78. *readChar*. *scan*⟧
else ⟦*msg*!“Error 7: strange character: ”; ?. *error*:= ⊤⟧
⟧⟧. ` end of *scan*

` NAME CONTROLLER

new *nSx*: *nat*, –1:= –1. `*n*ame*S*tack inde*x*. –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 *savedName*: *text*:= “”.

new *nameControl*
⟦`use: *name nameCode nameStack nSx scopeStack*
`assign: *error nameStack nSx scopeStack*

```
`output: msg
`call: stop

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
  `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

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

if nameCode=200 `open scope
 ⟦scopeStack:= scopeStack; #nameStack⟧

else ⟦if nameCode=201 `close scope
 ⟦nameStack:= nameStack (0;..scopeStack_(↔scopeStack–1)).
   scopeStack:= scopeStack_(0;..↔scopeStack–1)⟧

else ⟦if 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:= ⊤⟧⟧
 `new a\. ⟦new a\b:= 2. new a\. new a\b:= 3⟧ is legal, but the last definition is disallowed by 202

else ⟦if nameCode=203 `global lookup name to check that it is a dictionary
 ⟦globalLookup.
   if nSx = –1
    ⟦msg!"Error 16: "; name; " is not defined".  error:= ⊤⟧
   else ⟦if nameStack nSx "kind" ≠ "dictionary"
          ⟦msg!"Error 17: "; name; " is not a dictionary"  error:= ⊤⟧⟧⟧⟧
  `in a\b\c\d 203 checks unnecessarily that a and a\b are dictionaries

else ⟦if nameCode=204  `save simpleName as name
 ⟦name:= simpleName⟧
```

else ⟦**if** *nameCode*=205 `save *name* as *savedName*
⟦*savedName*:= *name*⟧

else ⟦**if** *nameCode*=206 `compound *name*
⟦*name*:= *name*; "\"; *simpleName*⟧

else ⟦**if** *nameCode*=207 `add *name* as data
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "data" | *nameDefault*]⟧

else ⟦**if** *nameCode*=208 `add *name* as dictionary
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "dictionary" | *nameDefault*]⟧

else ⟦**if** *nameCode*=209 `populate new dictionary *savedName* from old dictionary *name*
⟦*msg*!"Apology 5: dictionary population is not yet implemented". *error*:= ⊤⟧

else ⟦**if** *nameCode*=210 `add *savedName* as synonym for *name*
⟦*globalLookup*.
nameStack:= *nameStack*;; ["name" → *savedName* | *nameStack nSx*]⟧

else ⟦**if** *nameCode*=211 `forward definition
⟦*msg*!"Apology 3: forward definitions are not yet implemented". *error*:= ⊤⟧

else ⟦**if** *nameCode*=212 `add *name* as variable
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "variable" | *nameDefault*]⟧

else ⟦**if** *nameCode*=213 `add *name* as constant
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "constant" | *nameDefault*]⟧

else ⟦**if** *nameCode*=214 `add *name* as channel
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "channel" | *nameDefault*]⟧

else ⟦**if** *nameCode*=215 `add *name* as program
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "program" | *nameDefault*]⟧

else ⟦**if** *nameCode*=216 `add *name* as unit
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "unit" | *nameDefault*]⟧

else ⟦**if** *nameCode*=217 `hide *name* at this *nSx*. If it's a dictionary, this hides all names within it
⟦*nameStack*:= (*nSx*; "name") → *name*; "*" | *nameStack*⟧

else ⟦**if** *nameCode*=218 `should be concurrent composition, but apology for now
⟦*msg*!"Apology 4: concurrent composition is not yet implemented". *error*:= ⊤⟧

else ⟦**if** *nameCode*=219 `add *name* as input channel
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "input" | *nameDefault*]⟧

else ⟦**if** *nameCode*=220 `add *name* as output channel
⟦*nameStack*:= *nameStack*;; ["name" → *name* | "kind" → "output" | *nameDefault*]⟧

else ⟦**if** *nameCode*=221 `add *name* as dictionary

⟦*nameStack*:= *nameStack*;; [“name” → *name* | “kind” → “dictionary” | *nameDefault*]⟧

else ⟦**if** *nameCode*=222 `implicit *screen*
⟦*name*:= “*predefined**screen*”. *globalLookup*⟧ `once *screen* is predefined, replace *globalLookup*
`if *predefined* is redefined, this finds the wrong name

else ⟦**if** *nameCode*=223 `implicit *keys*
⟦*name*:= “*predefined**keys*”. *globalLookup*⟧ ` once *keys* is predefined, replace *globalLookup*
`if *predefined* is redefined, this finds the wrong name

else ⟦**if** *nameCode*=224 `global lookup *name* to check that it is a variable
⟦*globalLookup*.
if *nSx* = –1
⟦*msg*!“Error 1: ”; *name*; “ is not defined.”. *error*:= ⊤⟧
else ⟦**if** *nameStack nSx* “kind” ≠ “variable”
⟦*msg*!“Error 4: ”; *name*; “ is not a variable”. *error*:= ⊤⟧⟧⟧⟧

else ⟦**if** *nameCode*=225 `global lookup *name* to check that it is an (output) channel
⟦*globalLookup*.
if *nSx* = –1
⟦*msg*!“Error 0: ”; *name*; “ is not defined”. *error*:= ⊤⟧
else ⟦**new** *kind*:= *nameStack nSx* “kind”.
if *kind* ≠ “channel” ∧ *kind* ≠ “output”
⟦*msg*!“Error 18: ”; *name*; “ is not an output channel”. *error*:= ⊤⟧⟧⟧⟧

else ⟦**if** *nameCode*=226 `global lookup *name* to check that it is an (input) channel
⟦*globalLookup*.
if *nSx* = –1
⟦*msg*!“Error 19: ”; *name*; “ is not defined”. *error*:= ⊤⟧
else ⟦**new** *kind*:= *nameStack nSx* “kind”.
if *kind* ≠ “channel” ∧ *kind* ≠ “input”
⟦*msg*!“Error 20: ”; *name*; “ is not an input channel”. *error*:= ⊤⟧⟧⟧⟧

else ⟦**if** *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*:= ⊤⟧⟧⟧⟧

else ⟦**if** *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*⟧⟧

else ⟦**if** *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*:= ⊤⟧⟧

else ⟦**if** *nameCode*=230`start of definition. If in the persistent scope, save starting index of source
⟦**if** *scopeStack*_(↔*scopeStack*–1) = 0 ⟦*sourceStart*:= ↔*source*⟧⟧

else ⟦**if** *nameCode*=231 `populate dictionary *savedName* from dictionary *name*, resolving
⟦*msg*!“Apology 18: dictionary population is not yet implemented”. *error*:= ⊤⟧

else ⟦*msg*!“Apology 2: compiler error”. *stop*⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧.
`end of *nameControl*

` CODE GENERATOR

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

new *codeGenerator*
⟦`use: *actionCode fixupStack nameStack nat nil nl object*
`use: *CALL GO IF POP PRINT RETURN STOP*
`assign: *error fixupStack object*
`output: *msg*
`call: *stop*

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

if *actionCode*=300 `after **if** data
⟦*object*:= *object*; *IF*; 0. *fixupStack*:= *fixupStack*; ↔*object* – 1⟧

else ⟦**if** *actionCode*=301 `fix up address;
` end of **if**-program or **if**-**else**-program or **new** name ⟦ program ⟧ or name ⟦ program ⟧
⟦*object*:= *object*◃*fixupStack*_(↔*fixupStack*–1)▹↔*object*.
fixupStack:= *fixupStack*_(0;..↔*fixupStack*–1)⟧

else ⟦**if** *actionCode*=302 `after **if** data ⟦ program ⟧ **else**
⟦*object*:= *object*; *GO*; 0. *object*:= *object*◃*fixupStack*_(↔*fixupStack*–1)▹↔*object*.
fixupStack:= *fixupStack*◃↔*fixupStack*–1▹↔*object*–1⟧

else ⟦**if** actionCode=308 `Emit POP.
⟦*object*:= *object*; *POP*⟧

else ⟦**if** *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
                        ⟦if object_pc=STOP ∨ object_pc=RETURN ∨ object_pc=POP
                            ∨ object_pc=PRINT
                          ⟦pc:= pc+1⟧
                          else ⟦if object_pc=GO ∨ object_pc=IF ∨ object_pc=CALL
                                  ⟦object:= object◅pc+1▻object_(pc+1) + shift.  pc:= pc+2⟧
                                 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⟧⟧⟧

else ⟦if actionCode=311 `emit forward GO
 ⟦object:= object; GO; 0.  fixupStack:= fixupStack; ↔object – 1⟧

else ⟦if actionCode=312 `emit RETURN - end of program definition or named program
 ⟦object:= object; RETURN⟧

else ⟦if actionCode=313 `emit CALL at start of named program
 ⟦object:= object; CALL; ↔object+4⟧

else ⟦if 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
              ⟦`for efficiency, the following should be in order of decreasing frequency
                if nameStack nSx "object" _ pc = STOP ⟦pc:= pc+1⟧

                else ⟦if nameStack nSx "object" _ pc = GO
                        ⟦nameStack:=    (nSx; "object") → nameStack nSx "object" ◅pc+1▻
                                                          nameStack nSx "object" _ (pc+1) – objectStart
                                      | nameStack.
                         pc:= pc+2⟧

                     else ⟦if nameStack nSx "object" _ pc = IF
                             ⟦nameStack:=    (nSx; "object") → nameStack nSx "object" ◅pc+1▻
                                                               nameStack nSx "object" _ (pc+1) – objectStart
                                           | nameStack.
                              pc:= pc+2⟧

                     else ⟦if nameStack nSx "object" _ pc = CALL
```

```
                    ⟦nameStack:=    (nSx; "object") ⇀ nameStack nSx "object" ◅pc+1▻
                                                    nameStack nSx "object" _ (pc+1) – objectStart
                                    | nameStack.
                     pc:= pc+2⟧

              else ⟦if nameStack nSx "object" _ pc: RETURN, POP, PRINT ⟦pc:= pc+1⟧

              else ⟦msg! "Apology 8: compiler error".  stop⟧⟧⟧⟧⟧⟧.
             loop⟧⟧⟧

  else ⟦if 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
`                          2 empty
` 102 sequent              3 phrase parallelphrases
` 103 parallelphrases      4 || 218 sequent
`                          5 empty
` 104 phrase               6 new 230 name afternewname 228
`                          7 old name 229 217
`                          8 ⟦ 200 program 201 ⟧
`                          9 if data 300 ⟦ 200 program 201 ⟧ elsepart
`                         10 for 200 simplename 204 : data ⟦ program ⟧
`                         11 plan simplename parameterkind ⟦ program 201 ⟧ arguments
`                         12 ! 222 data
`                         13 ? 223 inputafterq
`                         14 simplename 204 progaftersimplename
` 105 name                15 simplename 204 compounder
` 106 compounder          16 \ 203 206 name
`                         17 empty
` 107 afternewname        18 : 202 212 data := data
`                         19 (| 315 202 207 data 314 |)
`                         20 := 202 213 data
`                         21 ? 202 214 data ! data
`                         22 ⟦ 202 215 200 313 311 program 201 ⟧ 312 301 316 314
`                         23 \ 202 208
`                         24 \\ 202 208 205 name 203 209
`                         25 #1 202 216
`                         26 simplename 205 204 compounder 210
`                         27 empty 202 211
` 108 elsepart            28 else ⟦ 200 302 program 301 201 ⟧
```

```
`                          29 empty 301
`109 parameterkind         30 : 213 data
`                          31 := 212 data
`                          32 ! 220 data
`                          33 ? 219 data
`                          34 \ 221
`110 progaftersimplename   35 ⟦ 200 215 313 311 program 201 ⟧ 312 301 316 317
`                          36 compounder programaftername
`111 programaftername      37 := 224 data
`                          38 ! 225 data
`                          39 ? 226 inputafterq
`                          40 \\ 203 208 205 name 203 231
`                          41 310 arguments
`112 inputafterq           42 ! echo
`                          43 data ⟨ data ⟩ data afterpattern
`113 afterpattern          44 ! echo
`                          45 empty
`114 echo                  46 simplename 204 compounder 225
`                          47 empty 222
`115 arguments             48 number arguments
`                          49 ∞ arguments
`                          50 text arguments
`                          51 ⊤ arguments
`                          52 ⊥ arguments
`                          53 value 200 simplename : 204 212 data := data ⟦ program 201 ⟧ arguments
`                          54 { data } arguments
`                          55 [ data ] arguments
`                          56 ( data ) arguments
`                          57 ⟨ 200 simplename : 204 213 data . data 201 ⟩ arguments
`                          58 simplename 204 dataaftersimplename arguments
`                          59 empty
`116 dataaftersimplename   60 ⦇ 200 207 data 201 ⦈
`                          61 compounder 227
`117 data                  62 data6 moredata
`118 moredata              63 ⊨ data ⊣ data
`                          64 empty
`119 data6                 65 data5 moredata6
`120 moredata6             66 = data5 moredata6
`                          67 ≠ data5 moredata6
`                          68 < data5 moredata6
`                          69 > data5 moredata6
`                          70 ≤ data5 moredata6
`                          71 ≥ data5 moredata6
`                          72 : data5 moredata6
`                          73 :: data5 moredata6
`                          74 ∈ data5 moredata6
`                          75 empty
`121 data5                 76 data4 moredata5
`122 moredata5             77 , data4 moredata5
`                          78 ,.. data4 moredata5
```

`	79	,_ data4 moredata5
`	80	‾, data3 moredata4
`	81	ǀ data4 moredata5
`	82	⧏ data ⧐data4 moredata5
`	83	empty
`123 data4	84	data3 moredata4
`124 moredata4	85	+ data3 moredata4
`	86	– data3 moredata4
`	87	;; data3 moredata4
`	88	; data3 moredata4
`	89	;.. data3 moredata4
`	90	‘ data3 moredata4
`	91	empty
`125 data3	92	data2 moredata3
`126 moredata3	93	× data2 moredata3
`	94	/ data2 moredata3
`	95	∧ data2 moredata3
`	96	∨ data2 moredata3
`	97	empty
`127 data2	98	# data2
`	99	– data2
`	100	~ data2
`	101	+ data2
`	102	□ data2
`	103	⚡ data2
`	104	* data2
`	105	¢ data2
`	106	$ data2
`	107	↔ data2
`	108	data1 moredata2
`128 moredata2	109	* data2 moredata2
`	110	→ data2 moredata2
`	111	^ data2 moredata2
`	112	^^ data2 moredata2
`	113	empty
`129 data1	114	data0 moredata1
`130 moredata1	115	% moredata1
`	116	? moredata1
`	117	?? moredata1
`	118	_ data0 moredata1
`	119	@ data0 moredata1
`	120	& data0 moredata1
`	121	arguments
`131 data0	122	number
`	123	∞
`	124	text
`	125	⊤
`	126	⊥
`	127	? 223
`	128	?? 223

```
`   129 value 200 simplename : 204 212 data := data ⟦ program 201 ⟧
`   130 { data }
`   131 [ data ]
`   132 ( data )
`   133 ⟨ 200 simplename : 204 213 data . data 201 ⟩
`   134 simplename 204 dataaftersimplename
```

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; 107; 105; 230; 3]; `6 phrase 104
[217; 229; 105; 4]; `7
[40; 201; 100; 200; 39]; `8
[108; 40; 201; 100; 200; 39; 300; 117; 2]; `9
[40; 201; 100; 39; 117; 18; 213; 204; 9; 200; 1]; `10
[115; 40; 201; 100; 39; 109; 204; 9; 200; 5]; `11
[117; 222; 27]; `12
[112; 223; 28]; `13
[110; 204; 9]; `14
[106; 204; 9]; `15 name 105
[105; 206; 203; 41]; `16 compounder 106
[nil]; `17
[117; 20; 117; 212; 202; 18]; `18 afternewname 107
[73; 314; 117; 207; 202; 315; 72]; `19
[117; 213; 202; 20]; `20
[117; 27; 117; 214; 202; 28]; `21
[314; 316; 301; 312; 40; 201; 100; 311; 313; 200; 215; 202; 39]; `22
[208; 202; 41]; `23
[209; 203; 105; 205; 208; 202; 10]; `24
[216; 202; 30]; `25
[210; 106; 204; 205; 9]; `26
[211; 202]; `27
[40; 201; 301; 100; 302; 200; 39; 0]; `28 elsepart 108
[301]; `29
[117; 213; 18]; `30 parameterkind 109
[117; 212; 20]; `31
[117; 220; 27]; `32
[117; 219; 28]; `33
[221; 41]; `34
[317; 316; 301; 312; 40; 201; 100; 311; 313; 215; 200; 39]; `35 progaftersimplename 110
[111; 106]; `36
[117; 224; 20]; `37 programaftername 111
[117; 225; 27]; `38
[112; 226; 28]; `39
[231; 203; 105; 205; 208; 203; 10]; `40
[115; 310]; `41
```

[114; 27]; `42 inputafterq 112
[113; 117; 76; 117; 75; 117]; `43
[114; 27]; `44 afterpattern 113
[*nil*]; `45
[225; 106; 204; 9]; `46 echo 114
[222]; `47
[115; 7]; `48 arguments 115
[115; 44]; `49
[115; 8]; `50
[115; 54]; `51
[115; 55]; `52
[115; 40; 201; 100; 39; 117; 20; 117; 212; 204; 18; 9; 200; 6]; `53
[115; 34; 117; 33]; `54
[115; 36; 117; 35]; `55
[115; 32; 117; 31]; `56
[115; 38; 201; 117; 17; 117; 213; 204; 18; 9; 200; 37]; `57
[115; 116; 204; 9]; `58
[*nil*]; `59
[73; 201; 117; 207; 200; 72]; `60 dataaftersimplename 116
[227; 106]; `61
[118; 119]; `62 data 117
[117; 71; 117; 70]; `63 moredata 118
[*nil*]; `64
[120; 121]; `65 data6 119
[120; 121; 21]; `66 moredata6 120
[120; 121; 22]; `67
[120; 121; 23]; `68
[120; 121; 24]; `69
[120; 121; 25]; `70
[120; 121; 26]; `71
[120; 121; 18]; `72
[120; 121; 19]; `73
[120; 121; 66]; `74
[*nil*]; `75
[122; 123]; `76 data5 121
[122; 123; 12]; `77 moredata5 122
[122; 123; 13]; `78
[122; 123; 77]; `79
[122; 123; 78]; `80
[122; 123; 42]; `81
[122; 123; 69; 117; 68]; `82
[*nil*]; `83
[124; 125]; `84 data4 123
[124; 125; 47]; `85 maredata4 124
[124; 125; 48]; `86
[124; 125; 15]; `87
[124; 125; 14]; `88
[124; 125; 16]; `89
[124; 125; 11]; `90
[*nil*]; `91

[126; 127]; `92 data3 125
[126; 127; 49]; `93 moredata3 126
[126; 127; 50]; `94
[126; 127; 56]; `95
[126; 127; 57]; `96
[*nil*]; `97
[127; 29]; `98 data2 127
[127; 48]; `99
[17; 62]; `100
[127; 47]; `101
[127; 67]; `102
[127; 63]; `103
[127; 61]; `104
[127; 64]; `105
[127; 65]; `106
[127; 53]; `107
[128; 129]; `108
[128; 127; 61]; `109 moredata2 128
[128; 127; 52]; `110
[128; 127; 58]; `111
[128; 127; 59]; `112
[*nil*]; `113
[130; 131]; `114 data1 129
[130; 45]; `115 moredata1 130
[130; 28]; `116
[130; 74]; `117
[130; 131; 51]; `118
[130; 131; 60]; `119
[130; 131; 46]; `120
[115]; `121
[7]; `122 data0 131
[44]; `123
[8]; `124
[54]; `125
[55]; `126
[223; 28]; `127
[223; 74]; `128
[40; 201; 199; 39; 117; 20; 117; 212; 204; 18; 9; 200; 6]; `129
[34; 117; 33]; `130
[36; 117; 35]; `131
[32; 117; 31]; `132
[38; 201; 117; 17; 117; 213; 204; 18; 9; 200; 37]; `133
[116; 204; 9]. `134

new *ntStart*:= ` for each parse code (nonterminal), its first production number, plus one more
0; 1; 3; 4; 6; 15; 16; 18; 28; 30; 35; 37; 42; 44; 46; 48; 60; 62; 63; 65; 66; 76; 77;
84; 85; 92; 93; 98; 109; 114; 115; 122; 135.

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 scanCode scanCodeText sCx sourceCodes
  `assign: actionCode error legals nameCode parseStack sCx top
  `call: codeGenerator nameControl nextScanCode pop stop
  `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:= ⊤⟧⟧⟧⟧⟧
         else ⟦if top<300 ⟦nameCode:= top. pop. nameControl. if –error ⟦parse⟧⟧
               else ⟦if top<999 ⟦actionCode:= top. pop. codeGenerator. if –error ⟦parse⟧⟧
                     else ⟦if top=999 ` bottom
                           ⟦if scanCode≠99 ` esc
                            ⟦msg!"Error 15: wrong symbol "; ~scanCodeText_scanCode;
                                  " Should be one of"; legals.
                              error:= ⊤⟧⟧
                           else ⟦msg!"Apology 0: compiler error". stop⟧⟧⟧⟧⟧⟧. `end of parse

source:= "". sourceCodes:= nil. sourceNumbers:= nil.
sourceTexts:= nil. sourceNames:= nil.
```

```
readChar. scan. `reads and scans and prettifies and prints input until escape is pressed
        `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
].`end of compile

` OPTIMIZER

new optimize
[`use: CALL IF GO object POP PRINT RETURN STOP
 `assign: object
 `call: stop
 `output: msg

  sweep
  [new changed: bin:= ⊥. `only those changes that require a new sweep
    new pc: nat:= 0.`program counter
    loop [if pc<↔object
          [if object_pc = STOP
            [pc:= pc+1. loop]

           else [if object_pc = GO
                  `GO a with a: GO b and a≠b becomes GO b
                  [if object_(object_(pc+1))=GO ∧ object_(pc+1)≠object_(object_(pc+1)+1)
                    [object:= object ◅ pc+1 ▻ object_(object_(pc+1) + 1)]
                   `GO a with a: RETURN becomes RETURN
                   else [if object_(object_(pc+1))=RETURN
                          [object:= object◅pc▻RETURN. pc:= pc+2]
                   `GO a with a: STOP becomes STOP
                   else [if object_(object_(pc+1))=STOP [object:= object ◅ pc ▻ STOP. pc:= pc+2]
                         else [pc:= pc+2]]]].
             loop]

           else [if object_pc = IF `IF a with a: GO b and a≠b becomes IF b
            [if object_(object_(pc+1))=GO ∧ object_(pc+1)≠object_(object_(pc+1)+1)
               [object:= object ◅ pc+1 ▻ object_(object_(pc+1) + 1)]
             else [pc:= pc+2].
            loop]

           else [if object_pc = CALL
            `CALL a with a: GO b and a≠b becomes CALL b
            [if object_(object_(pc+1))=GO ∧ object_(pc+1)≠object_(object_(pc+1)+1)
               [object:= object ◅ pc+1 ▻ object_(object_(pc+1) + 1)]
             `CALL a with a: RETURN becomes GO next (SKIP)
             else [if object_(object_(pc+1))=RETURN
                    [object:= object ◅ pc ▻ GO. object:= object ◅ pc+1 ▻ pc+2. changed:= ⊤]
             `CALL a with a: STOP becomes STOP
             else [if object_(object_(pc+1))=STOP [object:= object ◅ pc ▻ STOP. pc:= pc+2]
             `CALL a followed by RETURN becomes GO a
```

```
        else [[if object_(pc+2)=RETURN [[object:= object ◅ pc ▻ GO. changed:= ⊤]]
        else [[pc:= pc+2]]]]].
       loop]]

    else [[if object_pc: RETURN, POP, PRINT [[pc:= pc+1. loop]]

    else [[msg!"Apology 6: compiler error". stop]]]]].
    if changed [[sweep]]]]]]. `end of optimize

` EXECUTER

new execute
[[`use: all CALL IF GO nat nil object POP PRINT RETURN STOP
 `call: ok stop
 `input: keys
 `output: msg screen
  new valueStack: *[all]:= nil.
  new scopeStack: *nat:= 0. `scope numbers
  new baseStack: *nat:= 0. `synchronous with scopeStack, 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
        [[if object_pc = STOP [[ok]]

         else [[if object_pc = GO
               [[if pc+1<↔object [[pc:= object_(pc+1). loop]]
                 else [[msg!"Apology 16: execution error". stop]]]

         else [[if object_pc = IF `Pop valueStack. If it's ⊥ go to address.
               [[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]]]]

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

         else [[if object_pc = RETURN `Pop return address and go to it.
               [[pc:= returnAddressStack_(↔returnAddressStack–1).
                 returnAddressStack:= returnAddressStack_(0;..↔returnAddressStack–1). loop]]

          else [[if object_pc = POP [[valueStack:= valueStack_(0;..↔valueStack–1). loop]]

          else [[if object_pc = PRINT `Pop valueStack and print it. For now, print apology.
                [[msg!"Apology 15: PRINT op-code not implemented"]]
```

```
        else ⟦msg!"Apology 20: execution error". stop⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧⟧.`end of execute

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

` MAIN - EXECUTION STARTS HERE

```
` get login name and password
new login: text: "". new password: text: "".
!"Please enter your login name followed by escape: ".
?!. if ?="" ⟦!"No login name entered."; nl. stop⟧. !nl. login:= ?.
!"Please enter your password followed by escape: ".
getChar ⟦? "" ⟨char⟩ "".
          if ?=esc ⟦if password="" ⟦!"No password entered."; nl. stop⟧. !nl⟧
          else ⟦if ?=delete ⟦if 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" ∨ nameStack i "kind" = "input"
      `    ⟦drain SOMETHING⟧⟧.
      `but for now,
      drain keys.
      error:= ⊥.
      !nl; "➩ ".  ` the prompt
      compile.
      if ¬error ⟦optimize. execute⟧.
      loop⟧ ` end of ProTem implementation
```