

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.
- ∖ 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

- ∖ *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
- ∖ **case 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
- ∖ :: := = ≠ < > ≤ ≥ ! ? # #1 () { } [] ‹ › ‹› ‹› \ | || ∞ % & +
- ∖ 48 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 99
- ∖ - x / _ → ↔ T ⊥ ∧ ∨ ^ ^^ @ * ~ † ‡ \$ € □ ◁ ▷ ≡ = () ?? () end

- ∖ 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
- ∖ ' /= <= >= <: :> [|] - >< -> <> ^ ∨ // :~ [] <| |> |= =| (|) (: :)

- new scanCodeText**:= ∖for good error messages

```
["case"]; ["else"]; ["for"]; ["if"]; ["new"]; ["old"]; ["plan"]; ["value"]; ["number"];
["text"]; ["name"]; [""]; [","]; [",..."]; [";"]; [";,""]; [";.."]; ["."]; [".,""]; [":"]; [":=""]; [":=""];
["≠"]; [ "< "]; [ "> "]; [ "≤ "]; [ "≥ "]; [ "! "]; [ "?" ]; [ "# "]; [ "#1 "]; [ "(" ]; [ "(" "]; [ "{" ]; [ "{" "]; [ "[" ];
["["]; [ "< "]; [ "> "]; [ "[" ]; [ "]" ]; [ "\ "]; [ "|" ]; [ "|| "]; [ "∞ "]; [ "%" ]; [ "& "]; [ "+" ]; [ "-" ]; [ "x "];
["/"]; [ "_" ]; [ "→ "]; [ "↔ "]; [ "T "]; [ "⊥ "]; [ "∧ "]; [ "∨ "]; [ "∧ "]; [ "^^ "]; [ "@" ]; [ "*" ]; [ "~ "];
["$"]; [ "€ "]; [ "□ "]; [ "◁ "]; [ "▷ "]; [ "≡ "]; [ "=" ]; [ "(" ]; [ ")" ]; [ "??" ]; [ "(" ]; [ ")" ].
```

- new source**: *text*:= “”. ∖so persistent definitions can be saved

- new scanCode**: 0,..100:= 99. ∖end

- new sourceCodes**: *nat:= nil. ∖string 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 txt: text:= “”.
new sourceTexts: *[text]:= nil. `sequence of source texts
new sourceNumbers: *nat:= nil. `string of source numbers
new error: bin:=  $\perp$ . ` 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 txt
`call: readChar
`output: msg
`pre: ? has been output and joined to source but not scanned
`post: ?=end

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; 10;  $\leftrightarrow$ sourceNames.
sourceNames:= sourceNames; [simpleName]. readChar. scan]]
else [[if ?=“>”
[[readChar.
if ?=“>” [[!delete; delete; “>”. source:= source_(0;.. $\leftrightarrow$ source-2); “>”.
simpleName:= simpleName; “>”.
sourceCodes:= sourceCodes; 10;  $\leftrightarrow$ sourceNames.
sourceNames:= sourceNames; [simpleName]. readChar. scan]]
else [[simpleName:= simpleName; “>”. fancy]]]
else [[if ?=“«” [[error:=  $\top$ . msg!“Error 5: bare  $\ll$  within fancy name”]]
else [[if ?=“<”
[[readChar.
if ?=“<” [[!delete; delete; “<”.
source:= source_(0;.. $\leftrightarrow$ source-2); “<”.
error:=  $\top$ . msg!“Error 6: bare  $\ll$  within fancy name”]]
else [[simpleName:= simpleName; “<”. fancy]]]
else [[if ?=end [[error:=  $\top$ . 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:=  $\top$ ]]]
else [[if ?=“_” [[readChar.

```

```

if ?="'"' [!delete; delete; underline "'". txt:= txt; "'". moreText]
else [sourceCodes:= sourceCodes; 9; ↔sourceTexts.
      sourceTexts:= sourceTexts; [txt]. scan]]
else [if ?="'"'
      [readChar.
       if ?="'"' [!delete; delete; underline "'". txt:= txt; "'". moreText]
       else [!delete; delete; "'"; ?.
             sourceCodes:= sourceCodes; 9; ↔sourceTexts.
             sourceTexts:= sourceTexts; [txt]. scan]]]
      else [if ?=end [error:= ⊥. msg!“Error 3: unclosed text”]
            else [txt:= txt; ?. moreText]]]]]. `end of moreText

```

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

```
if ?=end [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=“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]]]]]]]]]. `simpleName
```

```
   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].

```

```
   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; 8; ↔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 ?=nl ∨ ?=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 [[if ?=";" [[sourceCodes:= sourceCodes; 15. readChar. 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 <
    [[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 [[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 (: 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 \
    [[readChar.
    if ?="/" [[!delete; delete; "√". sourceCodes:= sourceCodes; 57. readChar. scan]]
    else [[sourceCodes:= sourceCodes; 41. scan]]]]

else [[if ?="-" ` -> 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 ?="x" [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; 75. readChar. scan]]
else [[if ?="Ⓗ" [[sourceCodes:= sourceCodes; 76. readChar. scan]]
else [[msg!"Error 7: strange character: "; ? . error:= ⊤]]
)))))))))))))))))))))))))))))))))))))))))))))))))))).` end of 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`*end*

 | "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:= T, ⊥ 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” → “predefinedrandInt”
| “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” → “predefinedrandReal”
| “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_( $\leftrightarrow$ scopeStack–1)).
   scopeStack:= scopeStack_ (0;.. $\leftrightarrow$ scopeStack–1)]]

  `nameCode 202: local lookup name to check that it is new in current scope
  [[localLookup.
   if nSx  $\neq$  –1
    [[msg!“Error 8: ”; name; “ is already defined in this scope”. error:=  $\top$ ]]]
  `new a\. [[new a\b:= 2. new a\. new a\b:= 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:=  $\top$ ]]
   else [[if nameStack nSx “kind”  $\neq$  “dictionary”
    [[msg!“Error 17: ”; name; “ is not a dictionary” error:=  $\top$ ]]]]]]
  `in a\b\c\d 203 checks unnecessarily that a and a\b 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”  $\rightarrow$  name | “kind”  $\rightarrow$  “data” | nameDefault]]]

  `nameCode 208: add name as dictionary
  [[nameStack:= nameStack;; [“name”  $\rightarrow$  name | “kind”  $\rightarrow$  “dictionary” | nameDefault]]]

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

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

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

  `nameCode 212: add name as variable
  [[nameStack:= nameStack;; [“name”  $\rightarrow$  name | “kind”  $\rightarrow$  “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
[[nameStack:= nameStack;; [{"name" → name | "kind" → "program" | nameDefault}]

`nameCode 216: add name as unit
[[nameStack:= nameStack;; [{"name" → name | "kind" → "unit" | nameDefault}]

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

`nameCode 218: should be concurrent composition, but apology for now
[[msg!"Apology 4: concurrent composition is not yet implemented". error:= ⊤]]

`nameCode 219: add name as input channel
[[nameStack:= nameStack;; [{"name" → name | "kind" → "input" | nameDefault}]

`nameCode 220: add name as output channel
[[nameStack:= nameStack;; [{"name" → name | "kind" → "output" | nameDefault}]

`nameCode 221: add name as dictionary
[[nameStack:= nameStack;; [{"name" → name | "kind" → "dictionary" | nameDefault}]

`nameCode 222: implicit screen
[[name:= "predefinedScreen". globalLookup]] `once screen is predefined, replace globalLookup

`nameCode 223: implicit keys
[[name:= "predefinedKeys". globalLookup]] `once keys is predefined, replace globalLookup

`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:= ⊤]]]]]]

`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:= ⊤]]]]]]

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

```

```

if  $nSx = -1$ 
   $[[msg!{"Error 19: "; name; " is not defined". error:= \top}]]$ 
else  $[[new\ kind:= nameStack\ nSx\ "kind".$ 
  if  $kind \neq "channel" \wedge kind \neq "input"$ 
     $[[msg!{"Error 20: "; name; " is not an input channel". error:= \top}]]]]]$ 

```

\backslash nameCode 227: global lookup *name* to check that it has a value

$[[globalLookup.$

```

if  $nSx = -1$ 
   $[[msg!{"Error 21: "; name; " is not defined". error:= \top}]]$ 
else  $[[new\ kind:= nameStack\ nSx\ "kind".$ 
  if  $kind \neq "channel" \wedge kind \neq "input" \wedge kind \neq "constant" \wedge kind \neq "variable"$ 
     $\wedge kind \neq "data" \wedge kind \neq "unit"$ 
     $[[msg!{"Error 22: "; name; " does not have a value". error:= \top}]]]]]$ 

```

\backslash nameCode 228: end of a definition. If it's in the persistent scope, save its source

```

 $[[if\ scopeStack\_(\leftrightarrow scopeStack-1) = 0\ \backslash$ it's in the persistent scope
   $[[nameStack:= (\#nameStack - 1; "source") \rightarrow source_(sourceStart;.. \leftrightarrow source)$ 
     $| nameStack]]]$ 

```

\backslash nameCode 229: local lookup *name* to check that it is defined in current scope

$[[localLookup.$

```

if  $nSx = -1$   $[[msg!{"Error 24: "; name; " is not defined in this scope". error:= \top}]]]$ 

```

\backslash nameCode 230: start of a definition. If it's in the persistent scope, save starting index of source

```

 $[[if\ scopeStack\_(\leftrightarrow scopeStack-1) = 0\ [[sourceStart:= \leftrightarrow source]]]$ 

```

else $[[msg!{"Apology 2: compiler error". stop}]]].\backslash$ end of *nameControl*

\backslash CODE GENERATOR

\backslash instructions

new *STOP*:= 0. \backslash STOP: Stop execution.

new *GO*:= 1. \backslash GO a: Go to address a.

new *IF*:= 2. \backslash IF a: Pop *valueStack*. If it's \perp go to address a.

new *CASE*:= 3. \backslash CASE a: Look at top of *valueStack*. If it's 0, pop.
 \backslash If not, subtract 1 from it and go to address a.

new *CALL*:= 4. \backslash CALL a: Push return address and go to address a.

new *RETURN*:= 5. \backslash RETURN: Pop return address and go to it.

new *POP*:= 6. \backslash POP: Pop *valueStack*.

new *PRINT*:= 7. \backslash PRINT: Pop *valueStack* and print it.

new *actionCode*: 300,..999:= 998.

new *fixupStack*: **nat*:= *nil*. \backslash forward branch address fixup stack

new *caseCounterStack*: **nat*:= *nil*.

new *argCounterStack*: **nat*:= *nil*. \backslash counting arguments

\backslash *fixupStack* and *caseCounterStack* and *argCounterStack* could all be one stack

new *loaded*: **["nameStackIndex" \rightarrow nat | "address" \rightarrow 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).
  caseCounterStack:= caseCounterStack_(0;..↔caseCounterStack-1).
  loop [[object:= object<fixupStack_(↔fixupStack-1)> ↔object.
  fixupStack:= fixupStack_(0;..↔fixupStack-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

```

```

[[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
`                      2 empty
` 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

```

```

~
~
~ 105 afternewname 14 ? 223 inputafterq
~ 15 simplename 204 phraftsimname
~ 16 : 202 212 data := data
~ 17 ( 315 202 207 data 314 )
~ 18 := 202 213 data
~ 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 ifelse 30 else [ 200 302 program 301 201 ]
~ 31 empty 301
~ 109 caseelse 32 else [ 200 program 201 ]
~ 33 empty 309
~ 110 morecases 34 [ 304 303 306 200 program 201 ] morecases
~ 35 empty 304 308 306
~ 111 parameterkind 36 : 213 data
~ 37 := 212 data
~ 38 ! 220 data
~ 39 ? 219 data
~ 40 \ 221
~ 112 phraftsimname 41 [ 200 215 311 program 312 301 313 201 ]
~ 42 compounder aftername
~ 113 aftername 43 := 224 data
~ 44 ! 225 data
~ 45 ? 226 inputafterq
~ 46 310 arguments
~ 114 inputafterq 47 ! echo
~ 48 data ( data ) data afterpattern
~ 115 afterpattern 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 specificand arguments

```



```

`
` 64 empty
` 118 data 65 data6 moredata
` 119 moredata 66 ≡ data ≡ data
`
` 67 empty
` 120 data6 68 data5 moredata6
` 121 moredata6 69 = data5 moredata6
`
` 70 ≠ data5 moredata6
`
` 71 < data5 moredata6
`
` 72 > data5 moredata6
`
` 73 ≤ data5 moredata6
`
` 74 ≥ data5 moredata6
`
` 75 : data5 moredata6
`
` 76 :: data5 moredata6
`
` 77 ∈ data5 moredata6
`
` 78 empty
` 122 data5 79 data4 moredata5
` 123 moredata5 80 , data4 moredata5
`
` 81 ,.. data4 moredata5
`
` 82 | data4 moredata5
`
` 83 ◁ data ▷ data4 moredata5
`
` 84 empty
` 124 data4 85 data3 moredata4
` 125 moredata4 86 + data3 moredata4
`
` 87 − data3 moredata4
`
` 88 ;; data3 moredata4
`
` 89 ; data3 moredata4
`
` 90 ;.. data3 moredata4
`
` 91 ‘ data3 moredata4
`
` 92 empty
` 126 data3 93 data2 moredata3
` 127 moredata3 94 × data2 moredata3
`
` 95 / data2 moredata3
`
` 96 ∧ data2 moredata3
`
` 97 ∨ data2 moredata3
`
` 98 empty
` 128 data2 99 # data2
`
` 100 − data2
`
` 101 ~ data2
`
` 102 + data2
`
` 103 □ data2
`
` 104 † data2
`
` 105 * data2
`
` 106 ¢ data2
`
` 107 $ data2
`
` 108 ↔ data2
`
` 109 data1 moredata2
` 129 moredata2 110 * data2 moredata2
`
` 111 → data2 moredata2
`
` 112 ^ data2 moredata2
`
` 113 ^^ data2 moredata2

```

```

~
~ 130 data1      114 empty
~ 131 moredata1 115 data0 moredata1
~               116 % moredata1
~               117 ? moredata1
~               118 ?? moredata1
~               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
[*nil*];`29
[40; 201; 301; 100; 302; 200; 39; 1];`30 ifelse 108
[301];`31
[40; 201; 100; 200; 39; 1];`32 caseelse 109
[309];`33
[110; 40; 201; 100; 200; 306; 303; 304; 39];`34 morecases 110
[306; 308; 304];`35
[118; 213; 18];`36 parameterkind 111
[118; 212; 20];`37
[118; 220; 27];`38
[118; 219; 28];`39
[221; 41]`40
[40; 201; 313; 301; 312; 100; 311; 215; 200; 39];`41 phraftsimname 112
[113; 107];`42
[118; 224; 20];`43 aftername 113
[118; 225; 27];`44
[114; 226; 28];`45
[117; 310];`46
[116; 27];`47 inputafterq 114
[115; 118; 76; 118; 75; 118];`48
[116; 27];`49 afterpattern 115
[*nil*];`50
[225; 107; 10];`51 echo 116
[222];`52
[117; 8];`53 arguments 117
[117; 44];`54
[117; 9];`55
[117; 54];`56
[117; 55];`57
[117; 40; 201; 100; 39; 118; 20; 118; 212; 204; 18; 10; 200; 7];`58
[117; 34; 118; 33];`59
[117; 36; 118; 35];`60
[117; 32; 118; 31];`61
[117; 38; 201; 118; 17; 118; 213; 204; 18; 10; 200; 37];`62
[117; 133; 204; 10];`63
[*nil*];`64
[119; 120];`65 data 118
[118; 71; 118; 70];`66 moredata 119
[*nil*];`67
[121; 122];`68 data6 120
[121; 122; 21];`69 moredata6 121
[121; 122; 22];`70
[121; 122; 23];`71
[121; 122; 24];`72
[121; 122; 25];`73

[121; 122; 26]; `74
[121; 122; 18]; `75
[121; 122; 19]; `76
[121; 122; 66]; `77
[*nil*]; `78
[123; 124]; `79 data5 122
[123; 124; 12]; `80 moredata5 123
[123; 124; 13]; `81
[123; 124; 42]; `82
[123; 124; 69; 118; 68]; `83
[*nil*]; `84
[125; 126]; `85 data4 124
[125; 126; 47]; `86 moredata4 125
[125; 126; 48]; `87
[125; 126; 15]; `88
[125; 126; 14]; `89
[125; 126; 16]; `90
[125; 126; 11]; `91
[*nil*]; `92
[127; 128]; `93 data3 126
[127; 128; 49]; `94 moredata3 127
[127; 128; 50]; `95
[127; 128; 56]; `96
[127; 128; 57]; `97
[*nil*]; `98
[128; 29]; `99 data2 128
[128; 48]; `100
[128; 62]; `101
[128; 47]; `102
[128; 67]; `103
[128; 63]; `104
[128; 61]; `105
[128; 64]; `106
[128; 65]; `107
[128; 53]; `108
[129; 130]; `109
[129; 128; 61]; `110 moredata2 129
[129; 128; 52]; `112
[129; 128; 58]; `113
[*nil*]; `114
[131; 132]; `115 data1 130
[131; 45]; `116 moredata1 131
[131; 28]; `117
[131; 74]; `118
[131; 132; 51]; `119
[131; 132; 60]; `120
[131; 132; 46]; `121
[117; 310]; `122
[8]; `123 data0 132
[44]; `124

```

[9];`125
[54];`126
[55];`127
[28];`128
[74];`129
[40; 201; 100; 39; 118; 20; 118; 212; 204; 18; 10; 200; 7];`130
[34; 118; 33];`131
[36; 118; 35];`132
[32; 118; 31];`133
[38; 201; 118; 17; 118; 213; 204; 18; 10; 200; 37];`134
[133; 204; 10];`135
[73; 201; 118; 207; 200; 72];`136 specificand 133
[227; 107].`137

```

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;.. \leftrightarrow *parseStack*-1). *top*:= *parseStack* (\leftrightarrow *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:= \top]]]

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;.. \leftrightarrow *parseStack*-1); ~*rp*.

top:= *parseStack* (\leftrightarrow *parseStack* - 1)].

if *rp* = [*nil*] [[*pop*. *parse*]]

else [[**new** *prodHead*:= *rp* (#*rp* - 1).

if *prodHead* \geq 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 ` end
                    [[msg!“Error 15: wrong symbol ”; ~scanCodeText_scanCode;
                    “ Should be one of”; legals.
                    error:= ⊤ ]]]
                  else [[msg!“Apology 0: compiler error”. stop]]]]]]]]]. ` end of parse

```

OPTIMIZER

new optimize

[[use: GO object RETURN STOP

`assign: object

sweep

[[new changed: bin:= ⊥. `only those changes that require a new sweep

new pc: nat:= 0. `program counter

loop [[if pc<<=>object

[[case object_pc

`0: STOP

[[pc:= pc+1. loop]]

`1: GO a

[[if object_(object_(pc+1))=GO ∧ object_(pc+1)≠object_(object_(pc+1)+1)

[[object:= object < pc+1 ▷ object_(object_(pc+1) + 1)]]

else [[if object_(object_(pc+1))=RETURN [[object:= object < pc ▷ RETURN. pc:= pc+2]]

else [[if object_(object_(pc+1))=STOP [[object:= object < pc ▷ STOP. pc:= pc+2]]

else [[pc:= pc+2]]]].

loop]]

`2: IF a

[[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]]

`3: CASE a

[[pc:= pc+2. loop]]

`4: CALL a

[[if object_(object_(pc+1))=GO ∧ object_(pc+1)≠object_(object_(pc+1)+1)

```

    [[object:= object < pc+1 > object_(object_(pc+1) + 1)]]
  else [[if object_(object_(pc+1))=RETURN
    [[object:= object < pc > GO. object:= object < pc+1 > pc+2. changed:= ⊤]]
    else [[if object_(object_(pc+1))=STOP [[object:= object < pc > STOP. pc:= pc+2]]
    else [[if object_(pc+2)=RETURN [[object:= object < pc > GO. changed:= ⊤]]
      else [[pc:= pc+2]]]]]].
  loop]]

`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]].
  if changed [[sweep]]]]]]. `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. `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

[[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 20: 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 [[if password=“” [[!“Empty password. Try again.”; nl. pswd]]
      else [[!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” v nameStack i “kind” = “input”
  ` [[drain SOMETHING]].
  `but for now,
  drain keys.
  error:= ⊥. source:= “”. sourceCodes:= nil. sourceNumbers:= nil.
  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

```