OUTPUT( $SAVELINE#, $SAVEF#, ## )

SYMFRD = #*** SYNTAX ERROR #

1(30)

NEXTKN ITEM = NEXTITEM()

IDENT( ITEM = ITEM ) IS(FRETURN)

NEXTKN = ST ITEM

IDENT( NEXTKN = ITEM ) IF(RETURN) S(FAIL)

NEATFN ITEM = NEXTITEM()

IDENT( ITEM = ITEM ) IS(FRETURN)

NEXTFN = FI ITEM

IDENT( NEXTFN = ITEM ) IF(RETURN) S(FAIL)

NEXTITEM LINE = SPAN(1, $) HREAK (1, $) = NEXTITEM = $ IS(RETURN)

NEATITEM LINE = INPUT

OUTPUT = LINE

LINE = POS(0) ## IF(NEXTITEM) S(NEXTITEM)

NESTATE = $ ( S4 OLD $ $ TKN )

IDENT( S( S4 OLD $ $ TKN ) = $ ) IF(RETURN)

STATECNT = STATECNT + 1

$ ( S2 STATECNT ) = $ ( S2 OLD ) = $ TKN

$ ( S4 OLD $ $ TKN ) = STATECNT

NEXT STATE = STATECNT+1 (RETURN)

EXTEND A LIST BY TRANSYTIVITY THROUGH NON TERMINALS

EXTEND TKN = TRANSTKN + 1

EXTEND BT( TKN, TRANSTKN ) IS(EXTEND2)

$ ( T6 TKN ) = KEY

TKN = TKN + 1

(EXTEND1)

EXTEND PLAS = suff

TKNH = TEMKTKN + 1

EXTEND GT( TKNH, TRANSTKN ) IS(EXTEND9)

CHRTKN = $ ( T6 TKNH )

$ ( T6 TKNH ) = $ ( KEY TKN )

TKNH = $ ( KEY TKN )

EXTEND INMET( TKN, CHRTKN ) IS(EXTEND8)

LET TKN, TERMINANT ) IS(EXTEND7)

SUBTKN = $ ( KEY TKN )

EXTEND INMET( SUBTKN, ## ) IS(EXTEND7)

INMET( S( KEY TKNH ## S( SUBTKN ) ## ) IF(EXTENDA)
### 3270 FFT 71 Language Source Code

**Page 3**

```plaintext
CAL $H001L 22 FFR 71 12:41:24 PAGE 3

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>06</td>
<td>S(KEY TKNHU) #2 SUBTKN = S(KEY TKNHU)</td>
<td></td>
</tr>
<tr>
<td>07</td>
<td>S(KEY TKNHU) = SUBTKN</td>
<td></td>
</tr>
<tr>
<td>08</td>
<td>FLAG = #OFF</td>
<td></td>
</tr>
<tr>
<td>09</td>
<td>EXTEND1:  S(HTKN) = $S(KEY TKNHU) #2 SUBTKN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(EXTEND1)</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>EXTEND2:  TKN = $S(KEY TKNHU) #2 TKN</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(EXTEND2)</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>EXTEND3:  TKN = TKNNU * 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(EXTEND3)</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>EXTEND4:  IDENT( FLAG = OFF#)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(RETURN), I(EXTEND2)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>EXTEND STATE SET represented in RARR SET</td>
<td></td>
</tr>
<tr>
<td></td>
<td>BY LAMBD CMOVES</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SETS SET AND SETA EMPTY AT ENP</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ASSUMES SO AT START FOR SETX</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RETURNS NUMBER OF EXTENDED SET</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>EXTEND5:  FLAG = #OFF#</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>STATE = 1</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>EXTEND5A: GT(STATE, STATECNT) I(S(EXTEND5A))</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>IDENT( SET[STATE] = # ) I(S(EXTEND5A))</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>IDENT( SET[STATE] = # ) I(S(EXTEND5A))</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>SETX[STATE] = 1</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>TKN = TKNNU*STATECNT * 1</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>EXTEND5B: GT(TKN, TKNNU) I(S(EXTEND5B))</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>STACK = SETX[STATE] #2 TKN</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>IDENT( SETX[STATE] = # ) I(S(EXTEND5B))</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>IDENT( SETX[STATE] = # ) I(S(EXTEND5B))</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>SETX[STATE] = 1</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>FLAG = #OFF#</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>EXTEND5S: TKN = TKN &lt; 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(EXTEND5S)</td>
<td></td>
</tr>
<tr>
<td>27</td>
<td>EXTEND5S4: STATE = STATE + 1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(EXTEND5S)</td>
<td></td>
</tr>
<tr>
<td>28</td>
<td>EXTEND5S5 IDENT( FLAG = OFF#)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>I(S(EXTEND5S5))</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>STATE = 1</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>STACK = **</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>EXTEND5S: GT(STATE = STATECNT) I(S(EXTEND5S))</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>IDENT( SET[STATE] = # ) I(S(EXTEND5S))</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>IDENT( SET[STATE] = # ) I(S(EXTEND5S))</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>STATE = STATE #2 #2 STATE</td>
<td></td>
</tr>
<tr>
<td>35</td>
<td>EXTEND5S7 SET[STATE] = #2</td>
<td></td>
</tr>
<tr>
<td>36</td>
<td>SETX[STATE] = #2</td>
<td></td>
</tr>
<tr>
<td>37</td>
<td>STATE = STATE + 1 I(EXTEND5S)</td>
<td></td>
</tr>
</tbody>
</table>
```

**Footnotes:**

- The code snippet above is a part of the 3270 FFT 71 language source code, which is used to process data in the IBM 3270 display terminal system. The code is designed to manipulate state variables and perform operations based on user input and system state.

- The `IDENT` function is used to set or clear a state variable based on the input condition.

- The `STACK` function is used to manage a stack of state variables, which is crucial for tracking the state transitions during the execution of the code.

- The `EXTEND` functions are used to extend the state variables and perform operations based on the current state of the system.

- The `SETX[STATE]` function is used to set the state of the system based on the current state variables.

- The `STACK = **` statement initializes a stack with a null value, which is a common practice in assembly or low-level programming languages to ensure that the stack is properly initialized before execution.

- The `STATE = STATE + 1` statement increments the state variable by one, indicating a transition to the next state in the sequence.

- The `IDENT( STATE = STATECNT )` statement identifies or sets the state variable based on the current state count, which is essential for managing state transitions and ensuring that the system remains in a correct state.

- The `STACK = SETX[STATE] #2 TKN` statement sets the stack based on the current state and the total number of tokens, which is used to manage the data flow and control the execution of the code.

- The `STATE = STATE + 1` statement increments the state variable by one, indicating a transition to the next state in the sequence. This is a fundamental operation in state-based systems to ensure that the system progresses through its states in a controlled manner.

- The `IDENT( SET[STATE] = # )` statement identifies or sets the state variable based on the input condition, which is used to manage the state transitions based on user input or system data.

- The `SETX[STATE] = 1` statement sets the state of the system to one, which is a common practice in state-based systems to ensure that the system transitions to a predefined state for further processing.

- The `STACK = #2 #2 STATE` statement sets the stack based on the state variable, which is used to manage the data flow and control the execution of the code.

- The `STATE = STATE + 1` statement increments the state variable by one, indicating a transition to the next state in the sequence. This is a fundamental operation in state-based systems to ensure that the system progresses through its states in a controlled manner.

- The `IDENT( STACK = STATECNT )` statement identifies or sets the stack based on the current state count, which is essential for managing state transitions and ensuring that the system remains in a correct state.

- The `STACK = SETX[STATE]` statement sets the stack based on the current state, which is used to manage the data flow and control the execution of the code.

- The `STATE = STATE + 1` statement increments the state variable by one, indicating a transition to the next state in the sequence. This is a fundamental operation in state-based systems to ensure that the system progresses through its states in a controlled manner.
COLUMN | END | INITIAL | INITIAL
11B | EXTEND | $SS$ | STATE
11C | IDENT( EXTEND$; Domain# | IF(RETURN)
120 | SS$ = SS$ + 1
121 | EXTEND | SS$ = SS$ + 1
122 | IF SS$ > STATE | = STATE + 1 | RETURN
123 | OUTPUT AN ASSEMBLY LINE

124 | ASSEM | A = $A$ + C * 
125 | A TAB(1) | = R #
126 | A = R NO # 
127 | A TAB(2) | = R #
128 | IDENT( COMM # | iS(ASSEM)
129 | A = A + # & NO #
130 | A TAB(2) | = B #
131 | ASSEMBLY LINE = B COMM | (RETURN)
132 | ASSEM | ASSEM | = B RAND | (RETURN)

SUBROUTINE TO SAVE INFO FOR SECOND PASS

133 | SAVE | REDUND( #SAVE# )
134 | PUT( FILE(2) )
135 | PUT( TERMINCNT )
136 | PUT( TITLE )
137 | PUT( TITLE )
138 | PUT( STATECNT )
139 | I = 1
140 | SAVE | GT( I , TERMINCNT ) | BE(SAVE3)
141 | SOUT( $II$ )
142 | SOUT( $II$ )
143 | I = I + 1 | (SAVE1)
144 | SAVE | I = 1
145 | SAVE | GT( I , TERMINCNT ) | BE(SAVE4)
146 | SOUT( T4 )
147 | SOUT( T4 )
148 | SOUT( T4 )
149 | SOUT( T4 I )
150 | PUT( TITLE )
151 | PUT( $II$ )
152 | I = I + 1 | (SAVE3)
153 | SAVE | GT( I , TERMINCNT ) | BE(SAVE7)
154 | SOUT( T7 )
155 | SOUT( T7 )
156 | SOUT( T7 I )

PROG: 22 FEB 71 1216124
<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>157</td>
<td>K = T10 I</td>
<td></td>
</tr>
<tr>
<td>158</td>
<td>SPUT( K )</td>
<td></td>
</tr>
<tr>
<td>159</td>
<td>J = %K( )</td>
<td></td>
</tr>
<tr>
<td>160</td>
<td>SAVE5 INENT( J * #X# )</td>
<td>IS(SAVE6)</td>
</tr>
<tr>
<td>161</td>
<td>L = K * #X# J</td>
<td></td>
</tr>
<tr>
<td>162</td>
<td>SPUT( )</td>
<td></td>
</tr>
<tr>
<td>163</td>
<td>J = %L( )</td>
<td>I(SAVE6)</td>
</tr>
<tr>
<td>164</td>
<td>SAVE6 I = I + 1</td>
<td>I(SAVE4)</td>
</tr>
<tr>
<td>165</td>
<td>SAVE7 I = I</td>
<td></td>
</tr>
<tr>
<td>166</td>
<td>SAVE8 GT( I, DILECNY )</td>
<td>IS(SAVE9)</td>
</tr>
<tr>
<td>167</td>
<td>SPUT( J# I )</td>
<td></td>
</tr>
<tr>
<td>168</td>
<td>SPUT( #E I )</td>
<td></td>
</tr>
<tr>
<td>169</td>
<td>SPUT( #A )</td>
<td></td>
</tr>
<tr>
<td>170</td>
<td>I = I + 1</td>
<td>I(SAVE6)</td>
</tr>
<tr>
<td>171</td>
<td>SAVE9 I = 0</td>
<td></td>
</tr>
<tr>
<td>172</td>
<td>SAVE10 GT( I, STATECNT )</td>
<td>IS(SAVE13)</td>
</tr>
<tr>
<td>173</td>
<td>SPUT( S# I )</td>
<td></td>
</tr>
<tr>
<td>174</td>
<td>SPUT( SA I )</td>
<td></td>
</tr>
<tr>
<td>175</td>
<td>SPUT( SR I )</td>
<td></td>
</tr>
<tr>
<td>176</td>
<td>J = 1</td>
<td></td>
</tr>
<tr>
<td>177</td>
<td>SAVE11 GT( J, TINDEX )</td>
<td>IS(SAVE12)</td>
</tr>
<tr>
<td>178</td>
<td>SPUT( SA I * # J )</td>
<td></td>
</tr>
<tr>
<td>179</td>
<td>J = J + 1</td>
<td>I(SAVE11)</td>
</tr>
<tr>
<td>180</td>
<td>SAVE12 I = I + 1</td>
<td>I(SAVE10)</td>
</tr>
<tr>
<td>181</td>
<td>SAVE13 PUT( #ccccc# )</td>
<td></td>
</tr>
<tr>
<td>182</td>
<td>PUT( #cccccccc# ) OUT = # ITEMCNT</td>
<td>I(RETURN)</td>
</tr>
</tbody>
</table>

**ROUTINES USED BY SAVE**

<table>
<thead>
<tr>
<th>Line</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>183</td>
<td>PUT SVELTNE = ITEM</td>
<td></td>
</tr>
<tr>
<td>184</td>
<td>ITEMCNT = ITEMCNT + 1</td>
<td>I(RETURN)</td>
</tr>
<tr>
<td>185</td>
<td>SPUT TEMP = %ITEM</td>
<td></td>
</tr>
<tr>
<td>186</td>
<td>INENT( TEMP; #E )</td>
<td>IS(RETURN)</td>
</tr>
<tr>
<td>187</td>
<td>PUT( ITEM )</td>
<td></td>
</tr>
<tr>
<td>188</td>
<td>PUT( TEMP )</td>
<td>I(RETURN)</td>
</tr>
</tbody>
</table>
JUMP FUNCTION DATA

169 dumpfn: Output = '#
170 Output = #
171 Output = 
172 Output = #
173 dumpfn1 get ( f : FNCONC ) is ( RETURN )
174 Output = [ $ ( F2 I ) ] [ $ ( F1 $ ( F2 I ) ) ]
175 I = I + 1 ( dumpfn1 )

JUMP TOKEN TABLE

190 dumpk1: Output = #
191 Output = #
192 Output = 
193 Output = #
194 dumpk11 one ( t ) TOKEN TERMINAL #
195 Output = 
196 Output = 
197 dumpk11 part ( t ) TOKEN TERMINAL #
198 Output = PART $ ( T1 $ ( T2 I ) $ ( T4 I ) $ ( T71 I )
199 I = I + 1 ( dumpk11 )
200 dumpk12 Output = #
201 Output = #
202 Output = #
203 dumpk13 get ( t ) TOKEN RETURN
204 Output = [ $ ( T2 I ) ] [ $ ( T3 I ) ] [ $ ( T4 $ ( T2 I ) )
205 I = I + 1 ( dumpk13 )
206 dumpstat Output = 
207 Output = 
208 dumpstat1 get ( t ) STATE RETURN
209 Output = [ $ ( S2 I ) ] [ $ ( S4 I )
210 Output = I =
211 dumpstat11 get ( t ) STATE RETURN
212 Output = [ $ ( S2 $ ( S2 I ) ) ] [ $ ( S4 I )
213 Output = I =
214 dumpstat2 get ( t ) STATE RETURN
215 Output = [ $ ( T2 I ) ] [ $ ( T4 $ ( T2 I ) ) ] [ $ ( T7 $ ( T2 I ) )
216 dumpstat21 get ( t ) STATE RETURN
217 Output = [ $ ( S1 $ ( S1 I ) ) ] [ $ ( S4 I )]
218 dumpstat211 get ( t ) STATE RETURN
219 Output = [ $ ( S1 $ ( S1 I ) ) ] [ $ ( S4 I )]
220 dumpstat3 Output = J = J + 1 ( dumpstat2 )
221 dumpstat31 Output = I = I + 1 ( dumpstat3 )
222 dumpstat311 Output = I = I + 1 ( dumpstat3 )

dumprules Output = #
224 Output = #
225 Output = #
226 Output = # RUL E TABLE
I = 1

DUMP RULES

GT( I; RULECNT ) IS( RETURN )

OUTPUT = I ≡ # $3(R6 I) ≡ $5(R5 I) ≡ $0(R6 I)

E = $ (R7 I)

E = Next( E ) ! ( DUMP RULES )

I = I + 1 ! ( DUMP RULES )

DUMP TERMINAL EXTRA INFO

OUTPUT ≡ #

OUTPUT = #

OUTPUT = #

I = TERM COUNT + 1

DUMP TERMINAL EXTRA INFO

OUTPUT ≡ #

OUTPUT = #

OUTPUT = #

J = $ (T5 I)

DUMP TERMINALS

IDENT( J ≡ $ ) IS( DUMP TERMINALS )

OUTPUT = #

J = $ (T5 I ≡ J ) ! ( DUMP TERMINALS )

DUMP TERMINALS

OUTPUT ≡ #

OUTPUT = #

OUTPUT = #

J = $ ( T7 I )

DUMP TERMINALS

IDENT( J ≡ $ ) IS( DUMP TERMINALS )

OUTPUT = #

J = $ ( T7 I ≡ J ) ! ( DUMP TERMINALS )

DUMP TERMINALS

OUTPUT ≡ #

OUTPUT = #

OUTPUT = #

J = $ ( T6 I )

DUMP TERMINALS

IDENT( J ≡ $ ) IS( DUMP TERMINALS )

OUTPUT = #

J = $ ( T6 I ≡ J ) ! ( DUMP TERMINALS )

DUMP TERMINALS

OUTPUT ≡ #

OUTPUT = #

OUTPUT = #

J = $ ( T6 I )

DUMP TERMINALS

IDENT( J ≡ $ ) IS( DUMP TERMINALS )
PRINT OUT STATE SET

PRINTS S
PRINTSS1
PRINTS
PRINTS1
PRINTS2
PRINTRULES
PRINTTHUS
PRINTTHUS1

PRINT OUT RULE TABLE

PRINT OUT TOKEN INFO
CAL SMOHUL 26 FEB 71 124124 PAGE 9

314    ASSEM( #VXX#, #SET#, $T1$, $T2$ )
315    1 = 1 + 1
316    I (PRINTTKS1)

ACTUAL CUDF STARTS HERE

FIRST READ IN TOKEN DEFINITIONS; TERMINAL FIRST

319    GO
320    ASSEM( #IDENT#, NEXTITEM() )
321    ASSEM( #ENTRY#, #SUBRAME# )
322    ASSEM( #ENTRY#, #ENTRY# )
323    ITEM = NEXTITEM()
324    IDENT(I TEM, #FLM# ) IF (Grl)
325    ASSEM( #ENTRY#, #FLM# )

329    GO1
330    KEYBASE = NEXTITEM()
331    TWCNY = 1
332    ASSEM( #KEYWOL#, #KEYBASE# )
333    TKNS1 ITEM1 = NEXTITEM()
334    IDENT(ITEM1, #FLM# ) IS(TKNS1)
335    ITEM2 = NEXTITEM()
336    IDENT(ITEM2, #FLM# ) [IF (FAIL1)]
337    ITEM3 = NEXTITEM()
338    IDENT(ITEM3, #ENTRY# ) I S (TKNS1)
339    ITEM4 = NEXTITEM()
340    IDENT(ITEM4, #ENTRY# ) [IF (FAIL2)]
341    ITEM5 = NEXTITEM()
342    IDENT(ITEM5, #ENTRY# ) [IF (FAIL3)]
343    ITEM6 = NEXTITEM()
344    IDENT(ITEM6, #ENTRY# ) [IF (FAIL4)]
• \(X(T1) = TKN\)
• \(X(T2) = ITEM1\)
• \(X(T3) = ITEM2\)
• \(X(T4) = ITEM3\)
• \(X(T7) = TKN\)
• \(X(T7) = TKN\)
• \(X(T7) = TKN\)

---

NO# READ IN NON TERMINAL TOKEN DEFINITIONS

TKNS2

TERM TKNCNT = TKN

TKNS3

ITEM1 = NEATITEM()

TERM ITEM1 = TKN

ITEM2 = ITEM1

ITEM3 = TKN

ITEM4 = ITEM1

ITEM5 = ITEM2

ITEM6 = ITEM1

ITEM7 = ITEM3

ITEM8 = ITEM1

ITEM9 = ITEM1

ITEM10 = ITEM1

---

NO# READ IN FUNCTION DEFINITIONS

FNC50

FNC = 0

FNC1

ITEM1 = NEATITEM()

FNC2

ITEM1 = ITEM1

FNC3

ITEM1 = ITEM1

FNC4

ITEM1 = ITEM1

FNC5

FNC = FNC + 1

FNC6

ASSEM( **, NEAT , ITEM2)

FNC7

FNC = FNC + 1

FNC8

FNC = FNC + 1

---

THIS CODE READS IN THE RULES
AND GENERATES THE SYMBOLE STATE TABLE

STATECNT = 0

RULECNT = 0
• RULE1
  RULEPN = NEXTFN() IF(RULE4)

• RULE2
  TNK = NEXTTKN() IF(FAIL)

• RULE3
  STATE = NEWSTATE(STATE,TOKEN)

• RULE4
  PRINTTKNS()
CONSTRUCTION OF LIST OF TERMINALS THAT CAN BEGIN A NON TERMINAL

41c HTKN = TFM(TKVCNT * 1)
41d FSTTERMS1 GT(HTNYK, TKNCNT) IS(FSTTERMS1)
41e TKN = $ (TK HDTKN)
41f FSTTERMS2 IDENT(TKN, ##) IS(FSTTERMS2)
41g GT(TKN, TERMKNCNT) IS(FSTTERMS2)
41h $ (T7 HDTKN $ TKN) = $ (T7 HDTKN)
41i $ (T7 HDTKN) = TKN
41j FSTTERMS3 TKX = $ (T7 HDTKN $ TKN) IF (FSTTERMS2)
41k FSTTERMS4 HTKN = HDTKN + 1 IF (FSTTERMS1)
41l FSTTERMS4

CONSTRUCTION OF LIST ON NON TERMINAL THAT CAN END A NON TERMINAL

42c HTKN = TFM(TKVCNT * 1)
42d LASTNONT1 GT(HTNYK, TKNCNT) IS(LASTNONT1)
42e TKN = $ (TB HDTKN)
42f LASTNONT2 IDENT(TKN, ##) IS(LASTNONT2)
42g LET TKN, TERMKNCNT) IS(LASTNONT2)
42h $ (T9 HDTKN $ TKN) = $ (T9 HDTKN)
42i $ (T9 HDTKN) = TKN
42j LASTNONT3 TKX = $ (T9 HDTKN $ TKN) IF (LASTNONT2)
42k LASTNONT4 HTKN = HDTKN + 1 IF (LASTNONT1)
42l LASTNONT4

CONSTRUCT LIST OF TERMINALS IMAT CAN FOLLOW EACH NON TERM

43a RULE = 1
43b NXTTERMS1 GT(RULE, RULECNT) IS(NXTTERMS1)
43c L = $ (R, RULE)
43d NXTTERMS2 NONTYP = VALUE(L)
43e L = NEXT(L)
43f LET NONTYP, TERMKNCNT) IS(NXTTERMS2)
43g IDENT(LV, ##) IS(NXTTERMS3)
43h FOLLOWER = VALUE(L)
43i TKN = $ (TY NONTYP)
43j NXTTERMS3 IDENT(TKN, ##) IS(NXTTERMS3)
43k TYP = $ (TY FOLLOWER)
43l NXTTERMS4 IDENT(TKN, ##) IS(NXTTERMS4)
```
444  IDENT( T10 TK1 # TK2 ) # IF(NXTERM5)
445  #(T10 TK2) = #TK1
446  #(T10 TK1) = TK2
447  NXTERM5: TK#2 = # IF(T7 FULLROWID # TK2 ) # IF(NXTERM4)
448  NXTERM6: TK#1 = # IF(T9 NONTERM # TK1 ) # IF(NXTERM3)
449  NXTERM: RULE = RULE + 1 # IF(NXTERM)
450  NXTERM5: IDENT(L#2#) IS(NXTERM7) F(NXTERM2)
451  NXTERM5:
452  SAVE()
453  SPTRNS = TOKEN = NEXTKEN( IF(SPTRNS)
454  EXTNME = NEXTITEM()
455  IDENT( EXTNME # ) IF(IFAIL)
456  IDENT( NEXTITEM( # ) # IF(IFAIL)
457  PUTI( TOKEN )
458  PUTI( EXTNME ) IF(SPTRNS)
459  SPTRNS = PUTI( # # )
460  IF(ENDIF)
461  FAIL1: OUT( SYNR # item NOT # TOKEN# ) IF(IFAIL)
462  FAIL2: OUT( SYNR # item NOT # FUNCTION# ) IF(IFAIL)
463  FAIL4: OUT( SYNR # PERIOD EXPECTED ) IF(IFAIL)
464  FAIL5: OUT( SYNR # # PERIOD EXPECTED ) IF(IFAIL)
465  FAIL6: OUT( SYNR # # PERIOD EXPECTED ) IF(IFAIL)
466  FAIL7: OUT( SYNR # PERIOD NOT EXPECTED ) IF(IFAIL)
467  FALI: OUTPUT = SYNR
468  ASSEM; #; PERR; #; SYNR
469  END
470  SUCCESSFULCompilation
```

```
CAL SN C U L 27 80 71 12141124 PAGE 13
```

```
HEADC LIST Of TOKENS TO BE LOOKED UP
IN THIS GRAMMAR
```

```
SPTRNS = TOKEN = NEXTKEN( IF(SPTRNS)
```

```
IDENT( EXTNME # ) IF(IFAIL)
```

```
IDENT( NEXTITEM( # ) # IF(IFAIL)
```

```
PUTI( TOKEN )
```

```
PUTI( EXTNME ) IF(SPTRNS)
```

```
SPTRNS = PUTI( # # )
```

```
IF(ENDIF)
```

```
FAIL1: OUT( SYNR # item NOT # TOKEN# ) IF(IFAIL)
```

```
FAIL2: OUT( SYNR # item NOT # FUNCTION# ) IF(IFAIL)
```

```
FAIL4: OUT( SYNR # PERIOD EXPECTED ) IF(IFAIL)
```

```
FAIL5: OUT( SYNR # # PERIOD EXPECTED ) IF(IFAIL)
```

```
FAIL6: OUT( SYNR # # PERIOD EXPECTED ) IF(IFAIL)
```

```
FAIL7: OUT( SYNR # PERIOD NOT EXPECTED ) IF(IFAIL)
```

```
FAIL: OUTPUT = SYNR
```

```
ASSEM; #; PERR; #; SYNR
```

```
END
```

```
SUCCESSFUL Compilation
```

```
```
TOKEN DEFINITIONS

BLANK
TIL DE 1002
CP
PARAM 501R
PACAP 400R+1
PDATA 400R+2
NEWB 400R+3
KILLF 400R+4
MODT 400R+5
MOD 400R+6
VIEW 400R+7
NEWF 400R+8
NEWLD 400R+9
KILLF 400R+10
KILLD 400R+11
NEWL 400R+12
KILLBK 400R+13
COPYFILE 400R+14
KILLOBJ 400R+15
DELOBJ 400R+16
DELLINK 400R+17
DELKEY 400R+18
ADDKEY 400R+19
DELKEY 400R+20
SENDLC 2000H+1
LINE 2000H+2

PULF INFO

1 LINE IDENTITY 2
2 SENTENCE USRCPAP 3
3 SENTENCE PROTWR 3
4 SENTENCE PROTWR 3
5 SENTENCE MODWR 3
6 SENTENCE KILLWR 3
7 SENTENCE MOVEWR 4
8 SENTENCE MOVECAO 4
9 SENTENCE WIF4STACK 3
10 SENTENCE NDSKFILE 3
11 SENTENCE NNEWDIP 3
12 SENTENCE KILLSKFILE 3
13 SENTENCE KILLP 3
14 SENTENCE NNEWLP 3
15 SENTENCE KILLBP 3
16 SENTENCE COPYFILE 7