เอกสารอ้างอิง

ทักษิณา สวนานนท์. พจนานุกรมศัพท์คอมพิวเตอร์. พิมพ์ครั้งแรก. บริษัท มีเดีย แอสโซซิเอตเต็ด จำกัด, 2527.

มานิจ มานิตเจริญ. พจนานุกรมไทย. พิมพ์ครั้งที่ 7. สำนักพิมพ์ บำรุงสาส์น, 2524.

Aho V.A. and Ullman D.J., Principle of Compiler Design, Addison-Wesley Publishing Company, 1977.

Brinch Hansen P, Programming a Personal Computer, Prentice-Hall Inc., 1982.

Digital Research, Pascal/MT + Language Programmer s Guide for the CP/M-86 Family of Operating System, Digital Research, 1983.

Digital Research, Pascal /MT + Language Reference Manual, Digital Research, 1983.

Jansen K. and Wirth N., PASCAL : user manual and report, Springer-Verlag, 1974.

Liffick B.W., The Byte Book of Pascal, Byte Publications Inc., 1979.

Pratt Terrence. W., Programming Language : Design and Implementation, 1sted., Prentice-Hall Inc., 1975.

Pratt Terrence W., Programming Language : Design and Implementation, 2nd ed., Prentice-Hall Inc., 1984.

Pyster A.B., Compiler Design and Construction, Van Nostrand Reinhold Company, 1980.

ภาคผนวก ก

ไวยากรณ์ของภาษาคอมพิวเตอร์ภาษาไทย

1. <โปรแกรม> → <ส่วนหัวโปรแกรม> <ส่วนตัวโปรแกรม> {'โปรแกรม'}

2. <ส่วนหัวโปรแกรม> → 'โปรแกรม' <ชื่อ> ';' {'โปรแกรม'}

3. <ส่วนตัวโปรแกรม> → <ส่วนการนิยาม> <เริ่มต้น-สิ้นสุด> {'ตัวแปร'}

→ <เริ่มต้น-สิ้นสุด> {'เริ่มต้น'}

4. <ส่วนการนิยาม> → 'ตัวแปร' <ชุดนิยามตัวแปร> {'ตัวแปร'}

5. <ชุดนิยามตัวแปร> → <ประโยคนิยามตัวแปร> <ชุดนิยามตัวแปร> {<ชื่อ>}

→ λ {'เริ่มต้น'}

6. <ประโยคนิยามตัวแปร> → <ชุดของชื่อ> ':' <ชนิด> ';'

(<ประโยคนิยามตัวแปร> {<ชื่อ>} | → {'เริ่มต้น'}) {<ชื่อ>}

7. <ชุดของชื่อ> → <ชื่อ> (',' <ชุดของชื่อ> {','} | λ {':'}){<ชื่อ>}

8. <ชนิด> → 'ชนิดเลขจำนวนเต็ม' {'ชนิดเลขจำนวนเต็ม'}

→ 'ชนิดเลขจำนวนจริง' {'ชนิดเลขจำนวนจริง'}

→ 'ชนิดตัวอักษร' {'ชนิดตัวอักษร'}

→ 'ชนิดตรรก' {'ชนิดตรรก'}

9. <เริ่มต้น-สิ้นสุด> → 'เริ่มต้น' <ชุดของประโยคคำสั่ง> 'สิ้นสุด' ('.' {'.'} |

';' {';'}){'เริ่มต้น'}

10. <ชุดของประโยคคำสั่ง> → <ประโยคคำสั่ง> <ชุดของประโยคคำสั่ง>

{'เริ่มต้น' |'ถ้า' | 'วนทำ' |

'เมื่อ' | 'อ่าน' |'อ่านบรรทัด'|

'พิมพ์' | 'พิมพ์บรรทัด'}

→ λ {'สิ้นสุด'}

11. <ประโยคคำสั่ง> → <เริ่มต้น-สิ้นสุด> {'เริ่มต้น'}

→ <ถ้า-แล้ว> {'ถ้า'}

→ <วนทำ> {'วนทำ'}

→ <เมื่อ-ออก> {'เมื่อ'}

→ <อ่าน> {'อ่าน'} | 'อ่านบรรทัด'}

→ <พิมพ์> {'พิมพ์' | 'พิมพ์บรรทัด'}

→ <การกำหนดค่า> {<ชื่อ>}

12. <การกำหนดค่า> → <ชื่อ> ':=' <นิพจน์> ';' {<ชื่อ>}

13. <นิพจน์> → <นิพจน์ย่อย> <นิพจน์ส่วนเหลือ> {'ไม่' , 'จริง', 'เท็จ', '(', '+', '-',ค่าคงที่เลขจำนวนเต็ม, ค่าคงที่เลขจำนวนจริง, ค่าคงที่ตัวอักษร, <ชื่อ> }

14. <นิพจน์ส่วนเหลือ> → <เครื่องหมายเปรียบเทียบ> <นิพจน์ย่อย>{'<>', '=', '<=', '>=', '<', '>', '><', '=<', '=>'}

→ λ {')', ';'}

15. <นิพจน์ย่อย> → <ตัวประกอบ> <นิพจน์ย่อยส่วนเหลือ> {'ไม่', 'จริง', 'เท็จ', '(', '+', '-', ค่าคงที่เลขจำนวนเต็ม, ค่าคงที่เลขจำนวนจริง, ค่าคงที่ตัวอักษร, <ชื่อ>}

16. <นิพจน์ย่อยส่วนเหลือ> → <เครื่องหมายระดับบวกลบ> <ตัวประกอบ> <นิพจน์ย่อย-ส่วนเหลือ>

{'-', '+', 'หรือ'}

→ λ {'<>', '=', '<=', '>=', '<', '>', '><', '=<', '=>', ')', ';'}

17. <ตัวประกอบ> → <ตัวประกอบย่อย> <ตัวประกอบส่วนเหลือ> {'ไม่', 'จริง', 'เท็จ', '(', '+', '-',ค่าคงที่เลขจำนวนเต็ม, ค่าคงที่เลขจำนวนจริง, ค่าคงที่ตัวอักษร, <ชื่อ>}

18. <ตัวประกอบส่วนเหลือ> → <เครื่องหมายระดับคูณหาร> <ตัวประกอบย่อย> <ตัวประกอบส่วนเหลือ> {'*', '/', 'และ'}

→ λ { '<>', '=', '<=', '>=', '<', '>',
'><', '=<', '=>', ')', ';', '-',
'+', 'หรือ'}

19. <ตัวประกอบย่อย> → <พจน์> <ตัวประกอบย่อยส่วนเหลือ> {'ไม่', 'จริง', 'เท็จ',
'(', '+', '-',
ค่าคงที่เลขจำนวนเต็ม,
ค่าคงที่เลขจำนวนจริง,
ค่าคงที่ตัวอักษร, <ชื่อ>}

20. <ตัวประกอบย่อยส่วนเหลือ> → <เครื่องหมายยกกำลัง> <พจน์>
<ตัวประกอบย่อยส่วนเหลือ> {'**'}
→ λ {'<>', '=', '<=', '>=', '<', '>',
'><', '=<', '=>', ')', ';', '-',
'+', 'หรือ', '*', '/', 'และ'}

21. <พจน์> → 'ไม่' <พจน์> {'ไม่'}
→ '(' <นิพจน์> ')' {'(' }
→ '+' ('(' <นิพจน์> ')' {'('} |
ค่าคงที่เลขจำนวนเต็ม {ค่าคงที่เลขจำนวนเต็ม} |
ค่าคงที่เลขจำนวนจริง {ค่าคงที่เลขจำนวนจริง} |
<ชื่อ> {<ชื่อ>}) {'+'}
→ '-' ('(' <นิพจน์> ')' {'('}|
ค่าคงที่เลขจำนวนเต็ม {ค่าคงที่เลขจำนวนเต็ม} |
ค่าคงที่เลขจำนวนจริง {ค่าคงที่เลขจำนวนจริง} |
<ชื่อ> {<ชื่อ>}) {'-'}
→ ค่าคงที่เลขจำนวนเต็ม {ค่าคงที่เลขจำนวนเต็ม}
→ ค่าคงที่เลขจำนวนจริง {ค่าคงที่เลขจำนวนจริง}
→ ค่าคงที่ตัวอักษร {ค่าคงที่ตัวอักษร }
→ ค่าคงที่ชนิดตรรก {'จริง', 'เท็จ' }
→ <ชื่อ> {<ชื่อ>}

22. <เครื่องหมายเปรียบเทียบ> → '=' {'='}
→ '<>' {'<>', '><'}
→ '<=' {'<=', '=<'}
→ '>=' {'>=', '=>'}
→ '<' {'<'}
→ '>' {'>'}

23. <เครื่องหมายระดับบวกลบ> → '+' {'+'}
→ '-' {'-'}
→ 'หรือ' {'หรือ'}

24. <เครื่องหมายระดับคูณหาร> → '*' {'*'}
→ '/' {'/'}
→ 'และ' {'และ'}

25. <เครื่องหมายยกกำลัง> → '**' {'**'}

26. <ถ้า-แล้ว> → 'ถ้า' <นิพจน์> 'แล้ว' <ประโยคคำสั่ง> <ส่วนหรือไม่ก็>
{'ถ้า'}

27. <ส่วนหรือไม่ก็> → 'หรือไม่ก็' <ประโยคคำสั่ง> {'หรือไม่ก็'}
→ λ {'สิ้นสุด', <ชื่อ>, ';', 'เริ่มต้น', 'ถ้า',
'วนทำ', 'เมื่อ', 'อ่าน', 'อ่านบรรทัด',
'พิมพ์', 'พิมพ์บรรทัด'}

28. <วนทำ> → 'วนทำ' <ประโยคคำสั่ง> 'สิ้นสุดวนทำ' {'วนทำ'}

29. <เมื่อ-ออก> → 'เมื่อ' <นิพจน์> 'ออก' {'เมื่อ'}

30. <อ่าน> → ('อ่าน' {'อ่าน'}| 'อ่านบรรทัด' {'อ่านบรรทัด'})
('(' <ชุดการอ่าน> ')' ';'{ '('}| λ {';'}
{'อ่าน', 'อ่านบรรทัด'}

31. <ชุดการอ่าน> → <ชื่อ> (',' <ชุดการอ่าน> {','}| λ {')'}
{<ชื่อ>}

32. <พิมพ์> → ('พิมพ์' {'พิมพ์'}| 'พิมพ์บรรทัด'{ 'พิมพ์บรรทัด'})
('(' ('พ' <ชุดการพิมพ์> ')' ';' {'พ'}|
<ชุดการพิมพ์> ')' ';' {<ชื่อ>, ค่าคงที่เลขจำนวนเต็ม,
ค่าคงที่เลขจำนวนจริง, ค่าคงที่ตัวอักษร}) {'('}| λ
{';'}) {'พิมพ์', 'พิมพ์บรรทัด' }

33. <ชุดการพิมพ์> → <พิมพ์ข้อมูล> <พิมพ์ส่วนเหลือ> {ค่าคงที่ตัวอักษร, สายตัวอักษร,
<ชื่อ>, ค่าคงที่เลขจำนวนเต็ม,
ค่าคงที่เลขจำนวนจริง}

34. <พิมพ์ข้อมูล> → ค่าคงที่ตัวอักษร {ค่าคงที่ตัวอักษร}

→ สายตัวอักษร {สายตัวอักษร}

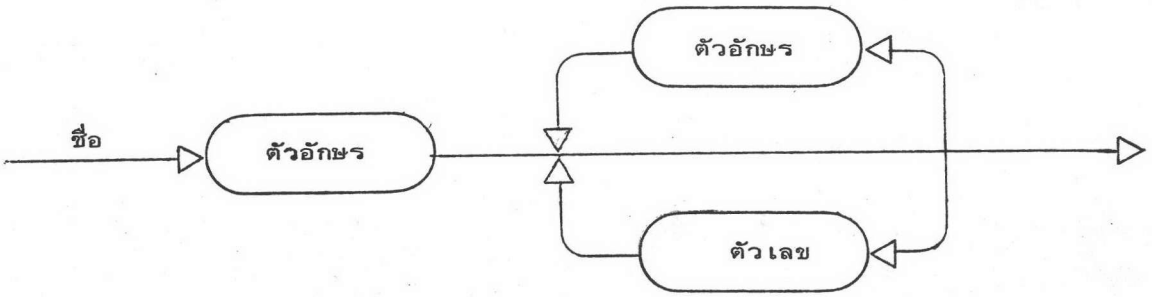
→ <นิพจน์> {<ชื่อ>, ค่าคงที่เลขจำนวนเต็ม,
ค่าคงที่เลขจำนวนจริง }

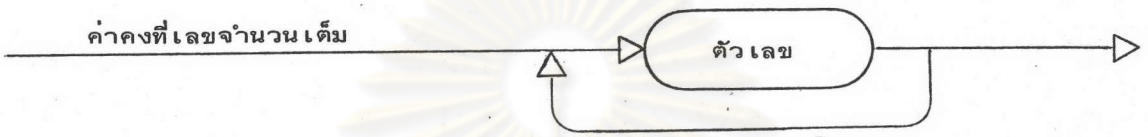
35. <พิมพ์ส่วนเหลือ> → ',' <ชุดการพิมพ์> {','}

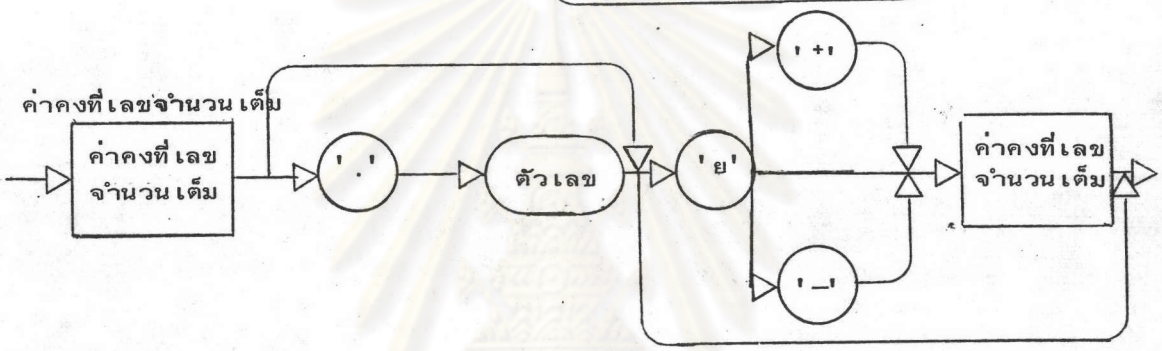
→ λ {')'}

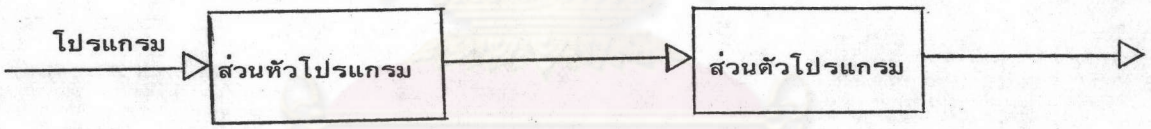
ภาคผนวก ข

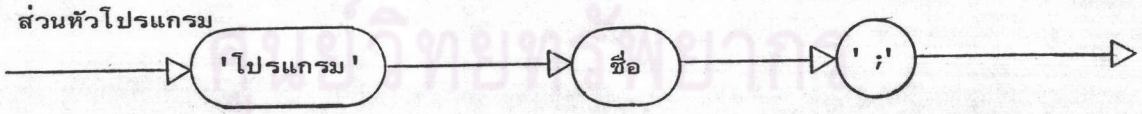
แบบแผนทางวากยสัมพันธ์

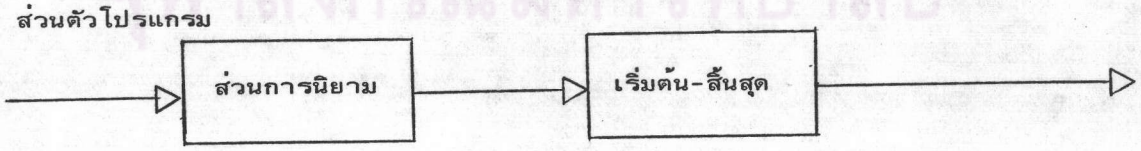

ตัวอักษร
ชื่อ
ตัวอักษร
ตัว เลข

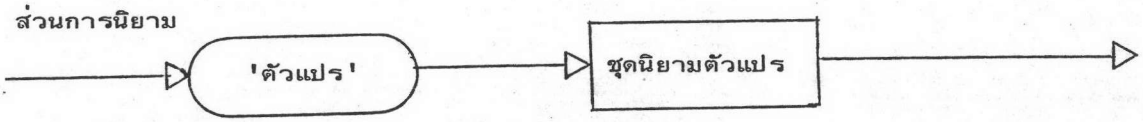

ค่าคงที่เลขจำนวน เต็ม
ตัว เลข

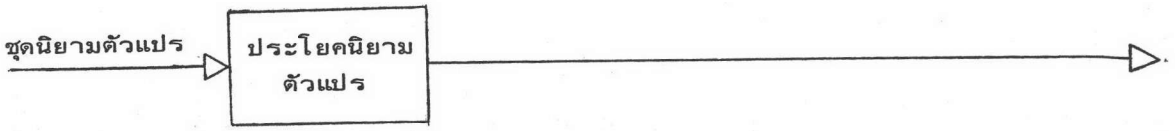

ค่าคงที่เลขจำนวน เต็ม
ค่าคงที่ เลข
จำนวน เต็ม
' . '
ตัว เลข
'ย'
' +'
' _'
ค่าคงที่เลข
จำนวน เต็ม

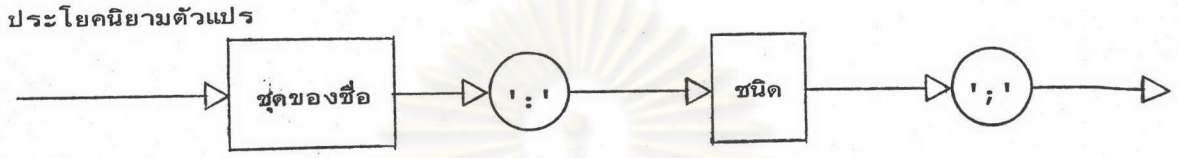

โปรแกรม
ส่วนหัวโปรแกรม
ส่วนตัวโปรแกรม

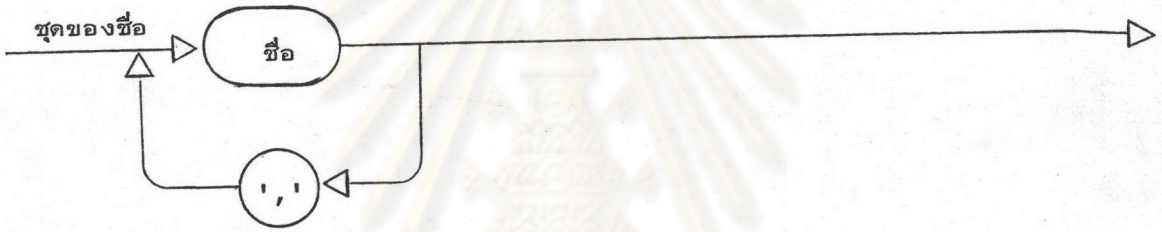

ส่วนหัวโปรแกรม
'โปรแกรม'
ชื่อ
' ;'

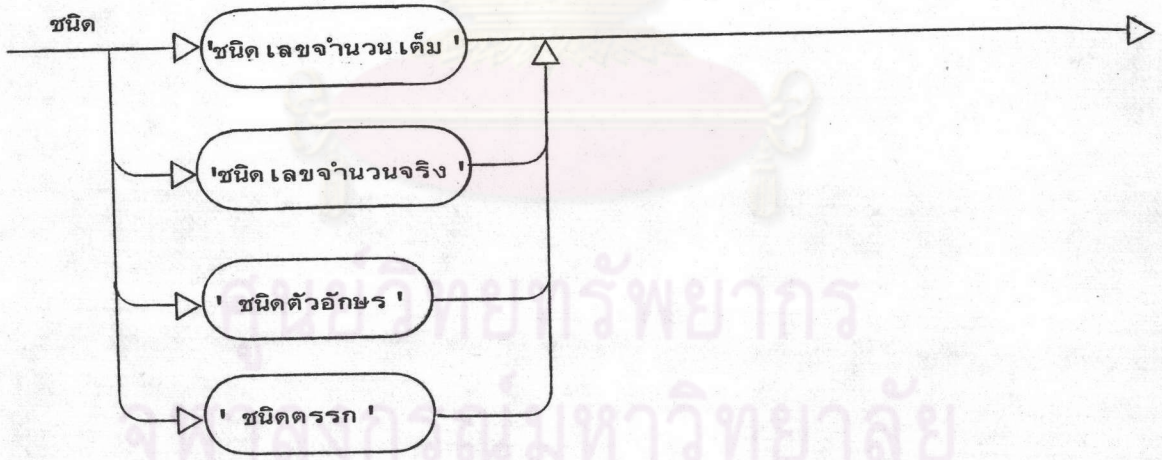

ส่วนตัวโปรแกรม
ส่วนการนิยาม
เริ่มต้น-สิ้นสุด

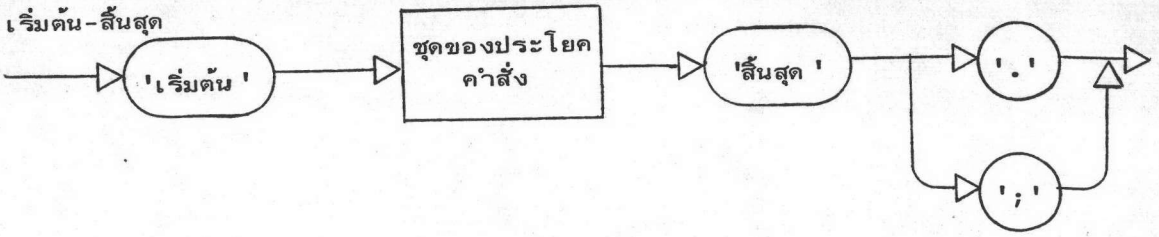

ส่วนการนิยาม
'ตัวแปร'
ชุดนิยามตัวแปร


ชุดนิยามตัวแปร
ประโยคนิยาม
ตัวแปร

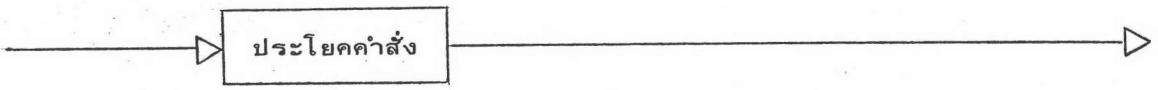

ประโยคนิยามตัวแปร
ชุดของชื่อ
':'
ชนิด
';'

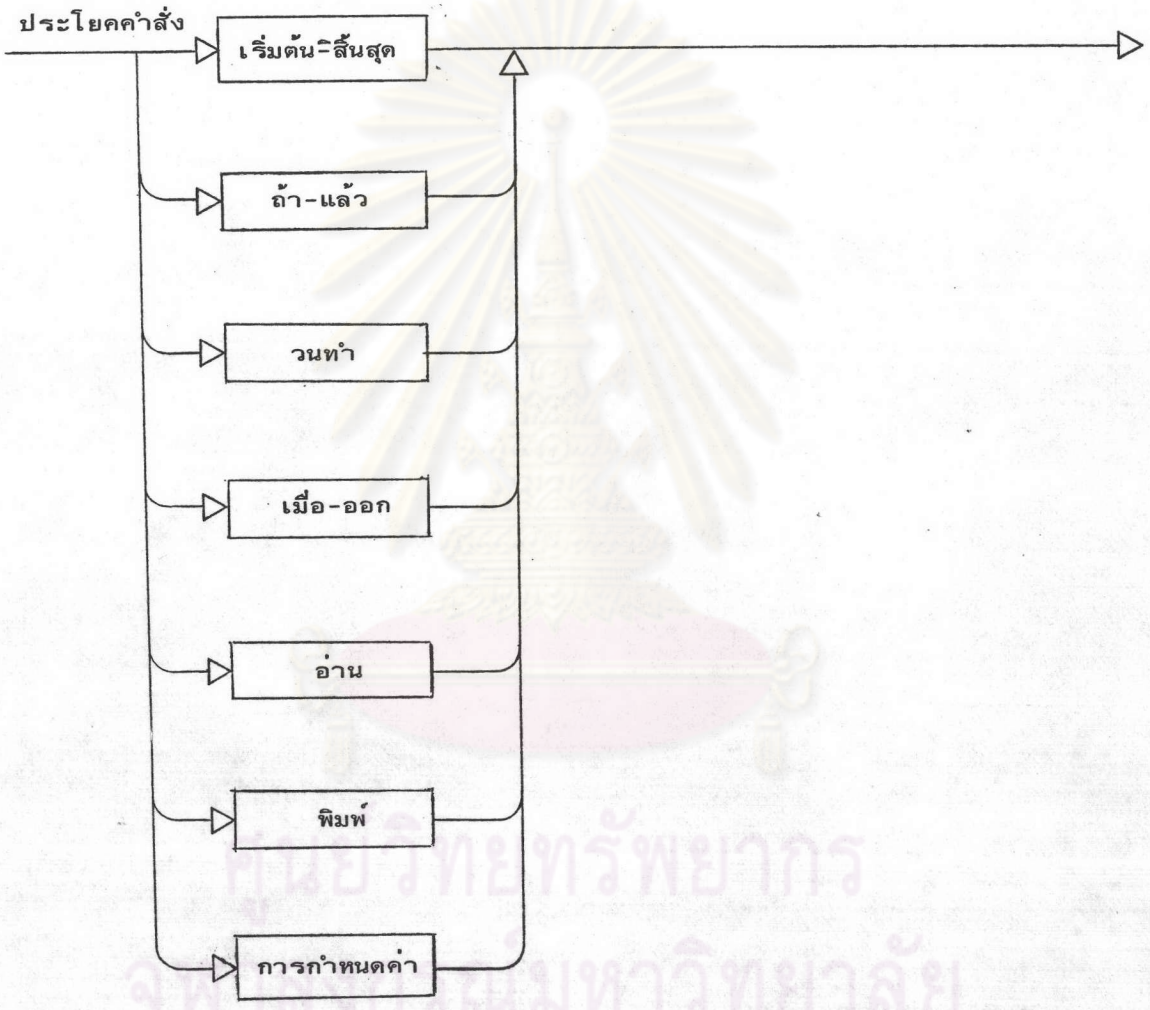

ชุดของชื่อ
ชื่อ
','


ชนิด
'ชนิด เลขจำนวน เต็ม '
'ชนิด เลขจำนวนจริง '
' ชนิดตัวอักษร '
' ชนิดตรรก '

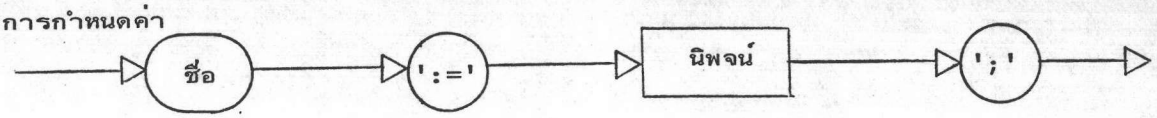

เริ่มต้น-สิ้นสุด
'เริ่มต้น '
ชุดของประโยค
คำสั่ง
'สิ้นสุด '
'.'
';'

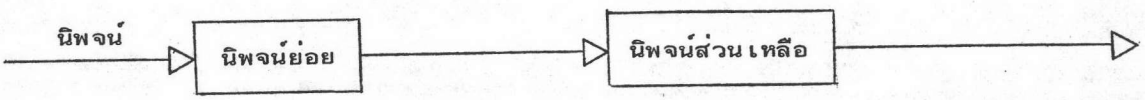
ชุดของประโยคคำสั่ง

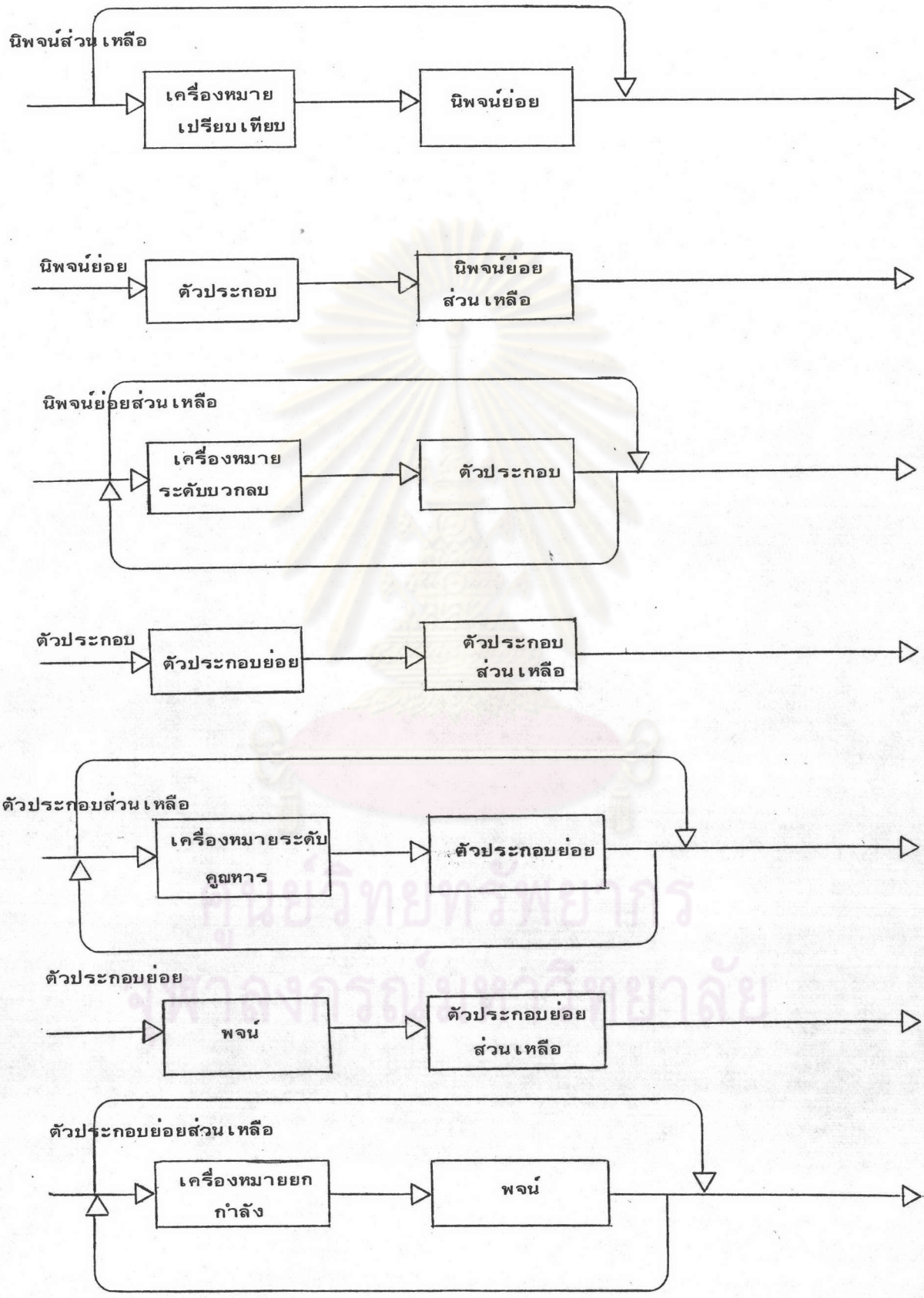


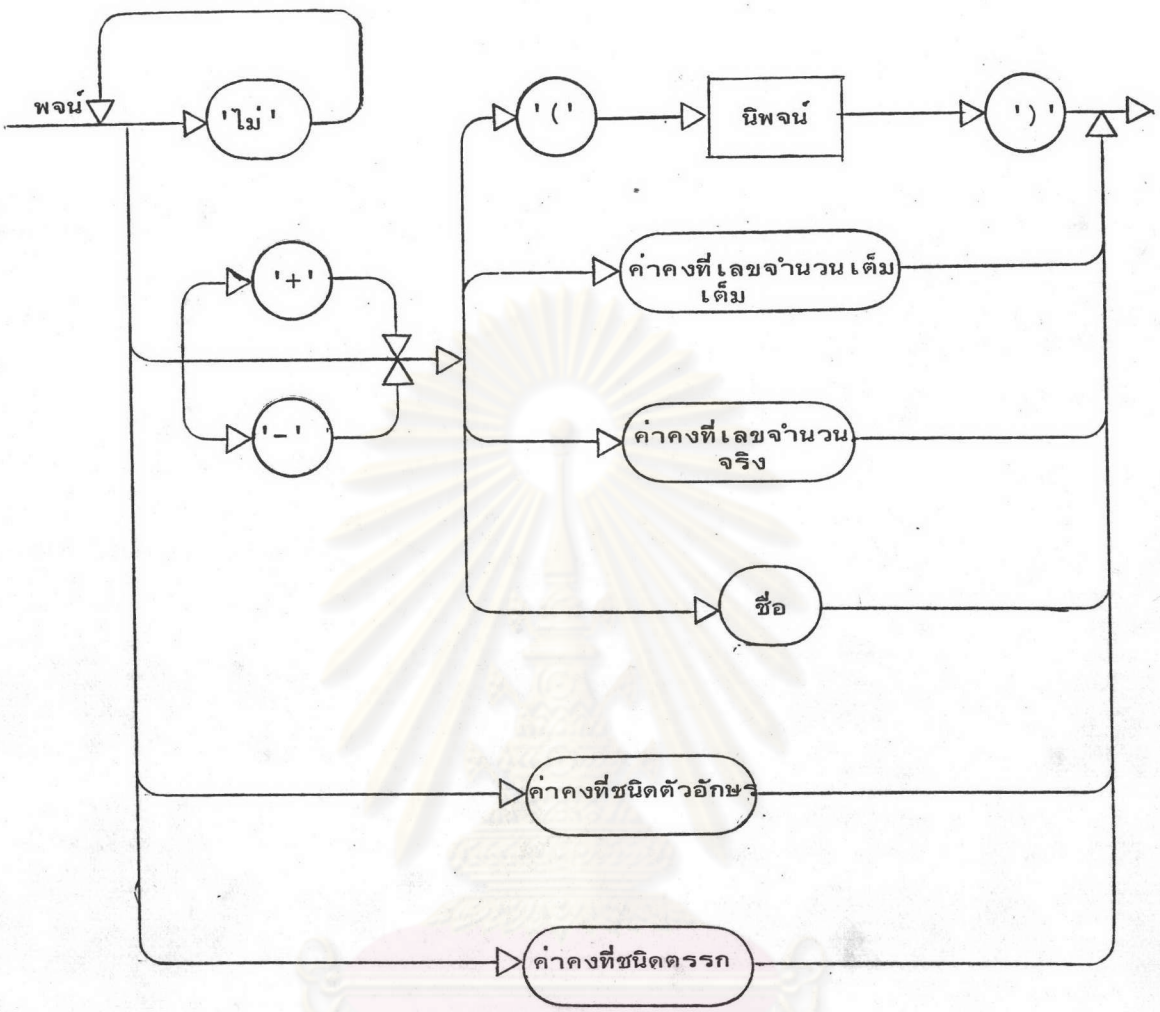


การกำหนดค่า

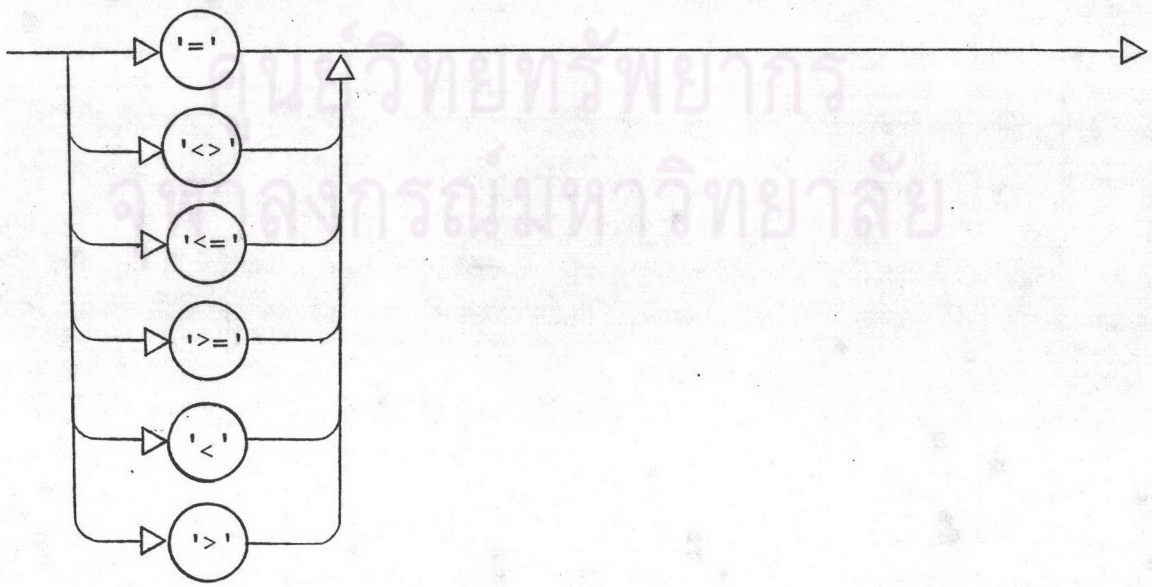




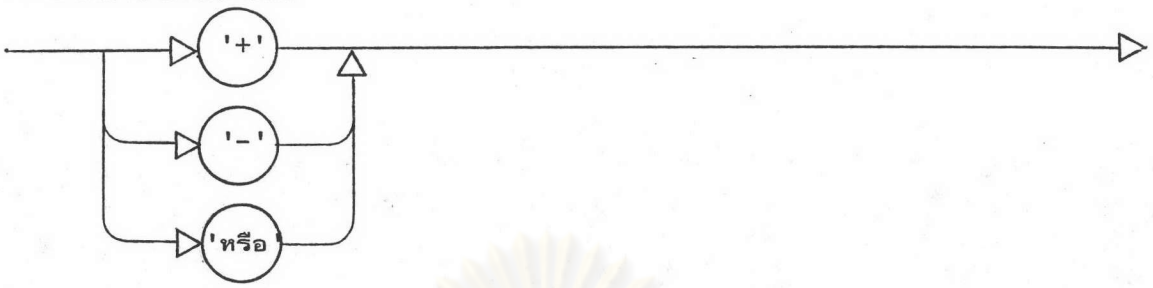

นิพจน์ส่วนเหลือ
เครื่องหมายเปรียบเทียบ
นิพจน์ย่อย
นิพจน์ย่อย
ตัวประกอบ
นิพจน์ย่อยส่วนเหลือ
นิพจน์ย่อยส่วนเหลือ
เครื่องหมายระดับบวกลบ
ตัวประกอบ
ตัวประกอบ
ตัวประกอบย่อย
ตัวประกอบส่วนเหลือ
ตัวประกอบส่วนเหลือ
เครื่องหมายระดับคูณหาร
ตัวประกอบย่อย
ตัวประกอบย่อย
พจน์
ตัวประกอบย่อยส่วนเหลือ
ตัวประกอบย่อยส่วนเหลือ
เครื่องหมายยกกำลัง
พจน์



เครื่องหมายเปรียบเทียบ

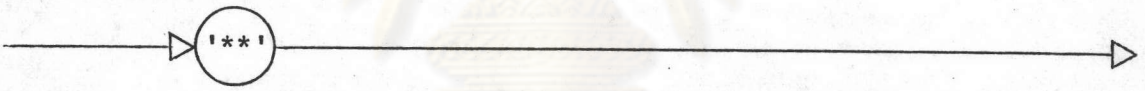


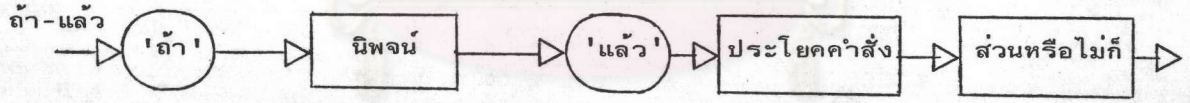
เครื่องหมายระดับบวกลบ

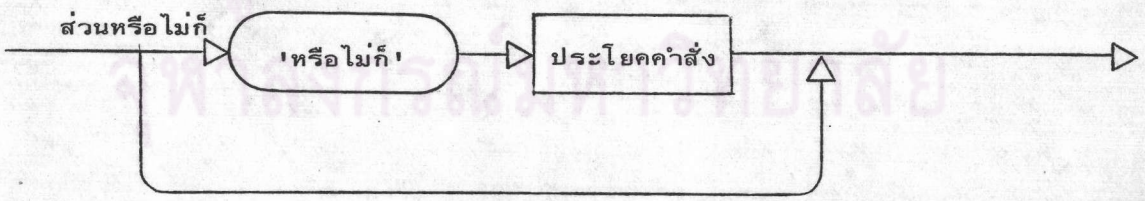


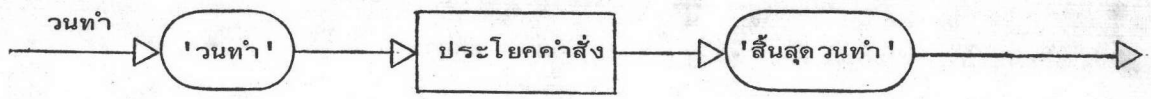
เครื่องหมายระดับคูณหาร

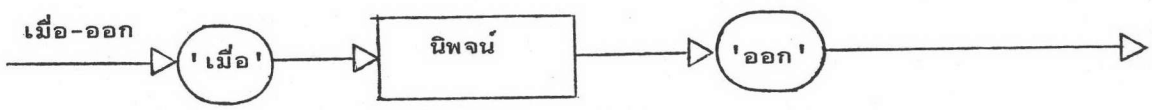
เครื่องหมายยกกำลัง

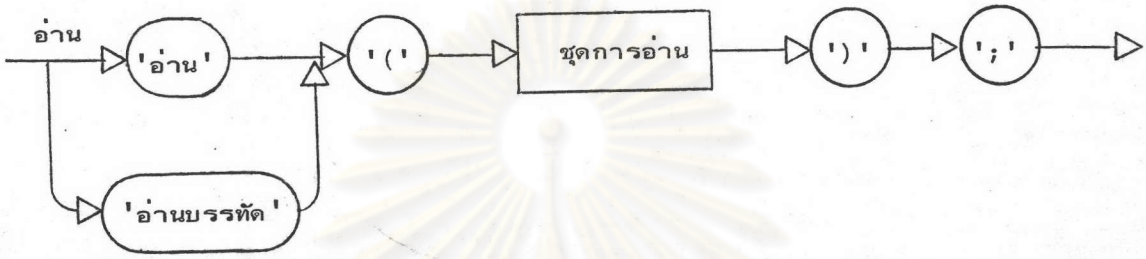


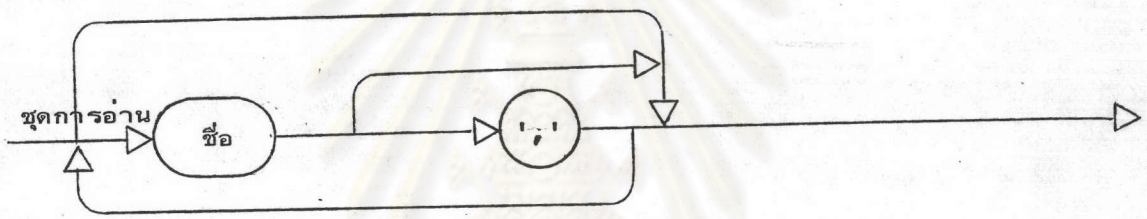


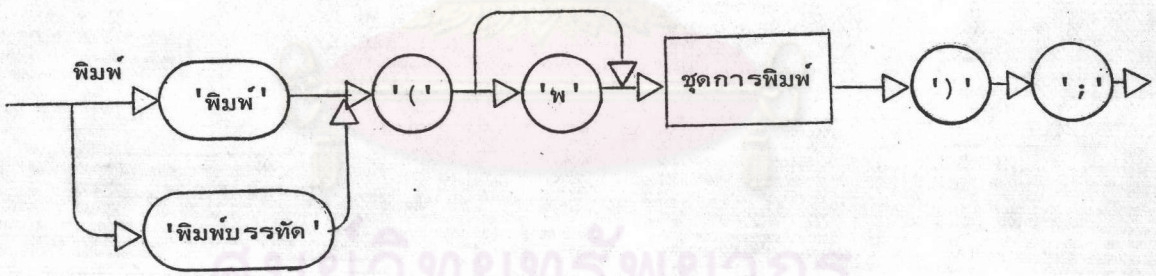


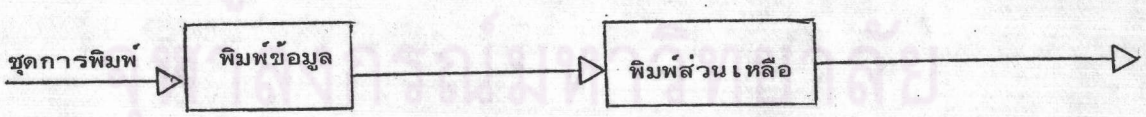


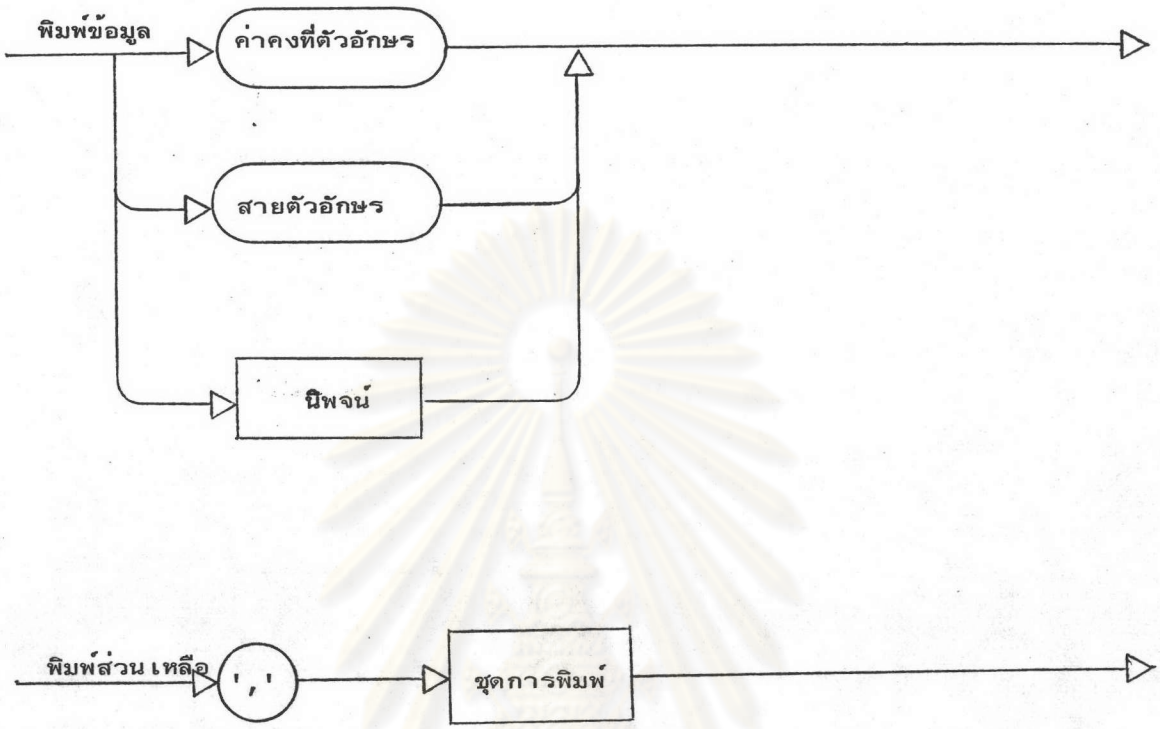












ภาคผนวก ค

ข้อความแสดงความผิดพลาด

ข้อความแสดงความผิดพลาดที่สแกนเนอร์ทำการตรวจสอบ

1. หมายเหตุไม่สมบูรณ์
2. ชุดของตัวอักษรไม่สมบูรณ์
3. ชื่อไม่ถูกต้อง
4. ชื่อไม่ได้กำหนดไว้ก่อน
5. ตัวเลขไม่ถูกต้อง
6. ตัวอักษรไม่ถูกต้อง
7. ไม่ปรากฏเครื่องหมายของส่วนยกกำลัง

ข้อความแสดงความผิดพลาดที่พาสเซอร์และโปรแกรมซีแมนติกซ์ตรวจสอบ

1. คาดว่าเป็น 'โปรแกรม'
2. คาดว่าเป็นชื่อโปรแกรม
3. คาดว่าเป็น ';'
4. ตามหลัง 'ตัวแปร' ต้องเป็นชื่อ
5. คาดว่าเป็น ':'
6. ชนิดไม่ถูกต้อง
7. คาดว่าเป็น 'เริ่มต้น' หรือชื่อ
8. คาดว่าเป็นชื่อ
9. คาดว่าเป็น ';' หรือ ':'
10. คาดว่าเป็น 'เริ่มต้น'
11. คาดว่าเป็น 'สิ้นสุด'
12. คาดว่าเป็น ':'
13. คำสำคัญไม่ถูกต้อง
14. ประโยคคำสั่งแบบนี้ไม่ปรากฏ
15. คาดว่าเป็น ':='
16. คาดว่าเป็น 'ไม่'
17. คาดว่าเป็น ')'
18. คาดว่าเป็น '(' หรือ ชื่อ หรือ ตัวเลข
19. คาดว่าเป็น 'แล้ว'
20. คาดว่าเป็น 'สิ้นสุดวนทำ'
21. ไม่มีทางออกจากการวน จะวนตลอดกาล
22. 'เมื่อ' ๑ คำสั่ง กำหนดในการวน ๑ ครั้งเท่านั้น
23. คาดว่าเป็น 'ออก'

24. คำสั่ง 'พิมพ์' แบบนี้ไม่ปรากฏ

25. ตัวดำเนินการมีมาก ที่เก็บมีไม่พอ

26. ตัวถูกกระทำมีมาก ที่เก็บมีไม่พอ

27. ที่เก็บตัวดำเนินการไม่มีตัวดำเนินการอยู่เลย

28. ที่เก็บตัวถูกกระทำไม่มีตัวถูกกระทำอยู่เลย

29. ไม่มีตัวแปรชั่วคราวให้ปลดปล่อย

30. มีการกำหนดชนิดซ้ำซ้อนกัน

31. ชนิดใช้ไม่ถูกต้อง

32. ตัวแปรคนละชนิดกัน

33. เลขจำนวนเต็มมีค่ามาก

34. เลขจำนวนจริงมีค่ามากหรือน้อยเกินไป

35. ตัวแปรไม่ได้ถูกกำหนดไว้ก่อน

ภาคผนวก ง

รายละเอียดโปรแกรม

ตัวแปลภาษาคอมพิวเตอร์ภาษาไทย

```
PROGRAM THAI_COMPILER;
(* A CONSTUCTION OF THAI_LANGUAGE COMPILER *)
(*              WRITTEN  BY                *)
(*          MR.ADISORN  KRUNGKASEM         *)
(*          VERSION 1.0  1987              *)
CONST
  CONTROL_C = #3;          (*CONTROL-C CODE*)
TYPE
  PT_STR    = ^STRING;
  TABLE     = ^PT_TABLE;
  PT_TABLE  = RECORD       (*FORMAT FOR EACH ENTRY OF IDENTIFIER,NUMBER AND LITERAL TABLE*)
    NM_PT   : INTEGER;     (*POINTER IN POOL*)
    LEN     : INTEGER;     (*LENGTH*)
    TYP     : CHAR;        (*TYPE*)
    LINK    : TABLE
  END;
  SYM_TAB   = RECORD       (*FORMAT OF SYMBOL TABLE*)
    NM      : STRING[10];  (*NAME*)
    CASE DEF: INTEGER OF (*TYPE DEFINE*)
    0,1,2,3,4:(CASE DAT:CHAR OF
               'I':(IN:INTEGER);
               'R':(RE:REAL);
               'B':(BL:BOOLEAN);
               'C':(C:CHAR);
               'L':(LT:STRING[255]));
  END;
  TMP_FL    = RECORD       (*FORMAT OF TEMPORARY FILE*)
    LABL_NM : STRING[10];  (*LABEL*)
    OP_CD   : CHAR;        (*OPERATION CODE*)
    OP1     : INTEGER;     (*OPERAND 1*)
    OP2     : INTEGER      (*OPERAND 2*)
  END;
VAR
  END_LINK  : BOOLEAN;     (*INDICATOR FOR END OF LINE IN DYNAMIC MEMORY*)
  END_FILE  : BOOLEAN;     (*INDICATOR FOR END OF FILE*)
  ERROR_FLAG: BOOLEAN;     (*INDICATOR WHETHER SOURCE PROGRAM HAS AN ERROR*)
  END_DECLARE: BOOLEAN;    (*INDICATOR FOR END OF DECLARATION PART*)
  FOUND_FLAG : BOOLEAN;    (*INDICATOR WHETHER TOKEN IS RESERVED WORD*)
  COMPLETE  : BOOLEAN;     (*INDICATOR WHETHER LITERAL OR COMMENT IS COMPLETE*)
  FIND_ID   : BOOLEAN;     (*INDICATOR WHETHER FOUND DATA IN TABLE*)
  PRTR_ON   : BOOLEAN;     (*INDICATOR FOR DISPLAYING AT PRINTER*)
  CHCOUNT   : INTEGER;     (*POINTER IN BUFFER*)
  SINDEX,INDEX,I: INTEGER;(*INDEX IN ANY PURPOSE*)
  POSIT     : INTEGER;     (*POSITION IN POOL*)
  LNG       : INTEGER;     (*LENGTH OF TOKEN*)
  LINE_LENGTH: INTEGER;    (*LENGTH OF BUFFER*)
  TOKEN_NO  : INTEGER;     (*TOKEN NUMBER*)
  LINE_NO   : INTEGER;     (*SOURCE PROGRAM LINE NUMBER*)
  ERROR_NO  : INTEGER;     (*NUMBER OF ERROR IN SOURCE PROGRAM*)
  STK_PT1   : INTEGER;     (*STACK POINTER IN OPERATION STACK*)
  STL_PT2   : INTEGER;     (*STACK POINTER IN OPERAND STACK*)
  NO_OF_BEGIN: INTEGER;    (*NUMBER OF BEGIN*)
  LOOP_COUNT : INTEGER;    (*LOOP COUNTER*)
  END_COUNT  : INTEGER;    (*ENDLOOP COUNTER*)
  EXIT_COUNT : INTEGER;    (*EXIT COUNTER*)
  CUR_TEMP_NO: INTEGER;    (*CURRENT NUMBER OF TEMPORARY VARIABLE*)
  TAB        : INTEGER;    (*TABLE NUMBER*)
  LABEL_COUNT: INTEGER;    (*CURRENT LABEL NUMBER*)
  RSV1       : INTEGER;    (*RESERVED WORD INDEX*)
  POOL_TOP   : INTEGER;    (*LATEST POSITION USED OF POOL*)
  RESULT     : INTEGER;    (*RETURN FROM CLOSE FUNCTION*)
  THAI_NUM   : SET OF CHAR;(*THAI NUMBER SET*)
  THAI_CHAR  : SET OF CHAR;(*THAI CHARECTER SET*)
  THI_VW_OVR : SET OF CHAR;(*THAI OVER VOWEL SET*)
  THI_VW_FNT : SET OF CHAR;(*THAI FRONT VOWEL SET*)
  THI_VW_BHD : SET OF CHAR;(*THAI BEHIND VOWEL SET*)
  THI_VW_UND : SET OF CHAR;(*THAI UNDER VOWEL SET*)
  SYMBOL     : SET OF CHAR;(*SET OF SYMBOL*)
  TOKEN      : STRING[255];(*TOKEN THAT IS AN OUTPUT OF SCANNER*)
```

```
  LINE        : STRING[255];(*STORE CURRENT RECORD READ FROM SOURCE FILE OR IS BUFFER*)
  FILENAME    : STRING[15]; (*FILENAME USED IN COMPILATION*)
  NAME        : STRING[8];  (*FILNAME WITH FILE EXTENSION '.THI'*)
  NAME_BUFF   : STRING[12]; (*FILENAME ACCEPT FROM USER*)
  IDENT       : STRING[255];(*COLLECT IDENTIFIERS THAT IS IN SAME DECLARATION STATEMENT*)
  OPT         : STRING[20]; (*STORE OPERATOR THAT POPED FROM OPERATOR STACK*)
  OPR         : STRING;     (*STORE OPERAND THAT POPED FROM OPERAND STACK*)
  A,B         : STRING[255];(*STORE OPERANDS THAT PASS TO SEMANTIC ROUTINE*)
  X           : STRING[10];
  NO_STR      : STRING;     (*STORE STRING THAT CONVERTED FROM NUMBER*)
  TEMP        : STRING;
  TEMP_COPY   : STRING[255];(*STORE TOKEN THAT READ FROM TABLES*)
  RSVWORD     : ARRAY[1..30] OF STRING[15];(*TABLE STORE ALL RESERVED WORD*)
  ERROR_MASS  : ARRAY[1..10] OF STRING[35];(*TABLE STORE ALL SCANNER ERROR MESSAGE*)
  ERR_TAB     : ARRAY[1..40] OF STRING[60];(*TABLE STORE ALL SYNTACTIC OR SEMANTIC ERROR MESSAGE*)
  OPR_STACK   : ARRAY[1..90] OF STRING;     (*OPERATOR STACK*)
  OPT_STACK   : ARRAY[1..50] OF STRING[20];(*OPERAND  STACK*)
  TBLE        : ARRAY[1..1024] OF SYM_TAB; (*SYMBOL TABLE*)
  LABEL_TAB   : ARRAY[1..1024] OF INTEGER; (*LABEL TABLE FOR STORING LABEL NAME AND PROGRAM COUNTER*)
  POOL_STR    : ARRAY[1..4096] OF CHAR;     (*POOL AREA*)
  TOKEN_TYPE  : CHAR;           (*TOKEN_TYPE THAT IS AN OUTPUT OF SCANNER*)
  TYPE_DATA   : CHAR;           (*TYPE OF DATA STORE IN TABLE*)
  TYPE_OF_ID  : CHAR;           (*TYPE OF IDENTIFIER*)
  A_KEY       : CHAR;           (*A CHARACTER CODE OF KEY THAT ACCEPTED FROM USER*)
  TYPES       : CHAR;           (*TYPE OF DATA READ FROM TABLES*)
  SOURCE      : TEXT;           (*SOURCE PROGRAM FILE*)
  LIST_FILE   : TEXT;           (*FILE USED TO STORE PROGRAM LISTING ON PRINTER*)
  PRINTFILE   : TEXT;           (*PRINTING FILE*)
  FILE_PTR    : PT_STR;         (*FOR USING WITH @CMD FUNCTION*)
  LIT_FIRST   : TABLE;          (*LITERAL TABLE*)
  NUM_FIRST   : TABLE;          (*NUMERIC TABLE*)
  IDEN_FIRST  : TABLE;          (*IDENTIFIER TABLE*)
  POINT_TEMP  : TABLE;          (*POINTER FOR NEW POINTER*)
  POINTS      : TABLE;          (*POINTER FOR USING IN SEARCHING DATA*)
  PROG_COUNT  : INTEGER;        (*PROGRAM COUNTER*)
  ADDR        : INTEGER;
  TEMP_FILE   : FILE OF TMP_FL;(*TEMPORARY FILE*)
  TMP         : TMP_FL;         (*TEMPORARY RECORD FOR ACCEPTING DATA FROM TEMPORARY FILE*)
EXTERNAL FUNCTION @CMD:PT_STR;
EXTERNAL PROCEDURE @HLT;


PROCEDURE INITIALIZE;
(* THIS SUBPROGRAM INITIALIZES VALUE IN CHARACTER SET,RESERVED WORD TABLE,
   LEXICAL SYNTACTIC AND SEMANTIC ERROR MESSAGE,ALL INDICATORS,COUNTERS
   AND STACK POINTER.IT DISPLAYS THE MESSAGES TO THE USER AND ASKS HIM
   TO ANSWER THE FILENAME THAT HE WANTS TO COMPILE*)
BEGIN
  THAI_NUM    := ['0'..'9'];
  THAI_CHAR   := ['ก'..'ฮ'];
  THI_VW_UPP  := ['ุ','ู','ิ','ี','ึ','ื','ั','่','้','๊','๋','็','์','ำ'];
  THI_VW_PRT  := ['ะ', 'า', 'เ', 'แ', 'โ'];
  THI_VW_BHD  := ['ไ', 'ใ', 'ฤ', 'ฦ', 'ๅ'];
  THI_VW_UND  := ['ฺ', 'ฯ'];
  SYMBOL      := ['+','-','*','/','=','<','>','(',')',' ',',','.',':',';','''','{','}','[',']'];
  RSVWORD[1]  := 'ประกาศ';
  RSVWORD[2]  := 'ตัวแปร';
  RSVWORD[3]  := 'เป็นสตริงขนาด';
  RSVWORD[4]  := 'เป็นตัวเลขขนาด';
  RSVWORD[5]  := 'ถ้า';
  RSVWORD[6]  := 'ให้';
  RSVWORD[7]  := 'ทำ';
  RSVWORD[8]  := 'ถึง';
  RSVWORD[9]  := 'ขณะ';
  RSVWORD[10]:= 'ทำซ้ำ';
  RSVWORD[11]:= 'ตั้งแต่';
```

```
RSVWORD[12] := 'ELSE';
RSVWORD[13] := 'END';
RSVWORD[14] := 'FOR';
RSVWORD[15] := 'IF';
RSVWORD[16] := 'INTEGER';
RSVWORD[17] := 'REAL';
RSVWORD[18] := 'PROCEDURE';
RSVWORD[19] := 'THEN';
RSVWORD[20] := 'WHILE';
RSVWORD[21] := 'DO';
RSVWORD[22] := 'BOOLEAN';
RSVWORD[23] := 'OF';
RSVWORD[24] := 'TO';
END_FILE    := FALSE;
END_DECLARE := FALSE;
COMPLETE    := FALSE;
PETR_ON     := FALSE;
CHCOUNT     := 0;
LINE_NO     := 0;
LINE_LENGTH := 0;
ERROR_NO    := 0;
NO_OF_BEGIN := 0;
LOOP_COUNT  := 0;
END_COUNT   := 0;
EXIT_COUNT  := 0;
STF_PT1     := 0;
STK_PT2     := 0;
CUR_TEMP_NO := 0;
LABEL_COUNT := 0;
PROG_COUNT  := 0;
POOL_TOP    := 1;
I           := 1;
RSV1        := 1;
NUM_FIRST   := NIL;
IDEN_FIRST  := NIL;
LIT_FIRST   := NIL;
ERROR_MASS[1] := 'ขนาดของตัวแปรเกินขอบเขต ';
ERROR_MASS[2] := 'จำนวนของตัวแปรเกินขอบเขต ';
ERROR_MASS[3] := 'ชื่อ,ตัวแปร,ซ้ำกัน ';
ERROR_MASS[4] := 'ชื่อ,ตัวแปร,ยาวเกินกำหนด ';
ERROR_MASS[5] := 'ตัวเลขผิด,ต้องมี ';
ERROR_MASS[6] := 'ตัวเลขเกินขอบเขต,ต้องมี ';
ERROR_MASS[7] := 'ผิด,ประเภท,ข้อมูล,ตามที่กำหนด ';
ERR_TAB[1]    := 'ควรจะมี''ไม่ควรจะมี'' ';
ERR_TAB[2]    := 'ควรจะมี ชื่อ,ประมวลผล ';
ERR_TAB[3]    := 'ควรจะมี '';'' ';
ERR_TAB[4]    := ควรจะมี''ตัวเลข''ตัวอักษร ชื่อ,ผิด ';
ERR_TAB[5]    := 'ควรจะมี ''('' ';
ERR_TAB[6]    := 'ผิด,ต้องมี ';
ERR_TAB[7]    := 'ควรจะมี ''ประโยค,ผิด'' ประโยค ชื่อ,ผิด ';
ERR_TAB[8]    := 'ควรจะมี ชื่อ,ผิด ';
ERR_TAB[9]    := 'ควรจะมี ''('' ประโยค ''' ';
ERR_TAB[10]   := 'ควรจะมี ''ประโยค,ผิด'' ';
ERR_TAB[11]   := 'ควรจะมี ''ตัวเลขผิด'' ';
ERR_TAB[12]   := 'ควรจะมี ''.'' ';
ERR_TAB[13]   := 'ควรจะมีชื่อ,ต้องมี ';
ERR_TAB[14]   := 'ประโยคผิดหลัก,ไม่ทราบ,ประเภท ';
ERR_TAB[15]   := 'ควรจะมี ''='' ';
ERR_TAB[16]   := 'ควรจะมี ''ผิด,'' ';
ERR_TAB[17]   := 'ควรจะมี ''('' ';
ERR_TAB[18]   := 'ควรจะมี ''('' ชื่อ ชื่อ,ผิด ประเภท ';
ERR_TAB[19]   := 'ควรจะมี ''ผิด,'' ';
ERR_TAB[20]   := 'ควรจะมี ''ตัวเลขผิดหลัก'' ';
ERR_TAB[21]   := 'ผิด,ประเภทของข้อมูลที่ต้องการ ';
ERR_TAB[22]   := ''' ควรจะมี,ประโยคผิดหลัก ';
ERR_TAB[23]   := 'ควรจะมี ''ผิด'' ';
```

```
  ERR_TABLE[24]    := '                                                ';
  ERR_TABLE[25]    := '                                                ';
  ERR_TABLE[26]    := '                                                ';
  ERR_TABLE[27]    := '                                                ';
  ERR_TABLE[28]    := '                                                ';
  ERR_TABLE[29]    := '                                                ';
  ERR_TABLE[30]    := '                                                ';
  ERR_TABLE[31]    := '                                                ';
  ERR_TABLE[32]    := '                                                ';
  ERR_TABLE[33]    := '                                                ';
  ERR_TABLE[34]    := '                                                ';
  ERR_TABLE[35]    := '                                                ';

  WRITE(CHR(12));
  WRITELN('                                                    ',VERSION);
  WRITELN('                                                    ');
  WRITELN('                                                    ');
  FILE_PTR := @CMD;
  NAME_BUFF := FILE_PTR^;
  IF LENGTH(NAME_BUFF)=0
    THEN
      BEGIN
        WRITE('                                                ');
        READLN(NAME_BUFF)
      END;
  INDEX:=1;
  WHILE NAME_BUFF[INDEX]=' ' DO
    INDEX:=INDEX+1;
  FILENAME:=COPY(NAME_BUFF,INDEX,LENGTH(NAME_BUFF)-INDEX+1);
  WHILE LENGTH(FILENAME)>8 DO
    BEGIN
      WRITE('                                                ');
      READLN(NAME_BUFF);
      INDEX:=1;
      WHILE NAME_BUFF[INDEX]=' ' DO
        INDEX:=INDEX+1;
      FILENAME:=COPY(NAME_BUFF,INDEX,LENGTH(NAME_BUFF)-INDEX+1)
    END;
  NAME:=CONCAT(FILENAME,'.TXT');
  ASSIGN(SOURCE,NAME);
  RESET(SOURCE);
  IF IORESULT=255
    THEN
      BEGIN
        WRITELN('                ',NAME,'                ');
        WRITELN('                ');
        HALT
      END;
  WRITELN('                ')
END;


PROCEDURE GET_TOKEN;
{* THIS PROGRAM IS LEXICAL ANALYZER OR SCANNER. SCANNER ACCEPTS SOURCE *}
{* PROGRAM IN THAT COMPUTER LANGUAGE AS INPUT THEN GROUPS THE CHARACTERS*}
{* INTO LEXICAL UNIT OR TOKEN. IT REMOVES BLANKS,COMMENTS AND REPORTS  *}
{* LEXICAL ERRORS.FINALLY IT GIVES TOKEN,TOKEN-TYPE AND TOKEN-NUMBER AS*}
{* AS OUTPUT.                                                          *}
PROCEDURE SERROR(NUM:INTEGER);
{* THIS SUBPROGRAM DISPLAYS SCANNER ERROR MASSAGE *}
BEGIN
  WRITELN('                ',LINE_NO:4,' : ',LINE);
  WRITELN('******',ERROR_MASS[NUM]);
  TOKEN_NO:=0;
  ERROR_NO:=ERROR_NO+1;
  WRITELN('                                                    ');
```

```
  READ(A_KEY);
  IF ORD(A_KEY)=CONTROL_C
    THEN
      QUIT
END;
PROCEDURE FILL_TABLE(VAR NODE:TABLE);
{* THIS SUBPROGRAM STORES THE DATA IN THE IDENTIFIER TABLE,NUMERIC TABLE*}
{* AND LITERAL TABLE.                                                  *}
BEGIN
  IF NODE<>NIL
    THEN
      BEGIN
        POINTS:=NODE;
        INDEX:=1;
        FIND_ID:=FALSE;
        END_LINE:=FALSE;
        LNG:=LENGTH(TOKEN);
        WHILE NOT END_LINE AND NOT FIND_ID DO
          BEGIN
            IF POINTS^.LEN=LNG
              THEN
                BEGIN
                  TEMP_COPY:='';
                  POSIT:=POINTS^.NM_PT;
                  SINDEX:=1;
                  WHILE SINDEX<=LNG DO
                    BEGIN
                      TEMP_COPY:=CONCAT(TEMP_COPY,POOL_STR[POSIT]);
                      POSIT:=POSIT+1;
                      SINDEX:=SINDEX+1
                    END;
                  WRITELN('NM_PT ',POINTS^.NM_PT,' TEMP-COPY',TEMP_COPY,' TYP ',POINTS^.TYP);
                  IF TEMP_COPY=TOKEN
                    THEN
                      FIND_ID:=TRUE
                END;
            IF NOT FIND_ID
              THEN
                IF POINTS^.LINK<>NIL
                  THEN
                    POINTS:=POINTS^.LINK
                  ELSE
                    END_LINE:=TRUE;
            INDEX:=INDEX+1
          END;
        IF NOT FIND_ID
          THEN
            IF NOT(END_DECLARE AND (TOKEN_TYPE='I'))
              THEN
                BEGIN
                  NEW(POINT_TEMP);
                  POINT_TEMP^.LINK:=NIL;
                  POINT_TEMP^.NM_PT:=POOL_TOP;
                  POINT_TEMP^.TYP:=TYPE_DATA;
                  SINDEX:=1;
                  WHILE SINDEX<=LNG DO
                    BEGIN
                      POOL_STR[POOL_TOP]:=TOKEN[SINDEX];
                      SINDEX:=SINDEX+1;
                      POOL_TOP:=POOL_TOP+1
                    END;
                  POINT_TEMP^.LEN:=LNG;
                  POINTS^.LINK:=POINT_TEMP;
                  TOKEN_NO:=INDEX+1
                END
              ELSE
```

```
                SERROR(4)
              ELSE
                TOKEN_NO:=INDEX
            END
          ELSE
            BEGIN
              IF NOT(END_DECLARE AND (TOKEN_TYPE='I'))
                THEN
                  BEGIN
                    NEW(POINT_TEMP);
                    POINT_TEMP^.LINE:=NIL;
                    POINT_TEMP^.NM_PT:=POOL_TOP;
                    POINT_TEMP^.TYP:=TYPE_DATA;
                    POINT_TEMP^.LEN:=LENGTH(TOKEN);
                    SINDEX:=1;
                    WHILE SINDEX<=LENGTH(TOKEN) DO
                      BEGIN
                        POOL_STR[POOL_TOP]:=TOKEN[SINDEX];
                        SINDEX:=SINDEX+1;
                        POOL_TOP:=POOL_TOP+1
                      END;
                    NODE:=POINT_TEMP;
                    TOKEN_NO:=1
                  END
                ELSE
                  SERROR(4)
              END
END;
PROCEDURE IDENTIFIER;
{  THIS SUBPROGRAM IS FOR MANIPULATION TOKEN THAT FIRST CHARACTER IS    }
{  }
{  THAI-CHARACTER.                                                      }
BEGIN
  WHILE NOT(LINE[CHCOUNT] IN SYMBOL) DO
    BEGIN
      IF LINE[CHCOUNT]='_'
        THEN
          BEGIN
            IF NOT(LINE[CHCOUNT] IN ['A'..'Z','a'..'z','0'..'9','_',THAI_VW_OVR,THAI_VW_ENT,THAI_VW_RHD,THAI_VW_URD,THAI_NOR
              THEN
                ERROR_FLAG:=TRUE;
            TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1))
          END;
      CHCOUNT:=CHCOUNT+1
    END;
  TOKEN_TYPE:='I';
  IF NOT ERROR_FLAG
    THEN
      BEGIN
        RSVI:=1;
        FOUND_FLAG:=FALSE;
        WHILE (RSVI<=25)AND(NOT FOUND_FLAG) DO
          BEGIN
            IF TOKEN=RSVWORD[RSVI]
              THEN
                BEGIN
                  TOKEN_TYPE:='R';
                  TOKEN_NO:=RSVI;
                  FOUND_FLAG:=TRUE
                END;
            RSVI:=RSVI+1
          END;
        IF (TOKEN_TYPE='R')AND((TOKEN_NO=23)OR(TOKEN_NO=24))
          THEN
            TOKEN_TYPE:='B';
        IF TOKEN_TYPE='I'
          THEN
```

```
            BEGIN
              IF LENGTH(TOKEN)>10
                THEN
                  TOKEN:=COPY(TOKEN,1,10);
              TYPE_DATA:=' ';
              FILL_TABLE(IDEN_FIRST)
            END
        END
      ELSE
        SERROR(3)
END;
PROCEDURE SPECIAL_CHAR;
(* THIS SUBPROGRAM IS FOR MANIPULATION TOKEN THAT FIRST CHARACTER IS    *)
(* SPECIAL CHARACTER                                                    *)
BEGIN
  TOKEN:=COPY(LINE,CHCOUNT,1);
  CHCOUNT:=CHCOUNT+1;
  SERROR(6)
END;
PROCEDURE SYMBOLIC;
(* THIS SUBPROGRAM IS FOR MANIPULATION TOKEN THAT FIRST CHARACTER IS    *)
(* SYMBOL CHARACTER.                                                    *)
VAR
  CNT:INTEGER;
BEGIN
  END_FILE:=FALSE;
  CASE LINE[CHCOUNT] OF
    '*':
      IF LINE[CHCOUNT+1]='*'
        THEN
          BEGIN
            TOKEN:=COPY(LINE,CHCOUNT,2);
            CHCOUNT:=CHCOUNT+2;
            TOKEN_TYPE:='X'
          END
        ELSE
          BEGIN
            TOKEN:=COPY(LINE,CHCOUNT,1);
            TOKEN_TYPE:=TOKEN[1];
            CHCOUNT:=CHCOUNT+1
          END;
    '/':
      BEGIN
        IF LINE[CHCOUNT+1]='*'
          THEN
            BEGIN
              COMPLETE:=FALSE;
              CHCOUNT:=CHCOUNT+2;
IF END_FILE THEN WRITELN('END_FILE');
              WHILE NOT END_FILE AND NOT COMPLETE DO
                BEGIN
                  WHILE (CHCOUNT<=LINE_LENGTH)AND(NOT COMPLETE) DO
                    BEGIN
                      IF (LINE[CHCOUNT]='*')AND(LINE[CHCOUNT+1]='/')
                        THEN
                          BEGIN
                            COMPLETE:=TRUE;
                            CHCOUNT:=CHCOUNT+1
                          END;
                      CHCOUNT:=CHCOUNT+1
                    END;
                  IF NOT COMPLETE
                    THEN
                      IF NOT(EOF(SOURCE))
                        THEN
                          NEW_LINE
```

```
                    ELSE
                      END_FILE:=TRUE
                  ELSE
                    TOKEN_TYPE:=' '
              END;
            IF NOT COMPLETE
              THEN
                SERROR(1);
          END
        ELSE
          BEGIN
            TOKEN:=LINE[CHCOUNT];
            TOKEN_TYPE:=TOKEN[1];
            CHCOUNT:=CHCOUNT+1
          END
      END;
    '/':
    BEGIN
      COMPLETE:=FALSE;
      CHCOUNT:=CHCOUNT+1;
      WHILE NOT END_FILE AND NOT COMPLETE DO
        BEGIN
          WHILE (CHCOUNT<=LINE_LENGTH)AND(NOT COMPLETE) DO
            BEGIN
              IF LINE[CHCOUNT]='/'
                THEN
                  BEGIN
                    COMPLETE:=TRUE;
                    CHCOUNT:=CHCOUNT+1
                  END;
              CHCOUNT:=CHCOUNT+1
            END;
          IF NOT COMPLETE
            THEN
              IF EOF(SOURCE)
                THEN
                  END_FILE:=TRUE
                ELSE
                  NEW_LINE
            ELSE
              TOKEN_TYPE:=' '
        END;
      IF NOT COMPLETE
        THEN
          SERROR(1)
    END;
  '=':
    IF LINE[CHCOUNT+1]='='
      THEN
        BEGIN
          TOKEN:=COPY(LINE,CHCOUNT,2);
          CHCOUNT:=CHCOUNT+2;
          TOKEN_TYPE:='E'
        END
      ELSE
        BEGIN
          TOKEN:=COPY(LINE,CHCOUNT,1);
          TOKEN_TYPE:=TOKEN[1];
          CHCOUNT:=CHCOUNT+1
        END;
  '''':
    BEGIN
      TOKEN:='''';
      CHCOUNT:=CHCOUNT+1;
      COMPLETE:=FALSE;
      CHT:=1;
```

```
WHILE (NOT END_FILE)AND(NOT COMPLETE)AND(CNT<=255) DO
  BEGIN
    WHILE (CHCOUNT<=LINE_LENGTH)AND(NOT COMPLETE) DO
      BEGIN
        IF LINE[CHCOUNT]='''
          THEN
            COMPLETE:=TRUE
          ELSE
            BEGIN
              CNT:=CNT+1;
              TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1))
            END;
        CHCOUNT:=CHCOUNT+1
      END;
    IF NOT COMPLETE
      THEN
        IF EOF(SOURCE) AND (CNT<255)
          THEN
            END_FILE:=TRUE
          ELSE
            NEW_LINE
        ELSE
          BEGIN
            TOKEN_TYPE:='L';
            IF (LENGTH(TOKEN))=1
              THEN
                TYPE_DATA:='C'
              ELSE
                TYPE_DATA:='L';
            FILL_TABLE(LIT_FIRST)
          END
  END
END;
'=','<','>':
  BEGIN
    TOKEN_TYPE:='P';
    CASE LINE[CHCOUNT] OF
      '=':
        IF LINE[CHCOUNT+1] IN ['<','>']
          THEN
            BEGIN
              CHCOUNT:=CHCOUNT+2;
              CASE LINE[CHCOUNT-1] OF
                '<':TOKEN:='<=';
                '>':TOKEN:='>='
              END
            END
          ELSE
            BEGIN
              TOKEN:='=';
              CHCOUNT:=CHCOUNT+1
            END;
      '<':
        IF LINE[CHCOUNT+1] IN ['=','>']
          THEN
            BEGIN
              CHCOUNT:=CHCOUNT+2;
              CASE LINE[CHCOUNT-1] OF
                '=':TOKEN:='<=';
                '>':TOKEN:='<>'
              END
            END
          ELSE
            BEGIN
              TOKEN:='<';
              CHCOUNT:=CHCOUNT+1
```

```
                END;
              ':':
                IF LINE[CHCOUNT+1] IN ['=','>']
                  THEN
                    BEGIN
                      CHCOUNT:=CHCOUNT+1;
                      CASE LINE[CHCOUNT-1] OF
                        '=':TOKEN:='=';
                        '>':TOKEN:='<>'
                      END
                    END
                  ELSE
                    BEGIN
                      TOKEN:=':';
                      CHCOUNT:=CHCOUNT+1
                    END
                END;
              ELSE
                BEGIN
                  TOKEN:=COPY(LINE,CHCOUNT,1);
                  TOKEN_TYPE:=TOKEN[1];
                  CHCOUNT:=CHCOUNT+1
                END
            END;
          IF END_FILE
            THEN
              ERROR(2)
      END;
      PROCEDURE NUMERIC;
      (* THIS SUBPROGRAM IS FOR MANIPULATION TOKEN THAT FIRST CHARACTER IS
      (* THAT_DIGIT.
      VAR
        NUM      : CHAR;
        NUM_TEMP : STRING[1];
        TEMP     : STRING;
      PROCEDURE CONVERT_NUMERIC;
      (* THIS SUBPROGRAM IS FOR CONVERSION THAT DIGIT TO GENERAL-DIGIT
      BEGIN
        TEMP:=COPY(LINE,CHCOUNT,1);
        NUM:=TEMP[1];
        NUM:=CHR(ORD(NUM)-48);
        NUM_TEMP:=NUM
      END;
      BEGIN
        WHILE NOT(LINE[CHCOUNT] IN SYMBOL) DO
          BEGIN
            IF NOT(LINE[CHCOUNT] IN THAT_NUM)
              THEN
                BEGIN
                  ERROR_FLAG:=TRUE;
                  TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1))
                END
              ELSE
                BEGIN
                  CONVERT_NUMERIC;
                  TOKEN:=CONCAT(TOKEN,COPY(NUM_TEMP,1,1))
                END;
            CHCOUNT:=CHCOUNT+1
          END;
        TOKEN_TYPE:='N';
        TYPE_DATA:='I';
        IF LINE[CHCOUNT]='.'
          THEN
            BEGIN
              TYPE_DATA:='R';
```

```
      TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1));
      CHCOUNT:=CHCOUNT+1;
      WHILE LINE[CHCOUNT] IN THAI_NUM DO
        BEGIN
          CONVERT_NUMERIC;
          TOKEN:=CONCAT(TOKEN,COPY(NUM_TEMP,1,1));
          CHCOUNT:=CHCOUNT+1
        END;
      IF LINE[CHCOUNT]='/'
        THEN
          BEGIN
            TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1));
            CHCOUNT:=CHCOUNT+1;
            IF NOT(LINE[CHCOUNT] IN ['0','-'])
              THEN
                BEGIN
                  SERROR(7);
                  TOKEN:=CONCAT(TOKEN,'0')
                END
              ELSE
                BEGIN
                  TOKEN:=CONCAT(TOKEN,COPY(LINE,CHCOUNT,1));
                  CHCOUNT:=CHCOUNT+1
                END;
            WHILE LINE[CHCOUNT] IN THAI_NUM DO
              BEGIN
                CONVERT_NUMERIC;
                TOKEN:=CONCAT(TOKEN,COPY(NUM_TEMP,1,1));
                CHCOUNT:=CHCOUNT+1
              END
          END
      END;
  IF NOT ERROR_FLAG
    THEN
      FILL_TABLE(NUM_FIRST)
    ELSE
      SERROR(5)
END;
PROCEDURE NEW_LINE;
{* THIS SUBPROGRAM CHECKS END OF FILE. IF NOT, IT WILL READ NEXT RECORD *)
{* AND STORES IN THE BUFFER                                             *}
BEGIN
  IF EOF(SOURCE)
    THEN
      END_FILE:=TRUE
    ELSE
      BEGIN
        READLN(SOURCE,LINE);
        LINE_LENGTH:=LENGTH(LINE);
        LINE_NO:=LINE_NO+1;
        CHCOUNT:=1;
        LINE:=CONCAT(LINE,' ');
        WRITELN(LINE_NO,' ',LINE)
      END
END;
BEGIN {* MAIN PROGRAM OF GET_TOKEN *}
  REPEAT
    BEGIN
      IF (CHCOUNT=0)OR(CHCOUNT>LINE_LENGTH)
        THEN
          NEW_LINE;
      IF NOT END_FILE
        THEN
          BEGIN
            TOKEN:='';
            ERROR_FLAG:=FALSE;
```

```
                TOKEN_TYPE:=' ';
                TOKEN_NO:=0;
                WHILE LINE[CHCOUNT]=' ' DO
                  CHCOUNT:=CHCOUNT+1;
                IF LINE[CHCOUNT] IN THAI_NUM
                  THEN
                    NUMERIC
                  ELSE
                    IF LINE[CHCOUNT] IN THAI_CHAR+THAI_VN_OVE+THAI_VN_FRT+THAI_VN_END+THAI_VN_UND
                      THEN
                        IDENTIFIER
                      ELSE
                        IF LINE[CHCOUNT] IN SYMBOL
                          THEN
                            SYMBOLIC
                          ELSE
                            SPECIAL_CHAR
              END
            END
  UNTIL TOKEN_TYPE<>' ';
      IF (TOKEN_TYPE='R')AND(TOKEN_NO=6)
        THEN
          END_DECLARE:=TRUE
END;


PROCEDURE ERROR(NUM:INTEGER);
{* THIS SUBPROGRAM DISPLAYS SYNTACTIC AND SEMANTIC ERROR MESSAGE.*}
BEGIN
  WRITELN('ข้อผิดพลาดที่ ',ERR_TABLE[NUM]);
  WRITELN('ข้อความที่ผิดพลาดอยู่ที่บรรทัดที่ ',LINE_NO);
  WRITE('กรุณาแก้ไขข้อผิดพลาดนี้ แล้วกดแป้นใดๆ เพื่อดำเนินการต่อ ');
  ERROR_NO:=ERROR_NO+1;
  READ(A_KEY);
  IF ORD(A_KEY)=CONTROL_C
    THEN
      HALT
END;


FUNCTION CONV_INT(VAR OPR1:STRING):INTEGER;
VAR
  MINUS:BOOLEAN;
  N,X,Y:INTEGER;
  C_TEMP:CHAR;
  I_TEMP:INTEGER;
  OVRFLW:BOOLEAN;
BEGIN
  N:=1;
  X:=1;
  Y:=0;
  IF OPR1[1]='-'
    THEN
      BEGIN
        MINUS:=TRUE;
        N:=2
      END
    ELSE
      MINUS:=FALSE;
  C_TEMP:=OPR1[N];
  I_TEMP:=ORD(C_TEMP)-48;
  OVRFLW:=FALSE;
  WHILE (N<=LENGTH(OPR1))AND(X<=5) DO
    BEGIN
      IF (X<5)OR((X=5)AND(Y<32766))OR((Y=32766)AND(I_TEMP<=7))
        THEN
```

```
            BEGIN
              Y:=I_TEMP*(EXP(10));
              N:=N+1;
              X:=X+1;
              C_TEMP:=OPR1[N];
              I_TEMP:=ORD(C_TEMP)-48;
            END
          ELSE
            OVRFLO:='TRUE'
      END;
    IF MINUS
      THEN
        CONV_INT:=-Y
      ELSE
        CONV_INT:=Y
END;
FUNCTION CONV_REAL(VAR OPR1:STRING):REAL;
VAR
  C_TEMP:CHAR;
  MINUS,PLUS:BOOLEAN;
  N,X,NI:INTEGER;
  Z,Y:REAL;
BEGIN
  N:=1;
  X:=1;
  NI:=0;
  Z:=0.0;
  Y:=0.0;
  IF OPR1[1]='-'
    THEN
      BEGIN
        MINUS:=TRUE;
        N:=2
      END
    ELSE
      MINUS:=FALSE;
  WHILE (N<=LENGTH(OPR1))AND(OPR1[N]<>'.') DO
    BEGIN
      C_TEMP:=OPR1[N];
      Y:=(ORD(C_TEMP)-48)+(Y*10);
      N:=N+1;
      NI:=NI+1
    END;
  N:=NI+1;
  WHILE (N<=LENGTH(OPR1))AND(OPR1[N]<>'0') DO
    BEGIN
      C_TEMP:=OPR1[N];
      Z:=Z+(ORD(C_TEMP)-48)*(EXP(X*LN(10)));
      N:=N+1;
      X:=X+1
    END;
  Y:=Y+Z;
  X:=0;
  Z:=0.0;
  IF OPR1[N]='0'
    THEN
      BEGIN
        N:=N+1;
        X:=1;
        PLUS:=FALSE;
        IF OPR1[N]='+'
          THEN
            PLUS:=TRUE;
        N:=N+1;
        WHILE (N<=LENGTH(OPR1))AND(X<=3) DO
          BEGIN
```

```
            C_TEMP:=OPR1[N];
            C:=(ORD(C_TEMP)-48)+(C*10.0);
            N:=N+1;
            X:=X+1
          END;
        IF N<LENGTH(OPR1)
          THEN
            BEGIN
              IF NOT PLUS
                THEN
                  C:=-C;
              IF ((C=N-1(30))OR((C=N-1)*-30))OR(((C=N-1)*30)AND(Y<>1))OR(((C=N-1)*-30)AND(Y<>-1))
                THEN
                  C:=Y*(EXP(C*LN(10)))
                ELSE
                  BEGIN
                    C:=0.0;
                    ERROR(34)
                  END
            END
          ELSE
            BEGIN
              C:=0.0;
              ERROR(34)
            END
      END
    ELSE
      C:=Y;
  IF MINUS
    THEN
      CONV_REAL:=-C
    ELSE
      CONV_REAL:=C
END;
PROCEDURE INSERTTAB(X:STRING;Y:CHAR);
VAR
  X_TEMP:STRING;
  N:INTEGER;
BEGIN
  N:=1;
  X_TEMP:='';
  X:=CONCAT(X,'*');
  WHILE N<=LENGTH(X) DO
    BEGIN
      IF (X[N]=',')OR(X[N]='*')
        THEN
          BEGIN
            POINTS:=IDEN_FIRST;
            FIND_ID:=FALSE;
            END_LINK:=FALSE;
            LNG:=LENGTH(X_TEMP);
            WHILE NOT END_LINK AND NOT FIND_ID DO
              BEGIN
                IF POINTS^.LEN=LNG
                  THEN
                    BEGIN
                      TEMP_COPY:='';
                      POSIT:=POINTS^.HM_PT;
                      SINDEX:=1;
                      WHILE SINDEX<=LNG DO
                        BEGIN
                          TEMP_COPY:=CONCAT(TEMP_COPY,POOL_STR[POSIT]);
                          SINDEX:=SINDEX+1;
                          POSIT:=POSIT+1
                        END;
                      IF TEMP_COPY=X_TEMP
```

```
                    THEN
                      FIND_ID:=TRUE
                  END;
                IF NOT FIND_ID
                  THEN
                    IF POINTS^.LINK<>NIL
                      THEN
                        POINTS:=POINTS^.LINK
                      ELSE
                        END_LINK:=TRUE
              END;
            IF FIND_ID
              THEN
                IF X_TEMP[1]<>'@'
                  THEN
                    IF POINTS^.TYP=' '
                      THEN
                        POINTS^.TYP:='V'
                      ELSE
                        ERROR(30)
                  ELSE
                    POINTS^.TYP:='C'
              ELSE
                IF END_LINK
                  THEN
                    IF X_TEMP[1]='@'
                      THEN
                        BEGIN
                          NEW(POINT_TEMP);
                          POINT_TEMP^.LINK:=NIL;
                          POINT_TEMP^.NM_PT:=POOL_TOP;
                          POINT_TEMP^.TYP:='V';
                          POINT_TEMP^.LEN:=LENGTH(X_TEMP);
                          POINTS^.LINK:=POINT_TEMP;
                          SINDEX:=1;
                          WHILE SINDEX<=LNG DO
                            BEGIN
                              POOL_STR[POOL_TOP]:=X_TEMP[SINDEX];
                              SINDEX:=SINDEX+1;
                              POOL_TOP:=POOL_TOP+1
                            END
                        END
                      ELSE
                        ERROR(35)
                  END
                ELSE
                  X_TEMP:=CONCAT(X_TEMP,COPY(X,N,1));
              N:=N+1
            END
END;


PROCEDURE PARSER;
VAR
  NUMBER:STRING;
PROCEDURE CODE(OPERATOR:STRING;OPERAN1:STRING;OPERAN2:STRING);
(* THIS PROGRAM IS CALLED SEMANTIC ROUTINE THAT GENERATES THE      *)
(* INTERMEDIATE CODE PROGRAM OR OBJECT PROGRAM AS AN OUTPUT OF     *)
(* THAT COMPUTER LANGUAGE COMPILER AND IT IS CALLED BY PARSER THAT IS *)
(* SYNTAX DIRECTED TRANSLATION.                                    *)
VAR
  TYPE1,TYPE2 : CHAR;
  TEMP_LABEL  : STRING;
  NO_STR      : STRING;
PROCEDURE CHG_TO_STR(NO:INTEGER);
(* THIS SUBPROGRAM CONVERTS NUMBER THAT IN INTEGER TYPE INTO CHARACTER *)
```

```
(* TYPE.                                                          *)
VAR
  FIRST,SECOND,THIRD:INTEGER;
BEGIN
  NO_STR:='';
  FIRST:=NO DIV 100;
  IF FIRST<>0
    THEN
      NO_STR:=CONCAT(NO_STR,CHR(FIRST+48));
  SECOND:=(NO-FIRST*100) DIV 10;
  IF NOT((NO_STR='')AND(SECOND=0))
    THEN
      NO_STR:=CONCAT(NO_STR,CHR(SECOND+48));
  THIRD:=NO-FIRST*100-SECOND*10;
  NO_STR:=CONCAT(NO_STR,CHR(THIRD+48))
END;
PROCEDURE GET_TEMP(P:CHAR);
(* THIS SUBPROGRAM GENERATES THE TEMPORARY VARIABLE FOR USING IN  *)
(* ARITHMETIC,RELATIONAL AND LOGICAL OPERATION.                   *)
BEGIN
  CUR_TEMP_NO:=CUR_TEMP_NO+1;
  CHG_TO_STR(CUR_TEMP_NO);
  TEMP:=CONCAT('@T',NO_STR);
WRITELN('TEMP ',TEMP);
  INSERTTAB(TEMP,P)
END;
PROCEDURE FREE_TEMP;
(* THIS SUBPROGRAM RELEASES THE TEMPORARY THAT HAS ALREADY USED TO *)
(* REUSE IN NEXT CALL.                                            *)
BEGIN
  CUR_TEMP_NO:=CUR_TEMP_NO-1;
  IF CUR_TEMP_NO<-1
    THEN
      ERROR(20)
END;
PROCEDURE FIND_TYPE(OPER:STRING);
(* THIS SUBPROGRAM FINDS THE DATA TYPE OF INPUT DATA AND PREPARES *)
(* STARTING POINT OF IDENTIFIER,NUMERIC AND LITERAL TABLE.        *)
BEGIN
  IF (OPER='$T@1')OR(OPER='$T@2')
    THEN
      BEGIN
        TAB:=4;
        TYPES:='@'
      END
    ELSE
      BEGIN
        IF OPER[1] IN ['0','1','2','3','4','5','6','7','8','9']
          THEN
            BEGIN
              POINTS:=NUM_FIRST;
              TAB:=3
            END
          ELSE
            IF OPER[1]=''''
              THEN
                BEGIN
                  POINTS:=LIT_FIRST;
                  TAB:=2
                END
              ELSE
                IF OPER[1]='@'
                  THEN
                    BEGIN
                      POINTS:=IDEN_FIRST;
                      TAB:=1
```

```
                    END
                  ELSE
                    BEGIN
                      POINTS:=IDEN_FIRST;
                      TAB:=0
                    END;
  FIND_ID:=FALSE;
  END_LINK:=FALSE;
  LNG:=LENGTH(OPER);
  WHILE NOT END_LINK AND NOT FIND_ID DO
    BEGIN
      IF POINTS^.LEN=LNG
        THEN
          BEGIN
            TEMP_COPY:='';
            POSIT:=POINTS^.NM_PT;
            INDEX:=1;
            WHILE INDEX<=LNG DO
              BEGIN
                TEMP_COPY:=CONCAT(TEMP_COPY,POOL_STR[POSIT]);
                POSIT:=POSIT+1;
                INDEX:=INDEX+1
              END;
            IF TEMP_COPY=OPER
              THEN
                FIND_ID:=TRUE
          END;
      IF NOT FIND_ID
        THEN
          IF POINTS^.LINK<>NIL
            THEN
              POINTS:=POINTS^.LINK
            ELSE
              END_LINK:=TRUE
        END;
      IF FIND_ID
        THEN
          TYPES:=POINTS^.TYP
        ELSE
          TYPES:=' ';
  WRITELN('TYPES ',TYPES,' OPER ',OPER);
    END
END;
PROCEDURE EMIT_ALL;
{ THIS SUBPROGRAM IS CALLED BY SEMANTIC ROUTINE TO GENERATE THE NEW   }
{ FILE WHICH IS IN INTERMEDIATE CODE. THE FIRST PART IS PROGRAM.THE   }
{ SECOND PART IS DATA.                                               }
TYPE
  OBJ_PROG=RECORD
    CASE INSTR_TYPE:CHAR OF
      'P':(OP_CODE:CHAR;OPR1:INTEGER;OPR2:INTEGER);
      'D':(CASE DATA:CHAR OF
             'T':();
             'I':(INT:INTEGER);
             'R':(REA:REAL);
             'B':(BOOL:BOOLEAN);
             'C':(CH:CHAR);
             'L':(LIT:STRING[255]))
      END;
  OBJ_FILE=FILE OF OBJ_PROG;
VAR
  DECI:OBJ_FILE;
  OBJ:OBJ_PROG;
  I:INTEGER;
BEGIN
  CLOSE(SOURCE,RESULT);
```

```
RESET(TEMP_FILE);
NAME:=CONCAT(FILENAME,'.OBJ');
ASSIGN(DECE,NAME);
REWRITE(DECE);
PROG_COUNT:=1;
READ(TEMP_FILE,TMP);
OBJ.INSTR_TYPE:='P';
OBJ.OP_CODE:=TMP.OP_CD;
OBJ.OPR1:=TMP.OP1;
OBJ.OPR2:=TMP.OP2;
WITH OBJ DO
  BEGIN
    WHILE OP_CODE<>'T' DO
      BEGIN
        IF OP_CODE IN ['J','C','F','G']
          THEN
            IF OPR1=0
              THEN
                OPR2:=LABEL_TABL[OPR2];
        WRITE(DECE,OBJ);
        PROG_COUNT:=PROG_COUNT+1;
        READ(TEMP_FILE,TMP);
        OBJ.OP_CODE:=TMP.OP_CD;
        OBJ.OPR1:=TMP.OP1;
        OBJ.OPR2:=TMP.OP2
      END
  END;
OBJ.OP_CODE:='T';
OBJ.OPR1:=0;
OBJ.OPR2:=0;
WRITE(DECE,OBJ);
I:=1;
OBJ.INSTR_TYPE:='D';
WHILE J<1 DO
  BEGIN
    CASE TBLE[J].DEF OF
      0:
       BEGIN
         OBJ.DATA:=TBLE[I].DAT;
         CASE OBJ.DATA OF
           'I':OBJ.INT:=0;
           'R':OBJ.REA:=0.0;
           'B':OBJ.BOOL:=FALSE;
           'C':OBJ.CH:=' '
         END
       END;
      1:OBJ.DATA:='T';
      2:
       BEGIN
         OBJ.DATA:=TBLE[I].DAT;
         CASE OBJ.DATA OF
           'C':
             OBJ.CH:=TBLE[I].C;
           'L':
             OBJ.LIT:=TBLE[I].LT;
         END
       END;
      3:
       BEGIN
         OBJ.DATA:=TBLE[I].DAT;
         CASE OBJ.DATA OF
           'I':
             OBJ.INT:=TBLE[I].ING;
           'R':
             OBJ.REA:=TBLE[I].RE;
         END
```

```
          END;
        4:
          BEGIN
            OBJ.DATA:=TBLE[J].DAT;
            OBJ.BOOL:=TBLE[J].BL
          END
        END;
      WRITE(DECL,OBJ);
      J:=J+1
    END;
  CLOSEDEL(TEMP_FILE,RESULT);
  CLOSE(DECL,RESULT)
END;
PROCEDURE FILL_STORAGE(OPE:STRING);
{ THIS SUBPROGRAM FILLS THE DATA IN THE SYMBOL TABLE.                    }
VAR
  I:INTEGER;
  SEARCH_NOT_FOUND:BOOLEAN;
  TEMPB:BOOLEAN;
BEGIN
  SEARCH_NOT_FOUND:=TRUE;
  J:=1;
  WHILE (J<=(I-1))AND SEARCH_NOT_FOUND DO
    BEGIN
      CASE TBLE[J].DEF OF
        0,1:
          BEGIN
            IF (TAB=TBLE[J].DEF)AND(OPE=TBLE[J].NM)
              THEN
                SEARCH_NOT_FOUND:=FALSE
          END;
        2,3,4:
          BEGIN
            IF (TAB=TBLE[J].DEF)AND(TYPES=TBLE[J].DAT)
              THEN
                CASE TBLE[J].DAT OF
                  'I':
                    BEGIN
                      IF (TBLE[J].ING=CONV_INT(OPE))
                        THEN
                          SEARCH_NOT_FOUND:=FALSE
                    END;
                  'R':
                    BEGIN
                      IF (TBLE[J].RE=CONV_REAL(OPE))
                        THEN
                          SEARCH_NOT_FOUND:=FALSE
                    END;
                  'B':
                    BEGIN
                      IF OPE='TRUE'
                        THEN
                          J:=1
                        ELSE
                          IF OPE='FALSE'
                            THEN
                              J:=2;
                      SEARCH_NOT_FOUND:=FALSE
                    END;
                  'C':
                    BEGIN
                      IF (TBLE[J].C=OPE[1])
                        THEN
                          SEARCH_NOT_FOUND:=FALSE
                    END;
                  'L':
```

```
                    BEGIN
                      IF (TBLE[I].LT=COPY(OPR,2,LENGTH(OPR)-1))
                        THEN
                          SEARCH_NOT_FOUND:=FALSE
                      END
                  END
              END
          END;
        I:=I+1
      END;
    IF NOT SEARCH_NOT_FOUND
      THEN
        ADDR:=I-1
      ELSE
        BEGIN
          CASE TBLE[I].DLT OF
            0:
              BEGIN
                TBLE[I].DAT:=TYPES;
                TBLE[I].NM:=OPR
              END;
            1:
              BEGIN
                TBLE[I].DAT:='T';
                TBLE[I].NM:=OPR
              END;
            2:
              BEGIN
                TBLE[I].DAT:=TYPES;
                CASE TBLE[I].DAT OF
                  'C':TBLE[I].C:=OPR[2];
                  'L':TBLE[I].LT:=COPY(OPR,2,LENGTH(OPR)-1)
                END
              END;
            3:
              BEGIN
                TBLE[I].DAT:=TYPES;
                CASE TBLE[I].DAT OF
                  'I':TBLE[I].ING:=CONV_INT(OPR);
                  'R':TBLE[I].RE:=CONV_REAL(OPR)
                END
              END;
            4:
              BEGIN
                TBLE[I].DAT:=TYPES;
                IF OPR='TRUE'
                  THEN
                    TBLE[I].BL:=TRUE
                  ELSE
                    IF OPR='FALSE'
                      THEN
                        TBLE[I].BL:=FALSE
              END
          END;
          ADDR:=I;
          I:=I+1
        END
  END;
PROCEDURE EMIT(LAB:STRING;OP_CODE:CHAR;OPR1:STRING;OPR2:STRING);
{* THIS SUBPROGRAM GENERATES THE INTERMEDIATE CODE STORES IT IN THE   *}
{* TEMPORARY FILE.                                                    *}
VAR
  I,J:INTEGER;
  C_TEMP:CHAR;
  N,Z:INTEGER;
  PLUS:BOOLEAN;
```

```
  S:STRING;
BEGIN
  WRITELN(LAB,'  ',OP_CODE,'  ',OPR1,'  ',OPR2);
  PROG_COUNT:=PROG_COUNT+1;
  I:=0;
  I2:=0;
  IF OP_CODE='O'
    THEN
      BEGIN
        TBLE[1].DEF:=4;
        TBLE[1].BL:=TRUE;
        TBLE[1].DAT:='1';
        I:=1+1;
        TBLE[I].DEF:=4;
        TBLE[I].BL:=FALSE;
        TBLE[I].DAT:='0';
        I:=I+1
      END;
  IF NOT(OP_CODE IN ['O','E','T','S','C','I','F','G'])
    THEN
      BEGIN
        FIND_TYPE(OPR1);
        TBLE[I].DEF:=TAB;
        FILL_STORAGE(OPR1);
        I:=ADDR
      END
    ELSE
      IF OP_CODE='F'
        THEN
          BEGIN
            C_TEMP:=OPR1[1];
            I:=ORD(C_TEMP)-48
          END
        ELSE
          IF OP_CODE IN ['S','C','I','G']
            THEN
              IF OPR1='4'
                THEN
                  BEGIN
                    N:=1;
                    I:=0;
                    PLUS:=FALSE;
                    IF OPR2[N]='+'
                      THEN
                        PLUS:=TRUE;
                    N:=N+1;
                    WHILE N<=LENGTH(OPR2) DO
                      BEGIN
                        C_TEMP:=OPR2[N];
                        I:=(ORD(C_TEMP)-48)+(I*10);
                        N:=N+1
                      END;
                    IF NOT PLUS
                      THEN
                        I:=-I;
                    I2:=PROG_COUNT+1;
                    I:=$FFFF
                  END
                ELSE
                  BEGIN
                    S:=COPY(OPR2,3,LENGTH(OPR2)-2);
                    N:=1;
                    I:=0;
                    WHILE N<=LENGTH(S) DO
                      BEGIN
                        C_TEMP:=S[N];
```

```
                                (* ... *)
                                I:=ORD(C_TEMP)-48+I*10;
                                N:=N+1
                              END;
                            I:=I
                          END;
    IF OP_CODE='T'
      THEN
        BEGIN
          TMP.LABEL_NM:='';
          TMP.OP_CD:='T';
          TMP.OP1:=0;
          TMP.OP2:=0;
          WRITE(TEMP_FILE,TMP);
          UNIT_ALL
        END
      ELSE
        BEGIN
          IF (OP_CODE IN ['M','+','-','*','/','Y','P','A','R'])OR((OP_CODE='F')AND(OPR2<>''))
            THEN
              BEGIN
                FIND_TYPE(OPR2);
                TABLE[I].BUF:=TAB;
                FILL_STORAGE(OPR2);
                I2:=ADDR
              END;
          TMP.LABEL_NM:=LAB;
          TMP.OP_CD:=OP_CODE;
          TMP.OP1:=I1;
          TMP.OP2:=I2;
          WRITE(TEMP_FILE,TMP);
          IF LAB<>''
            THEN
              BEGIN
                S:=COPY(LAB,3,LENGTH(LAB)-2);
                N:=1;
                I:=0;
                WHILE N<=LENGTH(S) DO
                  BEGIN
                    C_TEMP:=S[N];
                    I:=ORD(C_TEMP)-48+I*10;
                    N:=N+1
                  END;
                LABEL_TABE[I]:=PROG_COUNT
              END
            END
END;


PROCEDURE PROLOGUE_EMIT;
(* THIS SUBPROGRAM EMITS THE PROGRAM PROLOGUE *)
BEGIN
  EMIT(OPERAND1,'O','','')
END;
PROCEDURE EPILOGUE_EMIT;
(* THIS SUBPROGRAM EMITS THE PROGRAM EPILOGUE *)
BEGIN
  EMIT('','T','','')
END;
PROCEDURE THEN_CODE_EMIT;
(*_THIS SUBPROGRAM EMITS CODE WHICH FOLLOWS 'IF,T' AND STATEMENT      *)
(* PREDICATE.                                                         *)
BEGIN
  LABEL_COUNT:=LABEL_COUNT+1;
  CHG_TO_STR(LABEL_COUNT);
  TEMP_LABEL:=CONCAT('@L',NO_STR);
  EMIT('F',TL.OPERAND1,'')
```

```
    IF OPERAND1[1]='@'
      THEN
        FREE_TEMP;
    EMIT('','G','',TEMP_LABEL);
    PUSH_OPR(TEMP_LABEL)
  END;
  PROCEDURE ELSE_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE WHICH FOLLOWS 'ELSE' CLAUSE OF      *)
  (* 'IF-THEN' STATEMENT.                                           *)
  BEGIN
    LABEL_COUNT:=LABEL_COUNT+1;
    CHG_TO_STR(LABEL_COUNT);
    TEMP_LABEL:=CONCAT('@L',NO_STR);
    EMIT('','J','',TEMP_LABEL);
    EMIT(OPERAND1,'E','*','');
    PUSH_OPR(TEMP_LABEL)
  END;
  PROCEDURE FIF_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE WHICH FOLLOWS END OF 'IF-THEN' STATEMENT.*)
  BEGIN
    EMIT(OPERAND1,'E','*','')
  END;
  PROCEDURE LOOP_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE BEGINNING OF 'WHILE' STATEMENT.     *)
  BEGIN
    LABEL_COUNT:=LABEL_COUNT+1;
    CHG_TO_STR(LABEL_COUNT);
    TEMP_LABEL:=CONCAT('@L',NO_STR);
    EMIT(TEMP_LABEL,'E','*',  );
    PUSH_OPR(TEMP_LABEL)
  END;
  PROCEDURE ENDLOOP_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE WHICH FOLLOWS END OF 'WHILE' STATEMENT.  *)
  BEGIN
    EMIT('','J','',OPERAND1);
    EMIT(OPERAND2,'E','*','')
  END;
  PROCEDURE EXIT_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE FOLLOWING 'EXIT' STATEMENT.*)
  BEGIN
    LABEL_COUNT:=LABEL_COUNT+1;
    CHG_TO_STR(LABEL_COUNT);
    TEMP_LABEL:=CONCAT('@L',NO_STR);
    EMIT('','L',OPERAND1,'');
    IF OPERAND1[1]='@'
      THEN
        FREE_TEMP;
    EMIT('','G','',TEMP_LABEL);
    PUSH_OPR(TEMP_LABEL)
  END;
  PROCEDURE READ_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE FOLLOWING 'READ' STATEMENT.*)
  BEGIN
    IF OPERATOR='READ'
      THEN
        BEGIN
          IF OPERAND1<>''
            THEN
              EMIT('','F','0',OPERAND1)
        END
      ELSE
        EMIT('','F','1','')
  END;
  PROCEDURE WRITE_CODE_EMIT;
  (* THIS SUBPROGRAM EMITS CODE FOLLOWING 'WRITE' STATEMENT.*)
  BEGIN
```

```
  NUMBER:='';
  IF OPERATOR='WRITE'
    THEN
      BEGIN
        IF PETR_ON
          THEN
            NUMBER:='2'
          ELSE
            NUMBER:='3';
        IF OPERAND2<>''
          THEN
            EMIT('','F',NUMBER,OPERAND2)
      END
    ELSE
      EMIT('','F','6','')
END;
PROCEDURE ARITH_CHK_TYPE;
(* THIS SUBPROGRAM CHECKS THE OPERANDS' TYPE FOR ARITHMETIC OPERATION.*)
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  FIND_TYPE(OPERAND2);
  TYPE2:=TYPES;
  IF NOT((TYPE1 IN ['R','I']) AND (TYPE2 IN ['R','I']))
    THEN
      ERROR(31);
  IF OPERAND1[1]<>'@'
    THEN
      BEGIN
        GET_TEMP(TYPE1);
        EMIT('','M ',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END
END;
PROCEDURE ADD_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR ADDITION.*)
BEGIN
  ARITH_CHK_TYPE;
  EMIT('','+',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP;
  PUSH_OPR(OPERAND1)
END;
PROCEDURE SUB_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR SUBSTRACTION.*)
BEGIN
  ARITH_CHK_TYPE;
  EMIT('','-',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP;
  PUSH_OPR(OPERAND1)
END;
PROCEDURE MULT_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR MULTIPLICATION.*)
BEGIN
  ARITH_CHK_TYPE;
  EMIT('','*',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP;
  PUSH_OPR(OPERAND1)
END;
PROCEDURE DIV_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR DIVISION.*)
```

```
BEGIN
  ARITH_CHK_TYPE;
  EMIT('','-',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP;
  PUSH_OPR(OPERAND1)
END;
PROCEDURE EXP_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR EXPONENTIAL.*)
BEGIN
  ARITH_CHK_TYPE;
  EMIT('','X',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP;
  PUSH_OPR(OPERAND1)
END;
PROCEDURE ASSIGN_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOLLOWING ASSIGNMENT STATEMENT.*)
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  FIND_TYPE(OPERAND2);
  TYPE2:=TYPES;
  IF NOT((TYPE1=TYPE2)OR((TYPE1='R')AND(TYPE2='I')))
    THEN
      ERROR(32);
  IF OPERAND1<>OPERAND2
    THEN
      EMIT('','M',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP
END;
PROCEDURE REL_CHK_TYPE;
(* THIS SUBPROGRAM CHECKS THE OPERANDS' TYPE FOR RELATIONAL OPERATION.*)
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  FIND_TYPE(OPERAND2);
  TYPE2:=TYPES;
  IF NOT((TYPE1=TYPE2)OR((TYPE1='R')AND(TYPE2='I')))
    THEN
      ERROR(32);
  IF OPERAND1[1]<>'@'
    THEN
      BEGIN
        GET_TEMP(TYPE1);
        EMIT('','M',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END;
  EMIT('','P',OPERAND1,OPERAND2);
  IF OPERAND1[1]='@'
    THEN
      FREE_TEMP;
  GET_TEMP('B')
END;
PROCEDURE EQU_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING WHETHER OPERAND1 EQUALS    *)
(* OPERAND2.                                                         *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'TRUE');
  EMIT('','C','=','1');
  EMIT('','M',TEMP,'FALSE');
```

```
  PUSH_OPR(TEMP)
END;
PROCEDURE NEQ_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING WHETHER OPERAND1 UNEQUALS  *)
(* OPERAND2.                                                         *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'FALSE');
  EMIT('','C','*','#1');
  EMIT('','M',TEMP,'TRUE');
  PUSH_OPR(TEMP)
END;
PROCEDURE LEQ_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING OPERAND1 IS LESS THAN OR   *)
(* EQUAL TO OPERAND2.                                                *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'TRUE');
  EMIT('','C','*','#2');
  EMIT('','C','*','#1');
  EMIT('','M',TEMP,'FALSE');
  PUSH_OPR(TEMP)
END;
PROCEDURE GEQ_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING OPERAND1 IS GREATER THAN OR *)
(* EQUAL TO OPERAND2.                                                 *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'TRUE');
  EMIT('','C','*','#3');
  EMIT('','M',TEMP,'FALSE');
  EMIT('','C','*','#1');
  EMIT('','M',TEMP,'TRUE');
  PUSH_OPR(TEMP)
END;
PROCEDURE LSS_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING WHETHER OPERAND1 IS LESS   *)
(* THAN OPERAND2.                                                    *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'FALSE');
  EMIT('','C','*','#2');
  EMIT('','M',TEMP,'TRUE');
  EMIT('','C','*','#1');
  EMIT('','M',TEMP,'FALSE');
  PUSH_OPR(TEMP)
END;
PROCEDURE GTR_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR TESTING WHETHER OPERAND1 IS GREATER *)
(* THAN OPERAND2.                                                     *)
BEGIN
  REL_CHK_TYPE;
  EMIT('','M',TEMP,'FALSE');
  EMIT('','C','*','#2');
  EMIT('','C','*','#1');
  EMIT('','M',TEMP,'TRUE');
  PUSH_OPR(TEMP)
END;
PROCEDURE NOT_CODE_EMIT;
(* THIS SUBPROGRAM EMITS CODE FOR 'NOT' OPERAND1 WITH OPERAND2.*)
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPOS;
  IF TYPE1 <> 'B'
    THEN
      ERROR(5);
```

```
  IF OPERAND1[1] <> '@'
    THEN
      BEGIN
        GET_TEMP(TYPE1);
        EMIT('','M',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END;
  EMIT('','M',OPERAND1,'');
  PUSH_OPR(OPERAND1)
END;
PROCEDURE AND_CODE_EMIT;
{* THIS SUBPROGRAM EMITS CODE FOR 'AND' OPERAND1.*}
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  FIND_TYPE(OPERAND2);
  TYPE2:=TYPES;
  IF NOT((TYPE1='B')AND(TYPE2='B'))
    THEN
      ERROR(31);
  IF OPERAND1[1] <> '@'
    THEN
      BEGIN
        GET_TEMP('B');
        EMIT('','M',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END;
  EMIT('','A',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP
END;
PROCEDURE OR_CODE_EMIT;
{* THIS SUBPROGRAM EMITS CODE FOR 'OR' OPERAND1 WITH OPERAND2.*}
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  FIND_TYPE(OPERAND2);
  TYPE2:=TYPES;
  IF NOT((TYPE1='B')AND(TYPE2='B'))
    THEN
      ERROR(31);
  IF OPERAND1[1] <> '@'
    THEN
      BEGIN
        GET_TEMP('B');
        EMIT('','M',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END;
  EMIT('','A',OPERAND1,OPERAND2);
  IF OPERAND2[1]='@'
    THEN
      FREE_TEMP
END;
PROCEDURE NEG_CODE_EMIT;
{* THIS SUBPROGRAM EMITS CODE FOR OPERAND1 NEGATION.*}
BEGIN
  FIND_TYPE(OPERAND1);
  TYPE1:=TYPES;
  IF NOT(TYPE1 IN ['B','('])
    THEN
      ERROR(31);
  IF OPERAND1[1] <> '@'
    THEN
      BEGIN
        GET_TEMP(TYPE1);
```

```
        EMIT('','M',TEMP,OPERAND1);
        OPERAND1:=TEMP
      END;
    EMIT('','D',OPERAND1,'');
    PUSH_OPR(OPERAND1)
  END;
  BEGIN (* MAIN PROGRAM OF CODE OR SEMANTIC ROUTINE *)
    IF OPERATOR='+' THEN ADD_CODE_EMIT
      ELSE IF OPERATOR='-' THEN SUB_CODE_EMIT
        ELSE IF OPERATOR='*' THEN MULT_CODE_EMIT
          ELSE IF OPERATOR='/' THEN DIV_CODE_EMIT
            ELSE IF OPERATOR='**' THEN EXP_CODE_EMIT
              ELSE IF OPERATOR=':=' THEN ASSIGN_CODE_EMIT
                ELSE IF OPERATOR='=' THEN EQU_CODE_EMIT
                  ELSE IF OPERATOR='<>' THEN NEQ_CODE_EMIT
                    ELSE IF OPERATOR='<=' THEN LEQ_CODE_EMIT
                      ELSE IF OPERATOR='>=' THEN GEQ_CODE_EMIT
                        ELSE IF OPERATOR='<' THEN LSS_CODE_EMIT
                          ELSE IF OPERATOR='>' THEN GTR_CODE_EMIT
                            ELSE IF OPERATOR='NOT' THEN NOT_CODE_EMIT
                              ELSE IF OPERATOR='OR' THEN OR_CODE_EMIT
                                ELSE IF OPERATOR='AND' THEN AND_CODE_EMIT;
    IF OPERATOR='PROG' THEN PROLOGUE_EMIT
      ELSE IF OPERATOR='END' THEN EPILOGUE_EMIT
        ELSE IF OPERATOR='NEG' THEN NEG_CODE_EMIT
          ELSE IF OPERATOR='THEN' THEN THEN_CODE_EMIT
            ELSE IF OPERATOR='ELSE' THEN ELSE_CODE_EMIT
              ELSE IF OPERATOR='FI' THEN FI_CODE_EMIT
                ELSE IF OPERATOR='EXIT' THEN EXIT_CODE_EMIT
                  ELSE IF OPERATOR='LOOP' THEN LOOP_CODE_EMIT
                    ELSE IF OPERATOR='ENDLOOP' THEN ENDLOOP_CODE_EMIT
                      ELSE IF (OPERATOR='READ')OR(OPERATOR='READLN') THEN READ_CODE_EMIT
                        ELSE IF (OPERATOR='WRITE')OR(OPERATOR='WRITELN') THEN WRITE_CODE_EMIT
  END;


  PROCEDURE PROG_HEAD;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 2.4 *)
  (* TOKEN SHOULD BE 'PROGRAM' . *)
  BEGIN
    IF NOT((TOKEN_TYPE='R')AND(TOKEN_NO=1))
      THEN
        ERROR(1);
    GET_TOKEN;
    IF TOKEN_TYPE<>'I'
      THEN
        ERROR(2);
    CODE('PROG',TOKEN,'');
    GET_TOKEN;
    IF TOKEN_TYPE<>';'
      THEN
        ERROR(3);
    GET_TOKEN
  END;
  PROCEDURE PROG_BODY;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 3.4 *)
  (* TOKEN SHOULD BE 'BEGIN' . *)
  BEGIN
    DCL_PART;
    BEG_END_STMT
  END;
  PROCEDURE DCL_PART;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 4.4 *)
  (* TOKEN SHOULD BE 'BEGIN' . *)
  BEGIN
    IF (TOKEN_TYPE='R')AND(TOKEN_NO=2)
```

```
    THEN
      BEGIN
        GET_TOKEN;
        VAR_LIST
      END
END;
PROCEDURE VAR_LIST;
{* THIS SUBPROGRAM PROCESSES PRODUCTION 5.4 *}
{* TOKEN SHOULD BE AN IDENTIFIER.           *}
BEGIN
  IF TOKEN_TYPE<>'I'
    THEN
      ERROR(A);
  VAR_STMT
END;
PROCEDURE VAR_STMT;
{* THIS SUBPROGRAM PROCESSES PRODUCTION 5.4 *}
{* TOKEN SHOULD BE AN IDENTIFIER.           *}
BEGIN
  IDENT:='';
  IF TOKEN_TYPE<>'I'
    THEN
      ERROR(B);
  IDENT:=TOKEN;
  GET_TOKEN;
  IF (TOKEN_TYPE=',')OR(TOKEN_TYPE=':')
    THEN
      BEGIN
        WHILE TOKEN_TYPE=',' DO
          BEGIN
            GET_TOKEN;
            IF TOKEN_TYPE<>'I'
              THEN
                ERROR(B)
              ELSE
                IDENT:=CONCAT(IDENT,CONCAT(',',TOKEN));
            GET_TOKEN
          END;
        IF TOKEN_TYPE<>':'
          THEN
            ERROR(5)
      END
    ELSE
      ERROR(C);
  GET_TOKEN;
  IF NOT((TOKEN_TYPE='R')AND((TOKEN_NO=3)OR(TOKEN_NO=4)OR(TOKEN_NO=5)OR(TOKEN_NO=22)))
    THEN
      ERROR(6);
  CASE TOKEN_NO OF
    3:TYPE_OF_ID:='R';
    4:TYPE_OF_ID:='I';
    5:TYPE_OF_ID:='C';
   22:TYPE_OF_ID:='B';
   ELSE
     TYPE_OF_ID:='';
  END;
  INSERTTAB(IDENT,TYPE_OF_ID);
  GET_TOKEN;
  IF TOKEN_TYPE<>';'
    THEN
      ERROR(3);
  GET_TOKEN;
  IF NOT((TOKEN_TYPE='R')AND(TOKEN_NO=6))
    THEN
      IF TOKEN_TYPE='('
        THEN
```

```
        VAR_STMT
      ELSE
        ERROR(7)
END;
PROCEDURE BEG_END_STMT;
{* THIS SUBPROGRAM PROCESSES PRODUCTION 9.*}
{* TOKEN SHOULD BE 'BEGIN'.                 *}
BEGIN
  IF (TOKEN_TYPE='R')AND(TOKEN_NO=6)
    THEN
      NO_OF_BEGIN:=NO_OF_BEGIN+1
    ELSE
      ERROR(10);
  GET_TOKEN;
  STMT_LIST;
  IF (TOKEN_TYPE='R')AND(TOKEN_NO=7)
    THEN
      NO_OF_BEGIN:=NO_OF_BEGIN-1
    ELSE
      ERROR(11);
  GET_TOKEN;
  IF NO_OF_BEGIN<>0
    THEN
      BEGIN
        IF TOKEN_TYPE<>';'
          THEN
            ERROR(3);
        GET_TOKEN
      END
    ELSE
      IF TOKEN_TYPE<>'.'
        THEN
          ERROR(12)
        ELSE
          CODE('END','','')
END;
PROCEDURE STMT_LIST;
{* THIS SUBPROGRAM PROCESSES PRODUCTION 10.*}
{* TOKEN SHOULD BE 'IDENT','R','I','WRITE'  *}
{* 'IDENT','R','R','READ','WRITE'.          *}
BEGIN
  IF ((TOKEN_TYPE='R')AND(TOKEN_NO IN [6,8,9,10,11,12,13,14]))OR(TOKEN_TYPE='I')OR(TOKEN_TYPE='R')
    THEN
      BEGIN
        STMT;
        STMT_LIST
      END
END;
PROCEDURE STMT;
{* THIS SUBPROGRAM PROCESSES PRODUCTION 11.  *}
{* TOKEN SHOULD BE 'IDENT','R','I',         *}
{* 'IDENT','R','R','READ','WRITE'.          *}
BEGIN
  IF TOKEN_TYPE<>';'
    THEN
      IF TOKEN_TYPE='R'
        THEN
          CASE TOKEN_NO OF
            6:BEG_END_STMT;
            8:IF_STMT;
            9:LOOP_STMT;
           10:WHEN_STMT;
        11,12:READ_STMT;
        13,14:WRITE_STMT;
          ELSE
            BEGIN
```

```
                GET_TOKEN;
                ERROR(13);
                STMT
              END
            END
          ELSE
            IF TOKEN_TYPE='('
              THEN
                ASSIGNMENT
              ELSE
                BEGIN
                  GET_TOKEN;
                  ERROR(14);
                  STMT
                END
      ELSE
        GET_TOKEN
  END;
  PROCEDURE PUSH_OPT(NAME:STRING);
  BEGIN
    IF STK_PT1<50
      THEN
        BEGIN
          STK_PT1:=STK_PT1+1;
          OPT_STACK[STK_PT1]:=NAME
        END
      ELSE
        ERROR(25)
  END;
  PROCEDURE PUSH_OPR(NAME:STRING);
  BEGIN
    IF STK_PT2<80
      THEN
        BEGIN
          STK_PT2:=STK_PT2+1;
          OPR_STACK[STK_PT2]:=NAME
        END
      ELSE
        ERROR(26)
  END;
  PROCEDURE POP_OPT;
  BEGIN
    IF STK_PT1>0
      THEN
        BEGIN
          OPT:=OPT_STACK[STK_PT1];
          STK_PT1:=STK_PT1-1
        END
      ELSE
        ERROR(27)
  END;
  PROCEDURE POP_OPR;
  BEGIN
    IF STK_PT2>0
      THEN
        BEGIN
          OPR:=OPR_STACK[STK_PT2];
          STK_PT2:=STK_PT2-1
        END
      ELSE
        ERROR(28)
  END;
  PROCEDURE ASSIGNMENT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 12.4)
  (* TOKEN SHOULD BE AN IDENTIFIER.          *)
  BEGIN
```

```
  PUSH_OPT('TOKEN');
  GET_TOKEN;
  IF TOKEN_TYPE<>'E'
    THEN
      ERROR(15);
  PUSH_OPT('**');
  GET_TOKEN;
  EXPRESS;
  POP_OPR;
  B:=OPR;
  POP_OPR;
  A:=OPR;
  POP_OPT;
  CODE(OPT,A,B)
END;
PROCEDURE EXPRESS;
{ THIS SUBPROGRAM PROCESSES PRODUCTION 13.}
{ TOKEN SHOULD BE '(',',ไม่มี',ไม่มี',' ',}
{ '+','-',AN INTEGER,A REAL,A CHARACTER,   }
{ AN IDENTIFIER.                           }
BEGIN
  TERM;
  EXPRESS1
END;
PROCEDURE EXPRESS1;
{ THIS SUBPROGRAM PROCESSES PRODUCTION 14.}
{ TOKEN SHOULD BE A RELATIONAL OPERATOR.  }
BEGIN
  IF TOKEN_TYPE='R'
    THEN
      BEGIN
        PUSH_OPT('TOKEN');
        GET_TOKEN;
        TERM;
        POP_OPR;
        B:=OPR;
        POP_OPR;
        A:=OPR;
        POP_OPT;
        CODE(OPT,A,B)
      END
END;
PROCEDURE TERM;
{ THIS SUBPROGRAM PROCESSES PRODUCTION 15.}
{ TOKEN SHOULD BE '(',',ไม่มี',ไม่มี',' ',}
{ '+','-',AN INTEGER,A REAL,A CHARACTER OR}
{ AN IDENTIFIER.                          }
BEGIN
  FACTOR;
  TERM1
END;
PROCEDURE TERM1;
{ THIS SUBPROGRAM PROCESSES PRODUCTION 16.}
{ TOKEN SHOULD BE '-',')' OR 'ไม่มี'.     }
BEGIN
  IF (TOKEN_TYPE IN ['+','-']) OR ((TOKEN_TYPE='E') AND (TOKEN_NO=1))
    THEN
      BEGIN
        PUSH_OPT('TOKEN');
        GET_TOKEN;
        FACTOR;
        POP_OPR;
        B:=OPR;
        POP_OPR;
        A:=OPR;
        POP_OPT;
```

```
        ...-...;
        CODE(OPT,A,B);
        TERMS
      END
END;
PROCEDURE FACTOR;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 17.*)
(* TOKEN SHOULD BE '(',',','[','-',AN INTEGER,A REAL,A CHARACTER OR*)
(* AN IDENTIFIER.                                                  *)
BEGIN
  SUBFACT;
  FACTORS
END;
PROCEDURE FACTORS;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 18.*)
(* TOKEN SHOULD BE '*','/' OR 'AND'.                               *)
BEGIN
  IF (TOKEN_TYPE IN ['*','/'])OR((TOKEN_TYPE='R')AND(TOKEN_NO=20))
    THEN
      BEGIN
        PUSH_OPT(TOKEN);
        GET_TOKEN;
        SUBFACT;
        POP_OPR;
        B:=OPR;
        POP_OPR;
        A:=OPR;
        POP_OPT;
        CODE(OPT,A,B);
        FACTORS
      END
END;
PROCEDURE SUBFACT;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 19.*)
(* TOKEN SHOULD BE '(',',','[','-',AN INTEGER,A REAL,A CHARACTER OR*)
(* AN IDENTIFIER.                                                  *)
BEGIN
  PART;
  SUBFACTS
END;
PROCEDURE SUBFACTS;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 20.*)
(* TOKEN SHOULD BE '**'.                                           *)
BEGIN
  IF TOKEN_TYPE='*'
    THEN
      BEGIN
        PUSH_OPT(TOKEN);
        GET_TOKEN;
        PART;
        SUBFACTS;
        POP_OPR;
        B:=OPR;
        POP_OPR;
        A:=OPR;
        POP_OPT;
        CODE(OPT,A,B)
      END
END;
PROCEDURE PART;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 26.*)
(* TOKEN SHOULD BE '(',',','[','-',AN INTEGER,A REAL,A CHARACTER OR*)
(* AN IDENTIFIER.                                                  *)
BEGIN
```

```
CASE TOKEN_TYPE OF
  'N':
    BEGIN
      IF TOKEN_NO=0
        THEN
          BEGIN
            PUSH_OPT(TOKEN);
            GET_TOKEN;
            PART;
            POP_OPR;
            POP_OPT;
            CODE(OPT,OPR,'')
          END
        ELSE
          BEGIN
            ERROR(15);
            GET_TOKEN
          END
    END;
  'I','N','C','B':
    BEGIN
      PUSH_OPR(TOKEN);
      GET_TOKEN
    END;
  '(':
    BEGIN
      GET_TOKEN;
      EXPRESS;
      IF TOKEN_TYPE<>')'
        THEN
          ERROR(1);
      GET_TOKEN
    END;
  '+':
    BEGIN
      GET_TOKEN;
      IF TOKEN_TYPE='+'
        THEN
          BEGIN
            GET_TOKEN;
            EXPRESS;
            IF TOKEN_TYPE<>')'
              THEN
                ERROR(1)
          END
        ELSE
          BEGIN
            IF NOT((TOKEN_TYPE='N')OR(TOKEN_TYPE='I'))
              THEN
                ERROR(18);
            PUSH_OPR(TOKEN)
          END;
      GET_TOKEN
    END;
  '-':
    BEGIN
      GET_TOKEN;
      IF TOKEN_TYPE='-'
        THEN
          BEGIN
            GET_TOKEN;
            EXPRESS;
            IF TOKEN_TYPE<>')'
              THEN
                ERROR(1);
            POP_OPR;
```

```
                    ...');
                  CODE('NEG',OPR,'');
                END
              ELSE
                IF TOKEN_TYPE='N'
                  THEN
                    PUSH_OPR(CONCAT('-',TOKEN))
                  ELSE
                    IF TOKEN_TYPE='I'
                      THEN
                        ERROR(18)
                      ELSE
                        CODE('NEG',OPR,'');
            GET_TOKEN
          END
      END
  END;
  PROCEDURE IF_STMT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 16.*)
  (* TOKEN SHOULD BE 'IF'.                  *)
  BEGIN
    GET_TOKEN;
    EXPRESS;
    IF NOT((TOKEN_TYPE='R')AND(TOKEN_NO=15))
      THEN
        ERROR(17);
    POP_OPR;
    CODE('THEN',OPR,'');
    GET_TOKEN;
    STMT;
    ELSE_PT;
  END;
  PROCEDURE ELSE_PT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 17.*)
  (* TOKEN SHOULD BE 'ELSE' OR 'END'.       *)
  BEGIN
    IF (TOKEN_TYPE='R')AND(TOKEN_NO=16)
      THEN
        BEGIN
          POP_OPR;
          CODE('ELSE',OPR,'');
          GET_TOKEN;
          STMT;
          POP_OPR;
          CODE('FIF',OPR,'')
        END
      ELSE
        BEGIN
          POP_OPR;
          CODE('FIF',OPR,'')
        END
  END;
  PROCEDURE LOOP_STMT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 18.*)
  (* TOKEN SHOULD BE 'WHILE'.               *)
  BEGIN
    CODE('LOOP','','');
    LOOP_COUNT:=LOOP_COUNT+1;
    GET_TOKEN;
    STMT;
    IF NOT((TOKEN_TYPE='R')AND(TOKEN_NO=14))
      THEN
        ERROR(20);
    POP_OPR;
    B:=OPR;
    POP_OPR;
    A:=OPR;
```

```
      CODE('ENDLOOP',A,B);
      END_COUNT:=END_COUNT+1;
      IF LOOP_COUNT=END_COUNT
        THEN
          IF EXIT_COUNT<>LOOP_COUNT
            THEN
              ERROR(21);
        GET_TOKEN
    END;
    PROCEDURE WHEN_STMT;
    (* THIS SUBPROGRAM PROCESSES PRODUCTION 29.*)
    (* TOKEN SHOULD BE 'WHEN'.                *)
    BEGIN
      EXIT_COUNT:=EXIT_COUNT+1;
      IF LOOP_COUNT<>EXIT_COUNT
        THEN
          BEGIN
            ERROR(22);
            GET_TOKEN;
            WHILE NOT((TOKEN_TYPE='E')AND(TOKEN_NO=17)) DO
              GET_TOKEN
          END
        ELSE
          BEGIN
            GET_TOKEN;
            EXPRESS;
            IF NOT((TOKEN_TYPE='E')AND(TOKEN_NO=17))
              THEN
                ERROR(23);
            POP_OPR;
            CODE('EXIT',OPR,'');
            GET_TOKEN;
          END
    END;
    PROCEDURE READ_STMT;
    (* THIS SUBPROGRAM PROCESSES PRODUCTION 30.*)
    (* TOKEN SHOULD BE 'READ'.                *)
    BEGIN
      IF TOKEN_NO=11
        THEN
          X:='READ'
        ELSE
          IF TOKEN_NO=12
            THEN
              X:='READLN';
      GET_TOKEN;
      IF TOKEN_TYPE='('
        THEN
          BEGIN
            GET_TOKEN;
            READ_LIST;
            IF TOKEN_TYPE<>')'
              THEN
                ERROR(17);
            GET_TOKEN
          END
        ELSE
          CODE('READ','','');
      IF X='READLN'
        THEN
          CODE('READLN','','')
    END;
    PROCEDURE READ_LIST;
    (* THIS SUBPROGRAM PROCESSES PRODUCTION 31.*)
    (* TOKEN SHOULD BE AN IDENTIFIER.          *)
    BEGIN
```

```
    IF TOKEN_TYPE='('
      THEN
        CODE('READ',TOKEN,'')
      ELSE
        ERROR(8);
    GET_TOKEN;
    IF TOKEN_TYPE=','
      THEN
        BEGIN
          GET_TOKEN;
          READ_LIST
        END
  END;
  PROCEDURE WRITE_STMT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 32.*)
  (* TOKEN SHOULD BE 'WRITE'.                *)
  BEGIN
    IF TOKEN_NO=13
      THEN
        X:='WRITE'
      ELSE
        IF TOKEN_NO=14
          THEN
            X:='WRITELN';
    GET_TOKEN;
    IF TOKEN_TYPE='('
      THEN
        BEGIN
          GET_TOKEN;
          IF TOKEN='#'
            THEN
              BEGIN
                PRTR_ON:=TRUE;
                GET_TOKEN
              END;
          WRITE_LIST;
          IF TOKEN_TYPE<>')'
            THEN
              ERROR(17);
          PRTR_ON:=FALSE;
          GET_TOKEN
        END
      ELSE
        CODE('WRITE','','');
    IF X='WRITELN'
      THEN
        CODE('WRITELN','','')
  END;
  PROCEDURE WRITE_LIST;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 33.*)
  (* TOKEN SHOULD BE A CHARACTER,LITERAL,AN  *)
  (* INTEGER,A REAL,AN IDENTIFIER.           *)
  BEGIN
    WRITE_OUT;
    WRITE_TAIL
  END;
  PROCEDURE WRITE_OUT;
  (* THIS SUBPROGRAM PROCESSES PRODUCTION 34.1)
  (* TOKEN SHOULD BE A CHARACTER,LITERAL,AN  *)
  (* INTEGER,A REAL, AN IDENTIFIER.          *)
  BEGIN
    CASE TOKEN_TYPE OF
      'L','C':
        BEGIN
          CODE('WRITE','',TOKEN);
          GET_TOKEN
```

```
      END;
    'I','N':
      BEGIN
        EXPRESS;
        POP_OPR;
        CODE('WRITE','',OPR)
      END
  END
END;
PROCEDURE WRITE_TAIL;
(* THIS SUBPROGRAM PROCESSES PRODUCTION 35.*)
(* TOKEN SHOULD BE ','.                    *)
BEGIN
  IF TOKEN_TYPE=','
    THEN
      BEGIN
        GET_TOKEN;
        WRITE_LIST
      END
END;
BEGIN (* MAIN PROGRAM OF PARSER PROCESSES PRODUCTION 1.*)
  GET_TOKEN;
  PROG_HEAD;
  PROG_BODY
END;

BEGIN
  INITIALIZE;
  NAME:=CONCAT(FILENAME,'.TMP');
  ASSIGN(TEMP_FILE,NAME);
  REWRITE(TEMP_FILE);
  IF NOT(EOF(SOURCE))
    THEN
      PARSER;
  WRITELN('END OF COMPILE')
END.
```

```
PROGRAM INTERPRETE;
(* THIS PROGRAM INTERPRETES THE INTERMEDIATE CODE INTO THE INSTRUCTION *)
(* THAT INSTRUCTS COMPUTER TO WORK.                                    *)
TYPE
  PT_STR=^STRING;
  DAT=RECORD                  (* FORMAT OF SYMBOL TABLE.*)
    CASE DATATYPE:CHAR OF
      'I':(ING:INTEGER);
      'R':(RE:REAL);
      'B':(BL:BOOLEAN);
      'C':(C:CHAR);
      'L':(LT:STRING[255]);
      'T':(CASE TEMP_TYPE:CHAR OF (*FORMAT OF TEMPORARY VARIABLE.*)
             'I':(TIN:INTEGER);
             'R':(TRE:REAL);
             'B':(TBL:BOOLEAN)
           )
    END;
  OBJ_PROG=RECORD             (*FORMAT OF INTERMEDIATE CODE INSTRUCTION.*)
    CASE INSTR_TYPE:CHAR OF
      'P':(OP:CHAR;OP_RAND1:INTEGER;OP_RAND2:INTEGER);
      'D':(CASE DATA:CHAR OF
             'T':();
             'I':(INT:INTEGER);
             'R':(REA:REAL);
             'B':(BOOL:BOOLEAN);
             'C':(CH:CHAR);
             'L':(LIT:STRING[255])
           )
    END;
  OBJ_FILE=FILE OF OBJ_PROG;(*FORMAT OF OBJECT PROGRAM.*)
  INSTRUCTION=RECORD
    OP_CODE:CHAR;             (*OPERATION CODE.*)
    OPR1:INTEGER;             (*OPERAND 1 *)
    OPR2:INTEGER              (*OPERAND 2 *)
  END;
VAR
  RESULT:INTEGER;             (* STORE ERROR CODE IN FILE CLOSE *)
  INDEX,SINDEX:INTEGER;       (* INDEX FOR ANY PURPOSE *)
  INT_TEMP:STRING[6];         (* VARIABLE FOR STORING INTEGER STRING *)
  REA_TEMP:STRING[22];        (* VARIABLE FOR STORING REAL STRING *)
  DMY_FILE:TEXT;              (* DUMMY FILE FOR NUMERIC & STRING CONVERSION*)
  NAME_BUFF:STRING[20];       (* FILENAME ACCEPT FROM USER *)
  FILE_PTR:PT_STR;            (* FOR USING IN @CMD FUNCTION *)
  A_KEY:CHAR;                 (* A CHARACTER CODE OF KEY THAT ACCEPTED FROM USER*)
  I,J:INTEGER;                (* INSTRUCTION AND STACK POINTER *)
  FILENAME:STRING;            (* FILENAME USED IN INTERPRETATION *)
  NAME:STRING;                (* FILENAME WITH FILE EXTENSION '.OBJ'*)
  ZERO_FLAG,CARRY_FLAG:BOOLEAN;(*ZERO FLAG AND CARRY FLAG THAT USED IN INTERPRETATION*)
  TEST_FLAG:BOOLEAN;          (* TEST FLAG THAT USED IN INTERPRETATION *)
  INTEG1,INTEG2:INTEGER;      (* INTEGER VARIABLE *)
  REAL1,REAL2:REAL;           (* REAL VARIABLE    *)
  BOOL1,BOOL2:BOOLEAN;        (* BOOLEAN VARIABLE *)
  DECK:OBJ_FILE;              (* OBJECT FILE      *)
  OBJ:OBJ_PROG;               (* TEMPORARY RECORD FOR ACCEPTING DATA FROM OBJECT FILE*)
  SYM:ARRAY[1..40961OF DAT;   (* TABLE STORE DATA TYPE INSTRUCTION *)
  INST:ARRAY[1..40961OF INSTRUCTION;(* TABLE STORE COMMAND TYPE INSTRUCTION*)
  PRINTFILE : TEXT;
EXTERNAL FUNCTION @CMD:PT_STR;
EXTERNAL PROCEDURE @HLT;
```

```
PROCEDURE READ_PROC;
(* THIS SUBPROGRAM READS RECORDS FROM OBJECT FILE AND FILLS THEM INTO THE TABLE*)
BEGIN
  I:=1;
  J:=1;
  WHILE NOT EOF(DECK) DO
    BEGIN
      READ(DECK,OBJ);
      CASE OBJ.INSTR_TYPE OF
          'P':
            BEGIN
             WITH INSTR[I] DO
               BEGIN
                 WRITELN(OBJ.OP,'  ',OBJ.OP_RAND1,'  ',OBJ.OP_RAND2);
                 OP_CODE:=OBJ.OP;
                 OPR1:=OBJ.OP_RAND1;
                 OPR2:=OBJ.OP_RAND2;
               END;
             I:=I+1
            END;
          'D':
            BEGIN
              SYM[J].DATATYPE:=OBJ.DATA;
              WITH SYM[J] DO
                BEGIN
                  CASE DATATYPE OF
                    'T':
                      BEGIN
                      END;
                    'I':
                      BEGIN
                  ING:=OBJ.INT;
                      END;
                    'R':
                      BEGIN
                        RE:=OBJ.REA
                      END;
                    'B':
                      BEGIN
                        BL:=OBJ.BOOL
                      END;
                    'C':
                      BEGIN
                        C:=OBJ.CH
                      END;
                    'L':
                      BEGIN
                        LT:=OBJ.LIT
                      END;
                  END
                END;
                J:=J+1
              END;
            END
          END
  END;
  PROCEDURE INITIAL;
  (* THIS SUBPROGRAM INITIALIZES THE INTERPRETER PROGRAM *)
  BEGIN
    WRITE(CHR(12));
    WRITELN('ม.จุฬาลงกรณ์มหาวิทยาลัย, ภาควิชาวิศวกรรมคอมพิวเตอร์');
    WRITELN('ปริญญานิพนธ์วิศวกรรมศาสตรมหาบัณฑิต');
    WRITELN('การสร้างตัวแปลภาษาสำหรับเครื่องคอมพิวเตอร์');
    WRITELN('พ.ศ. ๒๕๒๖ ลิขสิทธิ์ของจุฬาลงกรณ์มหาวิทยาลัย');
    FILE_PTR:=@CHD;
    NAME_BUFF:=FILE_PTR^;
```

```
IF NOT(LENGTH(NAME_BUFF)=0)
  THEN
    BEGIN
      INDEX:=1;
      WHILE NAME_BUFF[INDEX]=' ' DO
        INDEX:=INDEX+1;
      FILENAME:=COPY(NAME_BUFF,INDEX,LENGTH(NAME_BUFF)-INDEX+1);
    END
  ELSE
    BEGIN
      WRITE('ชื่อแฟ้มข้อมูล : ');
      READLN(FILENAME)
    END;
NAME:=CONCAT(FILENAME,'.OBJ');
ASSIGN(DECK,NAME);
RESET(DECK);
ASSIGN(PRINTFILE,'LST:');
REWRITE(PRINTFILE);
IF EOF(DECK)
 THEN
   BEGIN
     WRITELN('ชื่อแฟ้มข้อมูล ',FILENAME,' ไม่มี');
     @HLT
   END
 ASSIGN(DMY_FILE,'DMMY');
END;
PROCEDURE ADDETH;
{* THIS SUBPROGRAM INTERPRETES 'A' INSTRUCTION *}
BEGIN
  WITH INSTE[I] DO
    BEGIN
      WITH SYM[OPR1] DO
        BEGIN
          CASE TEMP_TYPE OF
            'I':
              CASE SYM[OPR2].DATATYPE OF
                'I':TIN:=TIN+SYM[OPR2].INS;
                'R':
                  BEGIN
                    TEMP_TYPE:='R';
                    TRE:=TIN+SYM[OPR2].RE
                  END;
                'T':
                    CASE SYM[OPR2].TEMP_TYPE OF
                      'I':TIN:=TIN+SYM[OPR2].TIN;
                      'R':
                        BEGIN
                          TEMP_TYPE:='R';
                          TRE:=TIN+SYM[OPR2].TRE
                        END
                    END
              END;
            'R':
              CASE SYM[OPR2].DATATYPE OF
                'I':TRE:=TRE+SYM[OPR2].INS;
                'R':TRE:=TRE+SYM[OPR2].RE;
                'T':
                    CASE SYM[OPR2].TEMP_TYPE OF
                      'I':TRE:=TRE+SYM[OPR2].TIN;
                      'R':TRE:=TRE+SYM[OPR2].TRE
                    END
              END
          END
        END
    END;
```

```
PROCEDURE SUBRTN;
{* THIS SUBPROGRAM INTERPRETES '-' INSTRUCTION *}
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYM[OPR1] DO
        BEGIN
          CASE TEMP_TYPE OF
            'I':
              CASE SYM[OPR2].DATATYPE OF
                'I':TIN:=TIN-SYM[OPR2].ING;
                'R':
                  BEGIN
                    TEMP_TYPE:='R';
                    TRE:=TIN-SYM[OPR2].RE
                  END;
                'T':
                    CASE SYM[OPR2].TEMP_TYPE OF
                      'I':TIN:=TIN-SYM[OPR2].TIN;
                      'R':
                        BEGIN
                          TEMP_TYPE:='R';
                          TRE:=TIN-SYM[OPR2].TRE
                        END
                    END
                END;
              'R':
                CASE SYM[OPR2].DATATYPE OF
                  'I':TRE:=TRE-SYM[OPR2].ING;
                  'R':TRE:=TRE-SYM[OPR2].RE;
                  'T':
                      CASE SYM[OPR2].TEMP_TYPE OF
                        'I':TRE:=TRE-SYM[OPR2].TIN;
                        'R':TRE:=TRE-SYM[OPR2].TRE
                      END
                END
              END
            END
          END
        END;
PROCEDURE MULRTN;
{* THIS SUBPROGRAM INTERPRETES '*' INSTRUCTION *}
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYM[OPR1] DO
        BEGIN
          CASE TEMP_TYPE OF
            'I':CASE SYM[OPR2].DATATYPE OF
                  'I':TIN:=TIN*SYM[OPR2].ING;
                  'R':
                    BEGIN
                      TEMP_TYPE:='R';
                      TRE:=TIN*SYM[OPR2].ING;
                    END;
                  'T':CASE SYM[OPR2].TEMP_TYPE OF
                        'I':TIN:=TIN*SYM[OPR2].TIN;
                        'R':
                          BEGIN
                            TEMP_TYPE:='R';
                            TRE:=TIN*SYM[OPR2].TRE
                          END
                        END
                END;
            'R':CASE SYM[OPR2].DATATYPE OF
                  'I':TRE:=TRE*SYM[OPR2].ING;
```

```
                    'R':TRE:=TRE*SYMIOPR2].RE;
                    'T':CASE SYMIOPR2].TEMP_TYPE OF
                        'I':TRE:=TRE*SYMIOPR2].TIN;
                        'R':TRE:=TRE*SYMIOPR2].TRE
                        END
                    END
                  END
                END
              END;
PROCEDURE DIVRTN;
(* THIS SUBPROGRAM INTERPRETES '/' INSTRUCTION *)
BEGIN
  WITH INSTEI] DO
    BEGIN
      WITH SYMIOPR1] DO
        BEGIN
          CASE TEMP_TYPE OF
            'I':CASE SYMIOPR2].DATATYPE OF
                'I':TIN:=TIN DIV SYMIOPR2].ING;
                'R':
                  BEGIN
                    TEMP_TYPE:='R';
                    TRE:=TIN/SYMIOPR2].RE
                  END;
                'T':CASE SYMIOPR2].TEMP_TYPE OF
                    'I':TIN:=TIN DIV SYMIOPR2].TIN;
                    'R':
                      BEGIN
                        TEMP_TYPE:='R';
                        TRE:=TIN)SYMIOPR2].TRE
                      END
                  END;
                END;
            'R':CASE SYMIOPR2].DATATYPE OF
                  'I':TRE:=TRE/SYMIOPR2].ING;
                  'R':TRE:=TRE/SYMIOPR2].RE;
                  'T':CASE SYMIOPR2].TEMP_TYPE OF
                      'I':TRE:=TRE/SYMIOPR2].TIN;
                      'R':TRE:=TRE/SYMIOPR2].TRE
                      END
                  END
              END
          END
        END
    END;
PROCEDURE EXPRTN;
(* THIS SUBPROGRAM INTERPRETES '**' INSTRUCTION *)
BEGIN
  WITH INSTEI] DO
    BEGIN
      WITH SYMIOPR1] DO
        BEGIN
          CASE TEMP_TYPE OF
            'I':
              BEGIN
                TEMP_TYPE:='R';
                CASE SYMIOPR2].DATATYPE OF
                  'I':TRE:=EXP(SYMIOPR2].ING*LN(TIN));
                  'R':TRE:=EXP(SYMIOPR2].RE*LN(TIN));
                  'T':CASE SYMIOPR2].TEMP_TYPE OF
                        'I':TRE:=EXP(SYMIOPR2].TIN*LN(TIN));
                        'R':TRE:=EXP(SYMIOPR2].TRE*LN(TIN))
                      END
                END
              END;
```

```
                'R':CASE SYMTOPR2.TEMP_TYPE OF
                      'I':TRE:=EXP(SYMTOPR2.ING*LN(TRE));
                      'R':TRE:=EXP(SYMTOPR2.RE*LN(TRE));
                      'T':CASE SYMTOPR2.TEMP_TYPE OF
                         'I':TRE:=EXP(SYMTOPR2.TIN*LN(TRE));
                         'R':TRE:=EXP(SYMTOPR2.TRE*LN(TRE))
                          END
                    END
                  END
                END
            END;
PROCEDURE ANDRTN;
(* THIS SUBPROGRAM INTERPRETES 'A' INSTRUCTION *)
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYMTOPR[1] DO
        BEGIN
          CASE SYMTOPR2.DATATYPE OF
            'B':TBL:=TBL AND SYMTOPR2.BL;
            'T':TBL:=TBL AND SYMTOPR2.TBL
          END
        END
    END
END;
PROCEDURE ORRTN;
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYMTOPR[1] DO
        BEGIN
          CASE SYMTOPR2.DATATYPE OF
            'B':TBL:=TBL OR SYMTOPR2.BL;
            'T':TBL:=TBL OR SYMTOPR2.TBL
          END
        END
    END
END;
PROCEDURE MOVRTN;
(* THIS SUBPROGRAM INTERPRETES 'M' INSTRUCTION *)
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYMTOPR[1] DO
        BEGIN
          CASE DATATYPE OF
            'I':CASE SYMTOPR2.DATATYPE OF
                  'I':ING:=SYMTOPR2.ING;
                  'T':ING:=SYMTOPR2.TIN
                END;
            'R':CASE SYMTOPR2.DATATYPE OF
                  'I':RE:=SYMTOPR2.ING;
                  'R':RE:=SYMTOPR2.RE;
                  'T':CASE SYMTOPR2.TEMP_TYPE OF
                        'I':RE:=SYMTOPR2.TIN;
                        'R':RE:=SYMTOPR2.TRE
                      END
                END;
            'B':CASE SYMTOPR2.DATATYPE OF
                  'B':BL:=SYMTOPR2.BL;
                  'T':BL:=SYMTOPR2.TBL
                END;
            'C':C:=SYMTOPR2.C;
            'T':CASE SYMTOPR2.DATATYPE OF
                  'I':
```

```
                    BEGIN
                      TEMP_TYPE:='I';
                      TIN:=SYM[OPR2].ING
                    END;
                  'R':
                    BEGIN
                      TEMP_TYPE:='R';
                      TRE:=SYM[OPR2].RE
                    END;
                  'B':
                    BEGIN
                      TEMP_TYPE:='B';
                      TBL:=SYM[OPR2].BL
                    END;
                  'T':
                    BEGIN
                      TEMP_TYPE:=SYM[OPR2].TEMP_TYPE;
                      CASE TEMP_TYPE OF
                        'I':TIN:=SYM[OPR2].TIN;
                        'R':TRE:=SYM[OPR2].TRE;
                        'B':TBL:=SYM[OPR2].TBL
                      END
                    END
                  END
              END
            END
      END
  END;
  PROCEDURE NOTETH;
  (* THIS SUBPROGRAM INTERPRETES 'N' INSTRUCTION *)
  BEGIN
    WITH INST[I] DO
      BEGIN
        WITH SYM[OPR1] DO
          BEGIN
            CASE DATATYPE OF
              'B':BL:=NOT(BL);
              'T':TBL:=NOT(TBL)
            END
          END
      END
  END;
  PROCEDURE NEGETH;
  BEGIN
    WITH INST[I] DO
      BEGIN
        WITH SYM[OPR1] DO
          BEGIN
            CASE DATATYPE OF
              'I':ING:=-ING;
              'R':RE:=-RE;
              'T':CASE TEMP_TYPE OF
                    'I':TIN:=-TIN;
                    'R':TRE:=-TRE
                  END
            END
          END
      END
  END;
  PROCEDURE JZETH;
  (* THIS SUBPROGRAM INTERPRETES 'Z' INSTRUCTION *)
  BEGIN
    IF ZERO_FLAG
      THEN
        I:=INST[I].OPR2-1
  END;
```

```
PROCEDURE JCRTN;
(* THIS SUBPROGRAM INTERPRETES 'C' INSTRUCTION *)
BEGIN
  IF CARRY_FLAG
    THEN
      I:=INSTR[I].OPR2 1
END;
PROCEDURE JMPRTN;
(*THIS SUBPROGRAM INTERPRETES 'J' INSTRUCTION *)
BEGIN
  I:=INSTR[I].OPR2-1
END;
PROCEDURE EQURTN;
(* THIS SUBPROGRAM INTERPRETES 'E' INSTRUCTION *)
BEGIN
END;
(* THIS SUBPROGRAM INTERPRETES 'P' INSTRUCTION *)
PROCEDURE CMPRTN;
BEGIN
  WITH INSTR[I] DO
    BEGIN
      WITH SYM[OPR1] DO
        BEGIN
          CASE DATA_TYPE OF
            'I':INTEG1:=ING;
            'R':REAL1:=RE;
            'B':BOOL1:=BL;
            'T':CASE TEMP_TYPE OF
                  'I':INTEG1:=TIN;
                  'R':REAL1:=TRE;
                  'B':BOOL1:=TBL
                END
          END;
          CASE SYM[OPR2].DATATYPE OF
            'I':INTEG2:=SYM[OPR2].ING;
            'R':REAL2:=SYM[OPR2].RE;
            'B':BOOL2:=SYM[OPR2].BL;
            'T':CASE SYM[OPR2].TEMP_TYPE OF
                  'I':INTEG2:=SYM[OPR2].TIN;
                  'R':REAL2:=SYM[OPR2].TRE;
                  'B':BOOL2:=SYM[OPR2].TBL
                END
          END;
          CASE DATA_TYPE OF
            'I':IF(INTEG1=INTEG2)
                  THEN
                    ZERO_FLAG:=TRUE
                  ELSE
                    BEGIN
                      ZERO_FLAG:=FALSE;
                      IF (INTEG1)INTEG2
                        THEN
                          CARRY_FLAG:=FALSE
                        ELSE
                          CARRY_FLAG:=TRUE
                    END;
            'R':IF REAL1=REAL2
                  THEN
                    ZERO_FLAG:=TRUE
                  ELSE
                    BEGIN
                      ZERO_FLAG:=FALSE;
                      IF REAL1>REAL2
                        THEN
                          CARRY_FLAG:=FALSE
                        ELSE
```

```
                    CARRY_FLAG:=TRUE
                END;
            'B':IF BOOL1=BOOL2
                THEN
                  ZERO_FLAG:=TRUE
                ELSE
                  BEGIN
                    ZERO_FLAG:=FALSE;
                    IF BOOL1:BOOL2
                      THEN
                        CARRY_FLAG:=FALSE
                      ELSE
                        CARRY_FLAG:=TRUE
                  END;
            'T':CASE TEMP_TYPE OF
                  'I':IF INTEG1=INTEG2
                        THEN
                          ZERO_FLAG:=TRUE
                        ELSE
                          BEGIN
                            ZERO_FLAG:=FALSE;
                            IF INTEG1:INTEG2
                              THEN
                                CARRY_FLAG:=FALSE
                          END;
                  'R':IF REAL1=REAL2
                        THEN
                          ZERO_FLAG:=TRUE
                        ELSE
                          BEGIN
                            ZERO_FLAG:=FALSE;
                            IF REAL1:REAL2
                              THEN
                                CARRY_FLAG:=FALSE
                              ELSE
                                CARRY_FLAG:=TRUE
                          END;
                  'B':IF BOOL1=BOOL2
                        THEN
                          ZERO_FLAG:=TRUE
                        ELSE
                          BEGIN
                            ZERO_FLAG:=FALSE;
                              THEN
                                CARRY_FLAG:=FALSE
                              ELSE
                                CARRY_FLAG:=TRUE
                          END
                END;
            'C':IF C=SYM[OPR2].C
                THEN
                  ZERO_FLAG:=TRUE
                ELSE
                  BEGIN
                    ZERO_FLAG:=FALSE;
                    IF C:SYM[OPR2].C
                      THEN
                        CARRY_FLAG:=FALSE
                      ELSE
                        CARRY_FLAG:=TRUE
                  END
              END
          END
      END
END;
PROCEDURE LENTH;
```

```
(* THIS SUBPROGRAM INTERPRETES 'L' INSTRUCTION *)
BEGIN
  WITH INST[I] DO
    BEGIN
      CASE SYM[OPR1].DATATYPE OF
        'B':IF SYM[OPR1].BL
              THEN
                TEST_FLAG:=TRUE
              ELSE
                TEST_FLAG:=FALSE;
        'T':IF SYM[OPR1].TBL
              THEN
                TEST_FLAG:=TRUE
              ELSE
                TEST_FLAG:=FALSE
      END
    END
END;
PROCEDURE CSPRTN;
(* THIS SUBPROGRAM INTERPRETES 'T' INSTRUCTION *)
VAR
  INT_VALUE:INTEGER;
  REA_VALUE:REAL;
  CHA_VALUE:CHAR;
BEGIN
  WITH INST[I] DO
    BEGIN
      CASE OPR1 OF
        0:CASE SYM[OPR2].DATATYPE OF
            'I':
              BEGIN
                READ(INT_VALUE);
                SYM[OPR2].ING:=INT_VALUE
              END;
            'R':
              BEGIN
                READ(REA_VALUE);
                SYM[OPR2].RE:=REA_VALUE
              END;
            'C':
              BEGIN
                READ(CHA_VALUE);
                SYM[OPR2].C:=CHA_VALUE
              END
          END;
        1:READLN;
        2:CASE SYM[OPR2].DATATYPE OF
            'I':WRITE(PRINTFILE,SYM[OPR2].ING);
            'R':WRITE(PRINTFILE,SYM[OPR2].RE);
            'C':WRITE(PRINTFILE,SYM[OPR2].C);
            'L':WRITE(PRINTFILE,SYM[OPR2].LT);
            'T':CASE SYM[OPR2].TEMP_TYPE OF
                  'I':WRITE(PRINTFILE,SYM[OPR2].TIN);
                  'R':WRITE(PRINTFILE,SYM[OPR2].TRE);
                END
          END;
        3:CASE SYM[OPR2].DATATYPE OF
            'I':
              BEGIN
                REWRITE(DMY_FILE);
                WRITELN(DMY_FILE,SYM[OPR2].ING);
                CLOSE(DMY_FILE,RESULT);
                RESET(DMY_FILE);
                READLN(DMY_FILE,INT_TEMP);
                CLOSE(DMY_FILE,RESULT);
                SINDEX:=2;
```

```
      WHILE SINDEX<=6 DO
        BEGIN
          IF (INT_TEMP[SINDEX]>='0')AND(INT_TEMP[SINDEX]<='9')
            THEN
              INT_TEMP[SINDEX]:=CHR(ORD(INT_TEMP[SINDEX])+96);
          D:=D+1
        END;
      WRITE(INT_TEMP)
    END;
  'R':
    BEGIN
      REWRITE(DMY_FILE);
      WRITELN(DMY_FILE,SYM[OPR2].RE);
      CLOSE(DMY_FILE,RESULT);
      RESET(DMY_FILE);
      READLN(DMY_FILE,REA_TEMP);
      CLOSE(DMY_FILE,RESULT);
      SINDEX:=2;
      WHILE SINDEX<=22 DO
        BEGIN
          IF (REA_TEMP[SINDEX]>='0')AND(REA_TEMP[SINDEX]<='9')
            THEN
              REA_TEMP[SINDEX]:=CHR(ORD(REA_TEMP[SINDEX])+96)
            ELSE
              IF REA_TEMP[SINDEX]='E'
                THEN
                  REA_TEMP[SINDEX]:='0';
          D:=D+1
        END;
      WRITE(REA_TEMP)
    END;
  'C':WRITE(SYM[OPR2].C);
  'L':WRITE(SYM[OPR2].LT);
  'T':CASE SYM[OPR2].TEMP_TYPE OF
        'I':
          BEGIN
            REWRITE(DMY_FILE);
            WRITELN(DMY_FILE,SYM[OPR2].TIN);
            CLOSE(DMY_FILE,RESULT);
            RESET(DMY_FILE);
            READLN(DMY_FILE,INT_TEMP);
            CLOSE(DMY_FILE,RESULT);
            SINDEX:=2;
            WHILE SINDEX<=6 DO
              BEGIN
                IF (INT_TEMP[SINDEX]>='0')AND(INT_TEMP[SINDEX]<='9')
                  THEN
                    INT_TEMP[SINDEX]:=CHR(ORD(INT_TEMP[SINDEX])+96);
                D:=D+1
              END;
            WRITE(INT_TEMP)
          END;
        'R':
          BEGIN
            REWRITE(DMY_FILE);
            WRITELN(DMY_FILE,SYM[OPR2].TRE);
            CLOSE(DMY_FILE,RESULT);
            RESET(DMY_FILE);
            READLN(DMY_FILE,REA_TEMP);
            CLOSE(DMY_FILE,RESULT);
            SINDEX:=2;
            WHILE SINDEX<=22 DO
              BEGIN
                IF (REA_TEMP[SINDEX]>='0')AND(REA_TEMP[SINDEX]<='9')
                  THEN
                    REA_TEMP[SINDEX]:=CHR(ORD(REA_TEMP[SINDEX])+96)
```

```
                                ELSE
                                  IF REA_TEMP[SINDEX]='C'
                                    THEN
                                      REA_TEMP[SINDEX]:='D';
                                D:=D+1
                              END;
                            WRITE(REA_TEMP)
                          END;
                        END
                    END;
                  WRITELN
                END
              END
END;
PROCEDURE TESTRTN;
{* THIS SUBPROGRAM INTERPRETES 'G' INSTRUCTION *}
BEGIN
  IF NOT TEST_FLAG
    THEN
      I:=INST[I].OPR2-1
END;
BEGIN {*MAIN PROGRAM OF INTERPRETER *}
  INITIAL;
  READ_PROC;
  I:=1;
  T:=1;
  WHILE INST[I].OP_CODE<>'T' DO
    BEGIN
      WITH INST[I] DO
        BEGIN

          WRITELN(OP_CODE,'    ',OPR1,'    ',OPR2);

          IF OP_CODE='+' THEN ADDRTN
            ELSE IF OP_CODE='-' THEN SUBRTN
              ELSE IF OP_CODE='*' THEN MULRTN
                ELSE IF OP_CODE='/' THEN DIVRTN
                  ELSE IF OP_CODE='M' THEN MOVRTN
                    ELSE IF OP_CODE='X' THEN EXPRTN
                      ELSE IF OP_CODE='U' THEN NEGRTN
                        ELSE IF OP_CODE='P' THEN CMPRTN
                          ELSE IF OP_CODE='J' THEN JUMPRTN
                            ELSE IF OP_CODE='C' THEN JCBTN;
        IF OP_CODE='J' THEN JMPRTN
          ELSE IF OP_CODE='N' THEN NOTRTN
            ELSE IF OP_CODE='A' THEN ANDRTN
              ELSE IF OP_CODE='R' THEN ORRTN
                ELSE IF OP_CODE='F' THEN CSPRTN
                  ELSE IF OP_CODE='L' THEN LTRTN
                    ELSE IF OP_CODE='E' THEN EQURTN
                      ELSE IF OP_CODE='G' THEN TESTRTN

        END;
      I:=I+1
    END;
  CLOSEDEL(DMY_FILE,RESULT);
END.

A>
```

ภาคผนวก จ

ตัวอย่างโปรแกรมที่เขียนในภาษาคอมพิวเตอร์ภาษาไทย

```
โปรแกรม แฟคตอเรียล
(*โปรแกรมนี้ใช้หาค่าแฟคตอเรียล โดยผู้ใช้กำหนดค่าให้*)
ตัวแปร
    รับ๑ : ตัวอักษร ;
    ค่า๑, ค่า๒ :เลขจำนวนเต็ม ;
เริ่มต้น
    พิมพ์บรรทัด ('โปรแกรมคำนวณหาค่าแฟคตอเรียล') ;
    พิมพ์ ('โปรดให้ค่าที่ต้องการคำนวณ') ;
    อ่านบรรทัด (รับ๑) ;
    ค่า๑ := ๑;
    ค่า๒ := ๑;
    วนทำ
      เริ่มต้น
        เมื่อ ค่า๑  > รับ๑  ออก ;
        ค่า๒  :=  ค่า๒  * ค่า๑ ;
        ค่า๑  :=  ค่า๑  + ๑;
      สิ้นสุด;
    สิ้นสุดวนทำ;
    พิมพ์บรรทัด ('คำตอบ คือ', ค่า๒);
สิ้นสุด.
```

ภาคผนวก ฉ

คำศัพท์

ภาษาไทย	ภาษาอังกฤษ
ก	
กฎที่ใช้เปลี่ยนแปลง	Production Rule/Rewriting Rule
กรณี	Case
การกำหนดค่า	Assignment
การกำหนดเงื่อนไข	Conditional
การกำหนดชื่อและชนิดของตัวแปร	Declaration
การคืนสภาพจากข้อผิดพลาด	Error Recovery
การดำเนินการ	Operation
การดำเนินการคำนวณ	Arithmetic Operation
การดำเนินการคำนวณแบบมีข้อมูล ชนิดจำนวนเลข 1 ตัว	Unary Operation
การดำเนินการคำนวณแบบมีข้อมูล ชนิดจำนวนเลข 2 ตัว	Binary Operation
การดำเนินการทางตรรก	Logical Operation
การดำเนินการเปรียบเทียบ	Relational Operation
การทำงานวนซ้ำ	Repetition
การทำเอกสารในตัวเอง	Self-documentation
การแปลโดยอาศัยวากยสัมพันธ์	Syntax-directed Translation
การย้อนกลับไปตรวจสอบไวยากรณ์ใหม่	Backtrack
การรับข้อมูลจากแป้นกด	Keyboard Input
การวิเคราะห์วากยสัมพันธ์	Syntax Analyzer
การสร้างรหัสระหว่างกลาง	Intermediate Code Generating
การสิ้นสุดของแฟ้มข้อมูล	End of File

ภาษาไทย	ภาษาอังกฤษ
การแสดงผลออกทางจอภาพและเครื่องพิมพ์	Display Output and Printer Output
เก็บข้อมูลแบบพลวัต	Dynamic Allocation
ข	
ข้อกำหนดทางวากยสัมพันธ์	Syntactic Specification
ข้อความแสดงความผิดพลาด/คำอธิบายข้อผิดพลาด	Error Message
ข้อมูล	Data
ข้อมูลรับเข้า	Input Data
ค	
ค่าทางตรรก	Logical Value
ค่านิเสธ	Negation
ค่าแบบบูล	Boolean Value
ค่ารหัสแทนตัวอักษร	Character Code
เครื่องจักร-พี	P-machine
เครื่องพิมพ์	Printer
เครื่องหมายคำนวณ	Arithmetic Operator
เครื่องหมายเปรียบเทียบ	Relational Operator
คำ	Word
คำสำคัญ	Reserved Word
โครงสร้างที่เป็นลำดับชั้น	Hirarchical Structure
โครงสร้างแบบทรี	Tree Structure

ภาษาไทย	ภาษาอังกฤษ
จ	
จอภาพ	CRT-display
จำนวน เลข	Number
จำลองการทำงาน	Emulate
ช	
ชื่อ	Identifier
ซ	
ซีแมนติกซ์	Semantic
ต	
ตัวชี้ตำแหน่ง	Pointers
ตัวชี้ลำดับที่แสตค	Stack Pointer
ตัวดำ เนินการ	Operator
ตัวถูกกระทำ	Operand
ตัวถูกกระทำที่ เป็น เลขจำนวนจริง	Real Operands
ตัวถูกกระทำที่ เป็น เลขจำนวน เต็ม	Integer Operands
ตัวถูกกระทำ 1	First Operand
ตัวถูกกระทำ 2	Second Operand
ตัวแปรชั่วคราว	Temporary Variable
ตัวแปลคำสั่ง	Interpreter
ตัวแปลคำสั่งรหัส เทียม	Pseudo Code Interpreter
ตัวแปลคำสั่งรหัสระหว่างกลาง	Intermediate Code Interpreter
ตัวแปลภาษา	Compiler/Translator
ตัวแปลภาษาชนิดจวบกัน	Concurrent Structured Compiler

ภาษาไทย	ภาษาอังกฤษ
ตัวแปลภาษารหัสระหว่างกลาง	Translator
ตัวอักขระ	Character
ตาราง	Table
ตารางกำเนิดตัวอักษร	Character Generator Table
ตารางชื่อ/ตารางเลเบล	Label Table
ถ	
แถวลำดับ	Array
ท	
เทอร์มินอล	Terminal
โทเกน	Token
น	
นอนเทอร์มินอล	Nonterminal
นิพจน์	Expression
นิพจน์ตรรก	Logical Expression
นิพจน์ที่เป็นค่าของบูล	Boolean Expression
นิพจน์ทปรียบเทียบ	Relational Expression
นิพจน์เลขคณิต	Arithmetic Expression
ในระหว่างการแปลภาษา	Compile Time
บ	
บล้อก	Block
บล้อกในลักษณะซ้อน ๆ กัน	Nested Block
บัฟเฟอร์	Buffer
แบบแผนวากยสัมพันธ์	Syntax Diagram

ภาษาไทย	ภาษาอังกฤษ
ป	
ประโยคคำสั่ง	Statement
ประโยคหมายเหตุ	Comment Statement
ปาสคาล	Pascal
แป้นกด	Keyboard
โปรแกรมโครงสร้าง	Structured Program
โปรแกรมซีแมนติกซ์	Semantic Routines/Semantic Program
โปรแกรมดิบ	Source Program
โปรแกรมบรรณาธิกรไทย	EDTHAI
โปรแกรมผล	Obiect Program
ผ	
ฝนทอง	Single Quotation Mark
พ	
พาสเซอร์	Parser
พาสทรี	Parse Tree
พูล	Pool
พิสัยย่อย	Subrange
ฟ	
ฟันหนู	Double Quotation Mark
แฟลกค่าตรรก	Test Flag
แฟลกตัวทด	Carry Flag
แฟลกศูนย์	Zero Flag

ภาษาไทย	ภาษาอังกฤษ
ภ	
ภาษาคอมพิวเตอร์ภาษาไทย	Thai Computer Language
ภาษาเครื่อง	Machine Language
ภาษาที่มีโครงสร้างแบบบล็อก	Block-structured Language
ภาษาระดับสูง	High-level Language
ร	
รหัส	Code
รหัสเทียม	Pseudo Code
รหัสเทียมยูซีเอสดี	UCSD P-code
รหัสพี	P-code
รหัสสองตำแหน่ง	Two-address Code
รหัสระหว่างกลาง	Intermediate Code
ระบบปฏิบัติการ ซีพีเอ็ม-86	CP/M-86 Operating System
ระบบปฏิบัติการภาษาไทย	Thai Operating System
รูปแบบแบคลัส-นอร์	Backus Naur Form
ระเบียนชนิดแปรเปลี่ยน	Variant Record
ล	
เลเบล	Label
เล็กซิเคิล แอนนาไลซเซอร์	Lexical Analyzer
ว	
วนทำไม่มีที่สิ้นสุด	Infinite Loop
วากยสัมพันธ์	Syntax
วิธีการตรวจสอบจากบนลงล่าง	Top-down Parsing

ภาษาไทย	ภาษาอังกฤษ
ไวยากรณ์	Grammar
ไวยากรณ์แบบคอนเทคฟรี	Context-frce Grammar
ไวยากรณ์แบบแอลแอลเค	LL(k) Grammar
ไวยากรแบบแอลแอลวัน	LL(1) Grammar

ส

ส่วนการนิยาม	Declaration Part
ส่วนการกำหนดชื่อและชนิดของตัวแปร	Variable Declaration Part
ส่วนเชื่อม	Link
ส่วนที่ช่วยในการแก้ไขปรับปรุงตัวโปรแกรม	Debugging Aid
ส่วนประมวลผลซีแมนติกซ์	Semantic Processor
ส่วนประโยคคำสั่ง	Statement Part
ส่วนยกกำลัง	Exponent
ส่วนหัวโปรแกรม	Program Header
สแกนเนอร์	Scanner
สแตค	Stack
สแตคตัวดำเนินการ	Operator Stack
สแตคตัวถูกกระทำ	Operand Stack
สัญญลักษณ์ขีดเส้นใต้	Underscore
สัญญลักษณ์เริ่มต้น	Start Symbol
สายตัวอักษร	Character String

ห

หน่วยความจำสำรอง	Secondary Storage
หน่วยเล็กซิเคิล	Lexical Unit

ภาษาไทย	ภาษาอังกฤษ
อ	
ออฟติไมซเซอร์	Optimizer
อัลกอล	Algol
อุปกรณ์รับข้อมูล	Input Devices
เอกลักษณ์	Identity
เอด้า	Ada

ภาคผนวก ซ

รายละเอียดเพิ่มเติมของ

ตัวแปลภาษาคอมพิวเตอร์ภาษาไทยที่ได้สร้างขึ้นในการวิจัยนี้

ในการวิจัยนี้ การสร้างตัวแปลภาษาคอมพิวเตอร์ภาษาไทย กระทำบนเครื่องไมโครคอมพิวเตอร์ Future ซึ่งเป็น

- ไมโครคอมพิวเตอร์ 16 บิต
- หน่วยความจำ 256 กิโลไบต์

ตัวแปลภาษาคอมพิวเตอร์ภาษาไทย และตัวแปลคำสั่งรหัสระหว่างกลาง เขียนด้วยภาษาปาสคาลและผ่านการแปลด้วยตัวแปลภาษา 'PASCAL/MT+' ภายใต้ระบบปฏิบัติการ 'CP/M-86'

ข้อจำกัดของตัวแปลภาษาคอมพิวเตอร์ภาษาไทย

ข้อจำกัดมีเนื่องจากการออกแบบ เครื่องที่ใช้เป็นตัวทดลอง และระบบปฏิบัติการที่ใช้

1. ข้อจำกัดเนื่องมาจากการออกแบบ
   - ชื่อตัวแปรมีนัยสำคัญเพียง 15 ตัวอักษรแรกเท่านั้น เนื่องจากในทางปฏิบัติหน่วยความจำมีจำกัด การเก็บชื่อตัวแปรที่มีความยาวมากทั้งหมดทำให้สิ้นเปลืองเนื้อที่หน่วยความจำ และในกรณีที่ชื่อยาวมาก ๆ โอกาสซ้ำกันจะมีน้อย แต่ถ้าชื่อมีนัยสำคัญน้อยเกินไปจะมีโอกาสซ้ำกันมาก แต่นัยสำคัญดังกล่าวสามารถขยายได้ขึ้นกับลักษณะความยาวของตัวแปรในทางปฏิบัติ
   - การผสมชนิดข้อมูล
     - ในการดำเนินการ

       การดำเนินการเลขคณิต สามารถทำการดำเนินการเลขคณิต ผสมกันระหว่างข้อมูลชนิดเลขจำนวนเต็มกับข้อมูลชนิดเลขจำนวนจริงได้ โดยผลลัพท์ภายในที่ได้จากการดำเนินการเลขคณิตจะเก็บเป็นข้อมูลชนิดเลขจำนวนจริง

       การดำเนินการเปรียบเทียบกับการดำเนินการตรรก ข้อมูลที่ใช้ในการดำเนินการต้องเป็นชนิดเดียวกันเท่านั้น

     - ในประโยคคำสั่งกำหนดค่า

       ชื่อตัวแปรที่อยู่ทางซ้ายของประโยคคำสั่งกำหนดค่า และเป็นชื่อตัวแปรที่ใช้เก็บผลลัพท์จากนิพจน์ทางขวา จะต้องมีชนิดตัวแปรเดียวกันกับชนิดของผลลัพท์ที่ได้จากนิพจน์ทางขวาหรือในกรณีที่นิพจน์ที่อยู่ทางขวาของประโยคคำสั่งกำหนดค่า เป็นชนิดเลขจำนวนเต็ม หรือชนิดเลขจำนวนจริงแล้ว ตัวแปรทางซ้ายยังเป็นชนิดเลขจำ-

นวนจริงหรือ เลขจำนวน เต็มได้ด้วย การ เอาค่า เลขจำนวนจริงมา เก็บค่าไว้ใน ตัวแปรที่ เป็นชนิด เลขจำนวน เต็มจะประมาณโดยทำจะทำการปัด เศษ เศษตั้งแต่ .5 ขึ้นไป จะปัด เป็น 1 ส่วน เศษที่น้อยกว่า จะปัดทิ้ง

2. ข้อจำกัด เนื่องมาจากโครงสร้างของ เครื่องคอมพิว เตอร์ที่ใช้ในการวิจัย

- เลขจำนวน เต็ม มีค่าได้ตั้งแต่ - ๓๒๗๖๘ ถึง ๓๒๗๖๗
- เลขจำนวนจริง มีค่าได้ในช่วง ๑.๐ ย -๓๐๗ ถึง ๑.๐ ย +๓๐๗ และมีนัย สำคัญของ เลขจำนวนจริง ไม่รวม เครื่องหมายและส่วนยกกำลังได้ 15 ตำแหน่ง
- สายตัวอักษร มีค่าได้ตั้งแต่ 0 - 255 ไบต์
- การแสดงผลบนจอภาพในกรณีที่ เป็นตัว เลขจะแสดงผล เป็นตัว เลขไทย ส่วนการ แสดงผลบน เครื่องพิมพ์ในกรณีที่ เป็นตัว เลขจะแสดงผลเป็นตัว เลขอาระบิค ทั้งนี้ เนื่องมาจาก เครื่องพิมพ์ที่ใช้ในการวิจัยมีหน่วยความจำที่ใช้ เก็บรูปแบบตัวอักษร ได้จำกัด

3. ข้อจำกัด เนื่องมาจากระบบปฏิบัติการ 'CP/M-86' ที่สามารถใช้ตัวอักษรภาษาไทย ได้ (ปรับปรุงโดยอาจารย์บุญชัย โสวรรณวณิชกุล)

- ลำดับการรับตัวอักษรไทย เข้า เครื่องคอมพิว เตอร์ ในกรณีที่มีตัวสระกับวรรณยุกต์ รวมกัน ไม่ว่าจะกดตัววรรณยุกต์ไทยหรือสระไทยก่อน ลำดับตัวอักษรที่ เครื่อง คอมพิว เตอร์รับ เข้าไปจะถือว่าสระไทยมาก่อนและตามด้วยวรรณยุกต์ไทย เสมอ

ประวัติผู้เขียน

นายอดิศร กรุงเกษม เกิดที่กรุงเทพมหานคร เมื่อวันที่ 1 กุมภาพันธ์ พ.ศ. 2505 ได้รับปริญญาวิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมไฟฟ้า จากคณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย เมื่อปีการศึกษา 2525 และได้เข้าศึกษาต่อในระดับปริญญามหาบัณฑิต ภาควิชาวิศวกรรมคอมพิวเตอร์ บัณฑิตวิทยาลัย จุฬาลงกรณ์มหาวิทยาลัย ปี พ.ศ. 2526