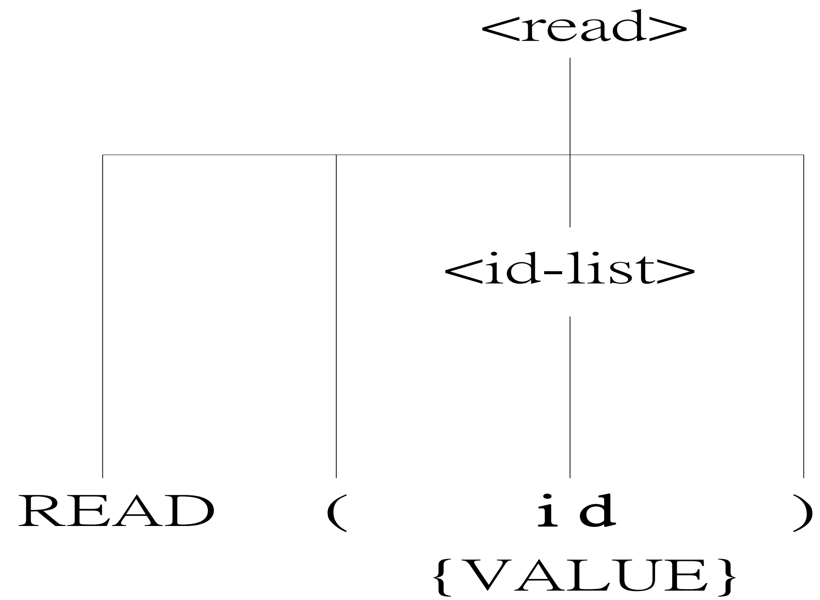


(a)

<id-list> ::=**id**

add S(**id**) to list

add 1 to LISTCOUNT

<id-list> ::= <id-list> , **id**

add S(**id**) to list

add 1 to LISTCOUNT

<<Code Generation.ptt>>

```
<read> ::= READ ( <id-list> )
            generate [ +JSUB   XREAD]
            record external reference to
XREAD
            generate [ WORD  LISTCOUT]
            for each item on list do
             begin
                remove S(ITEM) form list
                generate [ WORD S(ITEM)]
             end
            LISTCOUNT := 0
```

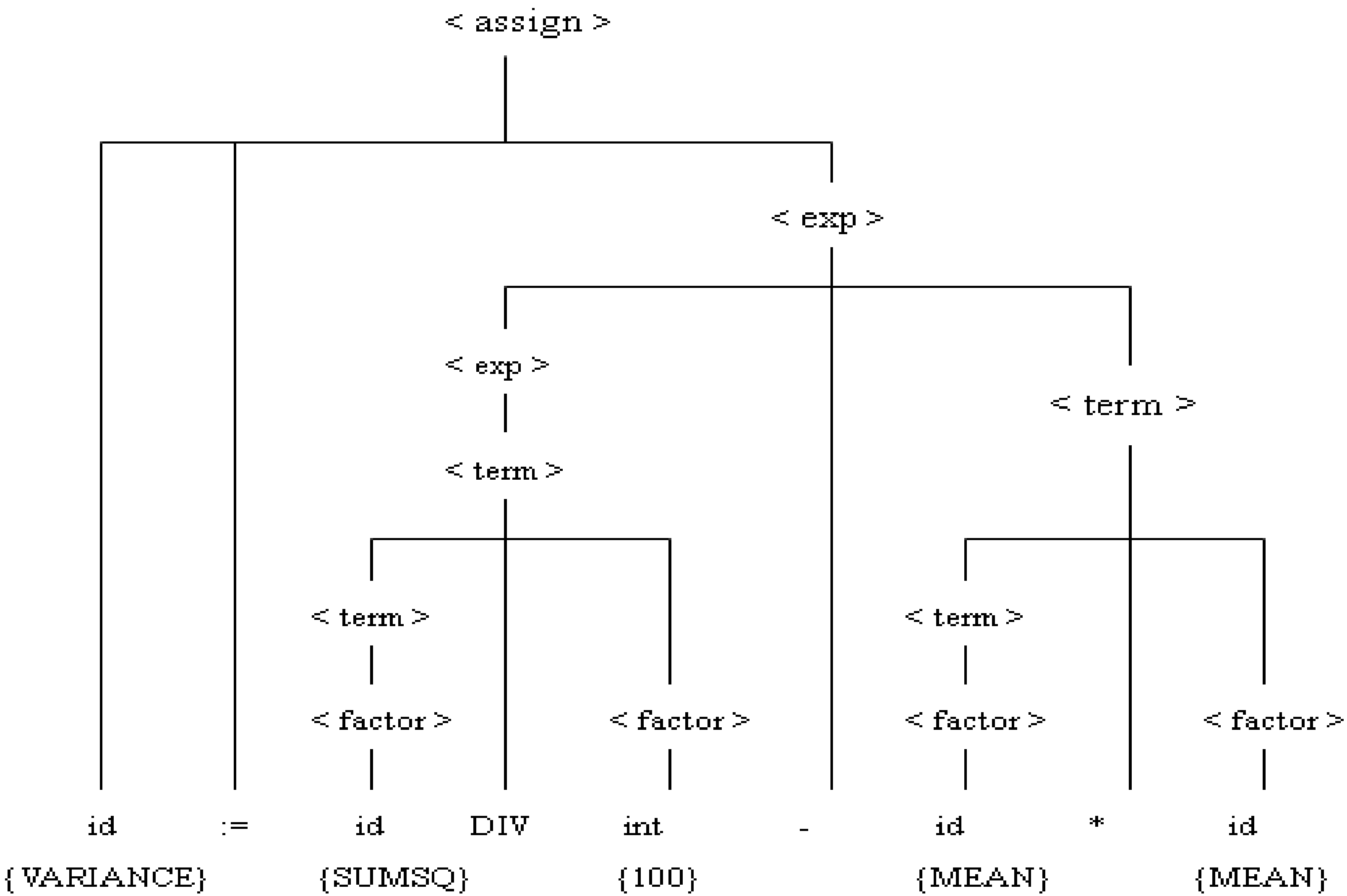
(b)

+JSB	XREAD
WORD	1
WORD	VALUE

(c)

FIGURE 5.12 Code generation for a READ statement.

FIGURE 5.13 Code generation for an assignment statement.



(a)

```
<assign> ::= id := <exp>
             GETA (<exp>)
             generate [STA  S(id)]
             REGA := null

<exp> ::=<term>
             S(<exp>) := S(<term>)
             if S(<exp>) = rA then
                  REGA := <exp>
```

```
<exp>1 ::=<exp>2 + <term>
        if S(<exp>2)= rA then
        generate [ADD  S(<term>)]
        else if S(<term>) =rA then
        generate [ADD  S(<exp>2)]
        else
            begin
               GETA (<exp>2)
               generate [ADDS(<term>)]
            end
        S(<exp>1) := rA
        REGA := <exp>1
```

```
<exp>_1 ::=<exp>_2 - <term>
        if S(<exp>_2)= rA then
                generate [SUB  S(<term>)]
        else
                begin
                        GETA (<exp>_2)
                        generate [SUB  S(<term>)]
                end
        S(<exp>_1)  := rA
        REGA := <exp>_1
<term> ::= <factor>
        S(<term>) := S(<factor>)
        if S(<term>)= rA then
                REGA := <term>
```

```
<term>1 ::=<term>2 * <factor>
      if S(<term>2)= rA then
            generate [MUL  S(<factor>)]
      else if S(<factor>) =rA then
            generate [MUL  S(<term>)2]
      else
            begin
                  GETA (<term>2)
                  generate [MUL  S(<factor>)]
            end
      S(<term>1)  := rA
      REGA := <term>1
```

<term>$_1$ ::=<term>$_2$ DIV <factor>

if S(<term>$_2$)= rA **then**

generate [DIV S(<factor>)]

else

begin

GETA (<term>$_2$)

generate [DIV S(<factor>)]

end

S(<term>$_1$) := rA

REGA := <term>$_1$

FIGURE 5.13(b) (Cont'd)

<factor> ::= **id**

S(<factor>) := S(<**id**>)

<factor> ::= **int**

S(<factor>) := S(<**int**>)

<factor> ::=(<exp>)

S(<factor>) := S(<exp>)

if S(<factor>)= rA **then**

REGA := <factor>

(b)

```
procedure GETA (NODE)
    begin
      if REGA = null then
      generate [LDA S(NODE)]
      else if S(NODE) ≠ rA  then
        begin
         create a new working variable Ti
         generate[STA Ti]
         record forward reference to Ti
         S(REGA) := Ti
         generate[LDA S(NODE)]
        end {if ≠ rA}
      S(NODE) := rA
      REGA := NODE
    end {GETA}
```

(c)

LDA	SUMSQ
DIV	#100
STA	T1
LDA	MEAN
MUL	MEAN
STA	T2
LDA	T1
SUB	T2
STA	VARIANCE

(d)

FIGURE 5.13 (Cont'd)

<prog> ::= PROGRAM <prog-name> VAR <dec-list> BEGIN <stmt-list> END

generate [LDL RETADR]

generate [RSUB]

for each Ti variable used **do**

generate [TiRESW   1]

insert [J EXADDR] {jump to first executable instruction} in bytes 3-5 of object program

fix up forward references to Ti variables

generate Modification records for external references

generate [END]

<prog-name> ::= **id**

generate [START 0]

generate [EXTREF XREAD,XWRITE]

generate [STL RETADR]

add 3 to LOCCTR {leave room for jump to first executable instruction}

generate [RETADR RESW 1]

<dec-list> ::= {either alternative}

save LOCCTR as EXADDR {tentative address of first executable instruction}

```
<dec > ::= {id-list} : <type>
      for each item on list do
        begin
           remove S(NAME) from list
           enter LOCCTR into symbol table
as address                            for NAME
           generate [S(NAME) RESW 1]
        end
      LISTCOUNT := 0
```

```
<type> ::= INTEGER
            {no code generation action}
<stmt-list> ::= {either alternative}
            {no code generation action}
```

FIGURE 5.14 Other code-generation routines for the grammar from Fig 5.2

```
<stmt> ::= {any alternative}
     {no code generation action}

<write> ::= WRITE (<id-list>)
     generate[+JUSB  XWRITE]
     record external reference to XWRITE
     generate[WORD  LISTCOUNT]
     for each item on list do
       begin
          remove S(ITEM) form list
          generate [WORD S(ITEM)]
       end
     LISTCOUNT := 0
```

```
<for> ::= FOR (<index-exp>) DO <body>
  pop JUMPADDR from stack {address of jump out
of loop}
  pop S(INDEX) from stack {index variable}
  pop LOOPADDR from stack {beginning address
of loop}
  generate[    LDA  S(INDXE)]
  generate[    ADD #1]
  generate[    J  LOOPADDR]
  insert[ JGT  LOCCTR] at location JUMPADDR
```

<index-exp> ::= **id** :=<exp>$_1$ TO <exp>$_2$

GETA (<exp>$_1$)

push LOCCTR onto stack {beginning address of loop}

push S(**id**) onto stack {index variable}

generate[STA S(**id**)]

```
generate[COMP  S(<exp>₂)]
   push LOCCTR onto stack {address of jump
out of loop}
   add 3 to LOCCTR {leave room for jump
instruction}
   REGA := null
<body> ::= {either alternative}
   {no code generation action}
```

FIGURE 5.14 (cont'd)

Symbolic Representation of Generated Code

STATS	START	0	{program header}
	EXTREF	XREAD,XWRITE	
	STL	RETADR	{save return address}
	J	{EXADDR}	
RETADR	RESW	1	
SUM	RESW	1	{variable declarations}
SUMSQ	RESW	1	
I	RESW	1	
VALUE	RESW	1	
MEAN	RESW	1	
VARIANCE	RESW	1	
{EXADDR}	LDA	#0	{SUM ：= 0}

Symbolic Representation of Generated Code

	STA	SUM	
	LDA	#0	{SUMSQ ：= 0}
	STA	SUMSQ	
	LDA	#1	{FOR I：= 1 TO 100 }
{L1}	STA	I	
	COMP	#100	
	JGT	{L2}	
	+JSUB	XREAD	{READ(VALUE)}
	WORD	1	
	WORD	VALUE	
	LDA	SUM	{SUM ：= SUM +VALUE}
	ADD	VALUE	
	STA	SUM	

Symbolic Representation of Generated Code

	LDA	VALUE	{SUMSQ ：= SUMSQ +VALUE *VALUE}
	MUL	VALUE	
	ADD	SUMSQ	
	STA	SUMSQ	
	LDA	I	{end of FOR loop}
	ADD	#1	
	J	{L1}	
{L2}	LDA	SUM	{MEAN ：= SUM DIV 100}
	DIV	#100	
	STA	MEAN	
	LDA	SUMSQ	{VARIANCE：= SUMSQ DIV 100-MEAN * MEAN}
	DIV	#100	

Symbolic Representation of Generated Code

	STA	T1	
	LDA	MEAN	
	MUL	MEAN	
	STA	T2	
	LDA	T1	
	SUB	T2	
	STA	VARIANCE	
	+JSUB	XWRITE	{WRITE(MEAN, VARIANCE)}
	WORD	2	
	WORD	MEAN	

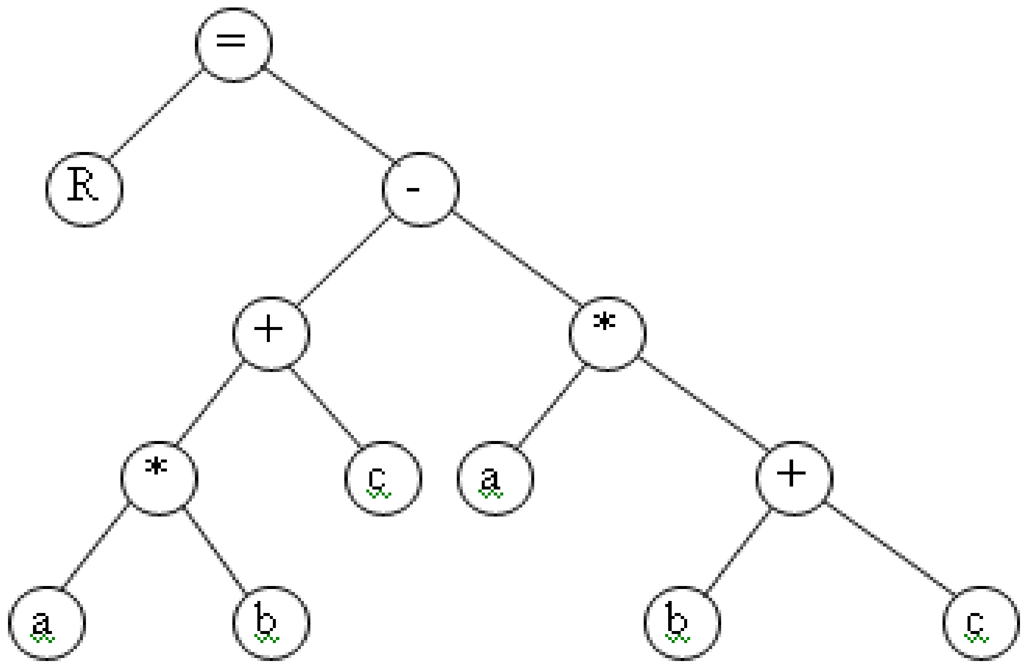
Symbolic Representation of Generated Code

	WORD	VARIANCE	
	LDL	RETADR	{return}
	RSUB		
T1	RESW	1	{working variables used}
T2	RESW	1	
	END		

Figure 5.15 Symbolic representation of object code generated for the program from Fig. 5.1.

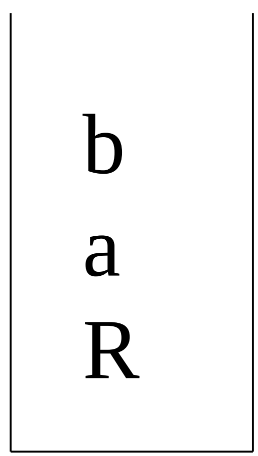
(referring Pascal3.doc and PaseTree.doc➔ Top-down approach)

以下舉例說明中間碼產生程式如何將剖析樹轉換成中間碼。假設有一運算式子R = (a * b + c) – (a * (b + c))被語法分析程式建成如下的剖析樹 (Bottom-up approach)：



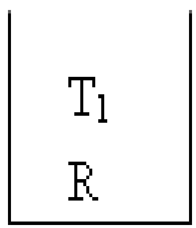
中間碼產生程式首先利用後序追蹤法找出此剖析樹的後置表示法（Postfix）： Rab * c + abc + * - = 然後準備一個堆疊。

掃瞄此後置表示式，如遇到變數符號則將其按入（push）堆疊裡。所以前三次掃瞄R，a，b分別被按入堆疊裏：



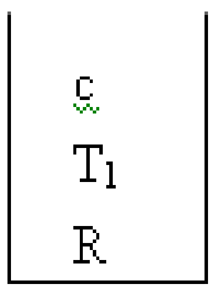
堆疊

如果掃瞄到運算符號則取出（pop）堆疊頂端兩個元素配合所掃瞄到的運算符號。組成一個中間碼，然後再將此中間碼的暫存位置符號按入堆疊裏。第四次掃瞄遇到*，於是取出b與a組成中間碼（*，a，b，T_1），T_1被按入堆疊：



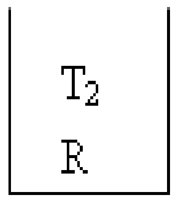
堆疊

第五次掃瞄，將c按入堆疊裏：



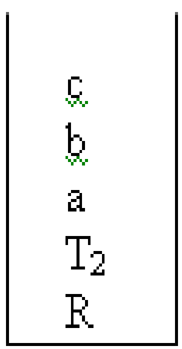
堆疊

第六次掃瞄遇到+，於是取出c與T_1組成中間碼（+，T_1，c，T_2），T_2被按入堆疊：



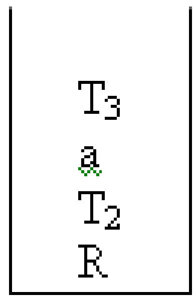
堆疊

第七、八、九掃瞄，將a，b，c分別按入堆疊：



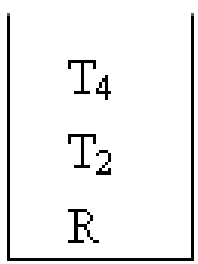
堆疊

第十次掃瞄遇到+，於是取出c與b組合中間碼（+，b，c，T_3），T_3被按入堆疊：



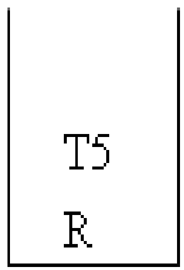
堆疊

第十一次掃瞄遇到*，於是取出T_3，a組成中間碼（*，a，T_3，T_4），T_4被按入堆疊：



堆疊

第十二次掃瞄遇到-，於是取出T_4，T_2組成中間碼（-，T_2，T_4，T_5），T_5被按入堆疊：



堆疊

第十三次掃瞄遇到＝，於是取出T_5與R組成中間碼（=，T_5，，R）該算術運算式子所得結果即存放於R內。