الفصل السابع

تعاريف وترجمة النحو الموجه الأعلى

SYNTAX-DIRECTED DEFINITIONS AND TRANSLATIONS

- المقدمة
- الترجمة القواعدية الموجهة
- تحسين الشفرة
- حذف متغيرات الحث
- توليد الشفرة الوسطية
- تمارين

الفصل السابع: تعاريف وترجمة النحو الموجه الأعلى

7.1 المقدمة

إن التحديد على ترجمة تركيبة معينة (Construct) في لغة برمجة ما يتضمن ماهية تلك التركيبة وتحديد قوانين الترجمة لها وان الترجمة هنا لا تعني بالضرورة توليد الشفرة الوسطية (Intermediate Code Generation) ولا شفرة الهدف (Object Code) ، كما تتضمن الترجمة إضافة معلومات إلى جدول الرموز (Symbol Table) وتنفيذ عمليات حسابية خاصة بالتركيبة. فإذا كانت التركيبة عبارة عن جملة وصفية (declarative) فان ترجمتها تضيف معلومات حول نوع التركيبة (Construct's Type) في جدول الرموز بينما لو كانت التركيبة تعبير (Expression) فان ترجمته تولّد شفرة لحساب ذلك التعبير.

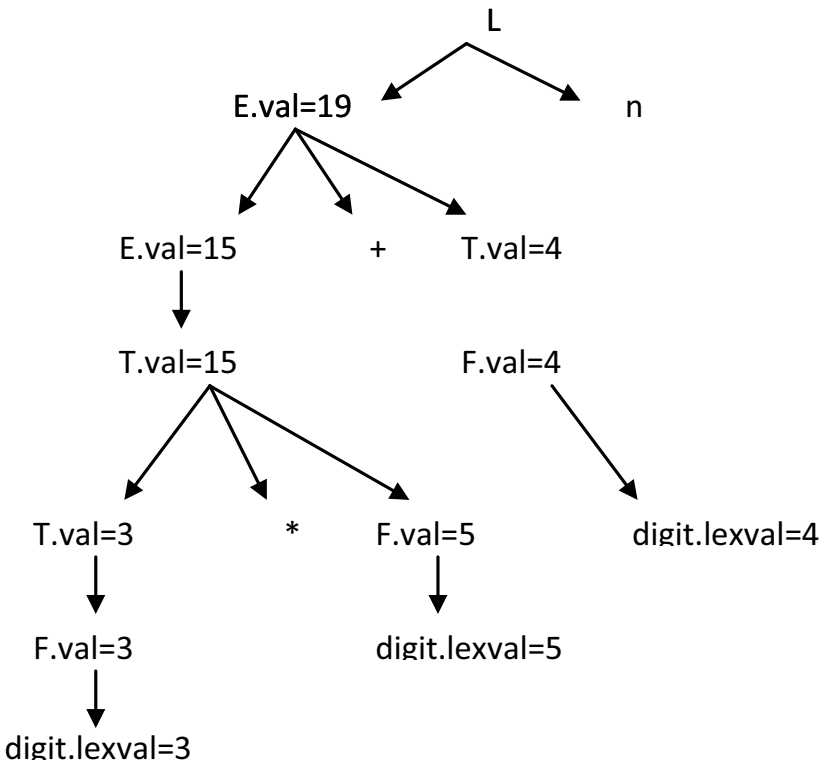
7.2 الترجمة القواعدية الموجهة (Syntax directed translation)

هي إحدى مراحل المترجم الضمنية وفيها يتم حساب التعابير (expressions)، وان لكل قاعدة إنتاج في القواعد هناك معنى :

قواعد المعنى	قاعدة الإنتاج
أطبع (E.val)	L→En
E.val := E_1.val+T.val	E→E_1+T
E.val := T.val	E→T
T.val := T_1.val*F.val	T→T_1*F
T.val := F.val	T→F
F.val := E.val	F→(E)
F.val := digit.lexval	F→digit

في قاعدة الإنتاج الأولى فان الـ n تعني علامة نهاية الجملة (end marker) والرمز (L) لا يؤثر على شيء (dummy) عدا من خلاله يتم طباعة ناتج التعبير، أما الأقواس في قاعدة الإنتاج F→(E) فلا تعني شيء في هذا المثال .

مثال7.1: لو كان لدينا التعبير 3*5+4n فان عملية حسابه تكون كما في الشكل (7.1).



شكل (7.1) حساب التعبير 3*5+4=19

بعد بناء الشجرة أعلاه نعوض عن قيم التعابير من الأسفل إلى الأعلى .

مثال7.2: لو كانت لدينا القواعد الآتية :

S→E$

E→E+E

E→E*E

$E \rightarrow (E)$

$E \rightarrow I$

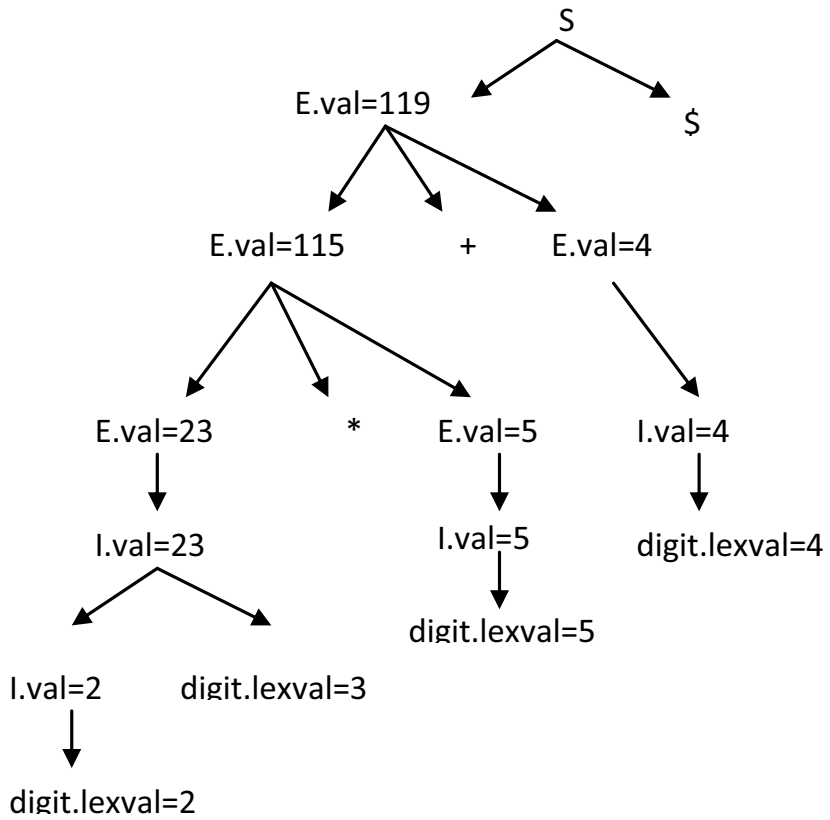
$I \rightarrow I\ digit$

$I \rightarrow digit$

وتم تفسير معنى تلك القواعد كما يلي :

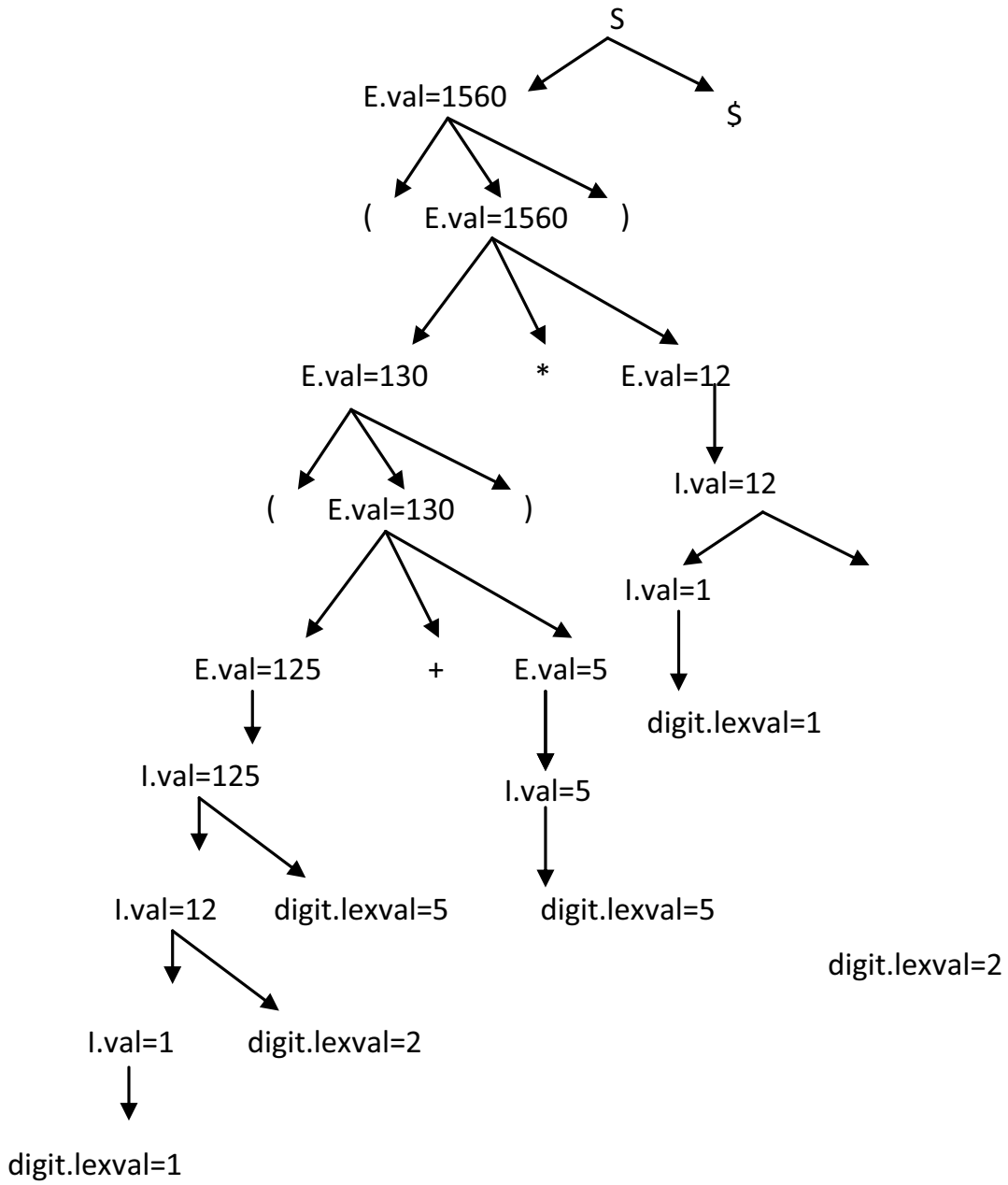
قواعد الانتاج	قواعد المعنى
$S \rightarrow E\$$	Print (E.val)
$E \rightarrow E_1+E_2$	$E.val := E_1.val+E_2.val$
$E \rightarrow E_1*E_2$	$E.val := E_1.val *E_2.val$
$E \rightarrow (E_1)$	$E.val := E_1.val$
$E \rightarrow I$	E.val := I.val
$I \rightarrow I_1\ digit$	$I.val := 10*I_1.val+digit.lexval$
$I \rightarrow digit$	I.val := digit.lexval

وكان لدينا التعبير التالي $4+5*23$ فان عملية حسابه تكون كما في الشكل (7.2).



شكل (7.2) حساب التعبير 23*5+4=119

أما إذا كان لدينا التعبير التالي $ ((125+5)*12) فان عملية حسابه تكون كما في الشكل (7.3).



شكل (7.3) حساب التعبير 1560= ((125+5)*12)

7.3 تحسين الشفرة (Code Optimization)

إن عملية ترجمة البرنامج المصدر (Source program) إلى الهدف تعتبر تطبيق من نوع (one to many) ، أي أن هناك أكثر من برنامج هدف لنفس البرنامج المصدر وان قسم من هذه البرامج الهدف أفضل من غيرها من البرامج الهدف من ناحية متطلبات الخزن (storage) وسرعة التنفيذ (execution speed) . تحسين الشفرة (code optimization) يشير إلى التقنيات التي يستخدمها المترجم (compiler) لإنتاج وتحسين برنامج الهدف وتحسين كفاءة التنفيذ للبرنامج الأصل المعطى .

تتضمن عملية تحسين الشفرة تحليل معقد للشفرة الوسطية وأداء مختلف التحويلات التي تقوم بها عملية تحسين الشفرة على أن يحافظ في كل عملية تحويل على معنى البرنامج (Semantics) ، أي أن المترجم لا يقوم بأي تحسينات تؤدي إلى تغيير في معنى البرنامج.

إن عملية تحسين الشفرة قد لا تعتمد على الماكنة (Machine independent) مثل حذف متغير الحث (induction variable elimination) كما سيتبين ذلك لاحقا ، وقد تعتمد على الماكنة (Machine dependent) وبالتالي يجب أن تكون عملية تحسين الشفرة على بينة ولديها معرفة عن الماكنة التي ستستخدم المترجم ، فمحاولة توليد شفرة الهدف التي تستفيد من مسجلات (registers) الماكنة بكفاءة أعلى هي مثال على عملية تحسين الشفرة التي تعتمد على الماكنة.

7.4 حذف متغيرات الحث (Eliminating Induction variables)

تعرف متغيرات الحث للدوارة (loop) بأنها تلك الأسماء التي يشار إليها داخل الدوارة وتأخذ الشكل $I = I \pm C$ حيث أن الـ C عبارة عن ثابت أو إسم قيمته لا تتغير داخل الدوارة. إن هناك خوارزمية لتحديد وحذف متغير الحث وهي :

خوارزمية تحديد وحذف متغيرات الحث

1. Find all of the basic variables by scanning the statements of loop L.

2. Find any additional induction variables, and for each additional induction variable A, find the family of some basic induction B to which A belongs. (If the value of A at the point of assignment is expressed as C1B+C2, then A is said to belong to the family of basic induction variable B). Specifically, we search for names A with single assignments to A within loop L, and which have one of the following forms:

$$A = B * C$$

$$A = C * B$$

$$A = B / C$$

$$A = B \pm C$$

$$A = C \pm B$$

Where C is a loop constant, and B is an induction variable, basic or otherwise. If B is basic, then A is in family of B, If B is not basic, let B be in the family of D, then the additional requirements to be satisfied are :

   a. There must be no assignment to D between the lone point of assignment to B in L and the assignment to A.

   b. There must be no definition of B outside of L reaches A.

3. Consider each basic induction variable B in turn. For every induction variable A in the family of B:

   a. Create a new name, temp.

b. Replace the assignment to A in the loop with A = temp.

c. Set temp to C1B+C2 at the end of the preheader by adding the statements:

```
temp = C1 * B

temp = temp + C2            /* omit if C2 = 0  */
```

d. Immediately after assignment B=B+D, where D is a loop invariant, append:

```
temp = temp + C1 * D
```

If D is a loop invariant name, and if C1≠1, create a new loop invariant name for C1* D, and add the statements:

```
temp1 = C1 * D

temp = temp + temp1
```

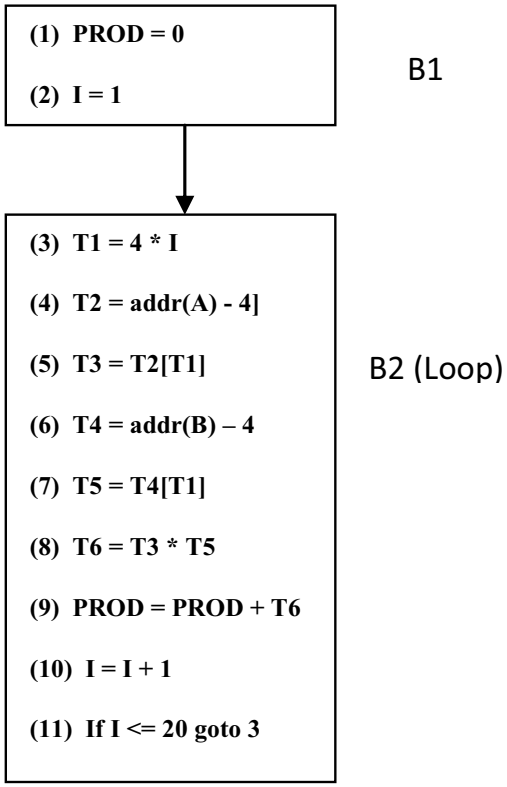
e. For each basic induction variable B whose only uses are to compute other induction variables in its family and in conditional branches, take some A in B's family, preferably one whose function expresses its value simply, and replace each test of the form B reloop X goto Y by:

```
temp2 = C1 * X

temp2 = temp2 + C2        /*  omit if C2 = 0 */

if temp reloop temp2 goto Y
```

Delete all assignments to B from the loop, as they will now be useless.

f. If there is no assignment to temp between the introduced statement A = temp (step 1) and the only use of A, then replace all uses of A by temp and delete the statement A = temp.

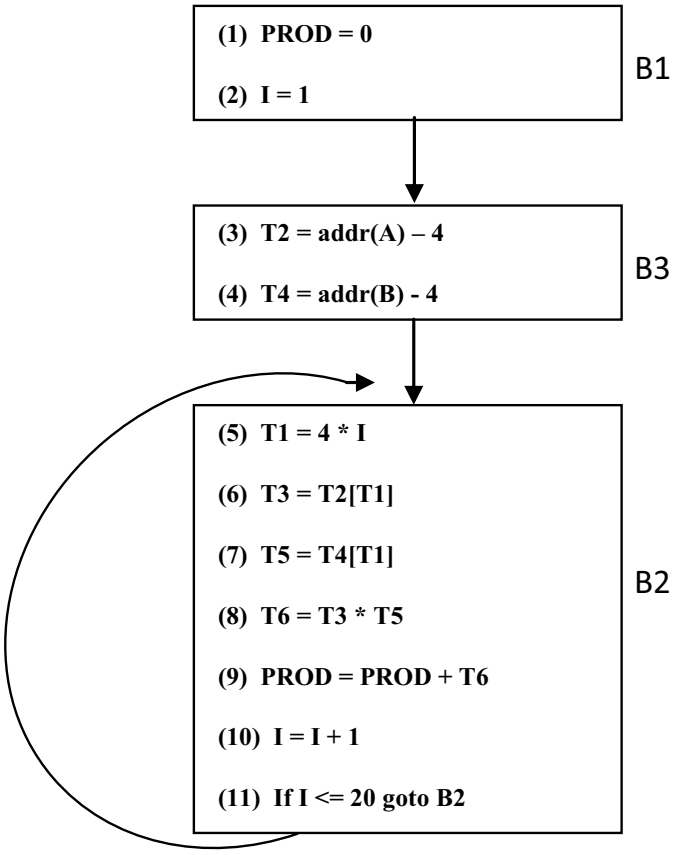
على سبيل المثال لو كان لدينا رسم التدفقِ البياني (flow graph) المبين في الشكل (7.4) . إن المتغير I يعتبر متغير حث في الدوارة المشكلة في البلوك B2.



شكل (7.4) رسم التدفقِ البياني للبلوكات الأساسية

الشكل (7.4) يمثل رسم التدفقِ البياني للبلوكات الأساسية لمقطع من برنامج فيه دوارة متمثلة في البلوك B2 وفيه الـI الذي يعتبر متغير حث كونه يمثل العنصر الأساسي في الدوارة والمتغير T1 يعتبر أيضا متغير حث في عائلة المتغير I (انظر خوارزمية تحديد وحذف متغير الحث) ،أما الشكل (7.5) فيمثل رسم التدفقِ البياني للبلوكات بعد تحديد متغيرات الحث (I) والجمل التي لا تتأثر بالدوارة مثل (PROD=0) وما يتعلق بتعريف

المصفوفات (T2 = addr(A) – 4) و (T4 = addr(B) – 4) وأعطيت أسماء جديدة للبلوكات بعد إعادة تقسيمها، ولأن المتغير T1 في جملة الإحلال (T1=4*I) يتأثرا بالمتغير I وطبقا للخطوة 3b في الخوارزمية أعلاه سيتم إحلال T1=4*I بـ T1=temp. وطبقا للخطوة 3c سيتم إضافة temp = 4 *I إلى ما يسمى بقبل العنوان الرئيسي (preheader) ثم يتم إضافة الجملة temp = temp + 4 بعد الجملة (10) كما في الشكل(7.6) .



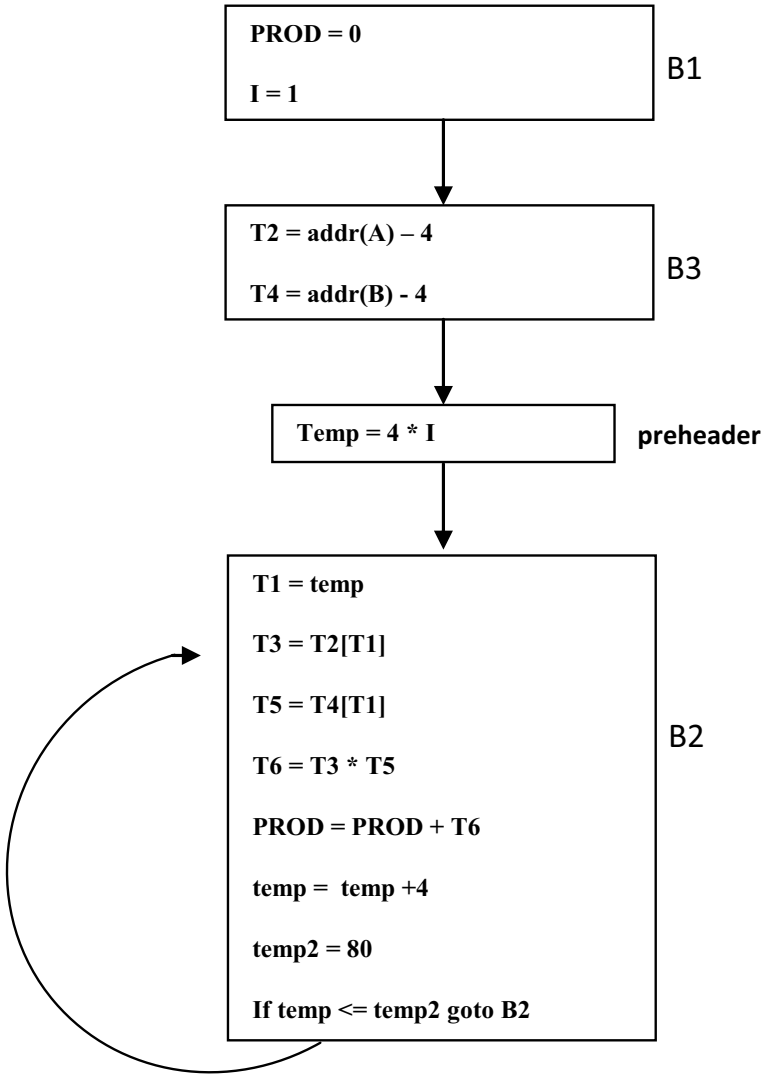
شكل (7.5) رسم التدفقِ البياني للبلوكات بعد تحديد متغيرات الحث

وطبقا للخطوة 3e في الخوارزمية سيتم إحلال الجملة if I<= 20 goto B2 بما يلي:

temp1 = 80

if (temp <= temp1) goto B2

ثم يتم حذف الجملة I = I+ 1 فتكون النتيجة كما في الشكل (7.6) ، وبهذا نكون قد تخلصنا من متغير الحث ومن معه في عائلته من داخل الدوارة.



شكل (7.6) رسم التدفقِ البياني للبلوكات بعد تحويرها

7.5 توليد الشفرة الوسطية (Intermediate Code Generation)

إن عملية توليد الشفرة الوسطية (Intermediate Code Generation) هي المرحلة التي تسبق مرحلة توليد الشفرة حيث أن هناك ثلاث طرق لتوليد الشفرة الوسطية هي:

- Postfix Notation
- Syntax Tree
- Three Address Code

مثال7.3 على الطريقة الأولى: لو كان لدينا التعبير الحسابي التالي:

(a-b)*(c+d)+(a-b)

فان توليد الشفرة الوسطية باستخدام Postfix Notation هي:

ab-cd+*ab-+

حيث أن هناك أسبقيات للعمليات الحسابية:

1. ( )
2. ^ (الأس)
3. * , /
4. + , -

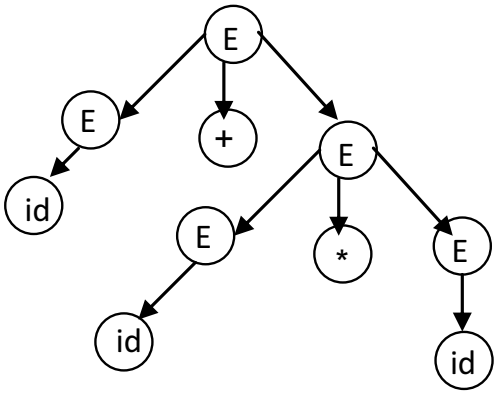
مثال7.4 على الطريقة الثانية: لو كانت لدينا القواعد التالية:

E → E+E | E*E | id

وكانت لدينا الجملة التالية :

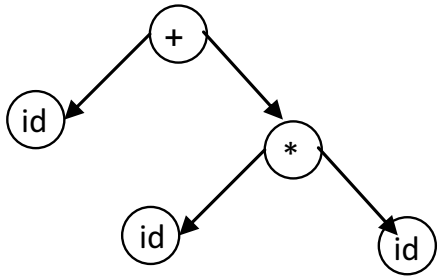
W = id + id * id $

فيمكن توليد ما يسمى بشجرة الإعراب (Parse Tree) كما في الشكل (7.7).



شكل (7.7) شجرة الإعراب(parse tree) للجملة id+id*id

لكن عندما تحذف الرموز القابلة للاشتقاق (E) في مثالنا فنحصل على شجرة تسمى شجرة النحو (Syntax Tree) وهي شجرة تتضمن فقط الرموز الغير قابلة للاشتقاق كما في الشكل (7.8).



شكل (7.8) شجرة النحو(Syntax tree) للجملة id+id*id

إن الطريقة الشائعة والمتعارف عليها هي الشفرة ذات الثلاث عناوين (Three Address Code) ومن اسمها فان كل إيعاز يحتوي على ثلاث معاملات كما موضح في المثال التالي:

مثال7.5: لو كان لدينا التعبير الحسابي التالي:

a+b*c+d

فان الشفرة ذات الثلاث عناوين (Three Address Code) المقابلة للتعبير أعلاه هي:

T1 = b * c

T2 = a + T1

T3 = T2 + d

كما هو واضح من المثال أعلاه فان كل إيعاز يتكون من ثلاث معاملات حيث أن الإيعاز الأول يتكون من المعاملات (T1, b, c) وهكذا بالنسبة للبقية.

هناك طريقتين لتمثيل الشفرة ذات الثلاث عناوين (Three Address Code) هما:

- التمثيل الثلاثي (Triple Representation)
- التمثيل الرباعي (Quadruple Representation)

إن الصيغة العامة للتمثيل الرباعي هي:

Operation Operand1 Operand2 Result

أي العملية ثم المعامل الأول ثم الثاني ثم الناتج هذا إذا كانت العملية Operation ثنائية المعامل (Binary Operation)، أما إذا كانت العملية أحادية المعامل (Unary Operation) فان الصيغة العامة للتمثيل الرباعي هي:

Operation Operand1 Result

أي العملية ثم المعامل الأول ثم فراغ ثم الناتج .

مثال7.6: لو كان لدينا التعبير التالي:

X = (a+b)*-c / d

فان الشفرة ذات الثلاث عناوين (Three Address Code) المقابلة للتعبير أعلاه هي:

T1 = a + b

T2 = - c

T3 = T1 * T2

T4 = T3 / d

X = T4

إن التمثيل الرباعي له يتمثل كما في الجدول التالي :

العملية	المعامل الأول	المعامل الثاني	الناتج
+	a	b	T1
-	c		T2
*	T1	T2	T3
/	T3	d	T4
=	T4		X

أما التمثيل الثلاثي له فانه يتمثل كما في الجدول التالي:

مؤشر	العملية	المعامل الأول	المعامل الثاني
(1)	+	a	b
(2)	-	c	
(3)	*	(1)	(2)
(4)	/	(3)	d
(5)	=	X	(4)

وهنا يظهر وجود المؤشّرات (Pointers) التي تأخذ حيزا من المساحة الخزنية إضافة إلى صعوبة إضافة أو حذف الايعازات لان ذلك سيتسبب في تغير كل

المؤشرات لذلك لا يفضّل استخدام هذا النوع من التمثيل في مرحلة توليد الشفرة الوسطية.

مثال : لو كان لديك التعبير الحسابي الآتي :

X = (A-C)*B+(A/(B+A)+(A-C)*B)

فان الشفرة ذات الثلاث عناوين (Three Address Code) المقابلة للتعبير أعلاه هي:

T1 = A - C

T2 = B + A

T3 = A / T2

T4 = T1 * B

T5 = T3 + T4

T6 = T1 * B

T7 = T6 + T5

X = T7

وان التمثيل الرباعي له يتمثل كما في الجدول التالي :

العملية	المعامل الأول	المعامل الثاني	الناتج
-	A	C	T1
+	B	A	T2
/	A	T2	T3
*	T1	B	T4
+	T3	T4	T5
*	T1	B	T6
+	T6	T5	T7
=	T7		X

أما التمثيل الثلاثي له فانه يتمثل كما في الجدول التالي:

مؤشر	العملية	المعامل الأول	المعامل الثاني
(0)	-	A	C
(1)	+	B	A
(2)	/	A	(1)
(3)	*	(0)	B
(4)	+	(2)	(3)
(5)	*	(0)	B
(6)	+	(5)	(4)
(7)	=	X	(6)

لو كانت لدينا عبارة الإحلال التالية:

X[i] = Y

فان التمثيل الثلاثي للعبارة أعلاه هي :

مؤشر	العملية	المعامل الأول	المعامل الثاني
(0)	[ ]	X	i
(1)	=	(0)	Y

ولو كانت لدينا عبارة الإحلال التالية:

Y = X[i]

فان التمثيل الثلاثي للعبارة أعلاه هي :

المعامل الثاني	المعامل الأول	العملية	مؤشر
i	X	[ ]	(0)
(0)	Y	=	(1)

مثال7.7 :لو كانت لدينا الايعازات التالية :

A = X[i]

X[i] = X[i-1]

X[i-1] = A

فان التمثيل الثلاثي للعبارة أعلاه هي :

المعامل الثاني	المعامل الأول	العملية	مؤشر
i	X	[ ]	(0)
(0)	A	=	(1)
1	i	-	(2)
(2)	X	[ ]	(3)
(3)	(0)	=	(4)
A	(3)	=	(5)

مثال7.8 : ولّد الشفرة الثلاثية للإيعاز التالي:

```
If  a = b then  x=1

        (0) if a=b goto (2)
        (1) goto (3)
        (2) x=1
        (3)
```

مثال7.9: ولّد الشفرة الثلاثية للإيعاز التالي:

```
for i = 1 to  n

x = x + i

        (1) i =1
        (2) if i > n  goto 6
        (3) x = x + i
        (4) i = i + 1
        (5) goto 2
        (6)
```

مثال7.10: ولّد الشفرة الثلاثية للإيعازات التالية المكتوبة بلغة C:

```
Main( )

{int i := 0;

  int B[10];

   while (i<10)

     B[i] =0;

}
```

(1) i =0
(2) if i <10 goto (4)
(3) goto (8)
(4) t1 = i * width
(5) t2 = addr(B) - width
(6) t2[t1] =0;
(7) goto (2)

حيث ان width يمثل عدد البايتات المطلوبة لكل عنصر من عناصر المصفوفة.

مثال7.11: ولّد الشفرة الثلاثية للبرنامج التالي :

```
Main( )
{  int i = 1;
    int a[10];
    While (i<10)
            a[i] = i;
}
```

إن الشفرة الثلاثية للبرنامج أعلاه هي:

(1) i=1
(2) if i<10 goto (4)
(3) goto (8)
(4) T1 = i * width
(5) T2 = addr (a) – width
(6) T2[T1] = i
(7) goto (2)

حيث أن width يمثل عدد البايتات المطلوبة لكل عنصر من عناصر المصفوفة.

مثال7.12: ولّد الشفرة الثلاثية للبرنامج التالي:

```
While (a < c  AND  b > d) do
    If a = 1
            then c = c + 1
    Else
            while a <= d  do
                    a = a + 3
```

إن الشفرة الثلاثية للبرنامج أعلاه هي:

(1) if a<c goto (3)
(2) goto (16)
(3) if b>d goto (5)
(4) goto (16)
(5) if a=1 goto (7)
(6) goto (10)
(7) T1 = c + 1
(8) c = T1
(9) goto (1)
(10) if a<=d goto (12)
(11) goto (1)
(12) T2=a+3
(13) a=T2
(14) goto (10)
(15) goto (1)