التناقضات في السياق والرمز Null

ان قواعد الكتلة التالية في لغة الكول

B ⟶ begin D;S end

D ⟶ d | D;d

S ⟶ s | Ss

تصبح قواعد اللغة بعد حذف التداخل من اليسار كما يلي:

B ⟶ begin D;S end

D ⟶ d{;d}*

S ⟶ s {;s}*

يبدو ان قواعد اللغة الجديدة تحقق شرط التطلع الى امام او شرط المسار الواحد ولكنها في الحقيقة ليست كذلك بسبب وجود تناقض في السياق يسببه الرمز (Null) والمحلل سينهج نهج معاكس عند اعطاءه مدخلات صحيحة مثل **begin d;d;s;s end**

يأخذ المحلل begin ويحللها على انها تحليل قاعدة B ثم ياخذ الرمز التالي فيجد d فيستدعي قاعدة D ثم يجد المحلل (;) فيدخل الاختيارالمتكرر في قاعدة D ثم يجد المحلل d على انها ضمن قاعدة D ثم يأخذ المحلل (;) بما ان المحلل داخل التكرار فيعتبر ان الفارزة المنقوطة هي بداية للتكرار من جديد ثم يجد المحلل s بدلا من d ومع ذلك لا يوجد خطأ فينهج المحلل نهجاً معاكساً لاغياً تحليله باعتبار ان الفارزة المنقوطة هي تعود الى الفارزة المنقوطة في تحليل B ثم يكمل ما تبقى من رموز المدخلات.

ان كل اختيار او تكرار في قواعد اللغة قد يسبب التناقض السياقي بسبب وجود رمز (Null) مخبأ .

ولازالة التناقضات في السياق وتحقيق شرط التطلع الى امام او شرط المسار الواحد فانه من الضروري تفحص الرموز التي تلي (Null) في صيغة جملية يمكن توليدها في قواعد اللغة .

والشروط التالية يجب تحققها لازالة التناقض السياقي

شروط ازالة التناقض في السياق

1- اذا وجد انتاج يحتوي على اختيار او تكرار كما يلي:

A ⟶ .....{B.....}C

A ⟶ .....{B.....}*C

فان رموز First* للرمز B يجب ان تكون مفصولة عن رموز First* للرمز C

مثال /هل القواعد التالية متناقضة في السياق و هل المدخلات التالية متناقضة سياقياً.

A ⟶ a {Bd}Cf

B ⟶ e | b

C ⟶ c | b

المدخلات : 1- (abdbf) 2-(abf)

الحل:

First*(B)=(B,e,b)

First*(C)=(C,c,b)

اذن القواعد فيها تناقض في االسياق لوجود رموز مشتركة(b)

المدخل الاول: abdbf

ياخذ المحلل a على انها تحليل للقاعدة الام الرئيسية A ثم يجد المحلل b فيدخل في الاختيار على انها تحليل B ثم يجد d ضمن الاختيار وبعدها يجد المحلل b كتحليل C ثم يجد المحلل f .

اذن لم يظهر هناك تناقض في السياق.

المدخل الثاني:abf

ياخذ المحلل a كتحليل للقاعدة الرئيسية A ثم يجد المحلل b (يتصور انها تحليل B)فيدخل الاختيار ثم يجد f ولا يجد d فينهج المحلل نهجا معاكسا ليحلل b على انها تحليل C ولا يدخل الاختيار ثم يجد f

اذن هناك تناقض في السياق بالنسبة للمدخل.

ملاحظة/ اذا كانت القواعد متناقضة في السياق فليس بالضرورة ان تكون المدخلات متناقضة فقط وانما يمكن ان تكون متناقضة او ليست متناقضة كما في المثال اعلاه واما اذا كانت القواعد ليست متناقضة في السياق فمن الضروري ان تكون المدخلات غير متناقظة في السياق .

2- اذا وجد انتاج ينتهي باختيار او تكرار كما يلي:

A⟶.....{D.....}

A⟶.....{D.....}*

فان رموز First* للرمز D يجب ان تكون مفصولة عن رموز First* لاي رمز يمكن ان يلي الرمز A في اية صيغة جملية.

مثال /هل القواعد التالية متناقضة في السياق و هل المدخدلات التالية متناقضة سياقياً.

S⟶ AEc

A⟶ a{Bd}

B⟶ b | e

E⟶ e | f

المدخلات : 1- (aedec) 2-(aec)

الحل:

First*(B)=(B,b,e)

First*(E)=(E,e,f)

اذن القواعد فيها تناقض في االسياق لوجود رموز مشتركة(e)

المدخل الاول: (aedec)

يجد المحلل a ويستدعي A ثم يجد e فيدخل في الاختيار على انها تحليل B ثم يجد d ثم يجد e فيستدعي E ثم يجد .

اذن لم يظهر هناك تناقض في السياق.

المدخل الثاني (aec)

يجد المحلل a فيستدعي A ثم يجد e فيدخل الاختيار على انها تحليل B ثم يجد c ولا يجد d ،فينهج المحلل نهجاً معاكساً لاغياً تحليله واعتبار e تحليل E ثم يجد d .

اذن هناك تناقض في السياق بالنسبة للمدخل.

ملاحظة: لا توجد طريقة معينة يمكن اتباعها لازالة التناقض السياقي ولكن يمكن اجراء بعض التغييرات على القواعد لمحاولة تحقيق الشروط السابقة.

مثلا يمكن اعادة صياغة القواعد السابقة في لغة الكول لازالة التناقض السياقي كما يلي:

B → begin D S end

D → d;{d;}*

S → s;{s;}*

تحليل اسبقيات العوامل

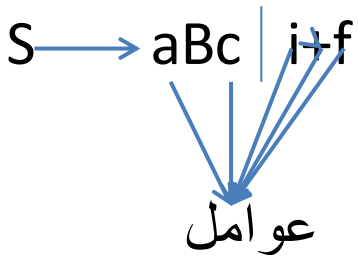
Operator Precede

تعتبر طريقة التحليل من اسفل الى اعلى هي اساس لاعراب اسبقيات العوامل والتي هي احدى اقدم طرق التحليل لكنها ما زالت مستعملة بسبب كفاءتها في تحليل التعبيرات اعتماداً على قواعد اللغة الاصلية غير المعدلة.

ان تحليل اسبقيات العوامل يعتمد على اسلوب تمييز انماط العبارات (اشكال العبارات)المدخلة .وعند تمييز نمط عبارة فهو يستبدل هذه العبارة بالرمز الموجود على الطرف الايسر للانتاج المناظر في القواعد وتسمى هذه العملية بالاختزال او التقليص.

في اعراب اسبقيات العوامل يتم تحديد انماط العبارات باستخدام اولوية(Priority) او اسبقية (Precedence) للرموز الختامية التي تسمى عوامل (Operator).

ملاحظة: لا يقصد بالعوامل (+،-،*،/، ....الخ) فقط وانما كل رمز ختامي في اللغة مثلاً :



بناء قواعد لغة العوامل

لبناء محلل يعتمد على اسبقية العوامل يجب ان تكون قواعد اللغة هي قواعد عوامل (Operator Grammar) والتي يجب ان تحقق الشرط التالي:

(لا توجد رموز غير ختامية متجاورة في انتاجات الطرف الايمن)

مثال / هل القواعد التالية هي قواعد العوامل؟

$S \rightarrow FT \mid F$

$F \rightarrow F* \mid F+ \mid F- \mid F/$

$T \rightarrow i \mid (s)$

القواعد اعلاه ليست قواعد عوامل لوجود رمزان غير ختاميان متجاوران

تحديد اسبقيات العوامل

يتم تحديد اسبقيات العوامل الموجودة في قواعد لغة العوامل باستخدام قوائم (Firstop+) و(Lastop+) وكما يلي:

1- لكل رمز غير ختامي كون قائمة (Firstop) والتي تتضمن الرمز الختامي الاول في كل انتاج في الطرف الايمن . اما اذا بدأ الانتاج برمز غير ختامي فيتم تضمينه بالاضافة الى الرمز الختامي الذي يليه . مثال/

X⟶a.... | Bc

Firstop(X)= (a,B,c)

2- لكل رمز غير ختامي كون قائمة (Lastop) والتي تتضمن الرمز الختامي الاخير في كل انتاج في الطرف الايمن، اما اذا انتهى الانتاج برمز غير ختامي فيتم تضمينه بالاضافة الى الرمز الختامي الذي يسبقه. مثال/

Y⟶ ....u | ......vW

Lastop(Y)=(u,v,W)

3- كون قوائم (Firstop+) و(Lastop+) وذلك بالتعويض عن كل رمز غير ختامي في قائمة (Firstop) بقائمة (Firstop) الخاصة به ، كذلك الحال مع القائمة (Lastop) .

مثال/

B⟶e …. | f….

X⟶a…. | Bc

Y⟶Xb… | d….

Firstop(B)=(e,f)

Firstop(X) =(a,B,c)

Firstop(Y) =(X,b,d)

Firstop+(B)=(e,f)

Firstop+(X) =(a,e,f,c)

Firstop+(Y) =(a,e,f,c,b,d)

4- كون مصفوفة لعلاقات اسبقيات العوامل من انتاجات قواعد اللغة وقوائم (Firstop+) و(Latop+) كما يلي:

أ- لكل انتاج U⟶....aB... ضع a<α حيث α كل رمز ختامي في قائمة (Firstop+) للرمز B.

ب- لكل انتاج U⟶.... Bc... ضع ɣ<c حيث ɣ كل رمز ختامي في قائمة (Latop+) للرمز B.

ج- لكل انتاج U⟶....ac.... أو U⟶.....aBc.... ضع a=c

د- ضع β<% حيث β كل رمز ختامي في قائمة (Firstop+) لرمز البداية (S) وان (%) هو رمز الفراغ.

هـ- Delta<% حيث Delta كل رمز ختامي في قائمة (Latop+) لرمزالبداية (S) .

توضيح خارجي للنقطة رقم 4

1- السطور

رمز الفراغ (%) علاقته <(اصغر من) قائمة (+Firstop) لرمز البداية.

كل رمز ختامي علاقته <(اصغر من) قائمة (+Firstop) للعنصر الذي بعد الرمز الختامي في القواعد.

2- الاعمدة

رمز الفراغ (%) علاقته >(اكبر من) قائمة (+Latop) لرمز البداية.

كل رمز ختامي علاقته >(اكبر من) قائمة (+Latop) للعنصر الذي قبل الرمز الختامي في القواعد.

3- اذا كان هناك رمزان ختاميان متجاوران او هناك رمز ختامي وبعده رمز غير ختامي ثم رمز ختامي فان علاقة الرمزان الختاميان متساويان.

مثال// لديك قواعد لغة عوامل الاتية:

S⟶A

A⟶T | A+T | A-T

T⟶F | T*F | T/F

F⟶P | P^F

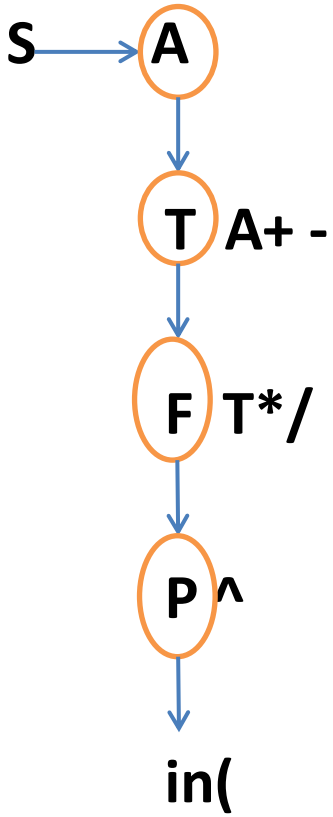
P⟶i | n | (A)

كون مصفوفة لعلاقات اسبقيات العوامل ؟

الحل:

الرمز	Firstop	Lastop
S	A	A
A	TA+ -	T+ -
T	FT*/	F*/
F	P^	PF^
P	in(	in)

لتكوين قوائم (Firstop+) و(Lastop+) نقوم يتعويض الرموز غير الختامية التي ظهرت في قوائم (Firstop) و(Lastop) وكما يلي :



Firstop+(S)=in(^T*/A+-

وهكذا يتم التعويض التراجعي لكل الرموز الغير ختامية في القوائم

الرمز	Firstop+	LastOp+
S	in(^T*/A+ -	in)F^*/+ -
A	in(^T*/A+ -	in)F^*/+ -
T	in(^T*/	in)F^*/
F	in(^	in)F^
P	in(	in)

مصفوفة الاسبقيات

	%	(	)	i,n	^	*,/	+,-
%		<		<	<	<	<
(		<	=	<	<	<	<
)	>		>		>	>	>
i,n	>		>		>	>	>
^	>	<	>	<	<	>	>
*,/	>	<	>	<	<	>	>
+,-	>	<	>	<	<	<	>

التداخل من اليسار غير المباشر

- يكشف عن التداخل غير المباشر من اليسار باستخدام قائمة (First+) لكل رمز غير ختامي. فان الرمز المتداخل من اليسار بصورة غير مباشرة سيظهر في قائمة (First+) الخاصة به ويتم ذلك عن طريق :

1- بناء قائمة (First)لكل رمز غير ختامي وذلك باخذ اول رمز من كل بديل موجود في الطرف الايمن لقاعدة ذلك الرمز غير الختامي .

مثال

A ⟶ abc | Bed | efy

B ⟶ b…. | d….

First(A)=(a,B,e)

First(B)=(b,d)

2- بناء قائمة (First+) وذلك بالتعويض عن كل رمز غير ختامي في قائمة (First)بقائمة (First)الخاصه به.

First+(A)=(a,b,d,e)

First+(B)=(b,d)

اذن لا يوجد تداخل من اليسار بصورة غير مباشرة لعدم ظهور رمز الطرف الايسر في القوائم.

مثال/اكشف عن التداخل غير المباشر في القواعد التالية:

S ⟶ c | L | R

L ⟶ SrE | SruE

R ⟶ wEds | iFds | Lx

First(S)=(c,L,R)

First(L)=(S)

First(R)=(w,i,L)

First+(S)=(c,S,w,i,L)

اذن يوجد تداخل من اليسار بصورة غير مباشرة.

طور مواصفات المستهدفات

يقوم هذا الطور بملئ جزء المواصفات في جدول الرموز لكل معرف من خلال معالجة اعلان ذلك المعرف في شجرة الاعراب .وان تفاصيل الاعلان ستحدد الصنف والنوع ومعلومات الوصول لكل معرف.

1- الصنف : اما ان يكون متغير بسيط،مصفوفة،قيد،دالة،نهج معالجة،..... الخ.

2- النوع: اما ان يكون حقيقي ،رمزي،صحيح،منطقي، ... الخ

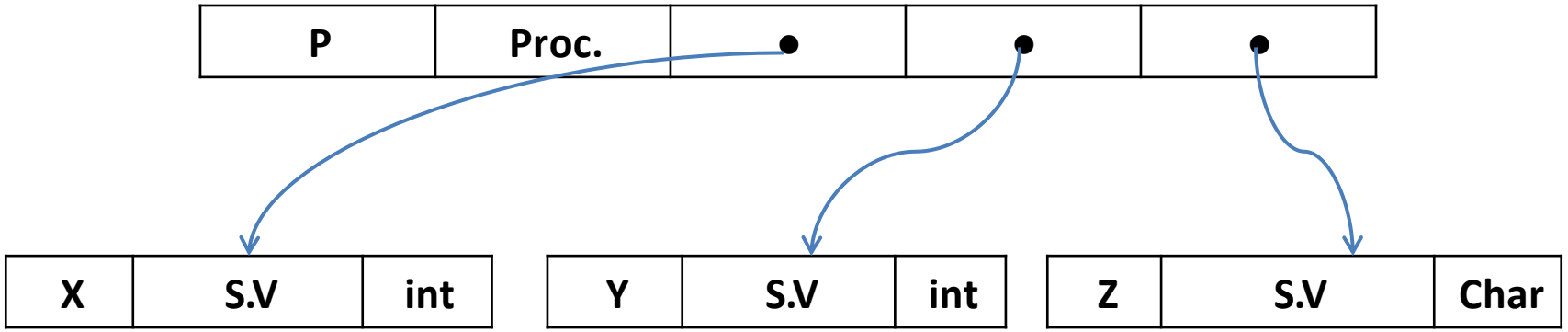
3- معلومات الوصول: لو كان لدينا مثلا:

أ- مصفوفة فنحتاج الى معرفة ابعادها وحجمها وحجم كل متغير (ابعاد العنصر)

ب- نهج معالجة فنحتاج قائمة واصفات المعلمات.

4- عنوان وقت التنفيذ: يحدد من قبل المحمل

ان في مؤلفات كثيرة ربما ترتبط واصفات المستهدفات معاً بطرق مختلفة فمثلا يحتوي واصف نهج المعالجة على مؤشر الى واصف كل معلمة من معلمات النهج.



كما ان الواصف يمكن ان يكون على شكل مكدس في اللغات الهيكلية الهيكل حيث تسمح باستخدام نفس الاسم في اعلانات مختلفة في كتل مختلفة مثلا :

B1

x:iteger;

B2

x:real;

B3

x:Boolean;

B4

x:string;

B1

X	DEC1	DEC2	DEC3	DEC4

B4

X	DEC1	DEC4

أ- مصفوفة ستاتيكية احادية البعد

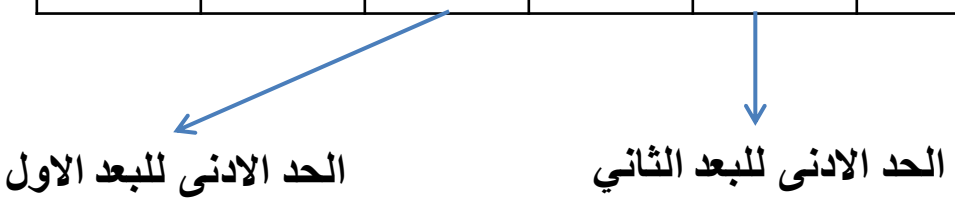
A:Array [1..10] of integer;

اسم المصفوفة	عدد الابعاد	الحد الادنى	الحد الاعلى	نوع المصفوفة	عنوان اول موقع
A	1	1	10	int	α

ب- مصفوفة ستاتيكية ثنائية البعد

B:Array [1..10,2..7] of real;

B	2	1	10	2	7	real	α



ج- مصفوفة ديناميكية احادية البعد

C:Array [1..N] of Char;

C	1	1	N	Char	α

الحد الاعلى(اعلان ديناميكي) يحدد وقت التنفيذ

تخصيص العناوين

1- **للمعرفات:**حسب مبدأ اطار البيانات حيث يكون لكل كتلة اطار بيانات خاص بها تنفذ فيه وتعامل كل كتلة بشكل مستقل فيتم تحديد العناوين النسبية لكل كتلة بشكل متسلسل على حدى مثلاً:

B1

X #1

Y #2

Z #3

B2

R #1

X #2

W #3

2- **للمصفوفات:** باستخدام متجة المعلومات (Pop Vector) والذي يحتوي على عنوان اول عنصر في المصفوفة ومعلومات اخرى مثل نوع المصفوفة وعدد الابعاد والحد الادنى والاعلى لكل بعد.

أعمال أضافيه للمؤلف

! أسناد وقت التنفيذ / Run Time Support

يقوم المؤلف بصيانه ومتابعه و التحضير في بعض امور التخزين التي تفيد في وقت التخزين ومن هذه الامور صيانه المكدس الذي يستخدم في تنفيذ البرامج الفرعيه المتداخله او البرامج كتليه الهيكل بأستخدام سجلات التنشيط.

? كشف الاخطاء / Error Detection

ان الاخطاء التي يكتشفها المؤلف نوعان

? أخطاء وقت التأليف

* الاخطاء اللغويه مثل

الخطاء

IF (x := y) Then

* اخطاء وقت الترجمه مثل

Type missmatch (عدم تطابق الانواع البيانيه)

Z := X + Y

integer Real

* اخطاء وقت التحميل مثل تجاوز الشفره للحدود المسموحه.

ب) أخطاء وقت التنفيذ
مثل الـ (Overflow) وتجاوز الحدود المسموحه

Example

```
A: Array [1..50] of integer;
.
.
.
.
ReadIn (I);
A[I]:= A[I]+1;
```

عند أدخال الرقم 100 سوف تتجاوز حدود المصفوفه

التأليف بدورتين

ان معظم المؤلفات ليست وحيده الدوره وانما تشتمل دورتين (بدلاً من اربع أو خمس دورات) كما في المخطط السابق للمؤلف ويكون تنظيم المؤلف للدورتين بحيث توجد دوره تحليل ودوره ترجمه وتشمل دوره التحليل اطوار القراءه وتحليل الفقرات و تحليل القواعد اللغويه ولكي توجد هذه الأطوار في دوره واحده يكون من المناسب جعل طور القراءه روتين فرعي لطور تحليل الفقرات وطور تحليل الفقرات يكون روتين فرعي لطور تحليل القواعد اللغويه.

اما دوره الترجمه فتشمل طور الترجمه البسيط فقط.

نظرة تفصيلية عن أطوار المؤلف

١) طور القراءة //

يعتبر طور القراءة روتين فرعي لطور تحليل الفقرات وهو يقوم بما يلي لكل رمز من رموز البرنامج.

تشويش فيزيائي ← شفرة تعتمد على الجهاز

الشفرة التي تعتمد على الجهاز ← شفرة رموز قياسية (ASCII)

٢) طور تحليل الفقرات //

يقوم هذا الطور بتحويل شفرة الرموز القياسية الى شفرة فقرات داخلية حيث يقوم بتجميع الرموز مع بعضها لتكوين فقرات وتحديد شفرة لكل فقرة مثل (K.W ، Id ، opr ، no ، ...)

Example

```
program ex ;
```

↓↓↓ ...

$C_1 C_2 C_3$ طور القراءة (شفرة أسكي)

K.W طور تحليل الفقرات

ان طور القراءه وتحليل الفقرات قد تستغرق (50%) من وقت المؤلف في حين تستغرق هاذين الطورين لا تتجاوز (10%) من تستغرق المؤلف وذلك لأن قراءه البرنامج لخيط رمزي طويل (الألف الرموز) ومن ثم تجميع هذه الرموز لتكوين فقرات تتطلب وقت كبير.

طور تحليل الفقرات ← يستدعي ← طور القراءه ← يقراء ← رمز

في لغات البرمجه أعلى كلفه (زمن) تكون لأستدعاء نهج معالجه.

طور تحليل الفقرات ← يستدعي ← طور القراءه ← يشكل ← سطر كامل (يوضع في Buffer)

ان طور تحليل المفردات سيكون اكثر كفاءه وذلك بجعل طور القراءه يقراء سطراً كاملاً (بدلاً من رمز واحد) في كل مره الى خزان انتقالي (Buffer) ومن ثم يقوم محلل الفقرات بفحص رموز هذا السطر في الـ (Buffer) وتقطيعها الى مجموعه من الفقرات.

ان هذه الطريقه تؤدي الى تقليل عدد مرات أستدعاء طور القراءه وكذلك تكون مفيده في انتاج تقارير بالأخطاء اللغويه في الأطوار القادمه وذلك لتحديد الخطاء ورقم السطر.

تمثيل الفقره

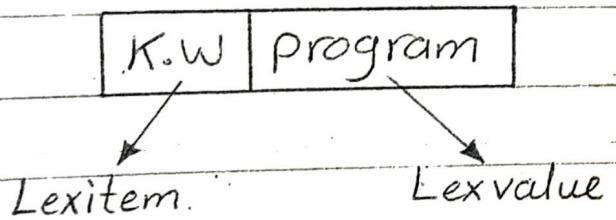
يتم تمثيل الفقرات بأستخدام جدول الفقرات حيث يكون لكل فقره في الجدول مايلي

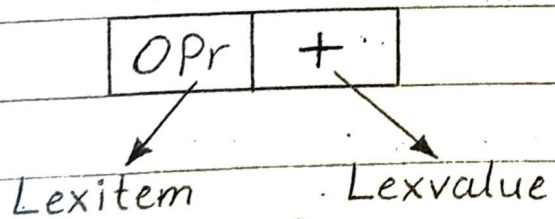
(1) نوع الفقره Lexitem :

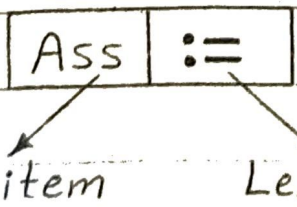
شفره الفقره او الفئه التي تنتمي لها الفقره مثل (no ، opr ، pun ، Id ، K.w ، ... ، الخ)

(2) الفقره Lexvalue :

وتعني قيمه اضافيه تعرف الفقره مثلاً







محلل الفقرات

هو محلل الفقرات يحدد فئة (Lexitem) لكل فقره بعد تجميعها
من خلال القواعد المايكروية للغة.

القاعدة المايكروية // هي قاعدة مستخدمة لوصف كل فئة من
فئات اللغة.

Example: القاعدة المايكروية المعرف في لغة باسكال تكون كمايلي

<Identifier> ::= <letter> { أي <letter> | <digit> | <underscore> } * → Loop
(or)

<letter> ::= A | B | ... | Z | a | b | ... | z

<digit> ::= 0 | 1 | ... | 9

<underscore> ::= _

تحليل التعبيرات من اسفل الى اعلى وانتاج ثلاثيات كشفرة وسطية:

تحتاج هذه الخوارزمية الى مكدسين احدهما لحفظ العوامل وترتيب الاسبقيات والاخر للمعاملات

وحسب الشرط التالي :(هل ان اسبقية قمة مكدس العوامل > او = العامل القادم والقمة ليست فارغة)

الجواب نعم: اسحب عامل ومعاملين وكون ثلاثية وادفع رقم الثلاثية الى مكدس المعاملات.

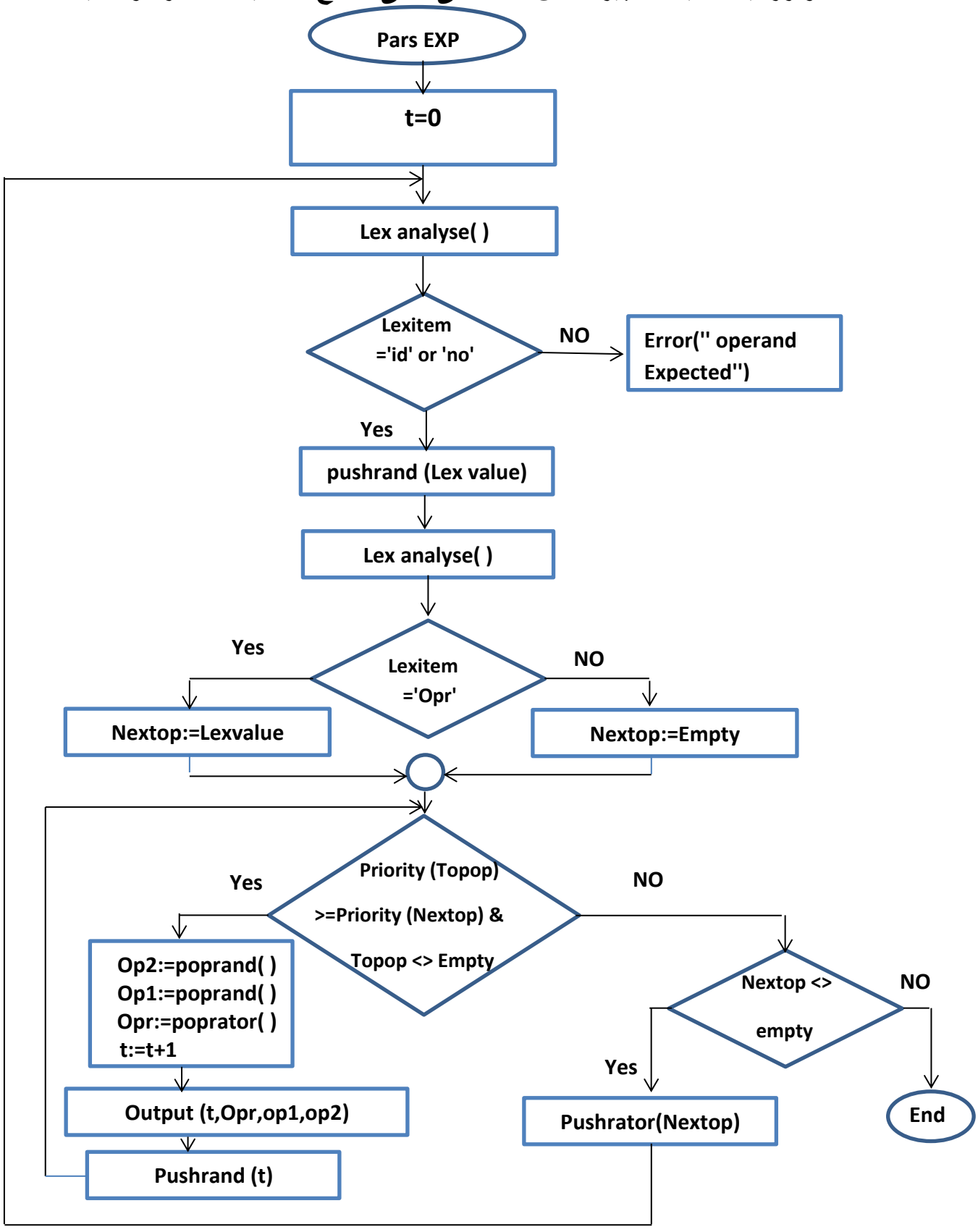
الجواب كلا: ادفع العامل القادم الى مكدس العوامل

مثال: حلل التعبير التالي من اسفل الى اعلى لانتاج ثلاثيات كشفرة وسطية

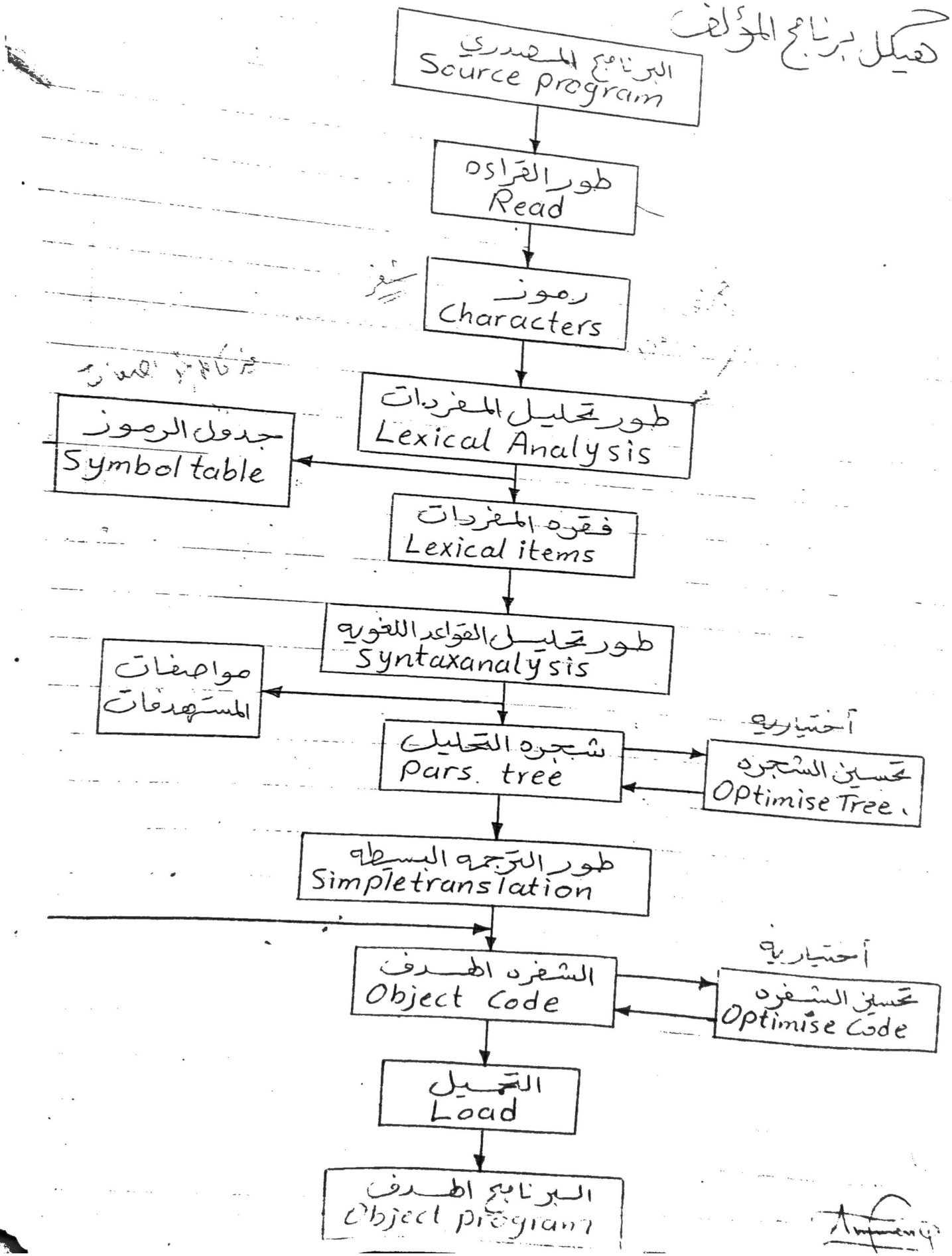
X*Y+Z-M

INPUT	STACK (operands)	STACK (operators)
X*Y+Z-M	EMPTY	EMPTY
*y+z-m	X	EMPTY
Y+z-m	X	*
+Z-M	Y X	*
Z-M	#1 #1=*XY	+
-M	Z #1	+
M	#2 #2=+#1Z	-
Ø	M #2	-
Ø	#3 #3=-#2M	EMPTY

خوارزمية تحليل التعبيرات من اسفل الى اعلى لانتاج الثلاثيات كشفرة وسطية:



هيكل برنامج المؤلف



ملاحظة:
الحروف الكبيرة تمثل طور او مرحلة فرعية اما الحروف الصغيرة تمثل هيكل معلومات او تمثيل وسطي للبرنامج.

نظرة عامة عن أطوار المؤلف

(1) طور القراءة (READ):
هذا الطور يستقبل البرنامج المصدري على شكل تسويشات فيزيائية وينتج هذا البرنامج المصدري بشكل خيط رمزي طويل ويعتبر طور القراءة طور جزئي من طور تحليل المفردات (يستدعى من قبله)

(2) طور تحليل المفردات (Lexical Analysis):
ينتج هذا الطور مفردات (عناصر) مثل مفردات الاعداد، مفردات العوامل .... الخ.
وتوضع داخل جدول يسمى جدول المفردات (المفردات) وبالإضافة الى هذا الجدول فأنه يكون جدول آخر هو جدول الرموز الذي يحتوي مدخل لكل اسم في البرنامج.
فعند ظهور اسم معين في البرنامج يقوم هذا الطور بفحص جدول الرموز فاذا كان الاسم غير موجود يقوم بفتح مدخل له في جدول الرموز، اما في حالة وجود الاسم فأنه يقوم بعمل مؤشر الى مدخل الاسم.
وجدول الرموز يحتوي اسماء المتغيرات والاجهزة والدوال و العلامات ويكون خالي من المواصفات في هذا الطور.

```
Program main;
 Var
   x:integer;
    y:real;
Begin
   x:=10;
    y:=2.5;
  x:=x*2+1;
 product(x,y);
End.
```

جدول المعرفات :

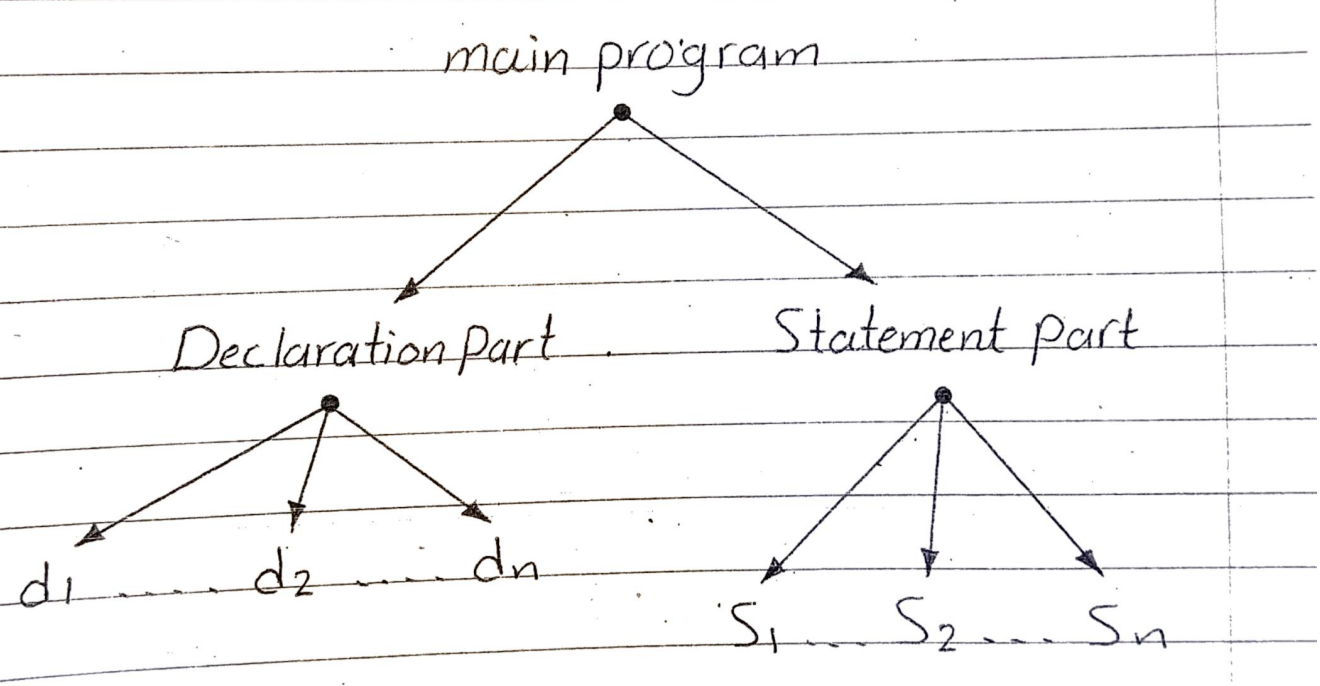
الاسم	المواصفات
Main	
x	
y	
product	

العقره	نوع العقره
program	K.W
main	Id
;	pun
Var	K.W
x	Id
y	Id
x	Id
y	Id
product	proc

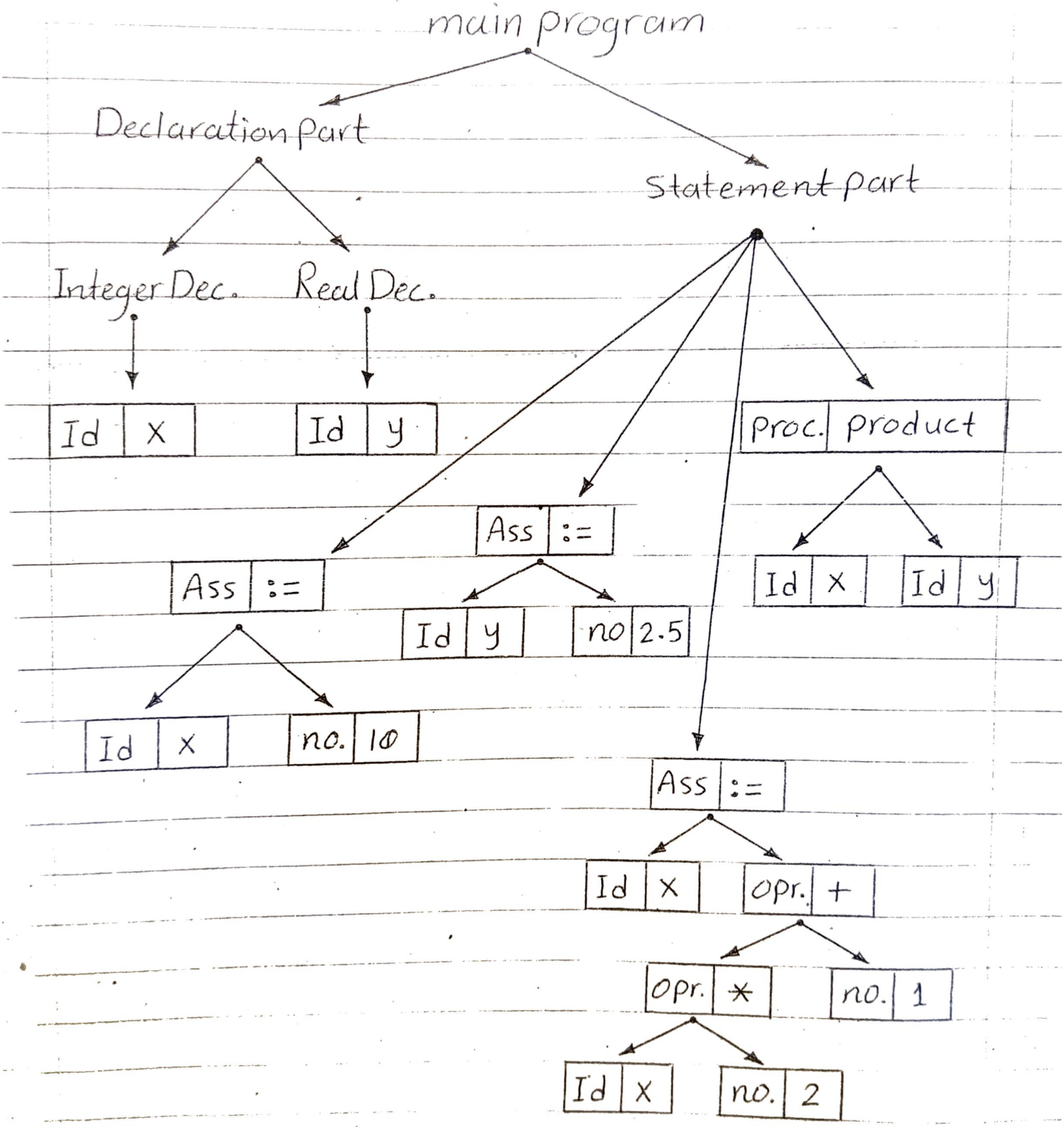
(٣) طور تحليل القواعد اللغوية (Syntaxanalysis):

ينتج هذا الطور شجره اعراب تمثل طريقه ربط اجزاء البرنامج (الفقرات) لتكوين تراكيب اكبر وكيفيه ترابط هذه التركيب لتكوين تراكيب اكبر منها اي ان الشجره توضح هيكل البرنامج ككل.

اذن مهمه هذا الطور هو فهم الكلمات الناتجه من تحليل الفقرات وهل هي مرتبه حسب قواعد اللغه.




```
Program main;
  Var
    x : integer;
    y : real;
Begin
    x:= 10;
    y:= 2.5;
    x:= x * 2 + 1;
    Product (x, y);
```



* ان كل عقده أسم مثل Id | x لزا مؤشر في جدول الرموز

أنواع الفقرات

هنالك نوعان من الفقرات

1) فقرات بسيطة /

وهي التي تتكون من رمز واحد فقط مثل
(; ، (،) ، [،] ، ... ، الخ)

2) فقرات مركبة /

وهي التي تتكون من مجموعة رموز مجمعة حسب قاعدة ماكروية معينة مثل
(Id ، num ، Ass ، ... ، الخ).

ملاحظة

من المناسب لمحلل الفقرات فحص الرمز التالي لآخر رمز في الفقرة الحالية لتحديد فيما اذا كانت فقرة جديدة او بداية لفقرة جديدة وذلك لتجنب حصول أخطاء في طور تحليل القواعد اللغوية كما في الحالة التالية

يعتبر محلل الفقرات ان

ولكن عند الوصول الى طور تحليل القواعد اللغوية سيحصل خطأ لعدم وجود التعاقب بين (Opr و Pun) ولذلك يجب على محلل الفقرات فحص الرمز التالي وتقريران

Ass	:=

تمييز المعرفات وكلمات المفاتيح

عندما يصادف محلل الفقرات حرف كأول رمز في فقره فأنه يتبع القاعدة المايكروية الخاصة بالمعرفات وهكذا يبدأ بتجميع الرموز التالية طالما كانت حرف او رقم وفي هذه الحالة اما ان تكون الفقره معرف (Id) او كلمة مفتاحية (K.W) [كلمات المفاتيح لا تحتوي ارقاماً] وهناك جدول خاص لكلمات المفاتيح المستخدمة في اللغة فيقوم المحلل بمقارنة الفقره (التي لا تحتوي على رقم) مع محتويات جدول كلمات المفاتيح فأذا كانت موجوده في الجدول يقرر أن (Lexitem هو K.W) اما في حاله انها غير موجوده في الجدول يقرران (Lexitem هو Id).

ان الأسماء الخاصة بالمعرفات تستخدم في اماكن متعددة من البرنامج (كمتغيرات، كأسماء أنهجة أو دوال، كعلاقات) ولذلك لابد من تحديد مواصفات كل معرف ويتم ذلك بفتح مدخل لكل معرف يظهر لأول مره في البرنامج في جدول الرموز حيث يحتوي مدخل اسم المعرف ومواصفاته (حالياً المواصفات خالية) اما في حاله تكرار ظهور الأسم في البرنامج فيتم عمل مؤشر الى مدخل الأسم في جدول الرموز.

اكتشاف الأخطاء في طور تحليل الفقرات

يكون اكتشافه للأخطاء محدود لانه يتعامل مع القواعد المايكرويه
ومن الأخطاء التي يكتشفها

1. عدم انتماء فقره الى أيه قاعده ميكرويه مثلاً

Program # ← لا تنتمي الى اية قاعده ميكرويه

2. يكتشف أخطاء الفيض (Overflow) والغيض (Underflow)
اي استعمال عدد خارج الحدود المسموح به.

3. في حاله وجود قيود على طول المعرفات فأن محلل الفقرات يكتشف الخطاء الناتج عند تجاوز الحد المسموح.

Note

من المناسب لطور تحليل القواعد اللغويه قيام محلل الفقرات بحذف التعليق الذي يحصر بين { }
او بين /* */ او بين (* *)

أستعمال جدول الرموز

يستعمل هذا الجدول من قبل كل الأطوار ابتداءً من تحليل العقرات الى طور الترجمة كما انه سيستخدم في وقت التنفيذ للمساعدة في تشخيص أخطاء البرنامج.

ومن المناسب القول ان جدول الرموز يعتبر مركز عمل المؤلف خاصة ان طور تحليل العقرات يجب ان يجد الأسماء في الجدول بأسرع ما يمكن حيث ان مصادفة كل معرف يتطلب من طور تحليل العقرات البحث في الجدول.

وتستعمل اطوار الترجمة البسيطة معلومات جدول الرموز في انتاج تقارير اخطاء عندما يستعمل اسم معين بصورة غير صحيحة وهذا يتطلب بحث الجدول ايضاً.

ولهذا فان تصميم جدول الرموز يؤثر على قدرات المؤلف في اصدار الأخطاء وسرعة الانجاز.

البحث في جدول الرموز

هنالك عدة طرق للبحث ومنها:-

1) البحث التعاقبي Sequential Search

هي الطريقة الأولى التي استخدمت في تصميم المؤلف والأكثر وضوحاً ولكنها الأسوء في انجازية البحث حيث تفاحص مداخل الجدول الواحد بعد الآخر.

فمثلاً في جدول يحتوي على (N) من اسماء المعرفات يكون معدل المداخل المفحوصة وصولاً الى اسم معين هو (N/2)

٢) البحث الثنائي Binary Search

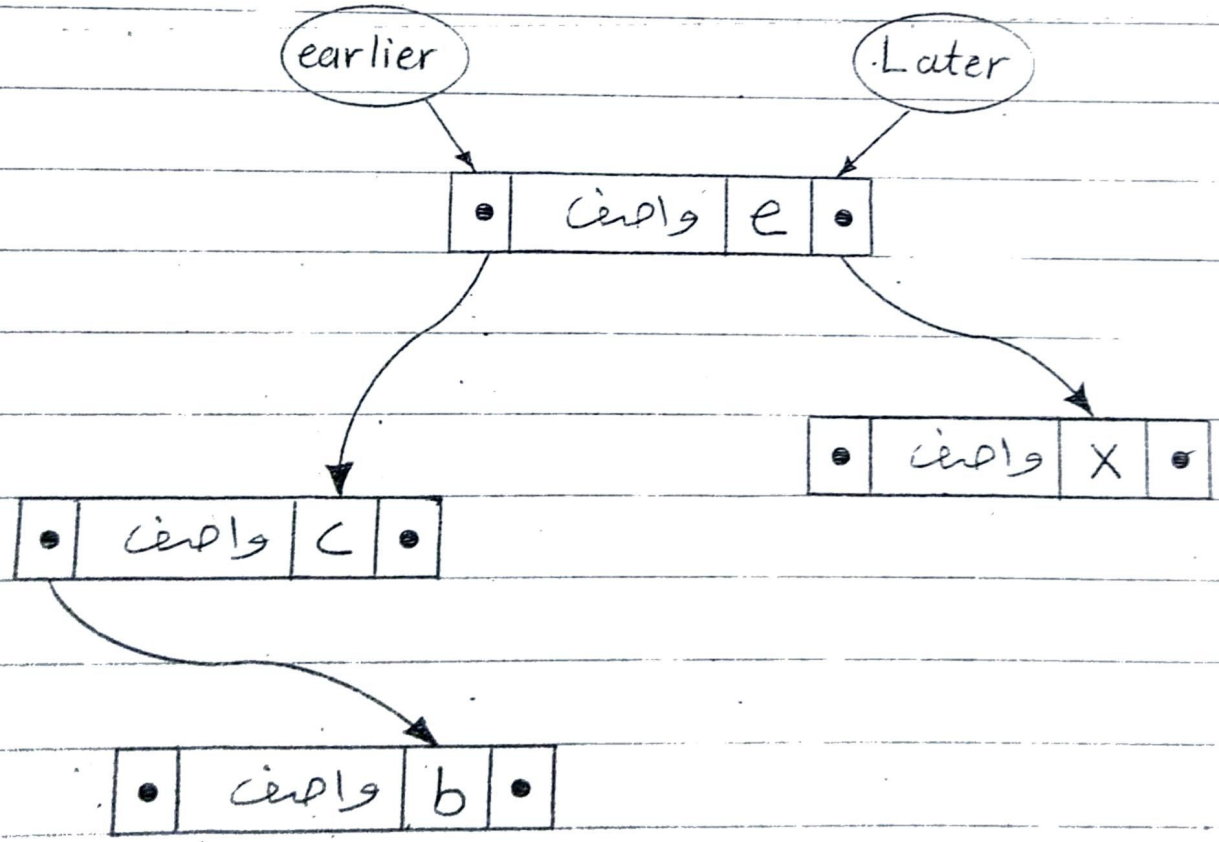
يستعمل البحث الثنائي مع الجداول المرتبة (بطريقتها) وبالرغم من ان خوارزمية البحث الثنائي لها انجازية بحث جديدة لكنها عديمة الفائدة عملياً بسبب ضرورة اضافة أسماء جديدة الى جدول الرموز من وقت لآخر.

لكي نضيف محال الفقرات مثلاً الأسم (D) في المثال التالي علينا تحريك كل المداخل من (F) الى (Z) لإفساح المجال للمدخل الجديد.

الأسم	المواصفات
A	
C	
F	
R	
Z	

٤) البحث في شجرة الأسماء

للتغلب على صعوبة اضافة عنصر الى الجدول المرتب يمكن استخدام عتيل آخر للجدول بشكل شجرة ثنائية مرتبة حيث يخزن كل اسم ومواصفاته مع مؤشرين أحدهما يشير الى شجرة الأسماء الفرعية السابقة والآخر يشير الى شجرة الأسماء الفرعية اللاحقة لم بالترتيب.



تحليل التعبيرات من اسفل الى اعلى وانتاج خيط تدوين لاحق كشفرة وسطية:

تحتاج هذه الخوارزمية الى المكدس لحفظ العوامل وترتيب الاسبقيات وحسب الشرط التالي :(هل ان اسبقية قمة مكدس العوامل > او = العامل القادم والقمة ليست فارغة)

الجواب نعم: اسحب من المكدس الى المخرجات

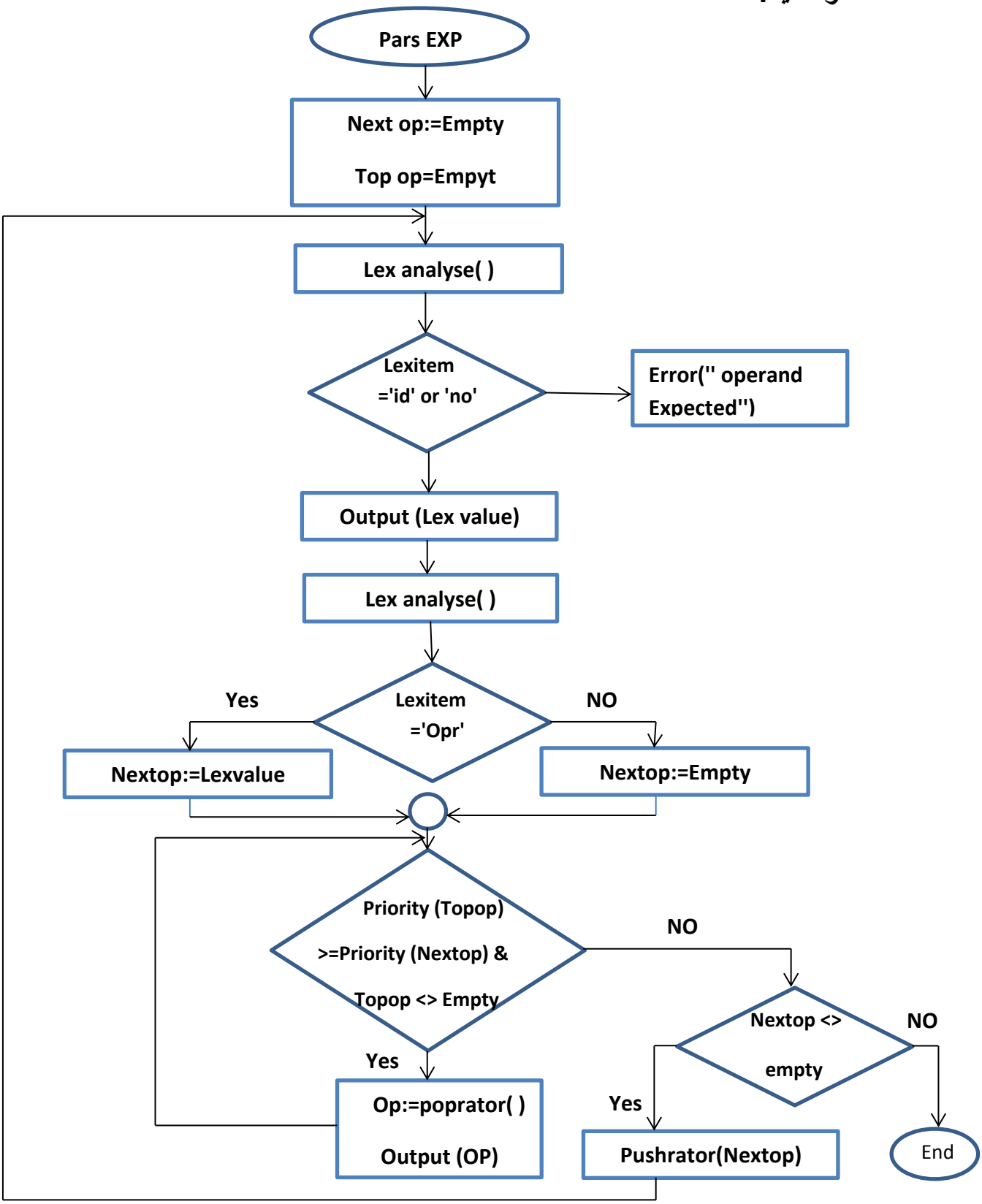
الجواب كلا: ادفع العامل القادم الى مكدس العوامل

مثال: ححل التعبير التالي من اسفل الى اعلى لانتاج خيط التدوين اللاحق كشفرة وسطية

X*Y+Z-M

INPUT	OUTPUT	STACK
X*Y+Z-M		EMPTY
*y+z-m	X	EMPTY
Y+z-m	X	*
+Z-M	XY	*
Z-M	XY*	+
-M	XY*Z	+
M	XY*Z+	-
Ø	XY*Z+M	-
Ø	XY*Z+M-	EMPTY

خوارزمية تحليل التعبيرات من اسفل الى اعلى لانتاج حيط التدوين اللاحق كشفرة وسطية:



2- طور تحليل القواعد اللغوية:

ان مهمة هذا الطور هو فهم الفقرات الناتجة من تحليل المفردات وكذلك كشف الطريقة التي ترتبط فيها الفقرات معا في عبارات وكيفية ترابط هذه العبارات في عبارات اكبر وهكذا.

مخرجات طور تحليل القواعد القواعد اللغوية(الشفرة الوسطية):

هناك عدة طرق لتمثيل الشفرة الوسطية منها

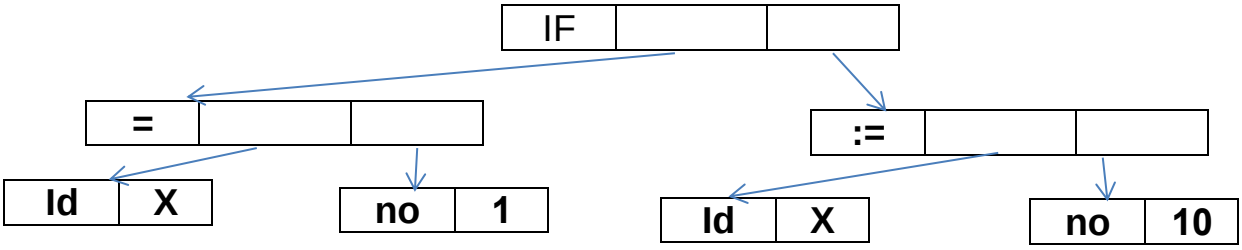
1- شجرة الاعرب parsing Tree : يقسم عمل محلل القواعد اللغوية الى قسمين:

أ- كيفية التعرف على عبارات اللغة(Recognize) .

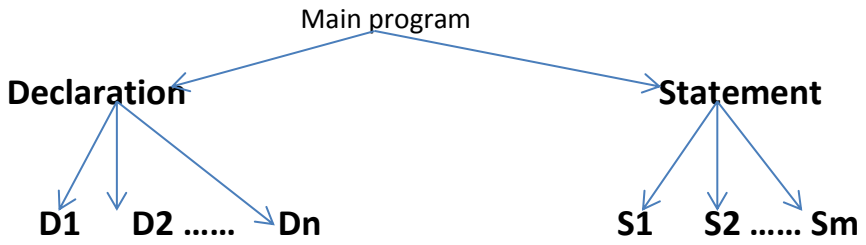
ب- بناء عقدة شجرة للعبارة المميزة .مثلا

If x=1 then x:=10;

بعد ان يتعرف المحلل على اول فقرة بالعبارة مثلا if فان المحلل يميز تلك العبارة وفق القواعد بانها تتكون من جزئين هما الشرط(condition) والاخر العبارة(statement) فبذلك يقوم المحلل ببناء عقدة الشجرة بناءا على تمييز تلك العبارة وكما يلي:

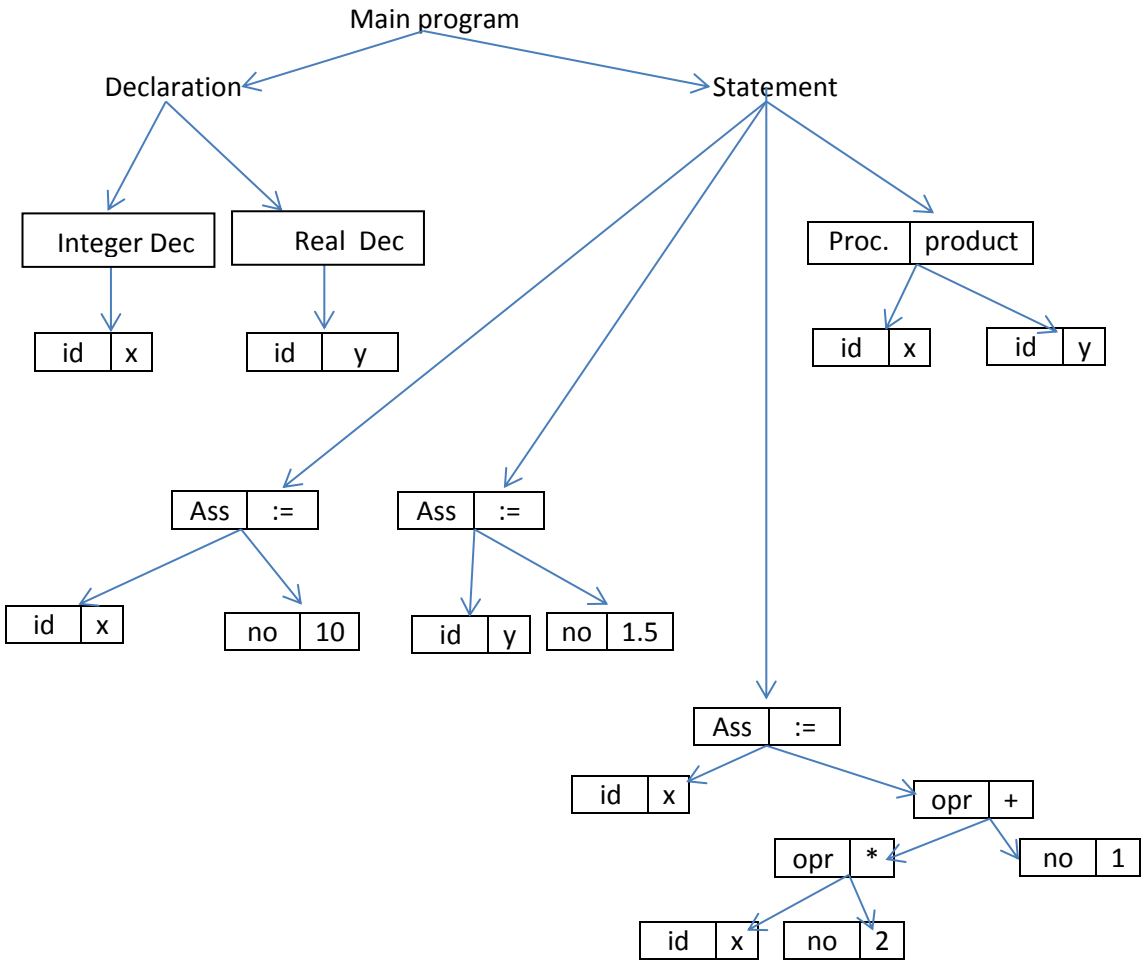


وبشكل عام فان الهيكل الرئيسي لرسم شجرة الاعراب تكون حسب المخطط التالي:



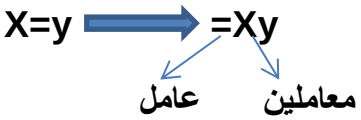
مثال/ لو اخذنا المثال الثالي والمطلوب بناء شجرة اعراب للفقرات الناتجة من محلل الفقرات فتكون بالشكل التالي:

```
Program math;
  Var
     X:integer;
     Y: Real;
 Begin
    X:=10;
    Y:=1.5;
    X:=X*2+1;
    Product(x,y);
End.
```


Main program
Declaration
Statement
Integer Dec
Real Dec
Proc.
product
id
x
id
y
id
x
id
y
Ass
:=
Ass
:=
id
x
no
10
id
y
no
1.5
Ass
:=
id
x
opr
+
opr
*
no
1
id
x
no
2

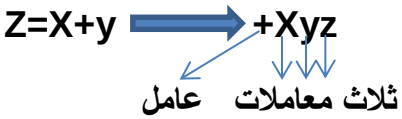
2- الثلاثيات Triples :

وتتكون الثلاثية من عامل (operator) ومعاملين (operands) مثال:



3- الرباعيات Quaternion :

وتتكون الرباعية من عامل (operator) وثلاث معاملات (operands) مثال:



4- خيط التدوين(التثبيت) اللاحق postfix string :

وتستخدم هذه الطريقة مع التعابير الحسابية : مثال/

ملاحظة: تستخدم الطرق 2،3،4 مع التعابير الحسابية.

❖ انواع محلل القواعد اللغوية:

1- محلل من اسفل الى اعلى Bottom-Up parsing .
2- محلل من اعلى الى اسفل Top-Down parsing .
3- محلل التطلع الى امام برمز واحد One symbol look head Analysis.
4- محلل المسار الواحد One Track Analysis .
5- محلل اسبقيات العوامل Operator Precede.

وسنقوم بشرح تفاصيل كل انواع المحللات اعلاه

1- محلل من اسفل الى اعلى Bottom-Up parsing :

يقوم بربط الفقرات الصغيرة مع بعضها لتكوين تراكيب اكبر (حسب قواعد اللغة) وصولا الى رمز البداية للتعليمة التي يتم تحليلها .

ويستخدم التحليل من اسفل الى اعلى مع التعابير الحسابية.

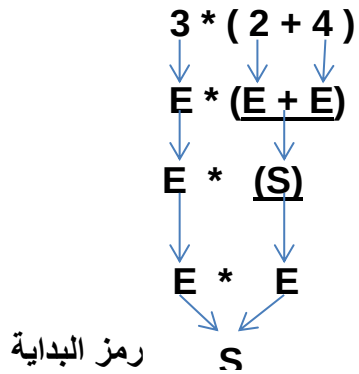
مثال/ القواعد التالية لتعبير حسابي في لغة برمجة معينة كما يلي:

S ⟶ E+E | E-E | E*E| E/E

E ⟶ S | (S) | 0 |……| 9

السؤال /هل التعبير الحسابي (2+4)*3 وفق هذه القواعد؟

الحل/

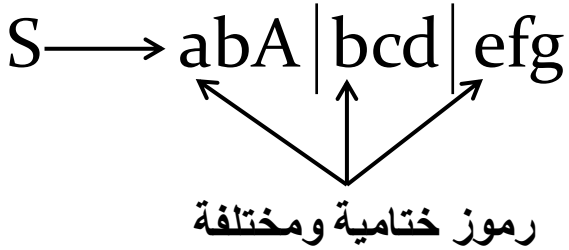


اذن التعبير الحسابي وفق القواعد.

محلل التطلع الى امام برمز واحد
One symbol look head analyser

- هذا المحلل يمكنه دائما تحديد مسار التحليل المناسب وذلك بتفحص رمز المدخلات الحالي (فقرة المفردات الحالية)ولبناء هذا المحلل يجب ان تكون قواعد اللغة هي قواعد التطلع الى امام برمز واحد وحسب الشرط التالي:
- (ان يبدأ كل بديل في كل انتاج في قواعد اللغة برمز ختامي وكذلك ان تكون الرموز التي تبدأ البدائل المتنافسة مختلفة)

مثال /



- ان محلل التطلع الى امام برمز واحد لا يحتاج مطلقاً الى النهج المعاكس حيث يبدأ كل بديل برمز ختامي ومختلف ولهذا فان المحلل لا يحتاج الى تخمين ما يفعل وانما عليه اتباع البديل الذي تشير اليه فقرة المفردات الحالية.

فاذا فشل في التحليل في اي مرحلة فلن تكون هناك ضرورة للنهج المعاكس الى موضع سابق في التحليل حيث لا يوجد موضع في التحليل يمكنه اتباع مسار بديل . ولهذا فبمجرد فشله في تحليل السطر الحالي فانه يقوم باصدار تقرير أخطاء.

Introduction

Compiler :

أن مهمة المؤلف الأكثر وضوحاً هي قراءة برنامج في احدى اللغات (البرنامج المصدري) في اللغة المصدرية وترجمته لأنتاج برنامج مكافئ في لغة اخرى (البرنامج الهدف).
عادتاً تكون اللغة الهدف هي لغة الآلة (Machain Languag) للحاسب او لغة قريبة منها وتكون عادتاً البرنامج المصدري في لغة عليا مثل فورتران، باسكال، الجول 68، سمبول 67، BcPL.

لأن الترجمة من لغة عليا الى لغة الآلة هي المسألة الحالية التي وجدت المؤلفات لحلها، يمكن كتابة مؤلفات للترجمة من اية لغة برمجة الى اخرى ولكن بدرجات كفاءة مختلفة للبرنامج الهدف الناتج.
المهمة الأخرى للمؤلف هي ضرورة ان يتأكد المؤلف من ان البرنامج المصدري ذو مغزى ومعنى وعندما يبدوا انه ليس كذلك فعلى المؤلف انتاج وصف للمشكلة (تقرير بالأخطاء) بحيث يمكن للمبرمج من تصحيح البرنامج.

لغات البرمجة :

لكل لغة الجديدة خاصة، فمثلاً لغة باسكال
* الأعداد : 10، 100، 5.2، 10E-2
* المعرفات : المتغيران، اسماء الدوال والأجهزة والعلامات
* العوامل : الحسابية والمنطقية.
* الفواصل : ; و , و . و [ ] .... الخ

المجمع Assembler : الترجمة من لغة التجميع الى لغة الآلة.

المفسر Interpreter
المؤلف Compiler } : الترجمة من لغة عليا الى لغة دنيا (لغة الآلة).

عملية الترجمة

Source Program → Translater → برنامج قابل للتنفيذ على الآلة

عملية التفسير

Source Program → Interpreter → الناتج النهائي للبرنامج المصدري

عملية التأليف

Source Program → Compiler → Object Program

برنامج الهدف

* Object Program / هو برنامج غير قابل للتنفيذ (لاحتواءه على عناوين رمزية نسبية).

* المحمل / هو برنامج يقوم باكمال عمل المؤلف حيث يقبل البرنامج الهدف الناتج من عملية التأليف ويقوم بتحويل العناوين النسبيه في هذا البرنامج الى عناوين حقيقيه في الذاكره ليطبع برنامج قابل للتنفيذ.

الفرق بين المفسر والمؤلف

(1) من حيث طبيعة العمل /

المؤلف يقبل برنامج مصدري بلغه عليا وينتج برنامج هدف بلغه الآله ولكنه غير قابل للتنفيذ لأحتوائه على عناوين رمزيه نسبيه ولذلك يحتاج الى مرحله اخرى هي مرحله التحميل لتحويل العناوين الرمزيه الى عناوين حقيقيه في الذاكره.

المفسر يكون عمله متواصل وبمرحله واحده حيث يقوم بتحليل المعلومات وتشفيرها وتخصيص العناوين وتنفيذها واستخراج ناتج التنفيذ.

(2) الخزن /

المؤلف يحتاج الى خزن كبير لانه يقوم بخزن البرنامج المصدري والبرنامج الهدف والبرنامج القابل للتنفيذ وملفات وسطيه أخرى.

المفسر لا يحتاج الى لخزن البرنامج المصدري فقط.

(3) الوقت /

المؤلف يقوم بترجمة كل البرنامج مرة واحدة والتأكد من خلوه من الأخطاء ثم ينفذ بعد ذلك وهو يترجم كل تعليمة مرة واحدة بغض النظر عن عدد مرات تكرارها.

المفسر قد يتسبب المفسر في ضياع الوقت لأنه يقوم بتحليل وتشفير وتنفيذ التعليمات الواحدة بعد الأخرى وقد يصادف ان تعليمة ما في نهاية البرنامج خطأ فأن ما تقدم من عمل المفسر يعتبر ضياع في الوقت وكذلك في حالة التعليمات التكرارية (الدورات) فأن المفسر يقوم بتحليل التعليمة وتشفيرها وتنفيذها عند كل مرة تنفذ برامج (بعدد مرات تكرار الدورة).

انواع المؤلفات

(1) مؤلف وحيد التمرير — Single Pass Compiler

(2) مؤلف متعدد التمرير — Mult pass Compiler

(3) مؤلف التحميل والتنفيذ — Load And Go pass Compiler

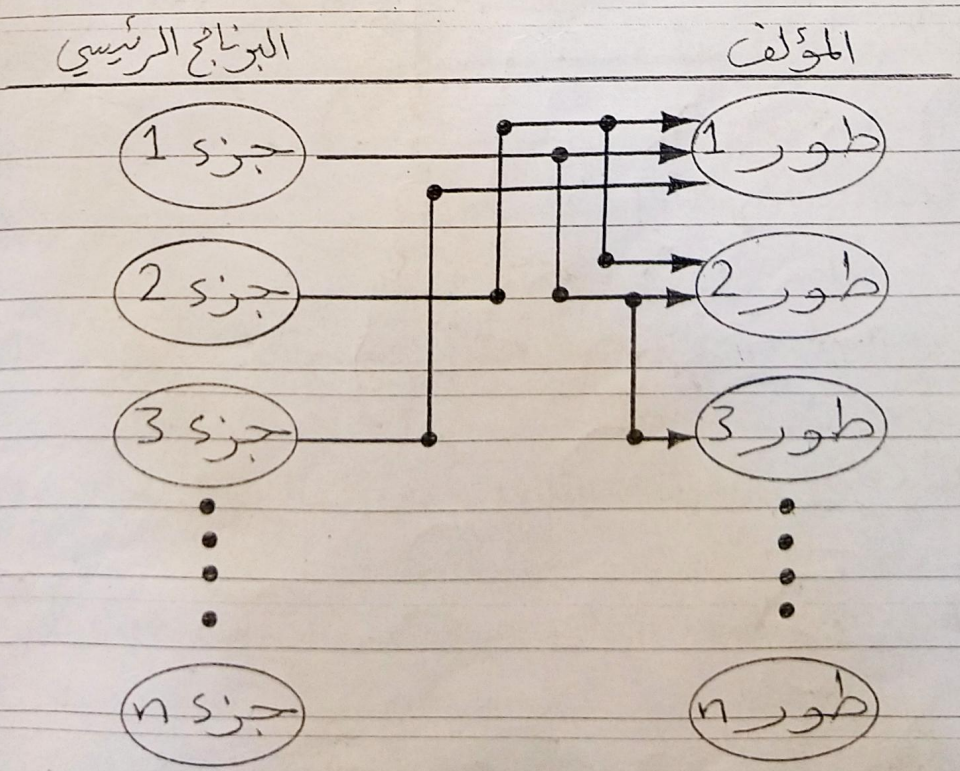
(4) مؤلف الأمثلية — Optimization Compiler

ان جميع المؤلفات تؤدي نفس الغرض (المهمة) ولكن الفرق يكون في كيفية ترتيب وربط مهمات المؤلف مع بعضها وبشكل عام المؤلف سيكون من عدة اطوار.

* الطور: هو عملية جزئية (مهمة من مهام المؤلف) يؤخذ الطور البرنامج المصدري بتمثيل معين وينتج هذا البرنامج بتمثيل أخر.

* المرير: هي طور او اكثر من اطوار المؤلف فعندما تدمج كل أطوار المؤلف مع بعضها سيمى مؤلف وحيد المرير اوان يكون كل طور مرير سيمى متعدد المريرات.

(1) مؤلف وحيد المرير Single Pass Compiler

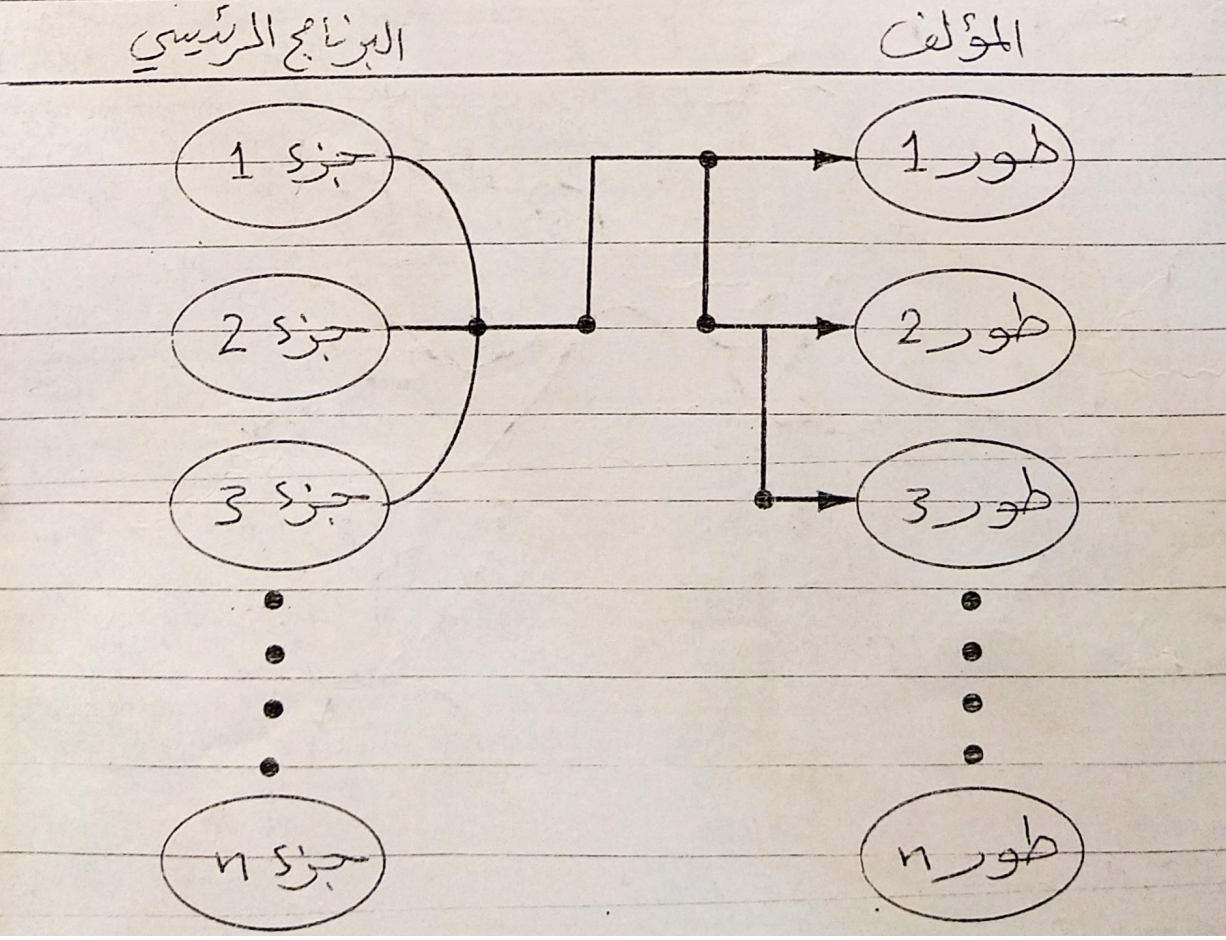


ان كل جزء يمر بكل الأطوار ، يدخل الجزء الأول الى الطور الأول وبعدها ينتقل الى الطور الثاني ثم ينتقل الجزء الثاني الى الطور الأول وبعد انتقال الجزء الأول الى الطور الثالث ينتقل الجزء الثاني الى الطور الثاني وفي نفس الوقت ينتقل الجزء الثالث الى الطور الأول وهكذا .

ملاحظة :

لا توجد ملفات وسطية بين الأطوار فقط خزن البرنامج المصدري والبرنامج الهدف .

٤ مؤلف متعدد المرورات Multi pass Compiler



ان كل البرنامج عبر بكل طور من أطوار المؤلف يحتاج الى خزن ملفات وسطيه لنتيجه كل طور التي هي مدخلات للطور الذي يليه.

عملية الترجمة

قبل طور الترجمة البسيط يعمل طور مواصفات المستهدفات حيث يقوم هذا الطور بأستخراج مواصفات كل الأسماء المستخدمة في البرنامج من خلال جزء الإعلانات في شجرة الإعراب ويضع مواصفات كل معرف في مدخل في جدول الرموز والمواصفات هي (الصنف، النوع، العنوان النسبي، عدد المعلمات بالنسبة للإجرائية ... الخ

* الصنف / يمكن ان يكون :

متغير بسيط ، Proc. ، Func. ، Array ، File ، Record

* النوع / يمكن ان يكون :

Integer ، Real ، Chr ، .... الخ .

* جدول الرموز .

الإسم	الصنف	النوع	العنوان النسبي	عدد المعلمات
main	Simple Var.	Chr	#1	—
x	Simple Var.	int	#2	—
y	Simple Var	Real	#3	—
Product	Proc.	—	#4	2

④ طور مواصفات المستهدفات

هي طور تحليل القواعد اللغوية الذي يتم فيه التعرف على اجزاء البرنامج ومنها جزء الاعلانات ويسبق طور الترجمة البسيطة لان عليه الترجمة تحتاج التعرف على انواع المعرفات ومواصفاتها قبل أضدار الشفرة.

ومواصفات المستهدفات يأخذ كمدخلات شجرة اعراب وجدول الرموز وينتج جدول رموز بمواصفات كاملة للمعرفات.

⑤ طور الترجمة البسيطة

بعد ان يكون طور مواصفات المستهدفات قد ملئ الواصفات في جدول الرموز يمكن لطور الترجمة البسيط أخذ شجرة التحليل مع جدول الرموز وانتاج تعاقب تعليمات.

فطور الترجمة البسيط سيخرج الشفرة المناسبة لكل تعليمة بأستخدام عناوين رمزية والترجمة البسيط لا تأخذ بالاعتبار كثيراً العلاقة بين اجزاء البرنامج بأستثناء ماهو مهم لانتاج شفرة صحيحة والتي قد تكون طويلة ومعقدة لكنها اقل كفاءة من الذي تنتجه اليه معينة تعطي بعض الاهتمام الى السياق الذي يجب ان يعمل فيه كل جزء.

من طور مواصفات المستهدفات ← #1 #1 #2

* لو اخذنا المثال التالي $X = X + Y$

```
Loadr 1, #1
Loadr 2, #2
Addn  1, 2
Store 1, #1
```

عناوين شبيهة رمزية

```
Addn 1, #2
```

الاستبدال ←

الأمثلية Optimization

تأخذ نظره اشمل للبرنامج وتعيد تنظيم الأجزاء او تحسين طريقة الاتصال بينها بحيث تنتج برنامج هدف اقصر او برنامج هدف اسرع مما قد تنتجه الترجمة البسيطة.

وهنالك طريقتان يمكن للمؤلف فيها انتاج ترجمة افضل من الترجمة البسيطة هي:

1) تحسين الشجرة (Optimize Tree):
اي ان المؤلف يستبدل اساساً خوارزمية البرنامج المصدر (الشجرة) بأخرى لها نفس التأثير ولكن يمكن ترجمتها الى شفرة اكثر كفاءة.

2) تحسين الشفرة (Optimize Code):
وهذا الطور يبحث اساساً عن الطرق التي تتصل بها اجزاء الشفرات بعضها ببعض وامكانية تقليص تلك الشفرات.

التحميل Loading

يأخذ المحمل شفره البرنامج الهدف الناتجه من طور الترجمه البسيط وينتج شفرات قابله للتنفيذ.

يقوم المحمل بمساعده نظام التشغيل (OS) في الحاسبه بإيجاد عنوان اول موقع ذاكره فارغ يبدأ تحميل الشفره منه.

فمثلاً لو كان اول خليه ذاكره فارغه تساوي (1000) اذن يكون البرنامج كالآتي:

عنوان نسبي ← Loadr 1, #1 ← عنوان حقيقي 1001 يستبدل

Loadr 2, #2 ← 1002 يستبدل

Addn 1, 2

Store 1, #1 ← 1001

فسوف تعتبر شفره قابله للتنفيذ لأحتوائها على عناوين حقيقيه.

محلل المسار الواحد
One Track Analyser

من غير الممكن دائماً انتاج قواعد التطابق الى امام حيث تعتبر اللغة في كثير من الحالات على الأختيار بين رموز غير خاصية في الإنتاج ولذلك فقواعد التطابق الى امام هي حالة خاصة من قواعد المسار الواحد والتي تسمح بالأختيار بين الرموز غير الخاصية عند بداية البدائل ولكنها ماز الت تحذف النهج المحاكس.

* أن شرط المسار الواحد :

في كل انتاج في قواعد اللغة يجب ان تكون قوائم (First*) للرموز التي تبداء البدائل المتنافسة منفصلة (ليس بينهما رموز مشتركة).

ملاحظة

أن (First*) هي نفسها (First+) مضافاً اليها الرمز نفسه (الذي تحسب له First*).

مثال// هل ان القواعد التاليه هي قواعد المسار الواحد برهن ذلك.

$S \longrightarrow E:=E \mid goto\ E \mid if\ E\ Then\ S\ else\ S1 \mid begin\ B$

$B \longrightarrow D;B \mid C$

$C \longrightarrow S;C \mid end$

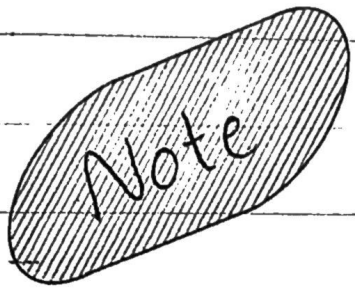
$D \longrightarrow let\ i=E$

$E \longrightarrow i \mid (E+E) \mid +E \mid -E$

الرمز	First * قاعدة
E	Ei(+-
S	SEi(+- goto if begin
D	Dlet
C	CSEi(+- goto if begin end
B	BDletCSEi(+- goto if begin end
goto	goto
if	if
begin	begin
end	end
let	let
i	i
(	(
+	+
-	-

أذن القواعد تحقق شرط المسار الواحد لعدم وجود رموز مشتركه بين البدائل.

* نلاحظ انه لا توجد رموز مشتركه بين البدائل المتنافسه في قوائم (First*) وهذا السبب تكون قواعد اللغه بالكامل ذات المسار الواحد.



ان محلل المسار الواحد لا ينهج نهجاً معاكساً مطلقاً فعندما يواجه المحلل الاختيار من بين بدائل في انتاج فأن فقره المفردات الحاليه ستكون في قائمه (First*) لأحد الرموز التي تبدأ البدائل المتنافسه او لا تكون في احد منها.
هذا فعندما يفشل التحليل قبل اكتمال العمل فلن يكون هناك سبب للنهج المعاكس لمحاوله تجريب أحد البدائل المتنافسه الأخرى (لأن قوائم First* منفصله) اي لا توجد نقطه يمكن للمحلل عندها اعاده اختياره مره ثانيه.