آریا ادیبی ۹۲۱۱۰۴۷۶
خشایار میرمحمدصادق ۹۲۱۰۰۹۸۶

گرامر

در زیر گرامر تغییر یافته‌ی زبان آمده است. برای هر قانون ابتدا شماره‌ی آن، سپس خود قانون و در خط بعدی مقادیر Tokenهایی که انتظار دیدن آن‌ها را داریم آمده است. در واقع این Tokenها همان‌هایی هستند که در LL(1) Table برای این متغیر مقدار دارند و بقیه error هستند. در همین پوشه فایل LL(1)-Table.xlsx این جدول را نشان داده است. در صورت تمایل برای دیدن First و Follow هر متغیر می‌توانید از کد زده شده استفاده کنید.

با بررسی گرامر جدید به راحتی می‌توان دید که همان زبان قبلی را تولید می‌کند. نکته‌های زیر برای فهم گرامر جدید مفید هستند:

- برای بهبود سرعت، آن Tokenهایی که رفتار یکسان در ساختار یاب داشتند را در یک گروه قرار دادم و ساختار یاب آن‌ها را به عنوان ۱ token می‌شناسد. برای مثال **basic_type** گروهی است که عضوهای آن **int, float, char, boolean** هستند. گروه‌های **reader** و **writer** نیز وجود دارند که عضوهای این ۲ مشخص است.
- قانون‌های $\langle expression \rangle \rightarrow \langle expr0 \rangle$، $\langle condOp0 \rangle \rightarrow \langle \textbf{||} \rangle$ و $\langle condOp1 \rangle \rightarrow \langle \textbf{\&\&} \rangle$ لزومی نداشتند و تنها برای خوانایی گذاشته شده‌اند. تأثیر این‌ها بر سرعت قابل توجه نیست.
- قسمت expression طوری طراحی شده که اولاً همان طور که خواسته شده بود تمامی عملگرها از چپ به راست اجرا شوند و دوماً اولویت آن‌ها نیز رعایت شود.

0	$\langle start \rangle$	$\rightarrow$	$\langle program \rangle$**EOF**

void basic_type EOF

1	$\langle program \rangle$	$\rightarrow$	**void id (**$\langle formalParameters \rangle$**)**$\langle block \rangle$ $\langle program \rangle$
2 3			| **basic_type id**$\langle rProgram \rangle$ | ϵ

void basic_type EOF

4	$\langle rProgram \rangle$	$\rightarrow$	$\langle dimDecleration \rangle \langle rVarList \rangle$**;**$\langle program \rangle$
5			| **(**$\langle formalParameters \rangle$**)**$\langle block \rangle \langle program \rangle$

[, ; (

6 7	$\langle formalParameteres \rangle$	$\rightarrow$	**basic_type id**$\langle rFormalParameters \rangle$ | ϵ

basic_type)

8 9	$\langle rFormalParameters \rangle$	$\rightarrow$	**,** **basic_type id**$\langle rFormalParameters \rangle$ | ϵ

) ,

10 11 $\langle rVarList \rangle \rightarrow$, **id**$\langle dimDeclaration \rangle$ | ϵ

; ,

12 13 $\langle dimDeclaration \rangle \rightarrow$ **[int_num]**$\langle dimDeclaration \rangle$ | ϵ

, ; [

14 $\langle block \rangle \rightarrow$ **{**$\langle blockContents \rangle$**}**

{

15 $\langle blockContents \rangle \rightarrow$ **basic_type id**$\langle dimDeclaration \rangle \langle rVarList \rangle$**;**$\langle blockContents \rangle$
16 17 | $\langle statement \rangle \langle blockContents \rangle$ | ϵ

basic_type) id if while for return break continue { reader writer

18 $\langle statement \rangle \rightarrow$ **id**$\langle assignmentOrMethodCall \rangle$
19 | **if (**$\langle expression \rangle$**)**$\langle block \rangle \langle optElse \rangle$
20 | **while(**$\langle expression \rangle$**)**$\langle block \rangle$
21 | **for(**$\langle assignment \rangle$**;**$\langle expression \rangle$**;**$\langle assignment \rangle$**)**$\langle block \rangle$
22 | **return**$\langle retExpr \rangle$**;**
23 | **break;**
24 | **continue;**
25 | $\langle block \rangle$
26 | **reader**$\langle location \rangle$**;**
27 | **writer**$\langle expression \rangle$**;**

id if while for return break continue { reader writer

28 $assignmentOrMethodCall \rightarrow \langle dimLocation \rangle$**=**$\langle expression \rangle$**;**
29 | **(**$\langle parameters \rangle textbf$**);**

([=

30 31 $\langle optElse \rangle \rightarrow$ **else**$\langle block \rangle$ | ϵ

else id if while for return break continue { reader writer }

32 33 $\langle dimLocation \rangle \rightarrow [\langle expression \rangle]\langle dimLocation \rangle$ | ϵ

[; = || && == != < <= => > + - * / % ,])

34 $\langle location \rangle \rightarrow$ **id**$\langle dimLocation \rangle$

id

35 $\langle assignment \rangle \rightarrow \langle location \rangle = \langle expression \rangle$

id

36 37 $\langle retExpr \rangle \rightarrow expression$ | ϵ

! - (id int_num real_num char_literal true false ;

38 $\langle expression \rangle \rightarrow \langle expr0 \rangle$

! - (id int_num real_num char_literal true false

39 $\langle expr0 \rangle \rightarrow \langle expr1 \rangle \langle rExpr0 \rangle$

! - (id int_num real_num char_literal true false

40 41 $\langle rExpr0 \rangle \rightarrow \langle condOp0 \rangle \langle expr1 \rangle \langle rExpr0 \rangle$ | ϵ

|| ;)] ,

42 $\langle expr1 \rangle \rightarrow \langle expr2 \rangle \langle rExpr1 \rangle$

! - (id int_num real_num char_literal true false

43 44 $\langle rExpr1\rangle \rightarrow \langle condOp1\rangle\langle expr2\rangle\langle rExpr1\rangle \mid \epsilon$

&& || ;)] ,

45 $\langle expr2\rangle \rightarrow \langle expr3\rangle\langle rExpr2\rangle$

! - (id int_num real_num char_literal true false

46 47 $\langle rExpr2\rangle \rightarrow \langle eqOp\rangle\langle expr3\rangle\langle rExpr2\rangle \mid \epsilon$

== != && || ;)] ,

48 $\langle expr3\rangle \rightarrow \langle expr4\rangle\langle rExpr3\rangle$

! - (id int_num real_num char_literal true false

49 50 $\langle rExpr3\rangle \rightarrow \langle relOp\rangle\langle expr4\rangle\langle rExpr3\rangle \mid \epsilon$

< <= >= > == != && || ;)] ,

51 $\langle expr4\rangle \rightarrow \langle expr5\rangle\langle rExpr4\rangle$

! - (id int_num real_num char_literal true false

52 53 $\langle rExpr4\rangle \rightarrow \langle arithOp0\rangle\langle expr5\rangle\langle rExpr4\rangle \mid \epsilon$

+ - < <= >= > == != && || ;)] ,

54 $\langle expr5\rangle \rightarrow \langle expr6\rangle\langle rExpr5\rangle$

! - (id int_num real_num char_literal true false

55 56 $\langle rExpr5\rangle \rightarrow \langle arithOp1\rangle\langle expr6\rangle\langle rExpr5\rangle \mid \epsilon$

* / % + - < <= >= > == != && || ;)] ,

57 58 $\langle expr6\rangle \rightarrow$ **!**$\langle expr6\rangle \mid \langle expr7\rangle$

! - (id int_num real_num char_literal true false

59 60 $\langle expr7\rangle \rightarrow$ **-**$\langle expr7\rangle \mid \langle expr8\rangle$

- (id int_num real_num char_literal true false

61 62 $\langle expr8\rangle \rightarrow \langle atomicExpr\rangle \mid$ **(**$\langle expr0\rangle$**)**

(id int_num real_num char_literal true false

63 *condOp*0 → ||

||

64 *condOp*1 → &&

&&

65 66 *eqOp* → == | !=

== !=

67 68 69 70 *relOp* → < | <= | >= | >

< <= >= >

71 72 *arithOp*0 → + | -

			+ -

73 74 75	$arithOp1$	→	* | / | %

	* / %

76 77 78 79 80 81	$\langle atomicExpr \rangle$	→	**id**$\langle locationOrMethodCall \rangle$ | **int_num** | **real_num** | **char_literal** | **true** | **false**

id int_num real_num char_literal true false

82 83	$\langle locationOrMethodCall \rangle$	→	$\langle dimLocation \rangle$ | ($\langle parameters \rangle$)

[(* / % + - < <= >= > == != && || ,] ;)

84 85	$\langle parameters \rangle$	→	$\langle expression \rangle \langle rParameters \rangle$ | ϵ

! - (id int_num real_num char_literal true false)

86 87	$\langle rParameters \rangle$	→	,$\langle expression \rangle \langle rParameters \rangle$ | ϵ

,)

برای آن گه نشان دهیم گرامری LL(1) است کافیست برای هر متغیر نشان دهیم که First قانون‌های آن دو به دو متمایز اند و اگر Firstای دارای ϵ بود این تمایز باید در قانون‌های دیگر و Follow متغیر نیز وجود داشته باشد. با توجه به گرامر و Tokenهایی که در پایین هر قانون ذکر شده است سخت نیست که از درستی این شرط اطمینان حاصل کنیم. از آن جایی که این شرط برقرار است می‌توان نتیجه گرفت که گرامر ابهام ندارد پس ابهام گرامر اولیه را نیز رفع کرده‌ایم.

برنامه‌ی نوشته شده

در این برنامه تمامی شرط‌های خواسته شده در پروژه رعایت شده است. به صورت موقت برنامه خروجی می‌دهد که قانون‌های انتخاب شده را انتخاب می‌کند. این برنامه طوری نوشته شده است که مستقل از داده ساختار LL(1) Table و سیاست error recovery وهمچنین گرامر زبان کار می‌کند و اگر مورد پسند بود می‌توان این ۳ را تغییر داد بدون این که نیازی به تغییر بقیه قسمت‌ها باشد.

Recovery Error

این برنامه شامل ۳ تابع getFirst()، getFollow() و getSynchronizingSet() است که یک متغیر به عنوان ورودی می‌گیرند و کار آن‌ها مشخص است. سیاست فعلی انتخاب شده ***"Panic mode Recovery"*** است. در این جا پیاده سازی به صورت مستقیم و با پشته بوده است. کاری که من در اینجا انجام می‌دهم از این قرار است. هنگامی که مشکل پیش می‌آید ۲ حالت دارد:

۱. *بالای پشته terminal ای قرار دارد و با Token ما فرق دارد:*

در این حالت آن قدر از ورودی می‌خوانیم تا این برابری حاصل شود. دقت کنید که اگر به انتهای فایل برسیم error recovery ما شکست خورده است.

۲. *بالای پشته variable ای قرار دارد و طبق Table LL(1) با Token خوانده شده به مشکل می‌خوریم:*

در این حالت به پشته دست نمی‌زنم و از ورودی آن مقدار می‌خوانم که برابر با یکی از Tokenهای در First این متغیر شود. دقت کنید که اگر به انتهای فایل برسیم error recovery ما شکست خورده است.

در آخر چند نکته‌ی نهایی قابل ذکر است. نکته‌ی اول این که این سیاست به کمک آن ۳ تابع ذکر شده در ابتدا در کمتر از حدود ۱۰ خط تقریباً به تمام ساست‌های معروف دیگر ساده تبدیل می‌شود؛ برای مثال یک سیاست این می‌تواند باشد که در حالت ۲ متغیر را از پشته حذف کنیم انگار که نبوده است و در ورودی تا اولین Token که در Follow متغیر وجود دارد را در نظر نگیریم و سپس ادامه دهیم. نکته‌ی دوم این که اجرای error recovery در هر مرحله دست صدا زننده‌ی ساختار یاب است از این رو اگر برای مشکلی نخواستیم recover کنیم به راحتی می‌توانیم. حتی می‌توان کلاً recovery نداشت.