توضیحات پیاده‌سازی فازهای پروژه درس کامپایلر

۱-۱- مفاهیم کامپایلری

قبل از این که به بررسی نرم‌افزار لکس و یَک بپردازیم، ابتدا تعدادی از مفاهیم کامپایلری را به طور اختصار مورد مطالعه قرار می‌دهیم.

الفبا: مجموعه‌ای متناهی از عناصر را الفبا می‌گویند. مانند الفبای زبان فارسی که از ۳۲ حرف تشکیل شده است. از خصوصیات الفبا، ساده و تجزیه‌ناپذیر بودن آن است.

رشته: به مجموعه‌ای متناهی از الفبا، رشته گفته می‌شود. مانند کلمات در زبان فارسی که ترکیبی از الفبای فارسی هستند. البته در زبان محاوره‌ای و یا نوشتاری روزمره و کتاب‌ها، رشته‌ها یا همان کلمات باید دارای معنا و مفهوم خاصی باشند که بیان‌کننده‌ی منظور شخص گوینده یا نویسنده است ولی در تعریف بالا، کلمات می‌توانند بدون معنا و یا حتی از نظر املایی غلط باشند؛ مانند اگل، ضصع، خاهر و... . در مثال‌های ذکرشده، دو مورد اول از نظر معنا غلط و مورد سوم از نظر املایی غلط هستند ولی با این حال، یک رشته به حساب می‌آیند.

مجموعه: مجموعه‌ای که در این‌جا موردنظر است، شبیه مجموعه‌ها در ریاضی است که از دو آکولاد باز و بسته "{}"، که بین این دو آکولاد عناصر الفبا قرار می‌گیرند، تشکیل شده است. مانند:

{۰، ۱، ۲، ۳، ۴، ۵، ۶، ۷، ۸، ۹}یا{ا ب پ ت}

همان‌طور که در ریاضیات بر روی مجموعه‌ها، عملیات اجتماع، اشتراک و ضرب[۱] تعریف می‌شود، برای این مجموعه‌ها نیز چنین اعمالی وجود دارد که شبیه عملیات مشابه در ریاضیات است.

مجموعه‌های باقاعده: مجموعه‌هایی را باقاعده گویند که دارای شرایط زیر باشد:

- مجموعه‌های تک عضوی و هم‌چنین مجموعه‌های تهی از دسته‌ی مجموعه‌های باقاعده هستند.

{}،{ب}

- اگر دو مجموعه به‌طور جداگانه هر یک مجموعه‌ای باقاعده باشند، در این صورت اجتماع، اشتراک و ضرب این دو مجموعه نیز مجموعه‌های باقاعده به شمار می‌روند.
- مجموعه‌ای باقاعده است که بتوان آن را از مجموعه‌های تک عضوی با استفاده‌ی مکرر از عملگرهای اتصال، اتحاد و ستاره تولید نمود.

عبارات باقاعده: همان مجموعه باقاعده است که فاقد علامت مجموعه "{}" باشد و تمام قوانین مربوط به مجموعه‌های باقاعده در مورد عبارات باقاعده صدق می‌کند.

زبان: مجموعه‌ی رشته‌هایی از الفبا است که از ساختار خاصی پیروی می‌کنند. برای تعریف یک زبان، مطلوب آن است که:

- الفبای آن تعریف شود.
- ابزاری جهت تعریف ساختار زبان تولید نمود.
- ابزاری جهت تولید رشته‌های زبان و جهت ارزیابی تعلق یک رشته به زبان تولید نمود. (رشته‌ای متعلق به یک زبان است که از ساختار آن زبان پیروی کند و از الفبای آن ساخته شده باشد.)

[۱] در بعضی از کتاب‌ها به عملیات ضرب مجموعه‌ها، اتصال یا الحاق مجموعه‌ها نیز گفته می‌شود.

گرامر: برای بیان ساختار یک زبان و تعریف عمومی و کلی یک زبان، از مفهومی به نام گرامر استفاده می‌شود.

برای درک بیشتر مفاهیم بالا، به مثال شکل ۱ توجه کنید:

پایانه = {کتاب، علی، شیشه، شکست، را}

ناپایانه = {<فاعل>,<فعل>,<مفعول>,<اسم>,<جمله>}

نماد شروع = <جمله>

قواعد : <جمله> → <فاعل><مفعول><را><فعل>

<فعل> → شکست

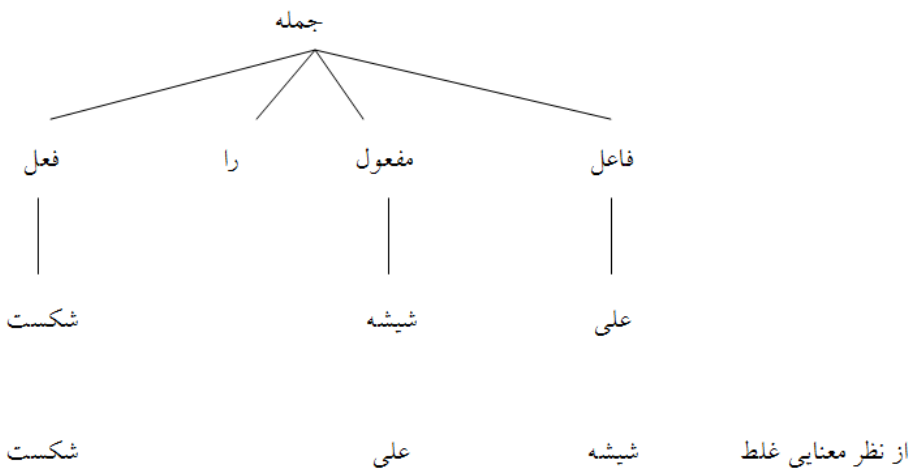
<فاعل> → <اسم>

<مفعول> → <اسم>

<اسم> → شیشه

<اسم> → کتاب

<اسم> → علی



شکل-۱ ساختار یک جمله در زبان فارسی

گرامر می‌تواند جملاتی را تولید نماید که ازلحاظ ساختار، کلیه‌ی مشخصات زبان را داشته باشد اما تضمینی بر تعلق رشته به زبان بر اساس معنای آن وجود ندارد. به عبارت دیگر، گرامر می‌تواند روی ساختار زبان صحبت کند ولی در مورد معنای آن نمی‌تواند بحث کند.

۱-۲- بررسی نرم‌افزار لکس و یَک

نرم‌افزار لکس و یَک ازجمله ابزارهایی هستند که برای تسهیل روند ساخت کامپایلرایجاد شده‌اند. این دو نرم‌افزار برای تحلیل لغوی و نحوی زبان‌ها به‌کار می‌روند. به این صورت که به ترتیب عبارات باقاعده و گرامر را از کاربر گرفته و زیربرنامه‌های تحلیل لغوی و نحوی را در اختیار قرار می‌دهند. لکس برای تهیه برنامه‌هایی که ورودی‌ها را تجزیه و تحلیل می‌کنند، به‌کار می‌رود. به عنوان مثال، اگر شما می‌خواهید یک برنامه‌ی ماشین‌حساب ساده بنویسید که وقتی عبارت ۸۵ + ۱۷۵ را به آن دادید، جواب محاسبه را به شما برگرداند، می‌توانید از لکس برای تشریح رشته حرفی ورودی استفاده کنید. عبارت بالا از سه کلمه "۱۷۵"، "+" و "۸۵" تشکیل شده است؛ بنابراین لکس، برنامه‌ای برای تحلیل ورودی و تبدیل کلمات به نشانه[۲] -که مشخص‌کننده نوع کلمه است- را تولید می‌کند.

مزیت استفاده از لکس در این است که می‌توان با دقت و کارآیی بیشتری، جزئیات فنی را که در تحلیل و ترجمه باید به‌کار روند، مورد استفاده قرار داد. کد تهیه‌شده توسط لکس، شناسایی داده‌های ورودی و مقایسه‌ی آن‌ها با چیزی که شما می‌خواهید را برآورده می‌کند. در یَک، قوانینی را که باید بر نشانه‌ها اعمال شوند، به صورت گرامر و به عنوان ورودی داده شده و ابزار یَک، به طور خودکار زیربرنامه‌ای تولید می‌کند که بتواند این قوانین را اعمال نماید. به بیان ساده‌تر، در ورودی یَک، طرز استفاده از نشانه‌ها مشخص می‌شوند.

این دو برنامه می‌توانند با یکدیگر و یا به طور جداگانه مورد استفاده قرار بگیرند. برنامه‌ی لکس می‌تواند وظیفه‌ی تحلیل لغوی ورودی را بر عهده داشته باشد و برنامه‌ای برای استفاده از این تحلیل نوشته شود، یا با نوشتن برنامه‌ای کار تحلیل لغوی انجام شود و سپس با استفاده از یَک، برنامه‌ای برای به‌کار بردن آن در اختیار قرار بگیرد. گرچه بهتر است این دو برنامه با یکدیگر مورد استفاده قرار بگیرند، زیرا در این صورت کارآیی برنامه بالاتر خواهد بود.

قابل ذکر است که کامپایلر از دو بخش تحلیل‌گر لغوی و تحلیل‌گر نحوی بهره می‌برد. از لکس برای نوشتن تحلیل‌گر لغوی و از یَک برای نوشتن تحلیل‌گر نحوی استفاده می‌شود.

۱-۳- خروجی‌های دو برنامه

این دو برنامه می‌توانند برنامه‌هایی به زبان‌های پاسکال، سی++[۳]، سی و جاوا تهیه کنند که در ساختن کامپایلرها استفاده می‌شوند. همچنین می‌توان به این برنامه‌ها که به صورت خودکار تولید شده‌اند، کدهای دیگری اضافه شود که نیاز خاصی موردنظر است. البته با ذکر این نکته که نباید کد اضافه‌شده، اختلالی در کدهای دیگر برنامه ایجاد کند.

خروجی لکس

هدف ابزار لکس، تولید کد برای زیربرنامه‌ای به نام yylex است که این زیربرنامه بدون آرگومان، و خروجی آن یک عدد صحیح است. اگر عدد صحیح خروجی صفر باشد، نشانه‌ی پایان پرونده، و در غیر این صورت، کدی که معرف یک نشانه مربوط به متن ورودی است، برگردانده می‌شود.

متغیرها و زیربرنامه‌های دیگر آن عبارتند از:

```
char   *ytext;
int    yyleng;
int    yylineno;
FILE   *yyin;
```

[۲] Token

[۳] C++

```
FILE   *yyout;
int    yylex(void);
void   yy_reset(void);
int    yygetc(void);
void   yyerror(char*, ...);
int    yywrap(void);
```

yytext:

متغیری که حروفی که یک نشانه را تشکیل می‌دهند، در خود نگه می‌دارد.

yyleng:

متغیری که طول نشانه را نگه می‌دارد.

yyin:

مکانی که نشانه‌ها باید از آن‌جا خوانده شوند را در خود نگه می‌دارد.

yyout:

مکانی که خروجی‌های لکس باید در آن قرار بگیرند.

yyreset():

این زیربرنامه برای مقداردهی اولیه به متغیرها به‌کار می‌رود. بعد از هر بار شناسایی، این زیربرنامه صدا زده می‌شود.

yygetc():

زیربرنامه‌ای که برای خواندن ورودی مورد استفاده قرار می‌گیرد.

yyerror():

زیربرنامه استاندارد پیغام خطا که به وسیله‌ی یَک نیز مورد استفاده قرار می‌گیرد.

yywrap:

این زیربرنامه زمانی فراخوانی می‌شود که در خواندن ورودی به انتهای پرونده رسیده شد.

yylval:

یک متغیر عمومی است که می‌توان از آن برای انجام کارهای مورد نظر استفاده کرد، برای نمونه، در مثال بالا می‌توان از آن برای قراردادنِ مقدار واقعی نشانه‌ی عددی که لکس پیدا می‌کند، استفاده کرد.

خروجی یَک

هدف یَک تولید کد برای زیربرنامه yyparse() است. این زیربرنامه از yylex() برای به‌دست‌آوردن نشانه استفاده می‌کند و همچنین در مواقعی که نیاز به مقدار واقعی نشانه‌ها داشته باشد، از yylval استفاده می‌کند.

در سی و پاسکال این زیربرنامه بدون هیچ آرگومانی صدا زده می‌شود. خروجی این زیربرنامه، اگر ورودی پردازش‌شده درست باشد، صفر است و مقدار آن اگر ورودی پردازش‌شده، نادرست باشد، یک است. لازم به ذکر است که می‌توان این مقادیر را تغییر داد.

۱-۴- تعریف نشانه‌ها

اگر به قسمت‌های قبل توجه کرده باشید، ()yylex کدی را کـه معـرف نشـانه‌ی پیـدا شـده اسـت را برمـی‌گردانـد و ()yyparse بر روی آن عمل می‌کند. بدیهی است که این دو زیربرنامه باید بـر مقـدارهای متفـاوت نشـانه‌هـا بـا یکـدیگر هماهنگ باشند.

نشانه‌ها را می‌توان در برنامه‌ی یَک به صورت زیر تعریف کرد:

```
%token      token_name
```

این نشانه‌ها در برنامه‌ی تولیدشده به زبان سی به صورت متغیرهای ثابت که با #define شروع می‌شوند، و در پاسکال با استفاده از CONST، نمایش داده می‌شوند. مقدار این نشانه‌ها از ۲۵۷ شروع مـی‌شـود. زیـرا از اعـداد ۱ تـا ۲۵۶ بـرای کدهای آسکی حرف‌ها استفاده می‌شود. وقتی یَک اجرا می‌شود، یک پرونده که شامل زیربرنامه تحلیل‌گـر نحـوی اسـت و همچنین یک سرآیند[۴] پرونده که شامل تعریف نشانه‌ها می‌باشد، ایجاد می‌کند.

جدول۱ - برنامه‌های تجزیه‌گر و سرآیند آن در زبان‌های مختلف

پرونده برنامه تجزیه‌گر	سرآیند پرونده	زبان
ytab.c	ytab.h	سی
ytab.cpp	ytab.hpp	سی++
ytab.pas	ytab.hp	پاسکال

برای مثال، در یک برنامه‌ی ماشین‌حساب ساده که ورودی‌های آن به‌صورت زیر هست:

789 + 45, 9045 - 234

دو نوع نشانه وجود دارد:

- عملوند عدد صحیح
- عملگر ریاضی

اگر yylex یک عملگر شبیه + یا – پیدا کند، یک کد مبنی بر پیداکردن آن برمـی‌گردانـد و اگـر یـک عملونـد عـدد صحیح پیدا کند، باید مقدار آن را در yylval قرار دهد و یک کد مبنی بر پیداکردن یک عـدد صـحیح برگردانـد. در ایـن صورت برای اطمینان از این‌که yylex و yyparse با یکدیگر در کد تعریف‌شده موافقت دارند، باید در یَک به صورت زیـر تعریف شوند:

```
%token      INTEGER_
```

لازم به ذکر است نمی‌توان از کلمات رزروشده در زبان‌های برنامه‌نویسی برای تعریف نشانه‌ها استفاده کرد. برای مثال، در پاسکال، کلمه‌ی رزروشده‌ی INTEGER وجود دارد؛ بنابراین در تعریف نشانه، علامت _ در آخر آن اضافه شده است.

اگر بخواهیم برنامه ماشین‌حساب را بسط داده و متغیرها را از اعداد تشخیص دهیم، می‌تـوانیم بـه‌صـورت زیـر عمـل کنیم:

[۴] Heading

```
%token    INTEGER_
%token    VARIABLE
```

۱-۵- ساختار برنامه لکس

ساختار کلی برنامه‌ی ورودی لکس به صورت زیر است:

```
definitions (تعاریف)
%%
translation (مترجم)
%%
functions (زیربرنامه‌ها)
```

قسمت‌های تعریف و زیربرنامه‌ها، قسمت‌های اختیاری هستند و می‌توانید آن‌ها را به‌کار نبرید. در قسمت زیربرنامه‌ها می‌توان کدهای موردنیاز خود را نوشت. این قسمت به انتهای پرونده لکسِ تولیدشده اضافه می‌شود. در قسمت مترجم، قوانین عباراتی که لکس باید آن‌ها را مقایسه کند، قرار می‌گیرد. در قسمت تعاریف، متغیرها و ساختارهایی قرار می‌گیرد که برای زبان موردنظر شما لازم است. برای مثال، مقدار دادن به yyin در پاسکال.

عبارت‌های حرفی: ساده‌ترین راه برای تشریح عبارت‌های حرفی، لیستی از حروف است که به‌وسیله‌ی "" مشخص و محصورشده باشند. مانند "علی" یا "حسن" یا "۱۲۳۴۵۶۷۸۹" که رشته‌های حرفی نامیده می‌شوند.

ابزار لکس، کدهای کنترلی داخل رشته را نیز تشخیص می‌دهد و می‌توان آن‌ها را به‌صورت زیر مشخص کرد:

\n	خط جدید
\b	یک فاصله به عقب[۵]
\t	هشت فاصله[۶]
…	**…**

این کدها در عبارت‌هایی که حرف‌های کنترلی و غیرقابل‌چاپ وجود دارند، می‌توانند مورد استفاده قرار بگیرند. همچنین می‌توان با استفاده از حرف‌های ^ و $، برای بیان شروع و پایان خط استفاده کرد. به عنوان مثال، "ke"^ نشان‌دهنده‌ی شروع یک عبارت باقاعده و $"abcdefg"^ نشان‌دهنده‌ی شروع و پایان یک خط است.

کلاس‌های حرفی: یک کلاس حرفی به وسیله‌ی تعدادی حرف که در داخل یک جفت براکت [] قرارگرفته‌اند، مشخص می‌شود. مانند [9-1]. این عبارت، بیان‌کننده‌ی کلاس حرفی اعداد است. همچنین برای حروف الفبا و چیزهای دیگر که نیاز است، می‌توان کلاس حرفی تشکیل داد؛ مانند کلاس حرفی حروف الفبای انگلیسی [A-Za-z][۷].

در کلاس‌های حرفی هر بار فقط یک حرف انتخاب می‌شود. یک عبارت باقاعده که با یک علامت * همراه باشد، می‌تواند صفر بار یا بیشتر تکرار شود.

---

[۵] Backspace

[۶] Tab

[۷] "-" به مفهوم تا است، یعنی هرچه حرف اسکی بین این دو حرف وجود دارند.

عبارت [0-9][0-9]* بیان‌کننده‌ی آن است که رشته‌ی مورد نظر، با یک عدد شروع‌شده و با تعدادی صفر یا بیشتر ادامه پیدا می‌کند. به‌عبارت‌دیگر عبارت بالا، الگویی از حرف‌ها برای بیان اعداد است. برای مثال، [A-Z][a-z]* بیان‌کننده‌ی رشته‌ای است که اول آن، حرف بزرگی است و به دنبال آن تعدادی صفر یا بیشتر حروف کوچک تکرار می‌شوند. بدین ترتیب، برای تعریف متغیرها در زبان سی باید به صورت زیر تعریف کرد:

[A-Za-z_][A-Za-z_0-9]*

این عبارت نشان‌دهنده‌ی یک حرف یا حرف _ است که به دنبال آن، حرف یا عدد یا حرف _ تکرار می‌شود.

حرف (عملگر) * نشان‌دهنده‌ی تکرار صفر یا بیشتر است و علامت + بیان‌گر تکرار تعداد یک و یا بیشتر است. برای مثال، [0-9]+ نشان‌دهنده‌ی تکرار یک یا بیشتر از حرف‌های عددی است که معادل با عبارت [0-9][0-9]* است. هم‌چنین، روش دیگری نیز وجود دارد که می‌توان تعداد تکرارها را نیز مشخص کرد. برای مثال عبارت [0-9]{3} بیانگر این است که عبارت موردنظر باید سه بار تکرار شود.

علاوه بر این، می‌توان برای عبارتی، یک محدوده‌ی تکرار قائل شد. این کار به‌صورت زیر انجام می‌گیرد:

[0-9]{1,9}

عبارت بالا نشان‌دهنده‌ی یک عدد یک تا نُه رقمی است. این روش را زمانی به‌کار می‌برند که استفاده‌کننده نمی‌خواهد عددش از محدوده‌ای بزرگ‌تر شود.

[A-Za-z_][A-Za-z_0-9]{0,31}

عبارت بالا نشان‌دهنده‌ی تعریف متغیرها در زبان سی است (۱ تا ۳۲ حرف).

عبارت‌های انتخابی: اگر در آخر یک عبارت باقاعده، یک علامت سؤال (؟) قرار بگیرد، نشان‌دهنده‌ی اختیاری بودن آن عبارت باقاعده است. برای مثال، A? نشان‌دهنده‌ی صفر یا یک اتفاق از A است؛ یعنی A می‌تواند باشد یا نباشد. اگر در بین دو عبارت باقاعده علامت | یا پایپ قرار داده شود، نشان‌دهنده‌ی این است که یکی از این عبارت‌های باقاعده می‌تواند به‌کار رود ولی هر دو به‌کار نمی‌روند[۸]؛ مانند [a-z]|[A-Z] که نشان‌دهنده‌ی حروف بزرگ یا کوچک است.

گروه‌بندی: از حرف () می‌توان برای گروه‌بندی عبارات باقی‌مانده استفاده کرد. برای مثال:

("high"|"medium"|"low")

که نشان‌دهنده‌ی هر سه رشته‌ی low، medium، high است.

توجه: علامت " عمل گروه‌بندی را انجام نمی‌دهد و یک اشتباهی که ممکن است رخ دهد، به صورت زیر است:

"is"?

این الگو نشان‌دهنده‌ی حرف i است که با s اختیاری دنبال می‌شود. برای این‌که کل رشته را اختیاری کنیم، باید از گروه زیر استفاده کنیم:

("is")?

تشریح قسمت‌های مختلف برنامه لکس

۱-۵-۱-۱- قسمت تعاریف:

این قسمت در ابتدای برنامه‌ی لکس است و برای خوانایی بیشتر برنامه‌ی لکس و همچنین تعریف متغیرهای موردنیاز به‌کار می‌رود. در این قسمت می‌توان به عبارات باقاعده یک نام اختصاص داد. آخرین خط این قسمت شامل %% است که علامت پایان قسمت تعاریف می‌باشد.

یک تعریف لکس شامل انتساب یک الگوی حرفی به یک نام است که قالب کلی آن به صورت زیر است:

[۸] مانند عملگر «یا» در مجموعه‌ها

```
name      regular_expression
```

regular_expression الگویی که نام قسمت name را به خود می‌گیرد، تشریح می‌کند. مانند:

```
digit       [0-9]
lower     [a-z]
upper     [A-Z]
```

این سه تعریف به الگوهای مختلف نام اختصاص می‌دهد.

یک تعریف در قسمت تعاریف می‌تواند از تعاریف دیگر با قرار دادن آن نام تعاریف در داخل {} دوباره استفاده کند. مانند:

```
letter      {lower}|{upper}
```

مثال قبل بیان می‌کند که letter شامل الگوی upper و lower که قبلاً تعریف‌شده‌اند، می‌باشد.

```
variable      {letter}({letter}|{digit})*
```

مثال بالا نیز نشان‌دهنده‌ی یک حرف است که به وسیله‌ی صفر یا بیشتر حرف یا رقم دنبال می‌شود.

۱-۵-۱-۲- مترجم:

بعد از ٪٪ که علامت پایان تعاریف است، قسمت مترجم شروع می‌شود. این قسمت توصیف‌کننده‌ی نشانه‌های ورودی و اعمالی که باید بر روی آن‌ها صورت گیرد، است. این قسمت با علامت ٪٪ پایان می‌پذیرد. قالب کلی این قسمت به صورت زیر است:

```
token-pattern    {action}
```

token-pattern یک عبارت باقاعده می‌باشد و ممکن است شامل تعاریفی باشد که پیش از این، در قسمت تعریف معرفی شده است.

action یک سری دستورات به زبان موردنظر است که باید بر روی الگوی موردنظر صورت گیرد و می‌تواند در چندین سطر باشد.

توجه: اگر شما دستورات پاسکال را در قسمت action قرار می‌دهید، بهتر است از (* و*) به‌جای {} استفاده کنید. برای مثال:

```
[\t\r]    (* ignored *)
```

مثال برای قسمت مترجم:

یک ماشین‌حساب ساده را در نظر بگیرید. قانون قسمت مترجم به‌صورت زیر خواهد بود:

```
[0-9]+ {
        yylval=atoi(yytext);
        return INTEGER_;
      }
```

عبارت باقاعده جمله‌ای از اعداد را مقایسه می‌کند و نشانه‌ی پیداشده در داخل yytext قرار می‌گیرد، شما باید مقـدار واقعی نشانه پیداشده را در yylval قرار دهید. بعد از این، تبدیل در قسمت action نوع نشانه را برمی‌گرداند.
برای مثال در پاسکال:

```
[-/*+]{
    getval := INTEGER (yytext[1]);
    goto yylexReturn; }
```

عبارت بالا شامل چهار حرف عملگر است. اگر yylex یکی از این عملگرها را پیدا کند، در قسمت action اولین حـرف موجود در yytext را برمی‌گرداند.
برای پاسکال، با استفاده از زیربرنامه ()INTEGER، به یک مقدار عددی تبدیل می‌شـود. در پاسـکال بـا اسـتفاده از goto yylexReturn از yylex خارج می‌شویم. اگر مقداری برای برگرداندن وجود داشته باشد، باید قبل از goto مقدار آن را در yylval قرار دهیم.

۱-۵-۱-۳- اظهارها[۹]

اظهارها می‌توانند با استفاده از ساختار

```
%{        %}
```

به قسمت تعاریف یا مترجم اضافه شوند.
اظهارها جهت تعریف متغیرها مورد استفاده واقع می‌شوند. در قسمت مترجم‌ها، ممکن اسـت اظهارهـا در ابتـدا قـرار بگیرند. در این صورت، متغیرهای تعریف‌شده در آن، به صورت محلی در زیربرنامه ()yylex به‌کار گرفته می‌شوند. مانند:

```
%%
%{
    int wordcount=0;
%}
[^|t|n] + {wordcount++}
[|t]+
[|n]{
    printf("%d|n",wordcount);
    wordcount=0;
}
```

این برنامه برای شمارش تعداد کلمات در هر خط به‌کار می‌رود. اگر ()yylex تعداد یک یا بیشتر از حرف‌هـایی را کـه حرف فاصله خالی، یا حرف هشت فاصله و یا حرف خط جدید نیستند را پیدا کند، به متغیر wordcount یک عدد اضـافه می‌کند و در جملاتی که حرف فاصله خالی یا حرف هشت فاصله وجود دارد، کاری انجام نمی‌دهد و وقتی به حـرف خـط جدید رسید، تعداد کلماتی که پیداکرده است را چاپ می‌کند و متغیر wordcount را صفر می‌کند.
اظهارها را در بخش تعاریف نیز می‌توان به‌کار برد. در این صورت، این متغیرها برای زیربرنامه ()yylex متغیر خـارجی به‌حساب می‌آیند. مانند:

[۹] Declarations

```
%{
   int characters = 0;
   int words = 0;
   int lines = 0;
%}
%%
\n  { ++lines;
    ++ characters;
     }
[\t]   characters+=yyleng;
[^\t\n]+{
       ++words;
characters+=yyleng;
         }
```

توجه: اگر سمت action تنها یک سطر باشد، می‌توان علامت {} را حذف کرد.

در مثال بالا ، بخش تعاریف فقط شامل اظهار است و بعد از ٪٪ مترجم می‌آید. در مترجم اول اگر \n را پیدا کند، به متغیرهای تعداد خط و متغیر تعداد حرف‌ها یک عدد اضافه می‌کند. در مترجم دوم، اگر فاصله یا \t پیدا کند، به متغیر تعداد حرف‌ها به‌اندازه‌ی yyleng اضافه می‌کند و در مترجم سوم، اگر دنباله‌ای از حرف‌ها را پیدا کند، به متغیر تعداد کلمه یک عدد و به متغیر تعداد حرف به‌اندازه‌ی yyleng اضافه می‌کند.

از این متغیرها می‌توان در زیربرنامه main() به‌صورت زیر استفاده کرد:

```
main()
      {
       extern int characters, words, lines;
      (void)yylex();
      printf ("%d characters, ", characters);
      printf(" %d words, ", words);
      printf(" % d lines \n ", lines);
       }
```

در این زیربرنامه ابتدا زیربرنامه yylex() برای تحلیل ورودی فراخوانی می‌شود. سپس جواب‌ها به زیربرنامه اصلی برگردانده می‌شوند و در زیربرنامه اصلی نتایج چاپ می‌شوند. با توجه به زبانی که لکس در آن کد تولید می‌نماید (به‌عنوان‌مثال زبان سی)، زبان لکس خاص آن زبان (سی) ایجاد شده است. برای آشنایی بیشتر با زبان‌های مختلف لکس، برنامه‌ای که عملیات ماشین‌حساب را انجام می‌دهد، به زبان‌های لکس خاص زبان‌های مختلف برنامه‌نویسی در ادامه آورده شده است.

۱-۶- برنامه لکس برای ماشین‌حساب

برای زبان سی (شکل ۷-۲)

```
%{
#include " ytab.h"
extern int yylval;
%}

 %%

[0-9]+{
      yylval = atoi(yytext);
      return INTEGER;
       }

[-+/*\n] return *yytext;

[\t]+;
```

شکل ۷-۲ برنامه ماشین‌حساب به زبان لکس مختص زبان سی

برای زبان سی++ (شکل ۷-۳)

```
%{
#include "ytab.hpp"
int yylval;
%}

 %%
[0-9] + {
       yylval = atoi((char *) yytext);
      return INTEGER;
      }

[-+/*\n] return * yytext;

[\t]+;
```

شکل ۷-۳ برنامه ماشین‌حساب به زبان لکس مختص زبان سی++

برای زبان پاسکال (شکل ۷-۴)

```
%{
unit dc1l;
interface

function yylex: integer;
procedure yysetup;
{$ I dc1.hp}
 var yylval: YYSTYPE;
 var code: integer;
 var valbuffer: string [40]

implementation

function yywrap: integer;
begin
 yywrap:=1
end;
%}

%%

[0-9]+{
        (* do not handle octal or hex constants (yet)*)
        valbuffer:=Copy(yytext,1,yyleng);
        val(valbuffer,yylval,code);
       retval:=CONSTANT;
      goto yylexReturn
       }

\n   {
      retval:=ord(^J);
     goto yylexReturn
      }
[-+/*]  {
        retval:=Integer(yytext[0]);
       goto yylexReturn
        }

[\t\r]+;
%%
procedure yysetup;
begin
```

```
  assign(yyin,");
  reset(yyin);
  assign(yyout,");
  rewrite(yyout)
end;
```

شکل ۷-۴ برنامه ماشین‌حساب به زبان لکس مختص زبان پاسکال

در زیربرنامه yysetup، متغیرهای yyin و yyout مقدار اولیه شده‌اند. در برنامه‌های بالا، زیربرنامه‌هایی برای تشخیص تمام نشانه‌هایی که در یک ماشین‌حساب وجود دارد، نوشته‌شده است. در ادامه، برنامه‌ی یَک برای این ماشین‌حساب نوشته شده است.

۱-۷- گرامر یَک

معمولاً برنامه‌ی یَک را گرامر می‌گویند. یَک برای تولید زیربرنامه تحلیل‌گر نحوی کامپایلرهای زبان‌های برنامه‌نویسی به وجود آمده است. در پرونده ورودی یَک ساختار این زبان‌ها تشریح می‌شود و یَک برنامه‌ی تحلیل‌گر نحوی موردنیاز را در اختیار کاربر قرار می‌دهد.

قالب کلی دستورات یَک به صورت زیر است:

```
DECLARATION اظهار
%%
RULES; قوانین
%%
FUNCTIONS زیربرنامه‌ها
```

تشریح قسمت‌های مختلف ساختار کلی یَک

۱-۷-۱-۱- قسمت اظهار

در این قسمت نشانه‌هایی که گرامر موردنظر را می‌سازند، تعریف می‌شوند. تعریف نشانه‌ها به‌صورت زیر است:

```
%token   name
```

name نام نشانه‌ها است. مانند:

```
%token  variable
```

مثال بالا تعریف نشانه یک متغیر در زبان می‌باشد.

پرونده تعریف نشانه‌ها:

خروجی یَک شامل دو پرونده است که در یک پرونده برنامه تحلیل‌گر نحوی است و یک سرآیند پرونده که شامل تعریف نشانه‌ها است. این سرآیند پرونده باید به برنامه‌ی لکس متصل شود تا لکس بتواند تعریف دقیقی از نشانه‌ها را

داشته باشد. برای انجام این کار، می‌توان به‌طور مستقیم از این سرآیند پرونده در برنامه‌ی لکس در قسمت تعاریف به صورت زیر استفاده کرد:

```
%{
#include "ytab.h"
%}
```

۱-۷-۱-۲- قوانین اولویت

در قسمت اظهار از یَک، می‌توانند قوانین اولویت و انجمن‌پذیری نیز قرار بگیرند. برای درک قوانین اولویت به این مثال توجه کنید. در قوانین ریاضی، ضرب و تقسیم از جمع و تفریق اولویت بیشتری دارند؛ مگر این‌که از پرانتز برای تغییر این اولویت استفاده شود. هم‌چنین، این دو نسبت به یکدیگر اولویت یکسانی دارند ولی هردوی آن‌ها از جمع و تفریق اولویت بیشتری دارند.

عبارات زیر را در زبان سی در نظر بگیرید:

```
A = 2 – 2 – 1;
A = B – 7 * 2;
```

برای محاسبه‌ی عبارت اول، محاسبه از چپ به راست انجام می‌شود:

```
A = ((2 – 2) – 1);
```

ولی در عبارت دوم، ابتدا عمل ضرب انجام می‌شود و سپس عملیات تفریق؛ یعنی محاسبه از راست به چپ انجام می‌شود:

```
A = (B - (7 * 2));
```

چون ضرب اولویت بیشتری نسبت به تفریق دارد، ضرب ابتدا انجام می‌شود و نتیجه از B کم می‌شود و سپس نتیجه به A اختصاص داده می‌شود. این کار به وسیله‌ی قوانین اولویت در یَک صورت می‌گیرد. عملیاتی که از راست به چپ انجام می‌گیرند را انجمن‌پذیری از راست و عملیاتی که از چپ به راست انجام می‌گیرند را انجمن‌پذیری از چپ می‌نامند. برای مثال عبارت a/b/c ممکن است به دو صورت مختلف پردازش شود.

اگر عملیات تقسیم، انجمن‌پذیری از راست داشته باشد، عملیات به‌صورت زیر انجام می‌شود:

```
a / (b / c)
```

و اگر عملیات تقسیم، انجمن پذیری از چپ داشته باشد، عملیات به‌صورت زیر انجام می‌شود:

```
(a / b )/ c
```

می‌توان انجمن‌پذیری و اولویت عملیات را در قسمت اظهار به‌صورت زیر مشخص کرد:

```
%left    operator
%right   operator
```

در یک خط می‌توانند چند عملگر بیایند که همه‌ی آن‌ها دارای اولویت یکسانی می‌شوند. مانند:

```
%left   '+', '-'
%left   '*', '/'
```

left و right به ترتیب نشان‌دهنده‌ی انجمن‌پذیری از چپ و انجمن‌پذیری از راست هستند. عملگرهایی که در یک سطر تعریف می‌شوند، از عملگرهایی که در سطرهای بالا تعریف می‌شوند، اولویت بیشتری دارند. این عبارت‌ها بیان‌کننده‌ی این مطلب هستند که * و / دارای اولویت بیشتری نسبت به + و – می‌باشند.

۱-۷-۱-۳- تعاریف زبان

در قسمت اظهـار از گرامر یَک، می‌توان تعاریف موردنظر خود را قرارداد. همان‌طور که در لکس این کار را با استفاده از {%}% انجام می‌دادیم.

۱-۷-۱-۴- قوانین

یک قانون، یک ساختار گرامری درست را تشریح می‌کند. این ساختار گرامری بـرای تشـخیص نشـانه‌هـای ورودی یـا ساختار گرامری و یا ترکیبی از آن دو به‌کار می‌رود.
برای درک این مطلب به قانون ماشین‌حساب توجه نمایید:

```
expression: INTEGER_
expression: expression '+' expression
expression: expression '-' expression
expression: expression '/' expression
expression: expression '*' expression;
```

این قوانین ساختار گرامری را تشریح می‌کنند که expression نامیده می‌شود. ساده‌ترین عبارت یـک نشـانه integer است که با _INTEGER شناسایی می‌شود. ممکن است عبارات پیچیده‌تری را با اضافه کردن جمـع و ضـرب و تقسـیم و تفریق ایجاد کنید. یک قانون مانند:

```
expression: expression'+' expression
```

دارای سه جزء است؛ یک expression و یک نشانه "+" و expression دیگر. اگر یک برنامه از گرامر قبلی برای تشـخیص عبارت ۳ + ۲ + ۱ استفاده کند، اول از همه، عدد ۱ وارد می‌شود که یک نشانه _INTEGER است. بنابراین می‌توانـد بـه یک expression تبدیل شود. درنتیجه ورودی به صورت زیر می‌شود:

```
expression + 2 + 3
```

البته ۲ نیز یک _INTEGER است. بنابراین یک expression است و به شکل زیر تبدیل می‌شود:

```
expression + expression + 3
```

برنامه، قسمت اول ورودی را به صورت عبارت درستی از expression تشخیص داده است. بنابراین به صورت زیـر تبـدیل می‌شود:

```
expression + 3
```

به روش مشابه عبارت بالا نیز به expression تبدیل می‌شود.

۱-۷-۱-۵- عمل[۱۰]

در قسمت قانون گرامر یَک، فقط ساختار گرامر تشریح نمی‌شود؛ بلکه بیان می‌شود وقتی ساختاری تشخیص داده شد چه‌کاری انجام شود. به‌عبارت‌دیگر، می‌توان قانون و عمل را با هم به کار برد. قالب کلی یک قانون به صورت زیر است:

```
name: definition{action};
```

name نام ساختار تعریف‌شده است.

definition تعریف ساختار گرامری با جملاتی از نشانه‌ها و یا ناپایانه‌ها است.

action شامل صفر یا بیشتر دستورالعمل است که وقتی ورودی به صورتی باشد که در definition تعریف‌شده است، انجام می‌شود.

در یَک متغیر $1 برای مقدار جزء اول، متغیر $2 برای مقدار جزء دوم، $3 برای مقدار جزء سوم و $$ برای نمایش مقدار ساختار تعریف‌شده به‌کار می‌رود. برای محاسبه جمع، جزء اول و جزء سوم با هم جمع می‌شوند.

```
expression : expression '+' experssion        { $$ = $1 + $3;};
```

برای محاسبه‌ی تمام عبارت می‌توان به‌صورت زیر آن را بیان کرد:

```
expression: expression '-' expression { $$ = $1 - $3; };
expression: expression '*' expression { $$ = $1 * $3; };
expression: expression '/' expression { $$ = $1 / $3; };
expression: INTEGER_              { $$ = $1; };
```

قانون آخر بیان‌کننده‌ی این مطلب است که اگر عبارتی دارای یک نشانه_INTEGER بود مقدار عبارت فقط مقدار نشانه می‌باشد.

اگر عمل مشخصی در قانون انجام نگیرد، عمل تعریف‌شده برابر با {$$ = $ 1;} خواهد بود. این جمله بیان‌گر این است که مقدار پیش‌فرض در یک ساختار، مقدار اولین جزء است.

مثال بالا برای زبان سی++ و سی بود. در زیر مثالی برای پاسکال آورده شده است:

```
expression: expression '-' expression { $$ = $1 - $3; };
expression: expression '*' expression { $$ = $1 * $3; };
expression: expression '/' expression {$$ = $1 div $3; };
expression: INTEGER_              { $$ = $1; };
```

۱-۷-۱-۶- فشرده‌سازی قوانین

اگر یک قانون دارای شکل‌های مختلفی در یک ساختار باشد، می‌توان آن را به شکل زیر ترکیب کرد:

```
name:
    definition1 {action}
  |definition2 {action}
```

[۱۰] Action

```
   /definion3 {action}
   /definition4 {action}
  / …
;
```

توجه کنید که نقطه‌ویرگول[۱۱] انتهای قانون را مشخص می‌کند و هر قانون دارای action مربوط به خود است. با این تعریف مثال بالا را می‌توان به صورت زیر نوشت:

```
expression:
        INTEGER_          { $$ = $1; }
      / expression'-' expression {$$ = $1 - $3; }
      / expression'+' expression{ $$ = $1 - $3; }
      / expression'*' expression { $$ = $1 * $3; }
      / expression'/' expression{ $$ = $1 / $3; }
```

۱-۷-۱-۷- نماد آغازگر[۱۲]

اولین ساختار گرامری تعریف‌شده در قسمت قوانین، باید شامل ساختار کلی باشد. برای مثال، اگر ورودی یَک، گرامر یک زبان باشد، قانون اول باید شامل یک برنامه‌ی کامل باشد. نام این قانون اول را نماد آغازگر می‌نامند. ()yyparse ورودی‌ها را در یک توصیف مناسب جمع‌آوری می‌کند. اگر گرامر یک زبان برنامه‌نویسی را تشریح کند و نماد آغازگر، یک برنامه کامل را توضیح دهد، ()yyparse بعد از انجام عملیات متوقف می‌شود.

نماد آغازگر باید طوری تعریف شود که همه‌ی ورودی‌های معتبر را شامل شود. در مثال ماشین‌حساب، باید یک برنامه با قانون زیر تعریف شود:

```
Program :
/*null*/
/ program expression '\n'
;
```

این دو قانون تعریف برنامه را بیان می‌کنند که می‌تواند شامل یک عبارت باشد که یک حرف خط جدید بعد از آن می‌آید (سطری که باید محاسبه شود)، و به دنبال آن، چنین خط‌های بیشتری می‌تواند بیاید یا اینکه می‌تواند شامل چیزی نباشد. تعریف «هیچی» در اولین قانون آمده است.

۱-۷-۱-۸- تعریف کد استفاده‌کننده[۱۳]

در قسمت قوانین می‌توان متغیرهایی که در داخل ()yyparse مورد استفاده قرار می‌گیرند را قرار داد. این کار به وسیله‌ی %{ }% صورت می‌گیرد. این متغیرها به‌صورت محلی در زیربرنامه yyparse مورد استفاده قرار می‌گیرند.

۱-۷-۱-۹- زیربرنامه‌ها:

این قسمت در گرامر یَک اختیاری است و اغلب استفاده نمی‌شود. در این قسمت زیربرنامه‌های کاربر قرار می‌گیرند که عیناً در انتهای زیربرنامه‌ی تولیدشده توسط یَک قرار می‌گیرد.

[۱۱] semicolon

[12] Start Symbol
[13] User code declaration

۱-۸- برنامه‌ی یَک برای ماشین‌حساب

همانند لکس، برای هر زبان برنامه‌نویسی متداول مانند سی، زبان یَکِ مختصِ آن زبان برنامه‌نویسی (سی) ایجاد شـده است. در ادامه، برنامه مربوط به یک ماشین‌حساب را در زبان‌های مختلف یَک بررسی می‌کنیم.

برنامه برای زبان سی (شکل ۷-۵)

```
%token INTEGER
%left '+' '-'
%left '*' '/'

%%

program:
  program expression '\n'    = {printf("%d\n",$2);}
|   /*NULL*/
;

expression :
     INTEGER
| expression '+' expression = { $$ = $1 + $3; }
| expression '-' expression = { $$ = $1 - $3; }
| expression '*' expression = { $$ = $1 * $3; }
| expression '/' expression ={ $$ = $1 / $3; }
;
```

شکل ۷-۵ برنامه ماشین‌حساب به زبان یَک مختص زبان سی

برنامه برای زبان سی++ (شکل ۷-۶)

```
%{
#include "dc11.hpp"
#include "dc1.hpp"
#ifdef_ZTC_
#include<stream.hpp>
#else
#include<iostream.h>
#endif
%}

%token INTEGER
%left'+''-'
```

```
%left'*''/'

%%


program:
       program expression '\n'     {cout << $2<<"\n";}
|       /*NULL*/
;

expression:
       INTEGER
|    expression '+' expression { $$ = $1 + $3; }
|    expression '-' expression { $$ = $1 - $3; }
|    expression '*' expression { $$ = $1 * $3; }
|    expression '\' expression { $$ = $1 / $3; }
;
%%
main()
{
yy_scan scan;
yy_parse parse;
return parse.yyparse(&scan);
}
```

شکل ۷-۶ برنامه ماشین‌حساب به زبان یَک مختص زبان سی++

برنامه برای زبان پاسکال (شکل ۷-۷)

```
%token CONSTANT
%left '+' '-'
%left '*' '/'
%{
program dc1;
uses dc1l;

procedure yyerror (msg: string);
begin
 writeln(msg);halt(1)
end;
%}

%%

program:
 program expression '\n' { writeln($2);}
```

```
|     /*NULL*/
;

expression:
        CONSTANT
|       expression '+' expression    { $$:= $1 + $3; }
|       expression '-' expression    { $$:= $1 - $3; }
|       expression '*' expression    { $$:= $1 * $3; }
|       expression '/' expression    { $$:= $1 div $3; }
;
%%
begin
 yysetup;
{$IFDEF YYDEBUG}
yydbug:= 1;
{$ENDIF}
 halt(yyparse)
end.
```

شکل ۷-۷ برنامه ماشین‌حساب به زبان یَک مختص زبان پاسکال