تعریف پروژه درس طراحی کامپایلر

دانشگاه صنعتی خواجه نصیرالدین طوسی - دانشکده ریاضی

ترم پاییز ۱۴۰۳

۱ گرامر زبان مورد نظر

پروژه درس طراحی کامپایلر شامل مرحله‌های تولید تحلیل‌گر لغوی، تحلیل‌گر نحوی، و کد میانی بر پایه گرامری است که در ادامه متن آورده شده و آن را $\mathcal{G}$ می‌نامیم. همه واژه‌هایی که به شکل پررنگ در گرامر $\mathcal{G}$ نوشته شده است واژه‌های کلیدی در زبان مورد نظر به شمار می‌آیند که از پیش نیز رزرو شده هستند. همچنین، زبان تولید شده توسط $\mathcal{G}$ حساس به متن است. از این روی، دو متغیر abc و aBc متفاوت با یکدیگر قلمداد می‌شوند. هر شناسه مجاز استفاده شده برای نام یک متغیر یا یک تابع، رشته‌ای است که با یک حرف لاتین (کوچک یا بزرگ) شروع می‌شود و در ادامه می‌تواند با حرف‌ها و رقم‌ها ادامه یابد. عددهای به کار رفته در برنامه‌های به دست آمده از $\mathcal{G}$ می‌توانند از گونه عدد صحیح و یا عدد حقیقی با علامت یا بدون علامت باشند. هر عدد حقیقی نیز در صورت داشتن ممیز دست کم باید یک رقم اعشاری در سمت چپ و راست ممیز داشته باشد. همچنین، اعداد حقیقی ممکن است دارای بخش توان با علامت یا بدون علامت نیز باشند. برای نمونه، عددهای 2.3E+16 ،284.098، 15.0-، و 73 عددهایی حقیقی یا صحیح معتبری به شمار می‌روند.

start	→	**program** id ; decList funcList block
decList	→	decs | decList decs
decs	→	type varList ; | ϵ
type	→	**integer** | **real** | **boolean**
varList	→	id | varList , id
funcList	→	funcList funcDec | ϵ
funcDec	→	**function** id parameters : type decList block
parameters	→	(decList)
block	→	**begin** stmtList **end**
stmtList	→	stmt | stmtList stmt
stmt	→	id := expr ;
	|	**if** expr **then** stmt
	|	**if** expr **then** stmt **else** stmt
	|	**while** expr **do** stmt
	|	**for** id:=expr **to** expr **do** stmt
	|	**return** expr ;
	|	block
expr	→	expr **and** expr | expr **or** expr
	|	expr $*$ expr | expr / expr
	|	expr + expr | expr − expr
	|	expr relop expr
	|	(expr)
	|	integerNumber | realNumber
	|	**true** | **false**
	|	id(actualparamlist)
	|	id
actualparamlist	→	expr | actualparamlist, expr | id | ϵ
relop	→	< | <= | = | <> | >= | >

۲ نمونه برنامه‌های تولید شده با گرامر داده شده

در این بخش دو برنامه برای نمونه آورده شده که با گرامر $\mathcal{G}$ قابل تولید هستند. برنامه نخست اول بودن عدد صحیح داده شده را بررسی می‌کند. برنامه دوم میانگین جمع عددهای صحیح بین (و شامل) دو عدد صحیح داده شده را به دست می‌آورد.

```
program prg1;
integer num, divisor, quotient;
begin
   num:=61;
   divisor:=2;
   quotient:=0;
   if num=1 then
      return false;
   else if num=2 then
      return true;
   while divisor<=(num/2) do
   begin
      quotient:=num/divisor;
      if divisor * quotient=num then
         return false;
      divisor:=divisor+1;
   end
   return true;
end
```

```
program prg2;

function avg(integer m; integer n;):real
integer sum, num;
real average;
begin
   sum:=0;
   average:=0;
   for num:=m to n do
      sum:=sum+num;
   average:=sum/(n-m+1);
   return average;
end

begin
   a:=avg(1,20);
end
```

۳ مرحله اول پروژه - تولید تحلیل‌گر لغوی

در این مرحله، با به‌کارگیری ابزار مناسب، تحلیل‌گر لغوی در یکی از زبان‌های Python، Java، C، و ... تولید خواهد شد. هر فایل ورودی به تحلیل‌گر لغوی برنامه‌ای است که با گرامر $\mathcal{G}$ قابل تولید است. تحلیل‌گر لغوی با خواندن فایل برنامه ورودی tokenهای برنامه را تشخیص داده و یک فایل در خروجی تولید می‌کند. توجه نمایید که خط نخست فایل خروجی باید بیانگر نام اعضای گروه و شماره دانشجویی آنها باشد. سپس، هر خط بعدی فایل خروجی، lexeme تشخیص داده شده به همراه token متناظر که به شکل دوتایی مرتب $< token\_name, token\_attribute >$ است را نشان خواهد داد. token_nameهای ممکن برای lexemeهای مشاهده شده توسط تحلیل‌گر لغوی در زیر آورده شده است.

lexeme	*token_name*	*lexeme*	*token_name*
program	PROGRAM_KW	:=	ASSIGN_OP
function	FUNCTION_KW	*	MUL_OP
begin	BEGIN_KW	/	DIV_OP
end	END_KW	+	ADD_OP
while	WHILE_KW	-	SUB_OP
do	DO_KW	<	LT_OP
for	FOR_KW	<=	LE_OP
to	TO_KW	<>	NE_OP
if	IF_KW	=	EQ_OP
then	THEN_KW	>=	GE_OP
else	LESE_KW	>	GT_OP
integer	INTEGER_KW	:	COLON
real	REAL_KW	;	SEMICOLON
boolean	BOOLEAN_KW	,	COMMA
return	RETURN_KW	(	RIGHT_PA
and	AND_KW	)	LEFT_PA
or	OR_KW		
true	TRUE_KW		
false	FALSE_KW		
id	IDENTIFIER		
integerNumber	INTEGER_NUMBER		
realNumber	REAL_NUMBER		

در این مرحله از انجام پروژه، منظور از $token\_attribute$ شماره ردیفی از جدول نمادها یا همان symbol table است که اطلاعات تکمیلی در مورد token دیده شده در آنجا نگهداری می‌شود. اگر چنین اطلاعاتی مورد نیاز نباشد، از خط تیره برای مؤلفه دوم دوتایی مرتب استفاده می‌نماییم. بدیهی است که اگر تحلیل‌گر لغوی با متغیری برخورد کند که پیش‌تر در جدول نمادها قرار داده شده است، آن متغیر را مجدداً در جدول نمادها قرار نمی‌دهد و آدرس پیشین آن را به عنوان $token\_attribute$ در نظر می‌گیرد.

همه فایل‌های مربوط به پروژه باید به شکل فایل فشرده‌ای با نام $SN - CompilerPhase1.tar$ آماده گردد که در این نام‌گذاری به جای SN شماره‌های دانشجویی اعضای تیم که با خط تیره از یکدیگر جدا شده‌اند قرار خواهد گرفت. اکنون، برای نمونه، بخشی از فایل خروجی تحلیل‌گر لغوی برای چند خط نخست برنامه اول در ادامه آورده شده است.

Lexeme	*Token*
program	<PROGRAM_KW, ->
prg1	<IDENTIFIER, 1>
;	<SEMICOLON, ->
integer	<INTEGER_KW, ->
num	<IDENTIFIER, 2>
,	<COMMA, ->
divisor	<IDENTIFIER, 3>
,	<COMMA, ->
quotient	<IDENTIFIER, 4>
;	<SEMICOLON, ->
begin	<BEGIN_KW, ->
num	<IDENTIFIER, 2>
:=	<ASSIGN_OP, ->
61	<INTEGER_NUMBER, 5>
;	<SEMICOLON, ->
divisor	<IDENTIFIER, 3>
:=	<ASSIGN_OP, ->
2	<INTEGER_NUMBER, 6>
;	<SEMICOLON, ->
quotient	<IDENTIFIER, 4>
:=	<ASSIGN_OP, ->
0	<INTEGER_NUMBER, 7>
;	<SEMICOLON, ->