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 برنامه‌ها می‌توانند صرفاً از گونه عدد صحیح بدون علامت و عدد حقیقی بدون علامت باشند. هر عدد حقیقی نیز در صورت داشتن ممیز باید در هر دو سمت ممیز دست کم یک رقم داشته باشد. همچنین، اعداد حقیقی ممکن است دارای بخش توان با علامت یا بدون علامت نیز باشند. بخش توان برای اعداد حقیقی با نماد حرف بزرگ E نمایش داده خواهد شد. عددهای صحیح نیز نمی‌توانند صفر زاید در سمت چپ داشته باشند. با توجه به آنچه گفته شد، عددهای 2.3E+16، 284.098، 15.0 و 73 عددهای حقیقی یا صحیح معتبری به شمار می‌روند، اما عددهای .23 و 05 نامعتبر هستند.

start	→	**program** id ; decList funcList block
decList	→	decs | decs decList
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. به منظور وجود یکپارچگی در پروژه‌های انجام شده توسط گروه‌های مختلف، لازم است از token_nameهای زیر برای lexemeهای مشاهده شده توسط تحلیل‌گر لغوی استفاده گردد.

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	ELSE_KW	>	GT_OP
integer	INTEGER_KW	:	COLON
real	REAL_KW	;	SEMICOLON
boolean	BOOLEAN_KW	,	COMMA
return	RETURN_KW	(	LEFT_PA
and	AND_KW	)	RIGHT_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. لازم به ذکر است که صرفاً برای tokenهایی که بیان‌گر شناسه یا عدد هستند ردیفی در جدول نمادها درنظر گرفته می‌شود. برای سایر tokenها، از خط تیره برای مؤلفه دوم دوتایی مرتب استفاده می‌نماییم. همچنین، اگر تحلیل‌گر لغوی با lexeme ای برخورد کند که پیش‌تر در جدول نمادها قرار داده شده است، آن را مجدداً در

جدول نمادها قرار نمی‌دهد و آدرس پیشین آن (ردیف متناظر در جدول) را به عنوان $token\_attribute$ در نظر می‌گیرد.

همه فایل‌های مربوط به پروژه باید به شکل فولدر 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. این فولدر فشرده شده دست کم شامل دو فایل خواهد بود: فایل نخست فایلی با نام input.txt است که شامل یک برنامه نمونه با استفاده از گرامر $\mathcal{G}$ است. فایل دوم با نام lexer.py همان برنامه تحلیل‌گر لغوی خواهد بود. با اجرای فایل lexer.py به طور خودکار فایل ورودی input.txt پردازش شده و یک فایل خروجی با نام output.txt تولید خواهد شد.

اکنون، بخشی از فایل output.txt که توسط تحلیل‌گر لغوی برای چند خط نخست برنامه اول تولید شده در ادامه آورده شده است.

Students' names and surnames and their IDs

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, ->

شایان ذکر است که برنامه‌های ورودی که به عنوان test case استفاده خواهند شد، برنامه‌هایی خواهند بود که ممکن است حاوی عددهای صحیح دارای صفر زاید باشند. در این صورت، باید تحلیل‌گر لغوی طراحی شده عبارت Illegal Lexeme را برای آن عدد (در فایل output.txt) چاپ نماید و سپس به پردازش باقی فایل ورودی بپردازد. سایر موارد موجود در برنامه‌ها منطبق بر گرامر $\mathcal{G}$ خواهد بود و به جز مورد ذکر شده در مورد عددها، بحث error handling در این مرحله مطرح نیست.

۴ مرحله دوم پروژه - تولید تحلیل‌گر نحوی

در این مرحله، با استفاده از یک تولید کننده تحلیل‌گر نحوی خودکار مناسب (همانند YACC)، برای گرامر $\mathcal{G}$، یک تحلیل‌گر نحوی بسازید. ورودی تحلیل‌گر نحوی تولید شده یک فایل خواهد بود که برنامه‌ای قابل تولید از گرامر $\mathcal{G}$ در آن نوشته شده است. خروجی تحلیل‌گر نحوی نیز فایلی خواهد بود که خط نخست آن بیانگر نام اعضای گروه و شماره دانشجویی آنها بوده و سپس، در هر خط بعدی، یکی از قانون‌های گرامر $\mathcal{G}$ به همراه شماره قانون نوشته شده نمایش داده می‌شود. به این ترتیب، فایل خروجی گام‌های عمل Parsing را یک به یک نشان خواهد داد. شیوه شماره‌گذاری قانون‌ها دلخواه است اما باید سازگار باشد. به این مفهوم که اگر یک قانون در جاهای مختلفی از فایل خروجی استفاده شده باشد، باید شماره‌های متناظر با آن قانون نیز یکسان باشند. برای نمونه، فایل خروجی تحلیل‌گر نحوی برای برخی از خط‌های برنامه اول می‌تواند به شکل زیر باشد.

Students' names and surnames and their IDs

6	type → **integer**
...	...
...	...
...	...
4	decs → type varList ;
...	...
...	...
...	...
1	start → **program** id ; decList funcList block

همه فایل‌های مربوط به پروژه باید به شکل فولدر فشرده‌ای با نام $SN-CompilerPhase2.tar$ آماده گردد که در این نام‌گذاری به جای SN شماره‌های دانشجویی اعضای تیم که با خط تیره از یکدیگر جدا شده‌اند قرار خواهد گرفت. این فولدر فشرده شده دست کم شامل دو فایل خواهد بود: فایل نخست فایلی با نام input.txt است که شامل یک برنامه نمونه با استفاده از گرامر $\mathcal{G}$ است. فایل دوم با نام parser.py همان برنامه تحلیل‌گر نحوی خواهد بود. با اجرای فایل parser.py به طور خودکار فایل ورودی input.txt پردازش شده و یک فایل خروجی با نام output.txt تولید خواهد شد. ممکن است برای انجام تحلیل نحوی، نیاز به فایل‌های دیگری اعم از فایل lexer.py که مربوط به فاز اول پروژه بود باشد. همچنین، ممکن است، نیاز به انجام برخی از تغییرات در lexer.py نیز وجود داشته باشد. از این روی، چنین فایل‌هایی نیز باید در فولدر فشرده شده قرار داده شوند.

به منظور انجام تحلیل نحوی درست، لازم است تا از دستورات مناسب برای بیان شرکت‌پذیری و اولویت عملگرهای ریاضی و منطقی گرامر $\mathcal{G}$ استفاده گردد. همچنین، اگر تولید کننده تحلیل‌گر نحوی به‌کار رفته به طور خودکار مشکل Dangling-else را حل نکرد، با استفاده از راه‌کار گفته شده (در کتاب مرجع و کلاس درس) قانون‌های مربوطه را نیز تغییر دهید.

۵ مرحله سوم پروژه - تولید کد میانی

انجام این مرحله از پروژه اختیاری بوده و حداکثر یک نمره اضافی خواهد داشت و هدف از آن، آشنایی با شیوه تولید کد میانی در قالب کدهای سه آدرسه برای برنامه‌های داده شده است. به منظور تسهیل انجام این مرحله، فایل‌های ورودی که به منظور ارزیابی استفاده خواهند شد، فاقد هر گونه تعریف تابع با استفاده از کلمه کلیدی **function** خواهند بود. همچنین، فایل‌های ورودی فاقد حلقه‌های ایجاد شده توسط کلمه کلیدی **for** خواهند بود. ممکن است در فرآیند پیاده‌سازی این مرحله، از quadruple برای نگهداری کدهای سه آدرسه در حافظه استفاده شود، اما در نهایت، باید فایل خروجی صرفاً به شکل کدهای سه آدرسه باشد.

همه فایل‌های پروژه باید به شکل فولدری فشرده با نام $SN-CompilerPhase3.tar$ آماده گردد که در این نام‌گذاری به جای SN شماره‌های دانشجویی اعضای تیم که با خط تیره از یکدیگر جدا شده‌اند قرار خواهد گرفت. این فولدر فشرده شده دست کم شامل دو فایل خواهد بود: فایل نخست فایلی با نام input.txt است که شامل یک برنامه نمونه با استفاده از گرامر $\mathcal{G}$ است. فایل دوم با نام codeGenerator.py همان برنامه تولید کننده کدهای سه آدرسه خواهد بود. با اجرای فایل codeGenerator.py به طور خودکار فایل ورودی input.txt پردازش شده و کدهای سه آدرسه متناظر در یک فایل خروجی با نام output.txt تولید خواهد شد.