Retro Basic Compiler Report

A Part of Programming Language Principle Project

By Pisit Wajanasara 5931042721 Section 33

ก่อนที่เราจะไปเริ่มทำการ Implement Compiler เราจะต้องมีการตรวจสอบก่อนว่า Grammar ที่จะนำไปใช้นั้น สามารถนำไปสร้าง Parser โดย Algorithm ที่ต้องการได้หรือไม่ เนื่องจาก Algorithm ที่เลือกใช้ในการทำ Parser ใน Project นี้คือ Algorithm LL(1) ซึ่งจะต้องทำให้ Grammar ไม่มี Left Recursion และ Left Factor อยู่ใน Grammar ของภาษา

จากการตรวจสอบ Grammar ที่ได้รับมาพบว่ามี Left Factor อยู่ใน Production Rule 2 ข้อคือ "exp := term + term | term – term | term” และ "cond := term < term | term = term” ทำให้เราต้องมีการแก้ไข Left Factor ที่อยู่ใน Grammar ภาษาก่อนที่จะเราจะไปเริ่มต้นสร้าง Compiler และ ได้ผลลัพธ์จากการแก้ไข Grammar ดังนี้

Grammar (LL(1) Parsable)

1. pgm := line pgm
2. pgm := EOF
3. line := line_num stmt
4. stmt := asgmnt
5. stmt := if
6. stmt := print
7. stmt := goto
8. stmt := stop
9. asgmnt := id = exp
10. exp := term exp’
11. exp’ := + term
12. exp’ := - term
13. exp’ := empty
14. term := id
15. term := const
16. if := IF cond line_num
17. cond := term cond’
18. cond’ := < term

19. cond’ := = term
20. print := PRINT id
21. goto := GOTO line_num
22. stop := STOP

หลังจากนั้น เราจะนำ Grammar นี้มาหา First Set และ Follow Set เพื่อนำไปสร้าง Parsing Table

Non-terminal	**First Set**	**Follow Set**
pgm	line_num, EOF	EOF
line	line_num	line_num, EOF
stmt	id, IF, PRINT, GOTO, STOP	line_num, EOF
asgmnt	id	line_num, EOF
exp	id, const	line_num, EOF
exp’	+, -, empty	line_num, EOF
term	id, const	+, -, line_num, EOF
if	IF	line_num, EOF
cond	id, const	line_num
cond’	<, =	line_num
print	PRINT	line_num, EOF
goto	GOTO	line_num, EOF
stop	STOP	line_num, EOF

เมื่อสามารถหา First Set และ Follow Set ได้แล้วสามารถสร้าง Parsing Table ได้ตามที่แสดงด้านล่าง

Parsing Table

	line_num	id	IF	PRINT	GOTO	STOP	const	<	Equal	+	-	EOF
pgm	1											2
line	3											
stmt		4	5	6	7	8						
asgmnt		9										
exp		10					10					
exp'	13									11	12	13
term		14					15					
if			16									
cond		17					17					
cond'								18	19			
print				20								
goto					21							
stop						22						

Compiler Constructing Part

1. Scanner

   ในการสร้าง Scanner นั้น เนื่องจาก Syntax ของตัวภาษาสามารถแยกแต่ละ Token ด้วย whitespace ได้เลย เราจึงสามารถที่จะ Implement Scanner ด้วยการอ่านข้อมูลเข้ามาทั้งหมดและแยกด้วย Whitespace ได้

2. Parser

   ในการสร้าง Parser นั้น Algorithm LL(1) ได้ถูกใช้ในการสร้าง Compiler ใน Project นี้ โดย Parser มีกระบวนการทำงานดังนี้

   1. เราจะทำการสร้าง stack ที่ข้างในประกอบไปด้วย pgm และ EOF ตามลำดับจาก top ของ stack
   2. Parser จะถูกวนเข้าไปพิจารณา token จาก scanner แต่ละตัว โดยที่เมื่อเข้าไปแล้วจะดูว่าในขณะที่ top ของ stack และ token ที่พิจารณาอยู่ไม่ใช่ token ที่สามารถ match กันได้ เราก็จะ pop stack ออกมาดู และ ดูว่าเราควรจะ Derive ด้วยกฎข้อใดตาม Parsing Table และ push กฎที่ derive ลงไปใน stack
   3. เมื่อ top ของ stack และ token match กันแล้ว เราจะทำการเก็บ top ของ stack และ token ไว้ด้วยกัน เพื่อจะนำไปแปลงเป็น bcode ต่อไป พร้อมทั้ง pop stack ออก และ เราก็จะไปพิจารณา token ตัวต่อไป และทำแบบเดิมซ้ำไปเรื่อยๆ จน match ครบทุก token
   4. ถ้าทำจนครบ token ทุกตัว และ stack จบที่ EOF แสดงว่าการ Compiler ผ่าน และ source code ที่นำมา compiler ถูกตามกฎของภาษา ถ้าไม่จบตามนี้แสดงว่ามี Syntax Error อยู่ใน Source code
   5. เราจะทำการแปลง bcode โดยนำค่าที่เก็บไว้ดังที่อธิบายในข้อ 3 ไป map เป็น bcode ของภาษา ตามหลักการของการแปลง bcode และ output ออกมาเป็นไฟล์ผลลัพธ์

Actual Compiler

เนื่องจากมีการ Implement Compiler จริงๆ จึงขอใช้ Code จริงๆในการแสดงการทำงานของ Compiler โดย compiler ถูก implement ด้วย ภาษา javascript และ ต้องทำงานบน interpreter Node.js version 8.6 ขึ้นไป ซึ่งสามารถเข้าไปดู Code ได้ที่ link ด้านล่าง ซึ่งจะมีอธิบายการใช้งานตัว Compiler สำหรับการใช้ compiler source code ภาษา Retro Basic อยู่ข้างในด้วย โดยหลักการทำงานของ compiler เป็นไปตามขั้นตอนที่ได้อธิบายไว้ด้านบนแล้ว

compiler project on Github: https://github.com/nekoteoj/RetroBasicCompiler