Compiler

ปวินท์ เปี่ยมไทย

5931037621

phase 1: language analysis

ทำการวิเคราะห์ grammar ที่ได้รับ และแก้ไขปรับปรุงให้ทำ LL(1) parser ได้ง่ายขึ้น

โดยมีการแก้ left factor ในกฎของ exp และ cond ได้ grammar ใหม่ดังนี้

pgm := line pgm | EOF

line := line\_num stmt

stmt := asgmnt | if | print | goto | stop

asgmnt := id = exp

exp := term exp'

exp' := + term | - term | EMPTY

term := id | const

if := IF cond line\_num

cond := term cond'

cond' := < term | = term

print := PRINT id

goto := GOTO line\_num

stop := STOP

ต่อมาทำการหา first set, follow set และ parsing table ได้ผลดังนี้

first set:

pgm = {line\_num, EOF}

line = {line\_num}

stmt = {id, IF, PRINT, GOTO, STOP}

asgmnt = {id}

exp = {id, const}

exp' = {+, -, EMPTY}

term = {id, const}

if = {IF}

cond = {id, const}

cond' = {<, =}

print = {PRINT}

goto = {GOTO}

stop = {STOP}

follow set:

pgm = {EOF}

line = {line\_num, EOF}

stmt = {line\_num, EOF}

asgmnt = {line\_num, EOF}

exp = {line\_num, EOF}

exp' = {line\_num, EOF}

term = {+, -, line\_num, EOF}

term = {+, -, line\_num, EOF}

if = {line\_num, EOF}

cond = {line\_num}

cond' = {line\_num}

print = {line\_num, EOF}

goto = {line\_num, EOF}

stop = {line\_num, EOF}

rules set:

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

Parsing table

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Line\_num | Id | Const | IF | PRINT | GOTO | STOP | + | - | < | = | 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 |  |  |  |  |  |

phase 2: lexical scanner

เนื่องจากกระผมใช้ภาษา C++ ในการ implement

จึงเลือกใช้โครงสร้าง string stream ของภาษาเป็นตัวช่วยในการแยก token

โดยทำการสร้าง class scanner ขึ้นมาใช้ช่วยในการดึงข้อมูลจากไฟล์

ขั้นแรกจะโหลดไฟล์เข้ามาในโปรแกรมและนำเนื้อหาในไฟล์ทั้งหมดไปเก็บไว้ใน string (temp)

จากนั้นทำการตัด white space ที่บริเวณท้ายไฟล์ออกทั้งหมด สุดท้ายจึงนำข้อมูลใน string ดังกล่าวใส่ลงใน string stream (ss)

เมื่อเวลาต้องการ token ตัวต่อไปจะเรียก method getNext() ของ scanner ซึ่งตัว method ดังกล่าวจะใช้ extraction operator (>>) ดึงข้อมูลออกมาจาก string stream ได้เป็น token หนึ่งตัวเก็บไว้ใน attribute token ของ scanner เพื่อให้ฟังชั่นอื่นมาเรียกใช้

phase 3: parser

3.1 preparation

ทำการสร้างโครงสร้างที่จำเป็นต่อการ parse อันได้แก่ parsing table และ rules set

โดย parsing table นั้นจะใช้ array ขนาด 13\*12 ซึ่งเท่ากับ parsing table ด้านบน

entry ที่ row\_i, column\_j จะหมายถึงหมายเลขกฎที่จะใช้หากมี terminal (column\_j) เข้ามาและตัวบนสุดของ stack ปัจจุบันคือ non terminal(row\_i) โดยเลข 0 คือไม่มีกฎใดรองรับ (หาก parser มาถึงช่องนี้แสดงว่า syntax ผิด)

โดยมี map สองตัวคือ terminalMap และ nonTerminalMap ทำการแปลงตัว terminal และ non terminal จากข้อความให้กลายเป็นตัวเลขที่สามารถมาใช้ index parsing table ข้างต้นได้

และสร้าง vector ซ้อน vector ไว้เก็บกฎต่างๆ ชื่อ rules

โดย vector แต่ละอันจะใช้เก็บกฎหนึ่งกฎ

3.2 parser implementation

ทำการสร้าง stack หนึ่งตัวไว้เก็บ token ที่ยังต้องพิจารณา

เราจะสร้าง LL(1) parser โดยมีหลักการทำงานโดยสังเขปดังนี้

1. ทำการ push EOF และ pgm ลง stack

2. หากยังไม่ถึง end of file ให้ทำการลอง match tokenปัจจุบัน กับ token บนสุดของ stack ดู

3. หาก match ได้ แสดงว่าทั้งคู่เป็น terminal แล้ว จะทำการ pop terminal ตัวดังกล่าวออกจาก stack และเรียกฟังชั่นเพื่อตรวจสอบว่า terminal ดังกล่าวมี BCode อย่างไร พร้อมทั้งเขียนลงไฟล์

4. หาก match ไม่ได้ จะทำการตรวจสอบ token ปัจจุบัน และตัวบนสุดของ stack ว่าใช้ production rule อันไหนแปลงได้บ้าง (จากฟังชั่น applyRule) เมื่อพบแล้วก็จะ pop ตัวบนสุด stack ออก และ push push ข้อมูลใหม่เข้าไป

5. ทำข้อ 2 - 4 ไปเ รื่อยๆ จนหมดไฟล์ ก็จะได้ BCode ของโปรแกรมออกมา

Full implementation

Full implementation ของ retro basic parser สามารถดาวโหลดได้จากลิงค์ Github ข้างล่างนี้:

https://github.com/pawin35/retro-basic-ll-1--parser