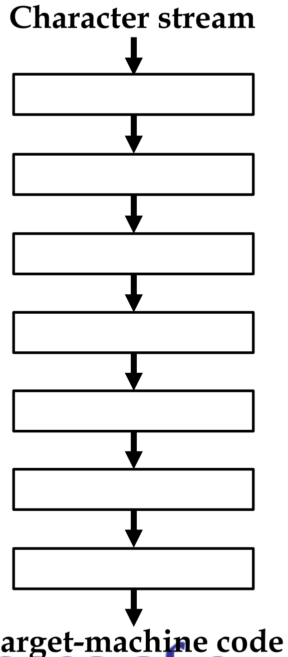
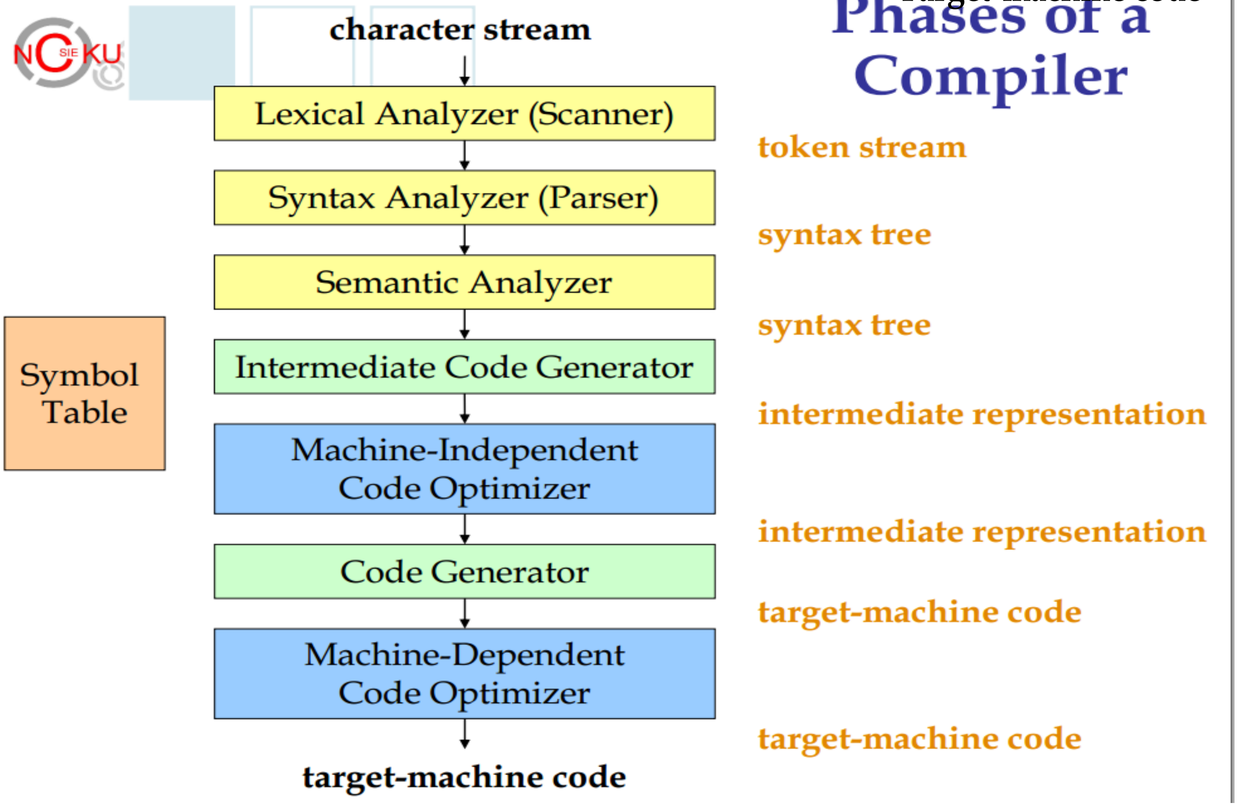
1. Please answer the following questions. (10pt)

a) Fill the missing phases of a compiler with the given items. (7pt)

b) Please mark the *frontend* and *backend* of the compiler on the right figure. (3pt)

I. Semantic Analyzer
II. Syntax Analyzer (Parser)
III. Lexical Analyzer (Scanner)
IV. Machine-Independent Code Optimizer
V. Intermediate Code Generator
VI. Machine-Dependent Code Optimizer
VII. Code Generator





Phases of a Compiler

character stream

Lexical Analyzer (Scanner) — token stream

Syntax Analyzer (Parser) — syntax tree

Semantic Analyzer — syntax tree

Intermediate Code Generator — intermediate representation

Machine-Independent Code Optimizer — intermediate representation

Code Generator — target-machine code

Machine-Dependent Code Optimizer — target-machine code

target-machine code

Symbol Table

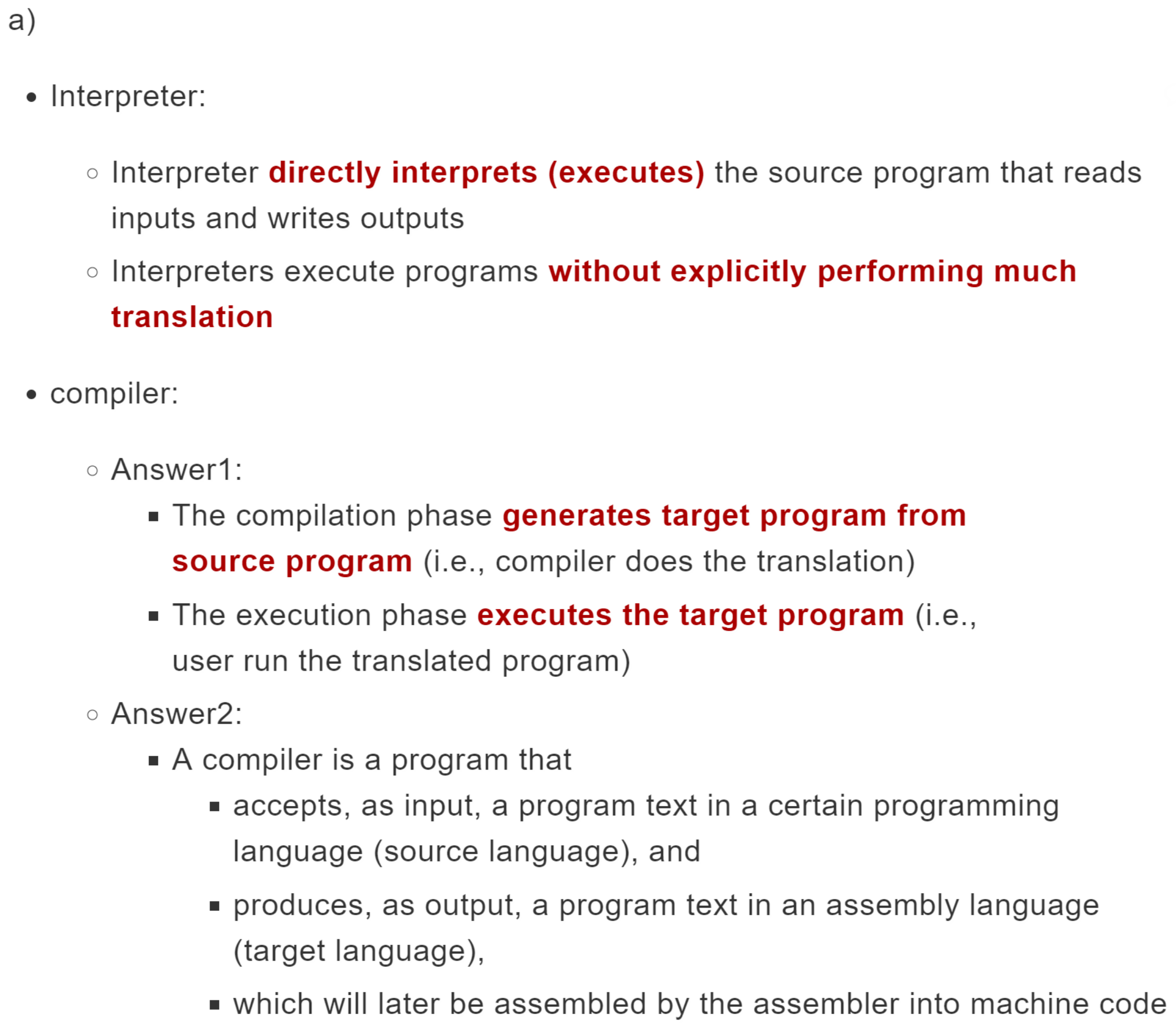
Front-end：semantic analyzer 之前（包括）

Back-end：Intermediate code generator 之後（包括），兩個部分依靠 AST 相連。

2. Please answer the following question.

a) What is a compiler? (2pt)

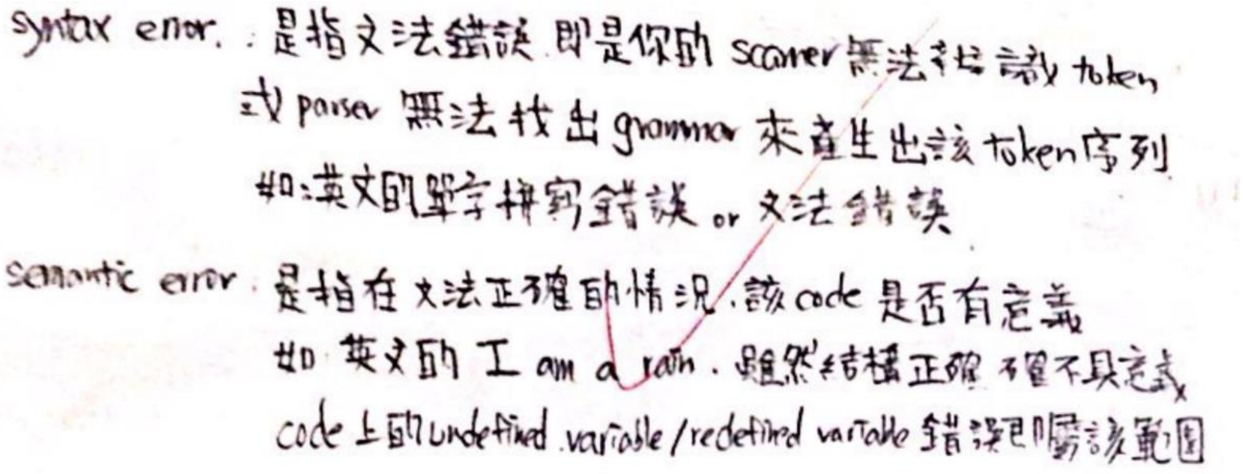
b) What is an interpreter? (2pt)



a)

- Interpreter:
  - Interpreter **directly interprets (executes)** the source program that reads inputs and writes outputs
  - Interpreters execute programs **without explicitly performing much translation**
- compiler:
  - Answer1:
    - The compilation phase **generates target program from source program** (i.e., compiler does the translation)
    - The execution phase **executes the target program** (i.e., user run the translated program)
  - Answer2:
    - A compiler is a program that
      - accepts, as input, a program text in a certain programming language (source language), and
      - produces, as output, a program text in an assembly language (target language),
      - which will later be assembled by the assembler into machine code

c) c) What are semantic error and syntax error? (2pt)



Syntax error：是指文法錯誤，即是你的 scanner 無法辨識 token 或 parser 無法找出 grammar 來產生出該 token 序列
如：英文的單字拼寫錯誤 or 文法錯誤

semantic error：是指在文法正確的情況，該 code 是否有定義
如：英文的 I am a rain. 雖然結構正確 確不具定義
code 上的 undefined variable / redefined variable 錯誤已屬該範圍

d) Describe, in one sentence, the strings captured by the following regular expression

$(ab)^+c^*$

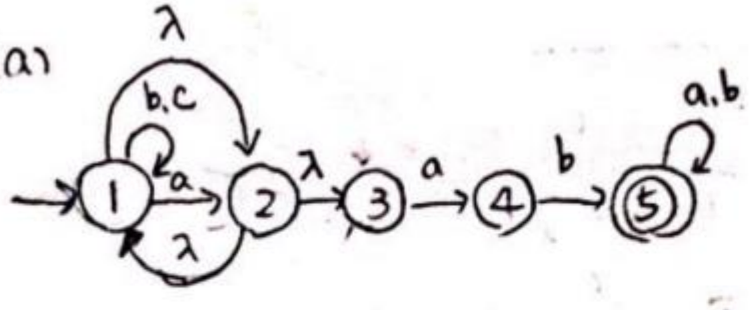
One or more instances of “ab” followed by zero or more “c”s

3. Please complete the following tasks with the given regular expression.

Regular expression: ((b|c)*a)*ab(a|b)*

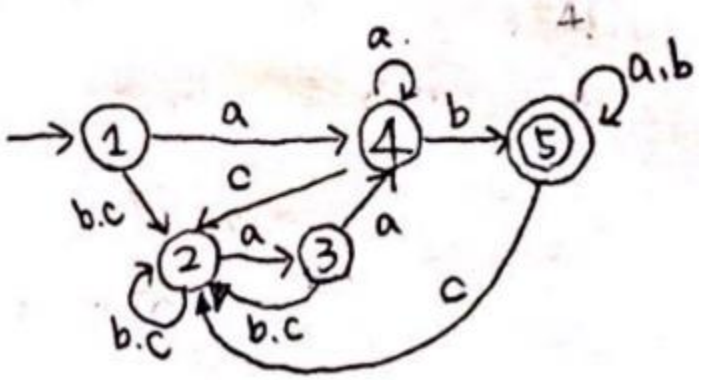
a) Transform the regular expression into the corresponding Non-deterministic Finite Automata (NFA). (15pt)

(a)



b) Convert the NFA in a) to the Deterministic Finite Automata (DFA). (15pt)

(b)



c) Write down the Transition Table for the DFA in b). (10pt)

(c)

e state	a	b	c
1	4	2	2
2	3	2	2
3	4.	2	2.
4	4	5	2.
5	5	5	2.

4. Let G be the grammar: $S \rightarrow AB$

$A \rightarrow xB \mid \lambda$

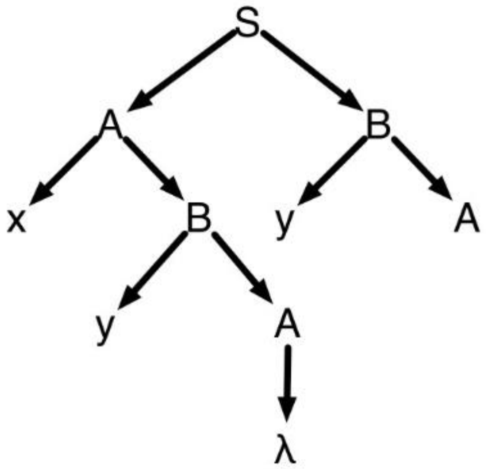
$B \rightarrow yA \mid zB$

a) What are the terminals and non-terminals of this grammar?

Terminals: x, y, z.

Non-terminals: S, A, B

b) Draw the parse tree for the following partial derivation (i.e., some of the leaves of your parse tree may be non-terminals). $S \Rightarrow xyyA$



c) Did this partial derivation get produced by left-derivation or right-derivation? (1 points)

Left derivation

5. Please help design the ac (adding calculator) language.

a) Filling out (A) to (G) in Table 1 uses regular expressions to define the ac tokens, i.e., assign, plus, minus, fnum (floating point numbers), and id (identifiers, which could be any alphabetic character except the reserved character for floatdcl, intdcl and print). (5pt)

A "=", B "+", C "–", D "*", E "/",
F [0-9]+"."[0-9]+ , G [a-e]|[g-h]|[j-o]|[q-z]

b) Based on the *ac* grammar α, please draw the parse tree for the input below. (10pt) (Please use the *leftmost derivation* method to derive the rules.)

c) Compare the *ac* grammars α and β, and describe the major difference between them. (5pt)

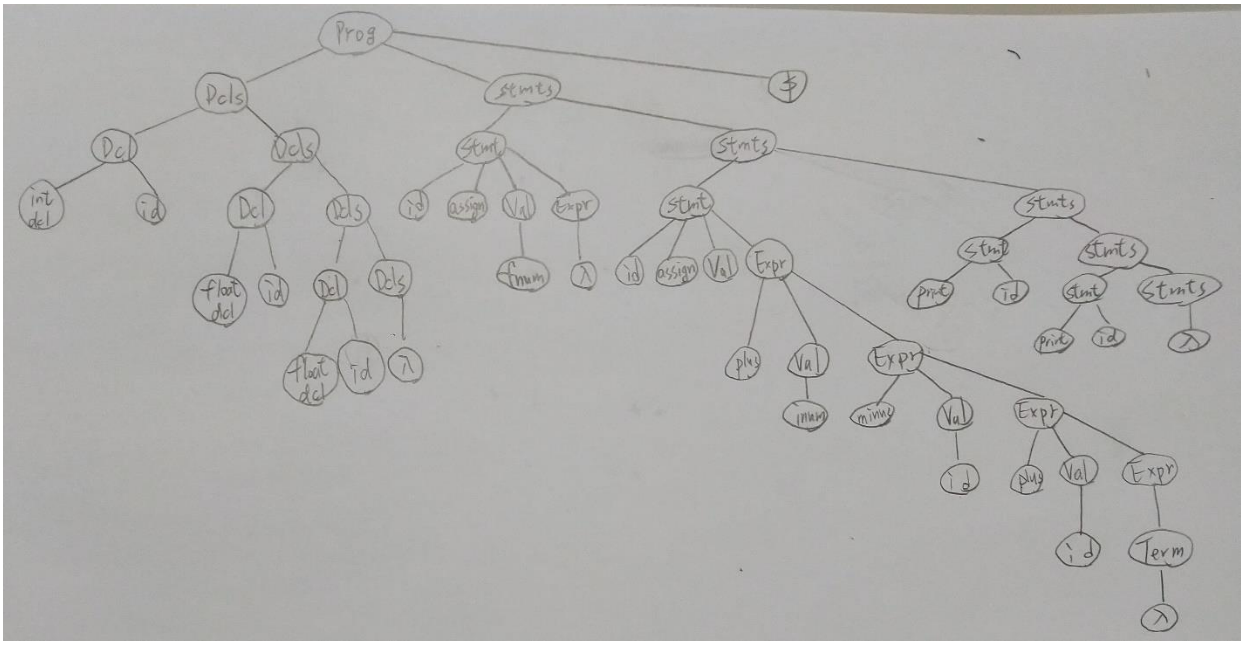
Input: i b f a f c a=2.5 b=3 c=b+2-a+b p b p c

Terminal	Regular Expression
floatdcl	"f"
intdcl	"i"
print	"p"
assign	**(A)**
plus	**(B)**
minus	**(C)**
multiply	**(D)**
div	**(E)**
inum	[0 - 9]+
fnum	**(F)**
blank	(" ")+
id	**(G)**

Table 1: Definition of *ac* tokens.

Context-free grammar α for *ac*.

```
 1| Prog   →  Dcls Stmts $
 2| Dcls   →  Dcl Dcls
 3|        |  λ
 4| Dcl    →  floatdcl id
 5|        |  intdcl id
 6| Stmts  →  Stmt Stmts
 7|        |  λ
 8| Stmt   →  id assign Val Expr
 9|        |  print id
10| Expr   →  Term
11|        |  plus Val Expr
12|        |  minus Val Expr
13| Term   →  Val
14|        |  multiply Val Expr
15|        |  div Val Expr
16|        |  λ
17| Val    →  id
18|        |  inum
19|        |  fnum
```



c)

The **precedence** of the plus, minus, multiply, and div is different between the two grammars.

In α, the precedence of multiply and div are the same and is higher than that of plus and minus.

In β, the precedence of +, -, * and / is identical.

Context-free grammar β for ***ac.***

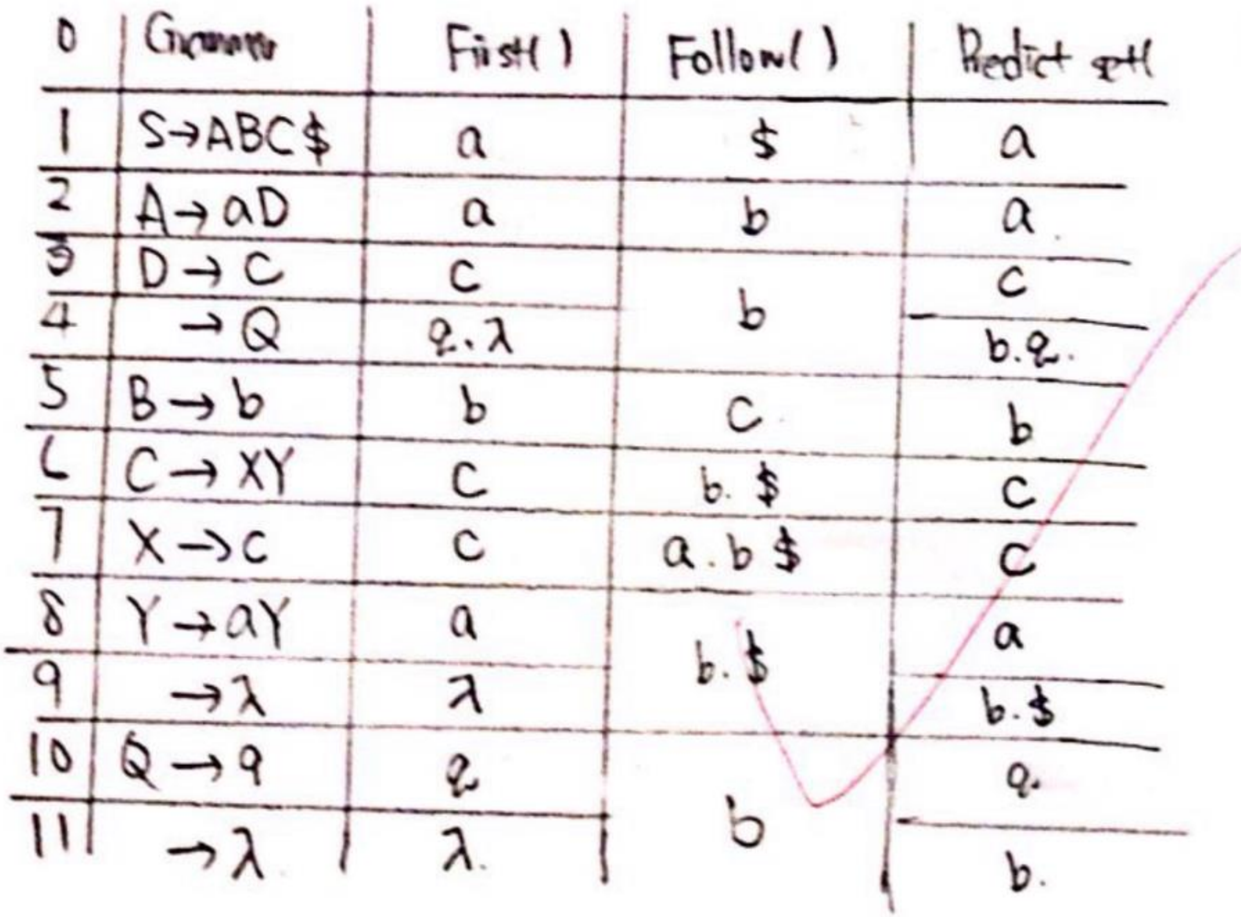
1|	Prog	→	Dcls Stmts $
2|	Dcls	→	Dcl Dcls
3|		|	λ
4|	Dcl	→	floatdcl id
5|		|	intdcl id
6|	Stmts	→	Stmt Stmts
7|		|	λ
8|	Stmt	→	id assign Val Expr
9|		|	print id
10|	Expr	→	plus Val Expr
11|		|	minus Val Expr
12|		|	multiply Val Expr
13|		|	div Val Expr
14|		|	λ
15|	Val	→	id
16|		|	inum
17|		|	fnum

6. Please answer the following questions for building the parser with the given context-free grammar γ . (25pt)

a) Is γ an ambiguous grammar? Why? (5pt)

NO. 不存在一 input string 產生多個 parse tree 的情況

b) Compute the First and Follow sets for γ . (15pt) (Note: When computing FIRST, you should show λ if necessary; otherwise, you will not get the corresponding points.)



0	Grammar	First( )	Follow( )	Predict set
1	S→ABC$	a	$	a
2	A→aD	a	b	a
3	D→c	c	b	c
4	→Q	q, λ		b, q
5	B→b	b	c	b
6	C→XY	c	b. $	c
7	X→c	c	a. b $	c
8	Y→aY	a	b. $	a
9	→λ	λ		b. $
10	Q→q	q	b	q
11	→λ	λ		b

c) Please use a concrete example to illustrate the scenario that describes the necessity of computing the Follow set. (5pt)

1. If there is a production A ⇒ αBβ,
   then everything in **FIRST(β)** except λ is added to FOLLOW(B)

2. If there is a production
   (a) A ⇒ αB, or
   (b) A ⇒ αBβ, where FIRST(β) contains λ,
   then everything in **FOLLOW(A)** is added to FOLLOW(B)

Input string: aqbc$

Context-free grammar, G.

```
 1|   S    ->   A B C $
 2|   A    ->   a D
 3|   D    ->   C
 4|         |   Q
 5|   B    ->   b
 6|   C    ->   X Y
 7|   X    ->   c
 8|   Y    ->   a Y
 9|         |   λ
10|   Q    ->   q
11|         |   λ
```

Example format for First, Follow, and Predict sets.

0|	Grammar	First()	Follow()	Predict set
1|	S -> A B C $	?	?	?
2|	A -> a D	?	?	?
3|	D -> C	?	?	?