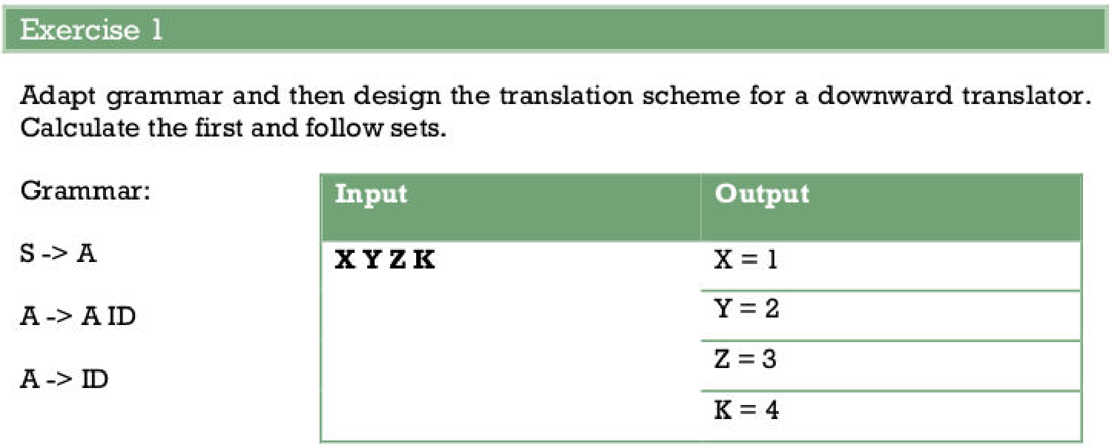


## Exercise 1

Adapt grammar and then design the translation scheme for a downward translator. Calculate the first and follow sets.

Grammar:

S -> A

A -> A ID

A -> ID

| Input | Output |
|---|---|
| **X Y Z K** | X = 1 |
| | Y = 2 |
| | Z = 3 |
| | K = 4 |

Solution:

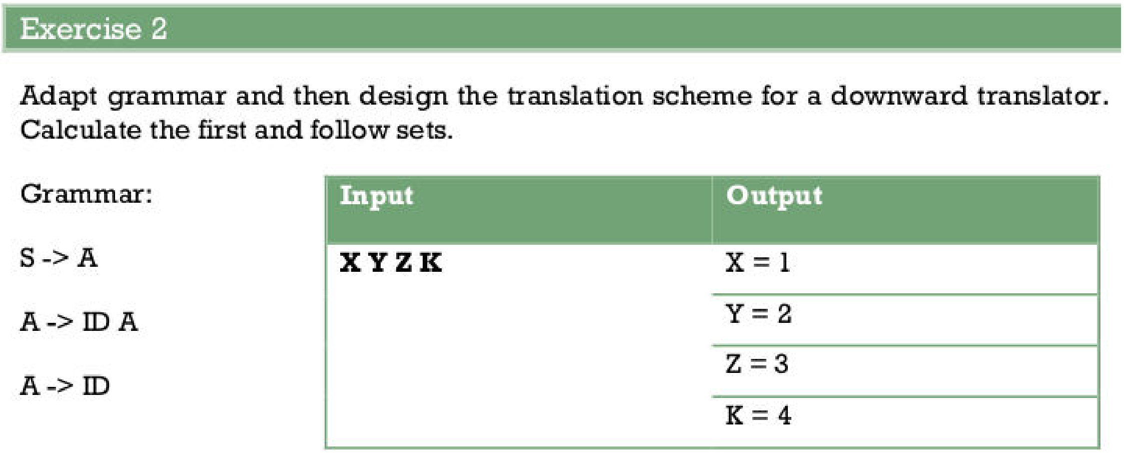
S → A
A → ID {escribe(ID.lexval,” = ”,contador)} A
A → epsilon

FIRST

| | |
|---|---|
| S | First(A) ={ID} |
| A | {ID, ε } |

FOLLOW

| | |
|---|---|
| S | {$} |
| A | Follow(S) = {$} |



## Exercise 2

Adapt grammar and then design the translation scheme for a downward translator. Calculate the first and follow sets.

Grammar:

S -> A

A -> ID A

A -> ID

| Input | Output |
|---|---|
| **X Y Z K** | X = 1 |
| | Y = 2 |
| | Z = 3 |
| | K = 4 |

Solution:

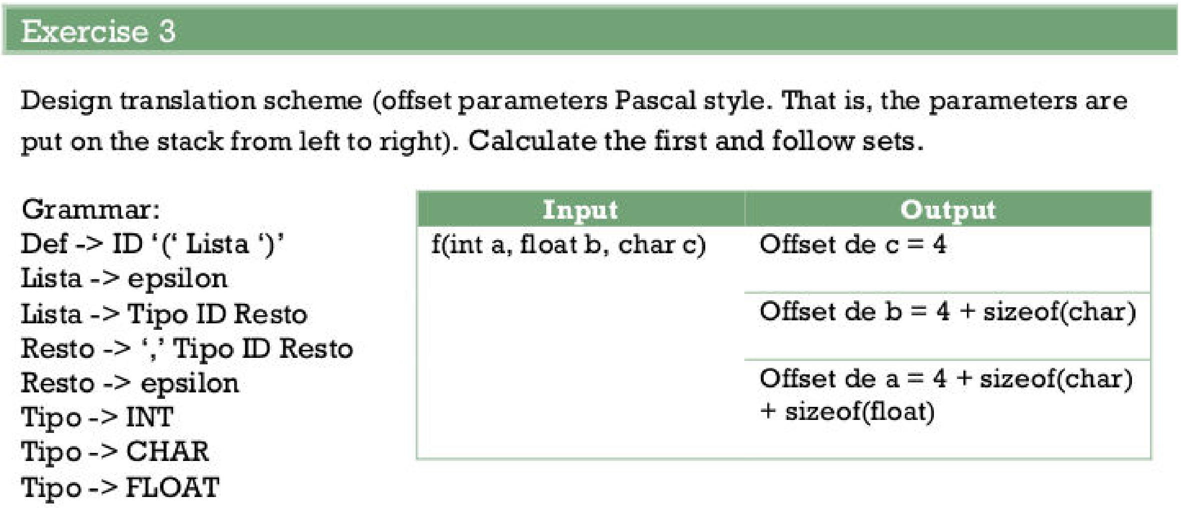
S → A<br>
A → ID {escribe(ID.lexval,” = ”,contador)} A<br>
A → epsilon

FIRST

| | |
|---|---|
| S | First(A) ={ID} |
| A | {ID,ε } |

FOLLOW

| | |
|---|---|
| S | {$} |
| A | Follow(S) = {$} |



## Exercise 3

Design translation scheme (offset parameters Pascal style. That is, the parameters are put on the stack from left to right). Calculate the first and follow sets.

Grammar:
Def -> ID ‘(‘ Lista ‘)’
Lista -> epsilon
Lista -> Tipo ID Resto
Resto -> ‘,’ Tipo ID Resto
Resto -> epsilon
Tipo -> INT
Tipo -> CHAR
Tipo -> FLOAT

| Input | Output |
|---|---|
| f(int a, float b, char c) | Offset de c = 4 |
| | Offset de b = 4 + sizeof(char) |
| | Offset de a = 4 + sizeof(char) + sizeof(float) |

FIRST

| | |
|---|---|
| Def | {ID} |
| Lista | First(Tipo) U epsilon = {INT,CHAR,FLOAT,ε } |
| Resto | {; , epsilon} |
| Tipo | {INT,CHAR,FLOAT} |

FOLLOW

| | |
|---|---|
| Def | {$} |
| Lista | { ) } |
| Resto | Follow(Lista) ={ ) } |
| Tipo | {ID} |

Def → ‘(‘ Lista ‘)’
Lista → epsilon
Lista → Tipo ID Resto
Resto -> ',' Tipo ID {Resto1.h = Resto.h} Resto {Resto.s.valor = ID.lexval; Resto.s.tipo = tipo.s }
Resto -> epsilon { Resto.s= Resto.h ; escribir("Offset de",Resto.s.valor,"= 4 ");}
Tipo → INT
Tipo → CHAR
Tipo → FLOAT