C minus BNF:

program → declaration-list
declaration-list → declaration-list declaration | declaration
declaration → var-declaration | fun-declaration
var-declaration → type-specifier **ID ;** | type-specifier **ID [NUM] ;**
type-specifier → **int** | **void**
fun-declaration → type-specifier **ID (** params **)** compound-stmt
params → param-list | **void**
param-list → param-list **,** param | param
param → type-specifier **ID** | type-specifier **ID [ ]**
compound-stmt → **{** local-declarations statement-list **}**
local-declarations → local-declarations var-declaration | empty
statement-list → statement-list statement | empty
statement → expression-stmt | compound-stmt | selection-stmt | iteration-stmt | return-stmt
expression-stmt → expression **;** | **;**
selection-stmt → **if (** expression **)** statement | **if (** expression **)** statement **else** statement
iteration-stmt → **while (** expression **)** statement
return-stmt → **return ;** | **return** expression **;**
expression → var **=** expression | simple-expression
var → **ID** | **ID [** expression **]**
simple-expression → additive-expression relop additive-expression | additive-expression
relop → **<=** | **<** | **>** | **>=** | **==** | **!=**
additive-expression → additive-expression addop term | term
addop → **+** | **-**
term → term mulop factor | factor
mulop → ***** | **/**
factor → **(** expression **)** | var | call | **NUM**
call → **ID (** args **)**
args → arg-list | empty
arg-list → arg-list **,** expression | expression

Passo 1: Eliminar recursão a esquerda e ambiguidade nas produções da gramática

program → declaration-list
declaration-list → declaration-list declaration | declaration
declaration-list → declaration declaration-list'
declaration-list' → declaration-list
declaration-list' → Ɛ
declaration → var-declaration
declaration → fun-declaration
var-declaration → type-specifier **ID ;** | type-specifier **ID [NUM] ;**
var-declaration → type-specifier **ID** var-declaration'
var-declaration' → **[NUM] ;**
var-declaration' → **;**
type-specifier → **int**

type-specifier → **void**
fun-declaration → type-specifier **ID** **(** params **)** compound-stmt
params → param-list
params → **void**
param-list → param-list **,** param | param
param-list → param param-list'
param-list' → **,** param param-list'
param-list' → Ɛ
param → type-specifier **ID** | type-specifier **ID** **[** **]**
param → type-specifier **ID** param'
param' → **[** **]**
param' → Ɛ
compound-stmt → **{** local-declarations statement-list **}**
local-declarations → local-declarations var-declaration | empty
local-declarations → var-declaration local-declarations'
local-declarations → Ɛ
local-declarations' → local-declarations
statement-list → statement-list statement | empty
statement-list → statement statement-list'
statement-list → Ɛ
statement-list' → statement-list
statement → expression-stmt
statement → compound-stmt
statement → selection-stmt
statement → iteration-stmt
statement → return-stmt
expression-stmt → expression **;**
expression-stmt → **;**
selection-stmt → **if** **(** expression **)** statement | **if** **(** expression **)** statement **else** statement
selection-stmt → **if** **(** expression **)** statement selection-stmt'
selection-stmt' → **else** statement
selection-stmt' → Ɛ
iteration-stmt → **while** **(** expression **)** statement
return-stmt → **return** **;** | **return** expression **;**
return-stmt → **return** expression-stmt
expression → var **=** expression
expression → simple-expression
var → **ID** | **ID** **[** expression **]**
var → **ID** var'
var' → **[** expression **]**
var' → Ɛ
simple-expression → additive-expression relop additive-expression | additive-expression
simple-expression → additive-expression simple-expression'
simple-expression' → relop additive-expression
simple-expression' → Ɛ
relop → **<=**
relop → **<**
relop → **>**

relop → **>=**
relop → **==**
relop → **!=**
additive-expression → additive-expression addop term | term
additive-expression → term additive-expression'
additive-expression' → addop additive-expression
additive-expression' → Ɛ
addop → **+**
addop → **-**
term → term mulop factor | factor
term → factor term'
term' → mulop term
term' → Ɛ
mulop → ***
mulop → **/**
factor → **(** expression **)**
factor → var
factor → call
factor → **NUM**
call → **ID (** args **)**
args → arg-list
args → Ɛ
arg-list → arg-list **,** expression | expression
arg-list → expression arg-list'
arg-list' → **,** arg-list
arg-list' → Ɛ

Resultado da Aplicação do Passo 1:

1. program → declaration-list
2. declaration-list → declaration declaration-list'
3. declaration-list' → declaration-list
4. declaration-list' → Ɛ
5. declaration → var-declaration
6. declaration → fun-declaration
7. var-declaration → type-specifier **ID** var-declaration'
8. var-declaration' → **[NUM] ;**
9. var-declaration' → **;**
10. type-specifier → **int**
11. type-specifier → **void**
12. fun-declaration → type-specifier **ID (** params **)** compound-stmt
13. params → param-list
14. params → **void**
15. param-list → param param-list'
16. param-list' → **,** param param-list'
17. param-list' → Ɛ

18. param → type-specifier **ID** param**'**
19. param' → **[ ]**
20. param' → Ɛ
21. compound-stmt → **{** local-declarations statement-list **}**
22. local-declarations → var-declaration local-declarations'
23. local-declarations → Ɛ
24. local-declarations' → local-declarations
25. statement-list → statement statement-list'
26. statement-list → Ɛ
27. statement-list' → statement-list
28. statement → expression-stmt
29. statement → compound-stmt
30. statement → selection-stmt
31. statement → iteration-stmt
32. statement → return-stmt
33. expression-stmt → expression **;**
34. expression-stmt → **;**
35. selection-stmt → **if (** expression **)** statement selection-stmt'
36. selection-stmt' → **else** statement
37. selection-stmt' → Ɛ
38. iteration-stmt → **while (** expression **)** statement
39. return-stmt → **return** expression-stmt
40. expression → var **=** expression
41. expression → simple-expression
42. var → **ID** var'
43. var' → **[** expression **]**
44. var' → Ɛ
45. simple-expression → additive-expression simple-expression'
46. simple-expression' → relop additive-expression
47. simple-expression' → Ɛ
48. relop → **<=**
49. relop → **<**
50. relop → **>**
51. relop → **>=**
52. relop → **==**
53. relop → **!=**
54. additive-expression → term additive-expression'
55. additive-expression' → addop additive-expression
56. additive-expression' → Ɛ
57. addop → **+**
58. addop → **-**
59. term → factor term'
60. term' → mulop term
61. term' → Ɛ
62. mulop → ***
63. mulop → */*
64. factor → **(** expression **)**
65. factor → var

66. factor → call
67. factor → **NUM**
68. call → **ID (** args **)**
69. args → arg-list
70. args → Ɛ
71. arg-list → expression arg-list'
72. arg-list' → **,** arg-list
73. arg-list' → Ɛ

Passo 2: Executar algoritmo para construir o conjunto Primeiro de cada produção

Algoritmo:
para cada não terminal A, Primeiro(A) = {};
enquanto houver alteração em algum Primeiro(A)
para cada escolha de produção A -> X1X2...Xn
acrescente Primeiro(X1) a Primeiro(A);

Observações:
Só terminais entram em conjuntos Primeiro.
O algoritmo de cálculo de Primeiro(α):
– É trivial quando α é um terminal t.
– varre as produções X → tω quando α é um não-terminal X;
– Inclui ε apenas quando X pode derivar em ε

1ª Iteração:

Primeiro(program) → {}
Primeiro(declaration-list) → {}
Primeiro(declaration-list') → {} = { Ɛ }
Primeiro(declaration) → {}
Primeiro(var-declaration) → {}
Primeiro(var-declaration') → {} = { **[** } ={ **[** , **;** }
Primeiro(type-specifier) → {} = { **int** } = { **int** , **void** }
Primeiro(fun-declaration) → {} = { **int** , **void** }
Primeiro(params) → {} = { **void** }
Primeiro(param-list) → {}
Primeiro(param-list') → {} = { **,** } = { , , Ɛ }
Primeiro(param) → {} = { **int** , **void** }
Primeiro(param') → {} = { **[** } = { **[** , Ɛ }
Primeiro(compound-stmt) → {} = { **{** }
Primeiro(local-declarations) → {} = { Ɛ }
Primeiro(local-declarations') → {} = { Ɛ }
Primeiro(statement-list) → {} = { Ɛ }
Primeiro(statement-list') → {} = { Ɛ }
Primeiro(statement) → {} = { **{** }

Primeiro(expression-stmt) → {} = { **;** }
Primeiro(selection-stmt) → {} = { **if** }
Primeiro(selection-stmt') → {} = { **else** } = { **else** , Ɛ }
Primeiro(iteration-stmt) → {} = { **while** }
Primeiro(return-stmt) → {} = { **return** }
Primeiro(expression) → {}
Primeiro(var) → {} = { **ID** }
Primeiro(var') → {} = { **[** } = { **[** , Ɛ }
Primeiro(simple-expression) → {}
Primeiro(simple-expression') → {} = { Ɛ }
Primeiro(relop) → {} = { **<=** } = { **<=** , **<** } = { **<=** , **<** , **>** } = { **<=** , **<** , **>** , **>=** } = { **<=** , **<** , **>** , **>=** , **==** } = { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → {}
Primeiro(additive-expression') → {} = { Ɛ }
Primeiro(addop) → {} = { **+** } = { **+** , **-** }
Primeiro(term) → {}
Primeiro(term') → {} = { Ɛ }
Primeiro(mulop) → {} = { * } = { * , / }
Primeiro(factor) → {} = { **(** } = { **(** , **ID** } = { **(** , **ID** , **NUM** }
Primeiro(call) → {} = { **ID** }
Primeiro(args) → {} = { Ɛ }
Primeiro(arg-list) → {}
Primeiro(arg-list') → {} = { **,** } = { **,** , Ɛ }

2ª Iteração:

Primeiro(program) → {}
Primeiro(declaration-list) → {}
Primeiro(declaration-list') → { Ɛ }
Primeiro(declaration) → {} = { **int** , **void** }
Primeiro(var-declaration) → {} = { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** }
Primeiro(param-list) → {} = { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ } = { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ } = { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ } = { Ɛ , **{** }
Primeiro(statement-list') → { Ɛ } = { Ɛ , **{** }
Primeiro(statement) → { **{** , **;** , **if** , **while** } = { **{** , **;** , **if** , **while** , **return** }
Primeiro(expression-stmt) → { **;** }
Primeiro(selection-stmt) → { **if** }

Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → {} = { **ID** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → {}
Primeiro(simple-expression') → { Ɛ } = { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → {}
Primeiro(additive-expression') → { Ɛ } = { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → {} = { **(** , **ID** , **NUM** }
Primeiro(term') → {} = { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ }
Primeiro(arg-list) → {} = { **ID** }
Primeiro(arg-list') → { **,** , Ɛ }

3ª Iteração:

Primeiro(program) → {}
Primeiro(declaration-list) → {} = { **int** , **void** }
Primeiro(declaration-list') → { Ɛ } = { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** } = { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** } = { Ɛ , **{** , **;** , **if** , **while** , **return** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** } = { Ɛ , **{** , **;** , **if** , **while** , **return** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** }
Primeiro(expression-stmt) → { **;** } = { **;** , **ID** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }

Primeiro(expression) → { **ID** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → {}
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → {} = { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ } = { Ɛ , **ID** }
Primeiro(arg-list) → { **ID** }
Primeiro(arg-list') → { **,** , Ɛ }

4ª Iteração:

Primeiro(program) → {} = { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** } = { **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(expression-stmt) → { **;** , **ID** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }

Primeiro(simple-expression) → {} = { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** }
Primeiro(arg-list) → { **ID** }
Primeiro(arg-list') → { **,** , Ɛ }

5ª Iteração:

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** } = { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** } = { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(expression-stmt) → { **;** , **ID** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** } = { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }

Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** }
Primeiro(arg-list) → { **ID** } = { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

6ª Iteração:

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(expression-stmt) → { **;** , **ID** } = { **;** , **ID** , **(** , **NUM** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }

Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** } = { Ɛ , **ID** , **(** , **NUM** }
Primeiro(arg-list) → { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

7ª Iteração:

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** } = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(expression-stmt) → { **;** , **ID** , **(** , **NUM** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }

Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** , **(** , **NUM** }
Primeiro(arg-list) → { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

8ª Iteração:

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** } = { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** } = { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(expression-stmt) → { **;** , **ID** , **(** , **NUM** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }

Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** , **(** , **NUM** }
Primeiro(arg-list) → { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

9ª Iteração: (Como não há mais alterações em nenhum dos conjuntos, a execução do algoritmo é encerrada)

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(expression-stmt) → { **;** , **ID** , **(** , **NUM** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }

Primeiro(args) → { Ɛ , **ID** , **(** , **NUM** }
Primeiro(arg-list) → { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

Conjuntos Primeiro resultantes da aplicação do passo 2:

Primeiro(program) → { **int** , **void** }
Primeiro(declaration-list) → { **int** , **void** }
Primeiro(declaration-list') → { Ɛ , **int** , **void** }
Primeiro(declaration) → { **int** , **void** }
Primeiro(var-declaration) → { **int** , **void** }
Primeiro(var-declaration') → { **[** , **;** }
Primeiro(type-specifier) → { **int** , **void** }
Primeiro(fun-declaration) → { **int** , **void** }
Primeiro(params) → { **void** , **int** }
Primeiro(param-list) → { **int** , **void** }
Primeiro(param-list') → { **,** , Ɛ }
Primeiro(param) → { **int** , **void** }
Primeiro(param') → { **[** , Ɛ }
Primeiro(compound-stmt) → { **{** }
Primeiro(local-declarations) → { Ɛ , **int** , **void** }
Primeiro(local-declarations') → { Ɛ , **int** , **void** }
Primeiro(statement-list) → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement-list') → { Ɛ , **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** }
Primeiro(expression-stmt) → { **;** , **ID** , **(** , **NUM** }
Primeiro(selection-stmt) → { **if** }
Primeiro(selection-stmt') → { **else** , Ɛ }
Primeiro(iteration-stmt) → { **while** }
Primeiro(return-stmt) → { **return** }
Primeiro(expression) → { **ID** , **(** , **NUM** }
Primeiro(var) → { **ID** }
Primeiro(var') → { **[** , Ɛ }
Primeiro(simple-expression) → { **(** , **ID** , **NUM** }
Primeiro(simple-expression') → { Ɛ , **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(relop) → { **<=** , **<** , **>** , **>=** , **==** , **!=** }
Primeiro(additive-expression) → { **(** , **ID** , **NUM** }
Primeiro(additive-expression') → { Ɛ , **+** , **-** }
Primeiro(addop) → { **+** , **-** }
Primeiro(term) → { **(** , **ID** , **NUM** }
Primeiro(term') → { Ɛ , ***** , **/** }
Primeiro(mulop) → { ***** , **/** }
Primeiro(factor) → { **(** , **ID** , **NUM** }
Primeiro(call) → { **ID** }
Primeiro(args) → { Ɛ , **ID** , **(** , **NUM** }
Primeiro(arg-list) → { **ID** , **(** , **NUM** }
Primeiro(arg-list') → { **,** , Ɛ }

Passo 3: Executar algoritmo para construir o conjunto Sequência de cada produção

Observações:
É reservado aos não-terminais X
Varre as produções onde X aparece à direita (A → ωX ω')
É chato quando X aparece no fim (ou logo antes de algo que deriva em ε)
Nunca inclui ε
1ª Iteração:

Sequência(program) → { $ }
Sequência(declaration-list) → {} = { $ }
Sequência(declaration-list') → {} = { $ }
Sequência(declaration) → {} = { **int** , **void** , $ } { Primeiro(declaration-list') – Ɛ } ∪ Sequência(declaration-list)
Sequência(var-declaration) → {} = { **int** , **void** , $ } Sequência(declaration)
Sequência(var-declaration') → {} = { **int** , **void** , $ } Sequência(var-declaration)
Sequência(type-specifier) → {} = { **ID** }
Sequência(fun-declaration) → {} = { **int** , **void** , $ } Sequência(declaration)
Sequência(params) → {} = { **)** }
Sequência(param-list) → {} = { **)** } Sequência(params)
Sequência(param-list') → {} = { **)** } Sequência(param-list)
Sequência(param) → {} = { **,** , **)** } { Primeiro(param-list') – Ɛ } ∪ Sequência(param-list)
Sequência(param') → {} = { **,** , **)** } Sequência(param)
Sequência(compound-stmt) → {} = { **int** , **void** , $ } Sequência(fun-declaration) = { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(statement)
Sequência(local-declarations) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } { Primeiro(statement-list) – Ɛ } ∪ Primeiro(})
Sequência(local-declarations') → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(local-declarations)
Sequência(statement-list) → {} = { **}** }
Sequência(statement-list') → {} = { **}** } Sequência(statement-list)
Sequência(statement) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } { Primeiro(statement-list') – Ɛ } ∪ Sequência(statement-list) = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** } { Primeiro(selection-stmt') – Ɛ } ∪ Sequência(selection-stmt)
Sequência(expression-stmt) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(statement)
Sequência(selection-stmt) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(statement)
Sequência(selection-stmt') → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(selection-stmt)
Sequência(iteration-stmt) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(statement)
Sequência(return-stmt) → {} = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } Sequência(statement)
Sequência(expression) → {} = { **;** } = { **;** , **)** } = { **;** , **)** , **]** } = { **;** , **)** , **]** , **,** } { Primeiro(arg-list') – Ɛ } ∪ Sequência(arg-list)
Sequência(var) → {} = { **=** } = { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } Sequência(factor)
Sequência(var') → {} = { **=** } Sequência(var)
Sequência(simple-expression) → {} = { **;** , **)** , **]** } Sequência(expression)
Sequência(simple-expression') → {} = { **;** , **)** , **]** } Sequência(simple-expression)

Sequência(relop) → {} = { **(** , **ID** , **NUM** } Primeiro(additive-expression)
Sequência(additive-expression) → {} = { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** }
{ Primeiro(simple-expression') – Ɛ } ∪ Sequência(simple-expression)
Sequência(additive-expression') → {} = { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** }
Sequência(additive-expression)
Sequência(addop) → {} = { **(** , **ID** , **NUM** } Primeiro(additive-expression)
Sequência(term) → {} = { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } { Primeiro(additive-expression') – Ɛ }
∪ Sequência(additive-expression)
Sequência(term') → {} = { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } Sequência(term)
Sequência(mulop) → {} = { **(** , **ID** , **NUM** } Primeiro(term)
Sequência(factor) → {} = { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } { Primeiro(term') – Ɛ } ∪
Sequência(term)
Sequência(call) → {} = { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } Sequência(factor)
Sequência(args) → {} = { **)** }
Sequência(arg-list) → {} = { **)** } Sequência(args)
Sequência(arg-list') → {} = { **)** } Sequência(arg-list)

2ª Iteração:

Sequência(program) → { $ }
Sequência(declaration-list) → { $ }
Sequência(declaration-list') → { $ }
Sequência(declaration) → { **int** , **void** , $ }
Sequência(var-declaration) → { **int** , **void** , $ } = { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** ,
(, **NUM** , **}** } { Primeiro(local-declarations') – Ɛ } ∪ Sequência(local-declarations)
Sequência(var-declaration') → { **int** , **void** , $ } = { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID**
, **(** , **NUM** , **}** } Sequência(var-declaration)
Sequência(type-specifier) → { **ID** }
Sequência(fun-declaration) → { **int** , **void** , $ }
Sequência(params) → { **)** }
Sequência(param-list) → { **)** }
Sequência(param-list') → { **)** }
Sequência(param) → { **,** , **)** }
Sequência(param') → { **,** , **)** }
Sequência(compound-stmt) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = {
int , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** } Sequência(statement)
Sequência(local-declarations) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(local-declarations') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(statement-list) → { **}** }
Sequência(statement-list') → { **}** }
Sequência(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = { **{** , **;** , **if** , **while**
, **return** , **ID** , **(** , **NUM** , **}** , **else** } Sequência(statement)
Sequência(selection-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = { **{** , **;** , **if** , **while** ,
return , **ID** , **(** , **NUM** , **}** , **else** } Sequência(statement)
Sequência(selection-stmt') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = { **{** , **;** , **if** , **while** ,
return , **ID** , **(** , **NUM** , **}** , **else** } Sequência(selection-stmt)

Sequência(iteration-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** } Sequência(statement)
Sequência(return-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** } = { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** } Sequência(statement)
Sequência(expression) → { **;** , **)** , **]** , **,** }
Sequência(var) → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** }
Sequência(var') → { **=** } = { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } Sequência(var)
Sequência(simple-expression) → { **;** , **)** , **]** } = { **;** , **)** , **]** , **,** } Sequência(expression)
Sequência(simple-expression') → { **;** , **)** , **]** } = { **;** , **)** , **]** , **,** } Sequência(simple-expression)
Sequência(relop) → { **(** , **ID** , **NUM** }
Sequência(additive-expression) → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(simple-expression)
Sequência(additive-expression') → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(additive-expression)
Sequência(addop) → { **(** , **ID** , **NUM** }
Sequência(term) → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(additive-expression)
Sequência(term') → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(term)
Sequência(mulop) → { **(** , **ID** , **NUM** }
Sequência(factor) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(term)
Sequência(call) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(factor)
Sequência(args) → { **)** }
Sequência(arg-list) → { **)** }
Sequência(arg-list') → { **)** }

3ª Iteração:

Sequência(program) → { $ }
Sequência(declaration-list) → { $ }
Sequência(declaration-list') → { $ }
Sequência(declaration) → { **int** , **void** , $ }
Sequência(var-declaration) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(var-declaration') → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(type-specifier) → { **ID** }
Sequência(fun-declaration) → { **int** , **void** , $ }
Sequência(params) → { **)** }
Sequência(param-list) → { **)** }
Sequência(param-list') → { **)** }
Sequência(param) → { **,** , **)** }
Sequência(param') → { **,** , **)** }
Sequência(compound-stmt) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else**}
Sequência(local-declarations) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(local-declarations') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }

Sequência(statement-list) → { **}** }
Sequência(statement-list') → { **}** }
Sequência(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(iteration-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(return-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression) → { **;** , **)** , **]** , **,** }
Sequência(var) → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(factor)
Sequência(var') → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** } = { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** } Sequência(var)
Sequência(simple-expression) → { **;** , **)** , **]** , **,** }
Sequência(simple-expression') → { **;** , **)** , **]** , **,** }
Sequência(relop) → { **(** , **ID** , **NUM** }
Sequência(additive-expression) → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(additive-expression') → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(addop) → { **(** , **ID** , **NUM** }
Sequência(term) → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(term') → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(mulop) → { **(** , **ID** , **NUM** }
Sequência(factor) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(call) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(args) → { **)** }
Sequência(arg-list) → { **)** }
Sequência(arg-list') → { **)** }

4ª Iteração: (Como não há mais alterações em nenhum dos conjuntos, a execução do algoritmo é encerrada)

Sequência(program) → { $ }
Sequência(declaration-list) → { $ }
Sequência(declaration-list') → { $ }
Sequência(declaration) → { **int** , **void** , $ }
Sequência(var-declaration) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(var-declaration') → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(type-specifier) → { **ID** }
Sequência(fun-declaration) → { **int** , **void** , $ }
Sequência(params) → { **)** }
Sequência(param-list) → { **)** }
Sequência(param-list') → { **)** }
Sequência(param) → { **,** , **)** }
Sequência(param') → { **,** , **)** }
Sequência(compound-stmt) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else**}
Sequência(local-declarations) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }

Sequência(local-declarations') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(statement-list) → { **}** }
Sequência(statement-list') → { **}** }
Sequência(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(iteration-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(return-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression) → { **;** , **)** , **]** , **,** }
Sequência(var) → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(var') → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(simple-expression) → { **;** , **)** , **]** , **,** }
Sequência(simple-expression') → { **;** , **)** , **]** , **,** }
Sequência(relop) → { **(** , **ID** , **NUM** }
Sequência(additive-expression) → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(additive-expression') → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(addop) → { **(** , **ID** , **NUM** }
Sequência(term) → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(term') → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(mulop) → { **(** , **ID** , **NUM** }
Sequência(factor) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(call) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(args) → { **)** }
Sequência(arg-list) → { **)** }
Sequência(arg-list') → { **)** }

Conjuntos Sequência resultantes da aplicação do passo 3:

Sequência(program) → { $ }
Sequência(declaration-list) → { $ }
Sequência(declaration-list') → { $ }
Sequência(declaration) → { **int** , **void** , $ }
Sequência(var-declaration) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(var-declaration') → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(type-specifier) → { **ID** }
Sequência(fun-declaration) → { **int** , **void** , $ }
Sequência(params) → { **)** }
Sequência(param-list) → { **)** }
Sequência(param-list') → { **)** }
Sequência(param) → { **,** , **)** }
Sequência(param') → { **,** , **)** }
Sequência(compound-stmt) → { **int** , **void** , $, **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else**}
Sequência(local-declarations) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(local-declarations') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** }
Sequência(statement-list) → { **}** }

Sequência(statement-list') → { **}** }
Sequência(statement) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(selection-stmt') → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(iteration-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(return-stmt) → { **{** , **;** , **if** , **while** , **return** , **ID** , **(** , **NUM** , **}** , **else** }
Sequência(expression) → { **;** , **)** , **]** , **,** }
Sequência(var) → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(var') → { **=** , ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(simple-expression) → { **;** , **)** , **]** , **,** }
Sequência(simple-expression') → { **;** , **)** , **]** , **,** }
Sequência(relop) → { **(** , **ID** , **NUM** }
Sequência(additive-expression) → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(additive-expression') → { **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(addop) → { **(** , **ID** , **NUM** }
Sequência(term) → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(term') → { **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(mulop) → { **(** , **ID** , **NUM** }
Sequência(factor) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(call) → { ***** , **/** , **+** , **-** , **<=** , **<** , **>** , **>=** , **==** , **!=** , **;** , **)** , **]** , **,** }
Sequência(args) → { **)** }
Sequência(arg-list) → { **)** }
Sequência(arg-list') → { **)** }

Passo 4: Construção da tabela de análise sintática LL(1)

Algoritmo:
para cada não terminal A e escolha de produção A -> α

> para cada token a em Primeiro(α), insira A -> α em M[A,a]
>
> se Ɛ pertencer a Primeiro(α), para cada elemento a em Sequência(A), insira A -> α em M[A,a]

Observações:
Para uma gramática ser LL(1) é necessário que:

1. para cada produção A -> α1 | α2 | ... |αn, Primeiro(αi) ∩ Primeiro(αj) = ∅
   para i >= 1, j =< n, i <> j(prefixos distintos)
2. para cada não terminal A tal que Ɛ ∈ Primeiro(A), Primeiro(A) ∩ Sequência(A) = ∅

Os números usados na construção da tabela são os números das produções da gramática:

1. program → declaration-list

2. declaration-list → declaration declaration-list'
3. declaration-list' → declaration-list
4. declaration-list' → Ɛ
5. declaration → var-declaration
6. declaration → fun-declaration
7. var-declaration → type-specifier **ID** var-declaration'
8. var-declaration' → **[NUM] ;**
9. var-declaration' → **;**
10. type-specifier → **int**
11. type-specifier → **void**
12. fun-declaration → type-specifier **ID (** params **)** compound-stmt
13. params → param-list
14. params → **void**
15. param-list → param param-list'
16. param-list' → **,** param param-list'
17. param-list' → Ɛ
18. param → type-specifier **ID** param**'**
19. param' → **[ ]**
20. param' → Ɛ
21. compound-stmt → **{** local-declarations statement-list **}**
22. local-declarations → var-declaration local-declarations'
23. local-declarations → Ɛ
24. local-declarations' → local-declarations
25. statement-list → statement statement-list'
26. statement-list → Ɛ
27. statement-list' → statement-list
28. statement → expression-stmt
29. statement → compound-stmt
30. statement → selection-stmt
31. statement → iteration-stmt
32. statement → return-stmt
33. expression-stmt → expression **;**
34. expression-stmt → **;**
35. selection-stmt → **if (** expression **)** statement selection-stmt'
36. selection-stmt' → **else** statement
37. selection-stmt' → Ɛ
38. iteration-stmt → **while (** expression **)** statement
39. return-stmt → **return** expression-stmt
40. expression → var **=** expression
41. expression → simple-expression
42. var → **ID** var'
43. var' → **[** expression **]**
44. var' → Ɛ
45. simple-expression → additive-expression simple-expression'
46. simple-expression' → relop additive-expression
47. simple-expression' → Ɛ
48. relop → **<=**
49. relop → **<**

50. relop → **>**
51. relop → **>=**
52. relop → **==**
53. relop → **!=**
54. additive-expression → term additive-expression’
55. additive-expression’ → addop additive-expression
56. additive-expression’ → Ɛ
57. addop → **+**
58. addop → **-**
59. term → factor term’
60. term’ → mulop term
61. term’ → Ɛ
62. mulop → ***
63. mulop → */*
64. factor → **(** expression **)**
65. factor → var
66. factor → call
67. factor → **NUM**
68. call → **ID (** args **)**
69. args → arg-list
70. args → Ɛ
71. arg-list → expression arg-list’
72. arg-list’ → **,** arg-list
73. arg-list’ → Ɛ

M[não-terminais, tokens]	$	id	[	num	]	;	int	void	else	(	…
program							1	1			
declaration-list							2	2			
declaration-list’	4										
declaration							5 * 6	5 * 6			
var-declaration							7	7			
var-declaration’			8			9					
type-specifier							10	11			
fun-declaration							12	12			
params							13	13 **			

								14			
…											

* Como logo no começo da construção da tabela obtemos que as produções 5 e 6 ocupam as mesmas posições M[declaration , int] e M[declaration , void], temos 2 produções para um mesmo espaço, que vai contra a regra “1. para cada produção A -> α1 | α2 | ... |αn, Primeiro(αi) ∩ Primeiro(αj) = ∅ para i >= 1, j =< n, i <> j(prefixos distintos)” pois Primeiro(var-declaration) ∩ Primeiro(fun-declaration) = { **int** , **void** } , o que resulta em ambas produções ocuparem as mesmas posições na tabela.
** Mais um caso que faz a Linguagem C- não ser LL(1) é o caso das produções 13 e 14 ocuparem a mesma posição M[params , void] pois a mesma regra “1. para cada produção A -> α1 | α2 | ... |αn, Primeiro(αi) ∩ Primeiro(αj) = ∅ para i >= 1, j =< n, i <> j(prefixos distintos)” não é seguida. Nesse caso Primeiro(param-list) ∩ Primeiro(**void**) = { **void** } , o que também resulta em ambas produções ocuparem a mesma posição na tabela.

Visto que esses dois casos obtidos na construção da tabela já descartam a possibilidade da Linguagem C- ser LL(1), não será necessário continuar construindo o resto da tabela de análise sintática LL(1).