Grammar:

program 🡪 program\_tail | meta\_statement program

First(program) = {int, void} Follow(program) = {}

program\_tail 🡪 type\_name identifier data\_or\_func program\_tail | $

data\_or\_func 🡪 beginning\_data\_decls | start\_func

First(program\_tail) = { comma, ( }

start\_func 🡪 ( parameter\_list ) func\_follow func\_list

First(func) = { ( }

func\_list 🡪 func func\_list | $

First(func\_list) = {$, ( }

func\_follow 🡪 ; | { data\_decls statements }

First(func\_follow) ={ comma , { }

func\_decl 🡪 type\_name identifier ( parameter\_list )

First(func\_decl) = { ( }

parameter\_list 🡪 void | non\_empty\_list | $

First(parameter\_list) = {void, int, $}

non\_empty\_list 🡪 type\_name identifier non\_empty\_list\_tail

First(non\_empty\_list) = {int, void}

non\_empty\_list\_tail 🡪 , type\_name identifier non\_empty\_list\_tail | $

First(non\_empty\_list\_tail) = { comma, $}

beginning\_data\_decls 🡪 , id\_list ; | ;

First(beginning\_data\_decls) = {$, comma}

data\_decls 🡪 type\_name id\_list ; data\_decls | $

type\_name 🡪 int | void

First(type\_name) = {int, void}

id\_list 🡪 identifier id\_list\_tail

First(id\_list) = { comma, ;, $}

id\_list\_tail 🡪 , identifier id\_list\_tail | $

First(id\_list\_tail) = { comma, $}

statements 🡪 statement statements | $

First(statements) = { $, letter, printf, scanf, if, while, return, break, continue}

statement 🡪 exp\_statement | print\_func\_call | scan\_func\_call | if\_statement | while\_statement | return\_statement | break\_statement | continue\_statement

First(statement) = { letter, printf, scanf, if, while, return, break, continue}

exp\_statement 🡪 identifier assign\_or\_func

First(exp\_statement) = {letter}

assign\_or\_func 🡪 = expression ; | ( expr\_list ) ;

First(assign\_or\_func) = { =, ( }

print\_func\_call 🡪 printf ( string printf\_tail

First(print\_func\_call) = { printf }

printf\_tail 🡪 ) ; | , expression ) ;

First(printf\_tail) = { ) , comma }

scan\_func\_call 🡪 scanf ( string, &expression)

First(scan\_func\_call) = { scanf }

expr\_list 🡪 non\_empty\_expr\_list | $

First(expr\_list) = { letter, $ }

non\_empty\_expr\_list 🡪 expression expr\_list\_tail

First(non\_empty\_expr\_list) = { letter }

expr\_list\_tail 🡪 , expression expr\_list\_tail | $

First(expr\_list\_tail) = { comma, $ }

if\_statement 🡪 if ( condition\_expression ) block\_statements else\_statement

First(if\_statement) = { if }

else\_statement 🡪 else block\_statements | $

First(else\_statement) = { else, $ }

while\_statement 🡪 while ( condition\_expression ) block\_statements

First(while\_statement) = { while }

return\_statement 🡪 return return\_tail

First(return\_statement) = { return }

return\_tail 🡪 ; | expression ;

First(return\_tail) = { ; , letter }

break\_statement 🡪 break ;

First(break\_statement) = { break }

continue\_statement 🡪 continue ;

First(continue\_statement) = { continue }

block\_statements 🡪 { statements }

First(block\_statements) = ( { )

condition\_expression 🡪 condition condition\_expression\_tail

First(condition\_expression) = {letter}

condition\_expression\_tail 🡪 condition\_op condition condition\_expression\_tail | $

First(condition\_expresson\_tail) = { &, | }

condition\_op 🡪 && | ||

First(condition\_op) = { &, | }

condition 🡪 expression comparison\_op expression

First(condition) = { letter }

comparison\_op 🡪 == | != | > | >= | < | <=

First(comparison\_op) = { =, !, >, < }

expression 🡪 term expression\_tail

First(expression) = { letter }

expression\_tail 🡪 addop term expression\_tail | $

First(expression\_tail) = { +, -, $}

addop 🡪 + | -

First(addop) = { +, - }

term 🡪 factor term\_tail

First(term) = { letter }

term\_tail 🡪 mulop factor term\_tail | $

First(term\_tail) = { \*, /, $ }

mulop 🡪 \* | /

First(mulop) = { \*, / }

factor 🡪 identifier factor\_tail | number | - number | ( expression )

First(factor) = { letter, digit, -, ( }

factor\_tail 🡪 [ expression ] | ( expr\_list ) | $

First(factor\_tail) = { [, (, $ }

identifier 🡪 id identifier\_tail

First(identifier) = { letter }

identifier\_tail 🡪 [ expression ] | $

First(identifier\_tail) = { [, $ }

id 🡪 letter let\_or\_dig

First(id) = { letter }

let\_or\_dig 🡪 letter let\_or\_dig | digit let\_or\_dig | \_ let\_or\_dig | $

First(let\_or\_dig) = { letter, digit, \_, $ }

number 🡪 digit number\_tail

First(number) = { digit }

number\_tail 🡪 digit number\_tail | $

First(number\_tail) = { digit, $ }

str 🡪 “ STRING “

meta\_statements