→Script - Tokens & Grammar

María Alejandra Pestana Viso A00824367

Armando Roque Villasana A01138717

Tokens

```
----------- →  foundations (macros)
digit       →  [0-9]
digits      →  {digit}+
letter      →  [a-zA-Z]
letters     →  {letter}+


----------- →  spacing
blank       →  [ ]
tab         →  \t
newline     →  \n
ws          →  ({blank} | {tab} | {newline})+
EOF         →  <<EOF>>


----------- →  reserved general words
program     →  program
main        →  main
var         →  var


----------- →  class reserved words
class       →  class
extends     →  extends
attributes  →  attributes
methods     →  methods


----------- →  function reserved words
void        →  void
func        →  func
return      →  return


----------- →  types
int         →  int
float       →  float
char        →  char
```

```
----------- →  IO reserved words
read        →  read
print       →  print


----------- →  control reserved words
if          →  if
else        →  else


----------- →  iteration reserved words
while       →  while
for         →  for
until       →  until


----------- →  IDs
id          →  {letter}({letter} | {digit})*

----------- →  literals
cte_f       →  {digits}\.{digits}
cte_i       →  {digits}
cte_string  →  \" ({letters} | {digits})+ \"


----------- →  ArrowScript markers
<-          →  <-
->          →  ->


----------- →  relational operators
<           →  <
>           →  >
=           →  =
==          →  ==
!=          →  !=
&           →  &
|           →  |


----------- →  arithmetic operators
*           →  *
/           →  /
-           →  -
+           →  +


----------- →  brackets
(           →  (
)           →  )
{           →  {
}           →  }
[           →  [
]           →  ]
```

```
----------- →  punctuation
:           →  :
;           →  ;
,           →  ,
.           →  .


----------- →  fallback
.           →  "error"
```

Grammar

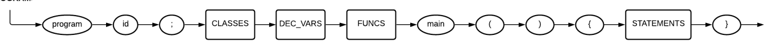
```
PROGRAM     →   program id ; CLASSES DEC_VARS FUNCS main ( ) { STATEMENTS }

CLASSES     →   class id { ATTRIBUTES METHODS }
             |  class id extends id { ATTRIBUTES METHODS }
             |  ϵ

ATTRIBUTES  →   attributes <- SIMPLE_VAR_LIST ->

METHODS     →   methods <- FUNCS ->

DEC_VARS    →   var <- SIMPLE_TYPE SIMPLE_ID_DEC SIMPLE_ID_LIST ; DEC_VARS_LIST ->
             |  var <- COMPOUND_TYPE COMPOUND_ID_DEC COMPOUND_ID_LIST ; DEC_VARS_LIST ->
             |  ϵ

DEC_VARS_LIST →   SIMPLE_TYPE SIMPLE_ID_DEC SIMPLE_ID_LIST ; DEC_VARS_LIST
             |  COMPOUND_TYPE COMPOUND_ID_DEC COMPOUND_ID_LIST ; DEC_VARS_LIST
             |  ϵ

SIMPLE_VAR_LIST →   SIMPLE_TYPE SIMPLE_ID_DEC SIMPLE_ID_LIST ; SIMPLE_VAR_LIST
             |  ϵ

COMPOUND_VAR_LIST →   COMPOUND_TYPE COMPOUND_ID_DEC COMPOUND_ID_LIST ; COMPOUND_VAR_LIST
             |  ϵ

SIMPLE_ID_LIST →   , SIMPLE_ID_DEC SIMPLE_ID_LIST
             |  ϵ

SIMPLE_ID_DEC →   id
             |  id [ int_cte ]
             |  id [ int_cte ] [ int_cte ]

COMPOUND_ID_LIST →   , COMPOUND_ID_DEC COMPOUND_ID_LIST
             |  ϵ
```

```
COMPOUND_ID_DEC →   id

SIMPLE_TYPE →   int
             |  float
             |  char

COMPOUND_TYPE →   id

FUNCS       →   FUNC FUNCS
             |  ϵ

FUNC        →   void func id ( PARAMS ) DEC_VARS { FUNC_STATEMENTS }
             |  SIMPLE_TYPE func id ( PARAMS ) DEC_VARS { FUNC_STATEMENTS }

PARAMS      →   var <- SIMPLE_TYPE SIMPLE_ID_DEC SIMPLE_ID_LIST ; SIMPLE_VAR_LIST ->
             |  ϵ

FUNC_STATEMENTS →   STATEMENTS RETURN_STATEMENT

RETURN_STATEMENT →   return EXPRESSION ; FUNC_STATEMENTS
             |  ϵ

STATEMENTS  →   STATEMENT STATEMENTS
             |  ϵ

STATEMENT   →   ASSIGNMENT
             |  VOID_FUNC_CALL
             |  IO
             |  CONTROL
             |  ITERATION

EXPRESSION  →   BOOL_EXP
             |  BOOL_EXP | EXPRESSION

BOOL_EXP    →   GENERAL_EXP
             |  GENERAL_EXP & BOOL_EXP

GENERAL_EXP →   EXP
             |  EXP > EXP
             |  EXP < EXP
             |  EXP == EXP
             |  EXP != EXP

EXP         →   TERM
             |  TERM + EXP
             |  TERM - EXP

TERM        →   FACTOR
             |  FACTOR * TERM
             |  FACTOR / TERM

FACTOR      →   ( EXPRESSION )
             |  int_cte
             |  float_cte
             |  VAR_NAME
             |  id ( PARAMS_CALL )
             |  id . id ( PARAMS_CALL )

PARAMS_CALL →   EXPRESSION
             |  EXPRESSION , PARAMS_CALL
             |  ϵ

ASSIGNMENT  →   VAR_NAME = EXPRESSION ;

VAR_NAME    →   SIMPLE_ID
             |  COMPOUND_ID

SIMPLE_ID   →   id
             |  id [ EXPRESSION ]
             |  id [ EXPRESSION ] [ EXPRESSION ]

COMPOUND_ID →   id . SIMPLE_ID

IO          →   READ
             |  PRINT

READ        →   read ( VAR_NAMES ) ;

VAR_NAMES   →   VAR_NAME , VAR_NAMES
             |  VAR_NAME

PRINT       →   print ( PRINT_PARAMS ) ;

PRINT_PARAMS →   EXPRESSION , PRINT_PARAMS
             |  EXPRESSION
             |  string_cte , PRINT_PARAMS
             |  string_cte

CONTROL     →   if ( EXPRESSION ) { STATEMENTS } ELSE ;

ELSE        →   else { STATEMENTS }
             |  ϵ

ITERATION   →   WHILE
             |  FOR

WHILE       →   while ( EXPRESSION ) { STATEMENTS }

FOR         →   for ( VAR_NAME = EXPRESSION until EXPRESSION ) { STATEMENTS }
```

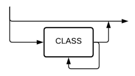
Syntax diagrams

https://lucid.app/lucidchart/invitations/accept/inv_bb674a1c-b067-4bb1-82bf-92e6cb51cdd1

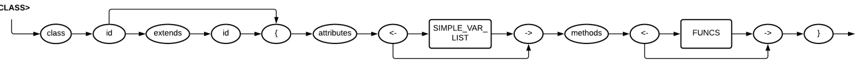
<PROGRAM>



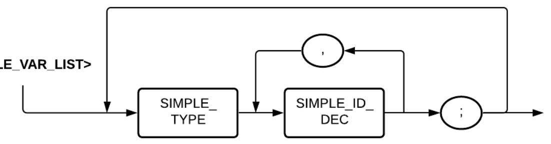
<CLASSES>



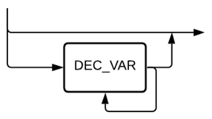
<CLASS>



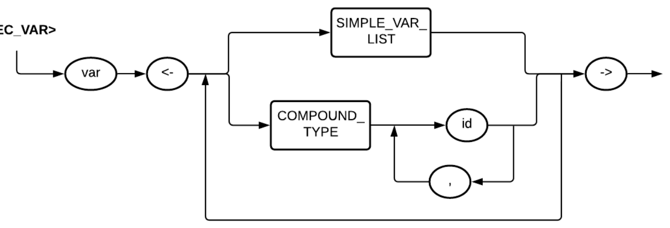
<SIMPLE_VAR_LIST>



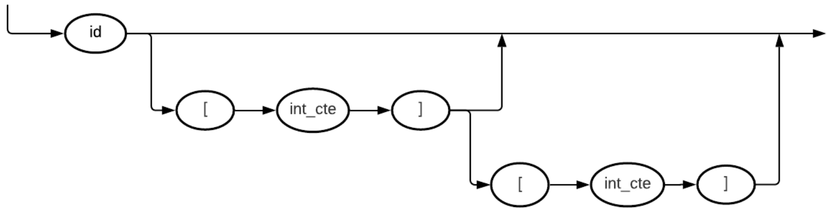
<DEC_VARS>



<DEC_VAR>

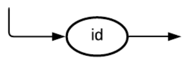


<SIMPLE_ID_DEC>



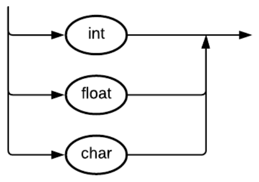
id
[
int_cte
]
[
int_cte
]

<COMPOUND_TYPE>



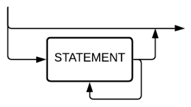
id

<SIMPLE_TYPE>



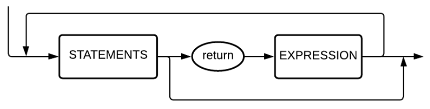
int
float
char

<STATEMENTS>



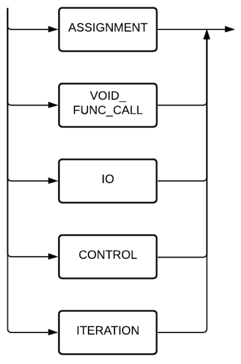
STATEMENT

<FUNC_STATEMENTS>



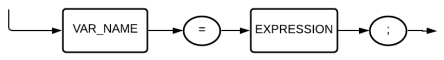
STATEMENTS
return
EXPRESSION

<STATEMENT>



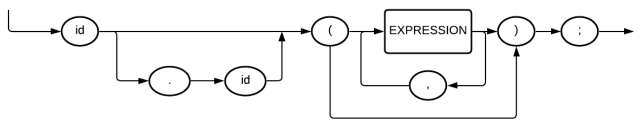
ASSIGNMENT
VOID_
FUNC_CALL
IO
CONTROL
ITERATION

<ASSIGNMENT>

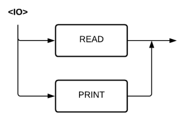


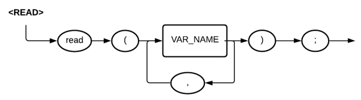
VAR_NAME
=
EXPRESSION
;

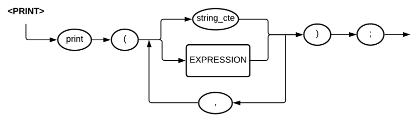
<VOID_FUNC_CALL>

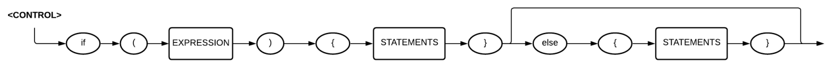


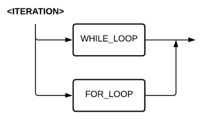
id
.
id
(
EXPRESSION
,
)
;

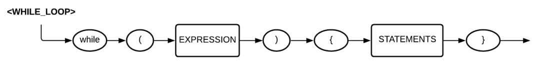

<IO>
READ
PRINT

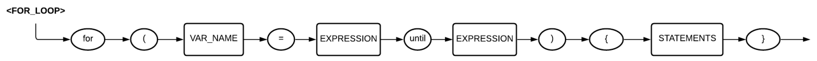

<READ>
read
(
VAR_NAME
,
)
;

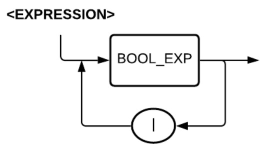

<PRINT>
print
(
string_cte
EXPRESSION
,
)
;

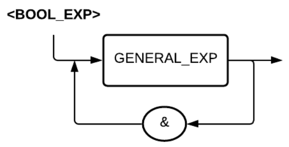

<CONTROL>
if
(
EXPRESSION
)
{
STATEMENTS
}
else
{
STATEMENTS
}

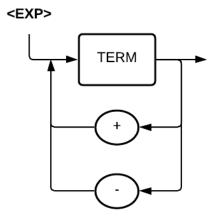

<ITERATION>
WHILE_LOOP
FOR_LOOP

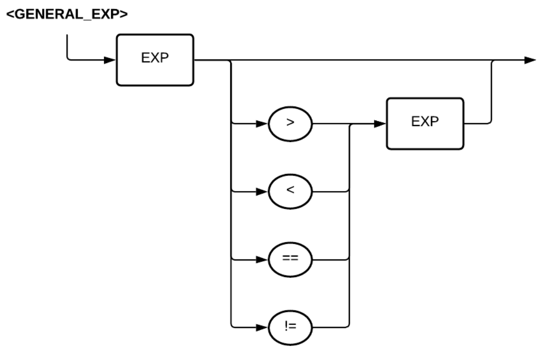

<WHILE_LOOP>
while
(
EXPRESSION
)
{
STATEMENTS
}


<FOR_LOOP>
for
(
VAR_NAME
=
EXPRESSION
until
EXPRESSION
)
{
STATEMENTS
}

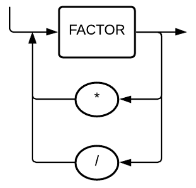

<EXPRESSION>
BOOL_EXP
|


<BOOL_EXP>
GENERAL_EXP
&

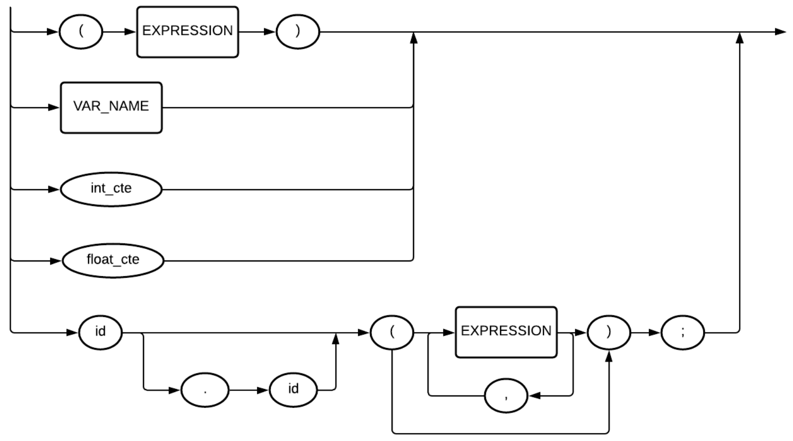

<EXP>
TERM
+
-


<GENERAL_EXP>
EXP
>
<
==
!=
EXP

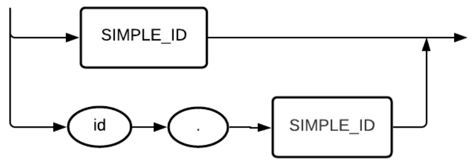
<TERM>


FACTOR
*
/

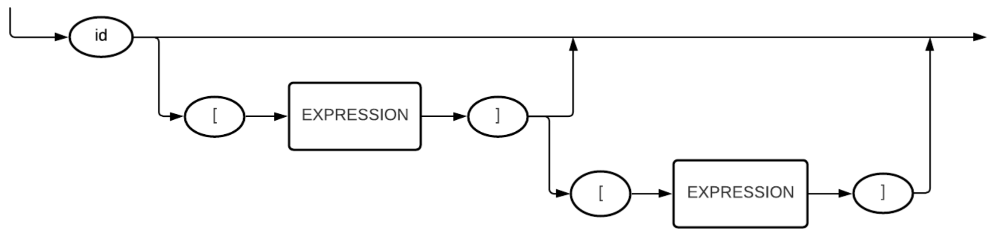
<FACTOR>


(
EXPRESSION
)
VAR_NAME
int_cte
float_cte
id
.
id
(
EXPRESSION
,
)
;

<VAR_NAME>


SIMPLE_ID
id
.
SIMPLE_ID

<SIMPLE_ID>


id
[
EXPRESSION
]
[
EXPRESSION
]