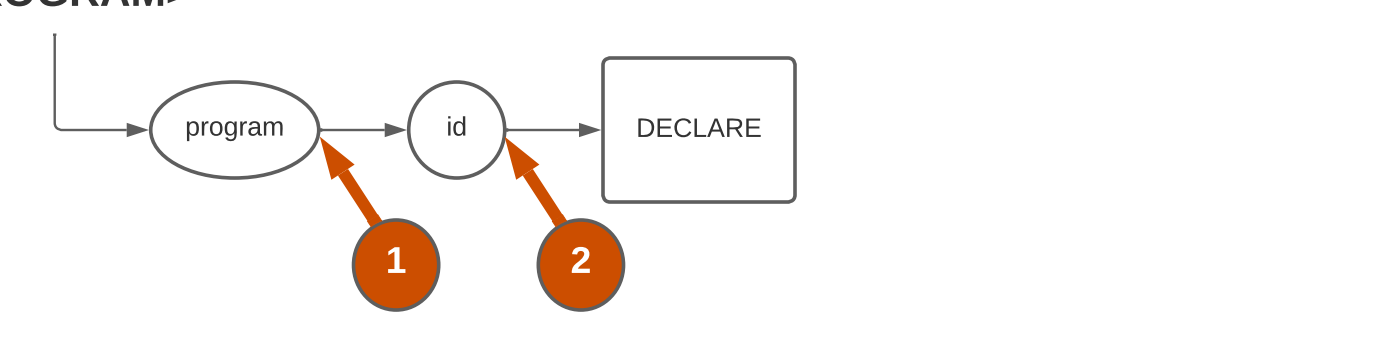
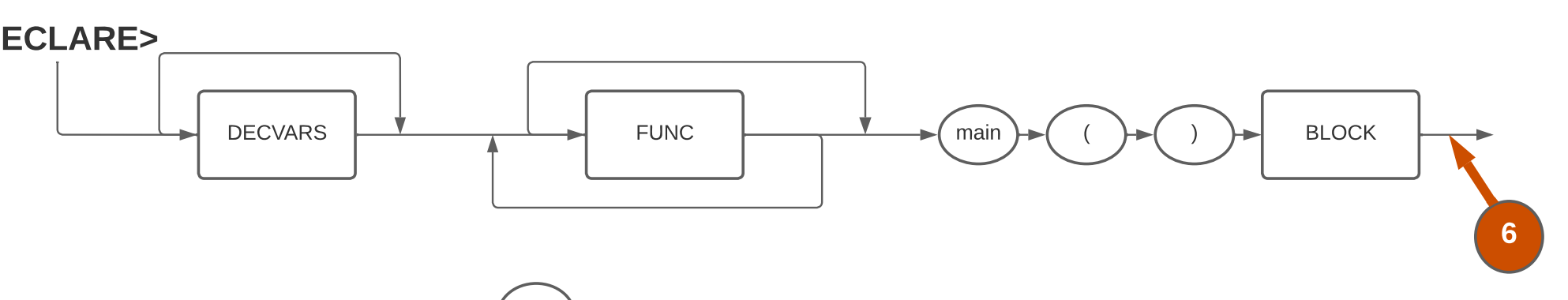
1. Create DirFunc
2. Add id-name and type program a DirFunc
3. If current Func doesn't have a VarTable then
   a. Create VarTable and link it to current Func
4. Current-type = type
5. Search for id-name in current VarTable
   a. if found -> Error "multiple declaration"
   b. if not, add id-name and current-type to current VarTable
6. Delete DirFunc and current VarTable (Global)

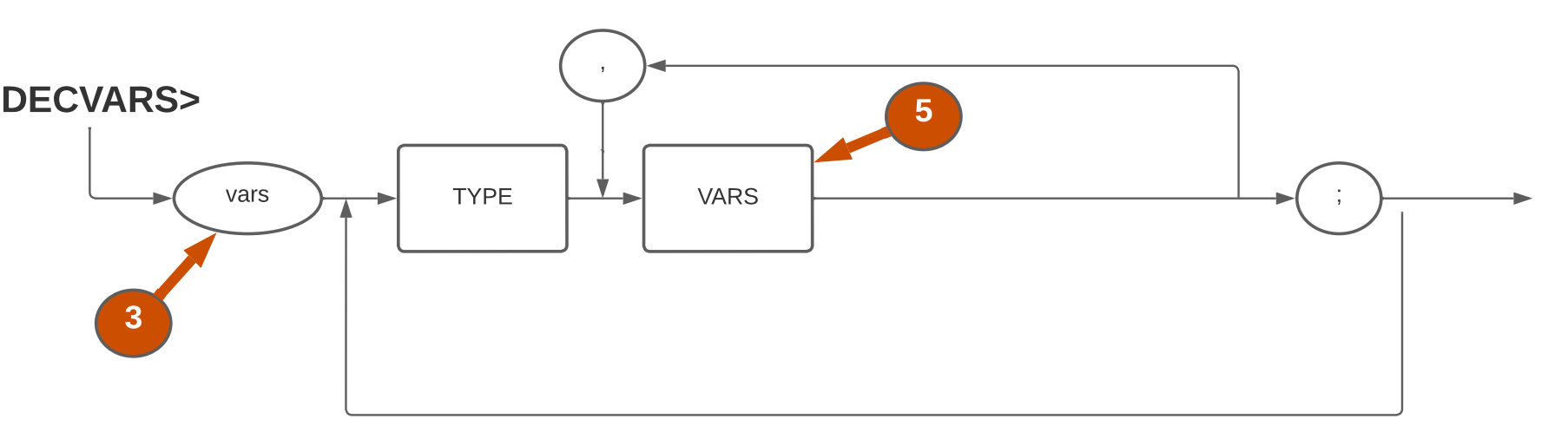
<PROGRAM>



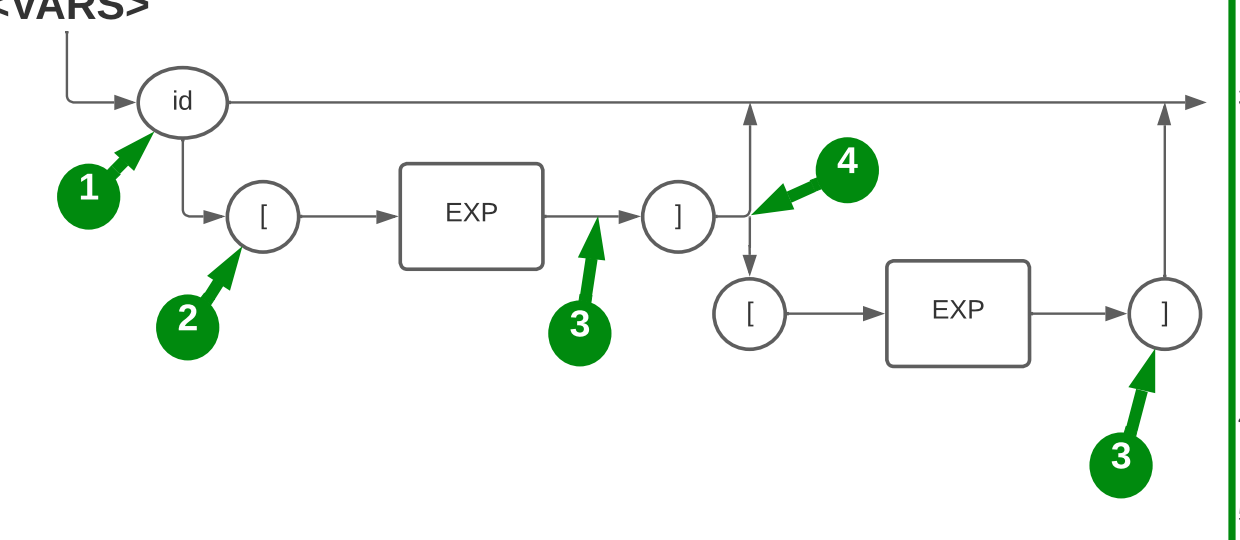
<DECLARE>



<DECVARS>

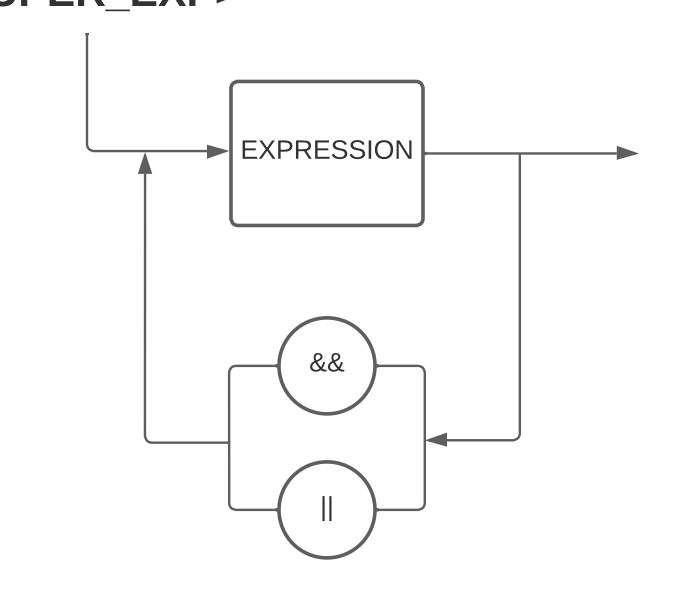


<VARS>

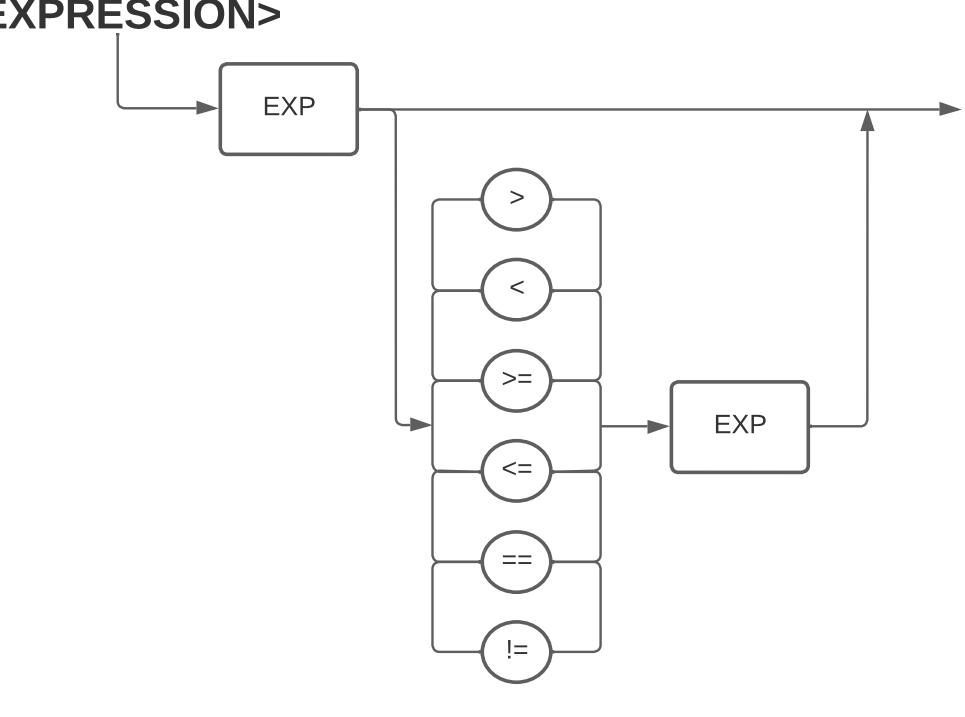


1. PilaO.push(id) and PTipos.push(type)
2. id = PilaO.pop(); type = PTipos.pop()
   verify that id has dimensions
   DIM = 1
   PilaDim.push(id, DIM)
   Get the first node of dimensions (list)
   POper.push(FakeBottom)
3. Create Quadruple:
   Verify PilaO.Top LiDIM LsDIM ** NO Pop at this time
   If NextPointer(List)
   { aux = PilaO.Pop()
   Create quadruple: * aux mDIM Tj | where Tj is next available temp
   PilaO.Push(Tj)}
   If DIM > 1
   { aux2 = PilaO.Pop(); aux1 = PilaO.Pop();
   Create quadruple: + aux1 aux2 Tk | where Tk is next available temp
   PilaO.Push(Tk)}
4. DIM = DIM + 1
   Update DIM in PilaDIM
   Move to next node in List
5. Aux1 = PilaO.Pop()
   Create quadruples;
   + aux1 K Ti | where Ti is next available temp. K is the offset
   + Ti VirtualAddress Tn | where Tn is next available temp
   PilaO.Push((Tn)). | Contains an Address, thats why we use () to distinguish it
   POper.Pop() | Eliminates fake bottom

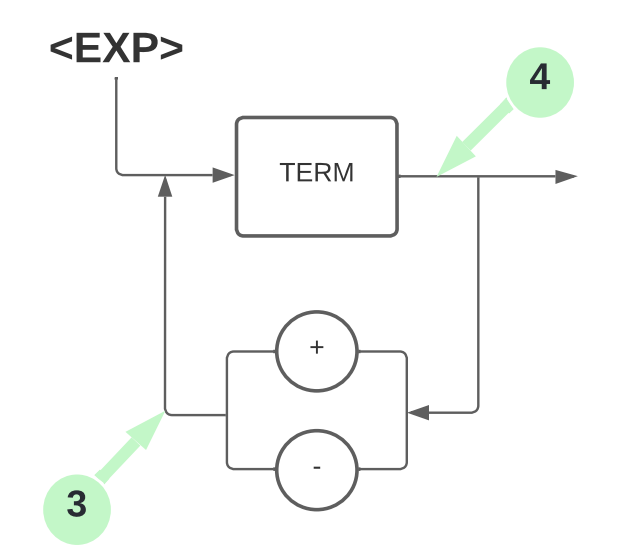
<SUPER_EXP>



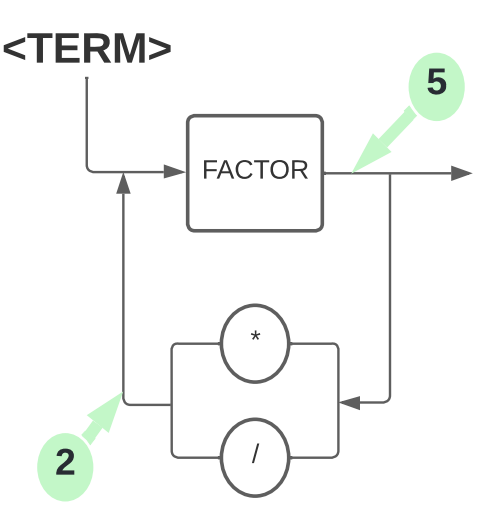
<EXPRESSION>



<EXP>

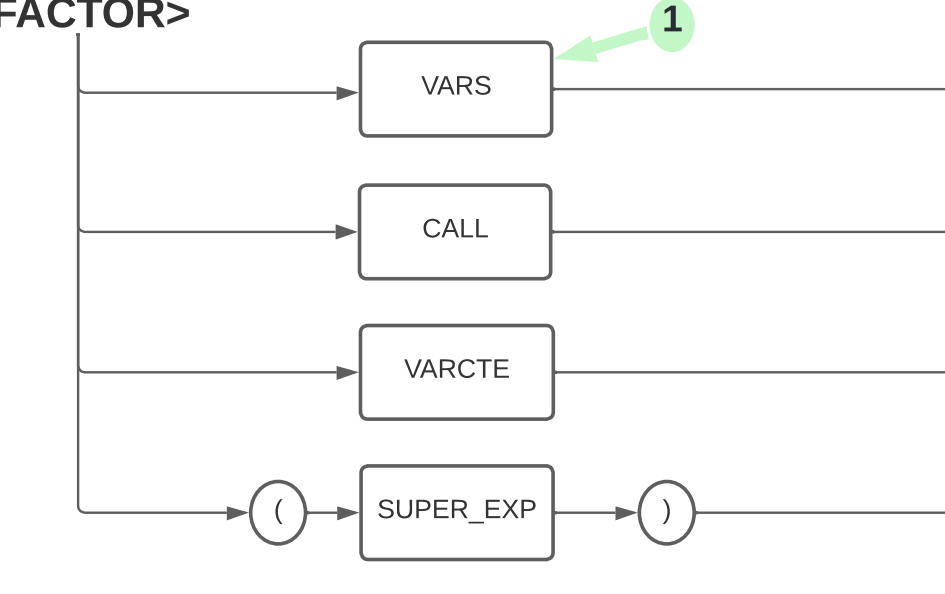


<TERM>

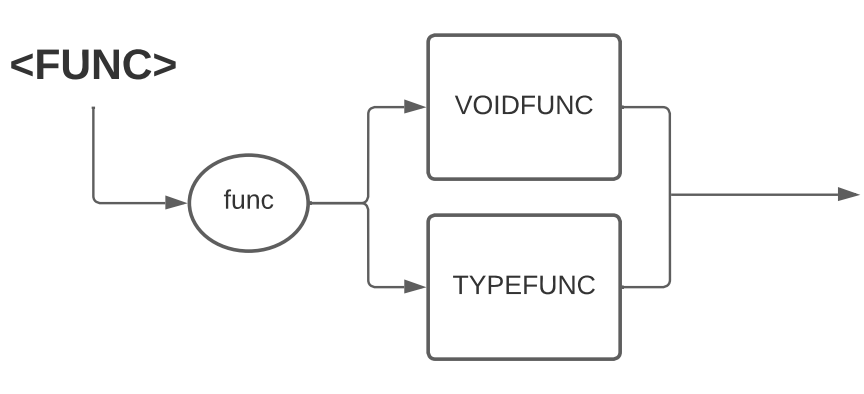


1. PilaO.Push(id.name) and PTypes.Push(id.type)
2. POper.Push(+ or -)
3. POper.Push(* or /)
4. If POper.top() == '+' or '-' then
   a.
      i. right_operand = PilaO.Pop()
      ii. right_Type = PTypes.Pop()
   b.
      i. left_operand = PilaO.Pop()
      ii. left_Type = PTypes.Pop()
   c. operator = POper.Pop()
   d. result_Type = Semantics[left_type, right_type, operator]
   e. if (result_Type != ERROR)
      i. result <-- AVAIL.next()
      ii. generate quad
         1. (operator, left_op, right_op, result)
      iii. Quad.Push(quad)
      iv.
         1. PilaO.Push(result)
         2. PTypes.Push(result_Type)
      v. If any operand were a temporal space, return it to AVAIL
   f. Else
      i. ERROR ("Type mismatch")
5. If POper.top()== '*' or '/' then
   a. Sames as 4., but with * and /

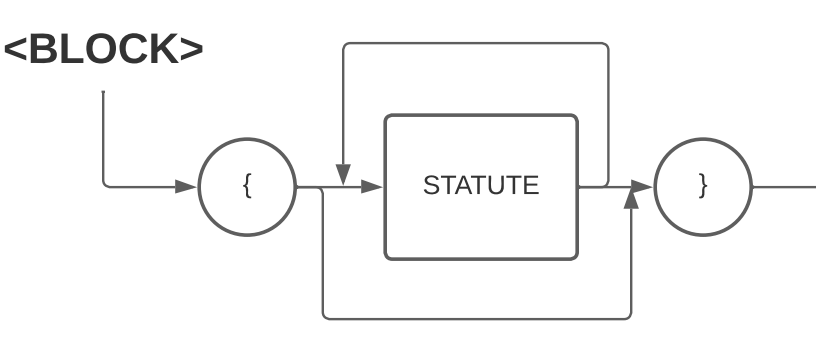
<FACTOR>



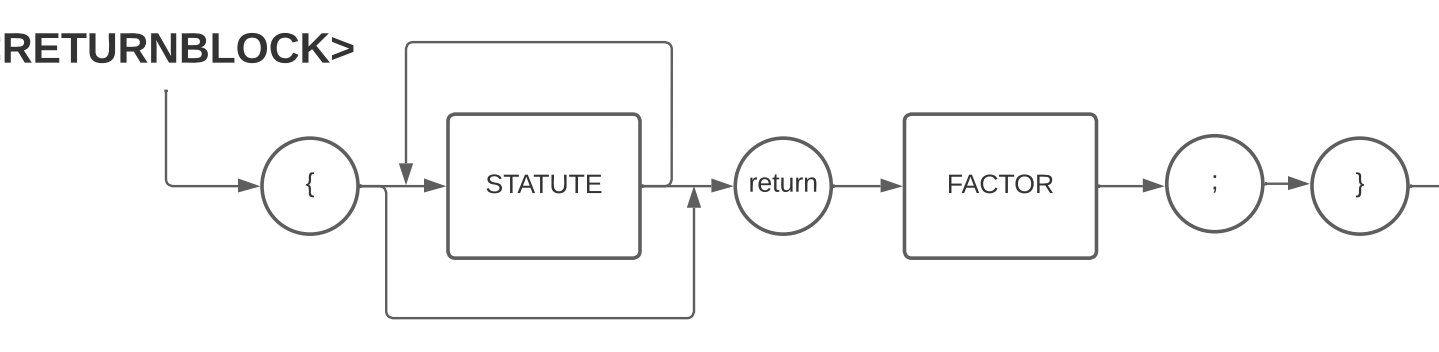
<FUNC>



<BLOCK>

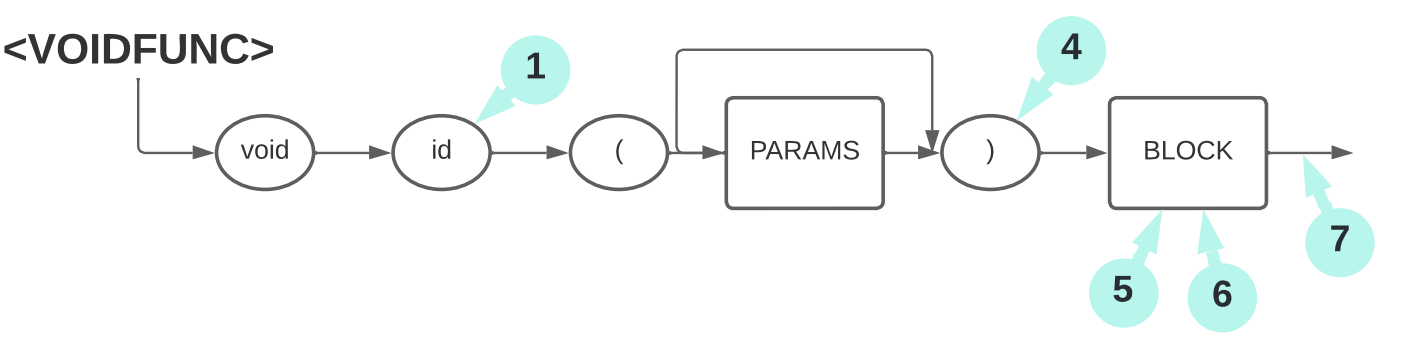


<RETURNBLOCK>

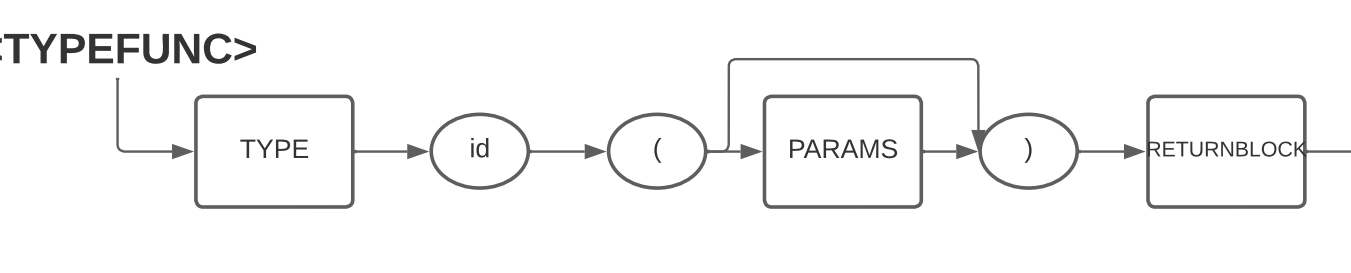


1. Insert Function name into the DirFunc table (and its type, if any), verify semantics
2. Insert every parameter into the current (local) VarTable
3. Insert the type to every parameter uploaded into the VarTable
   At the same time into the ParameterTable (to create the Function's signature)
4. Insert into DirFunc the number of parameters defined. **to calculate the workspace required for execution
5. Insert into DirFunc the current quadruple counter (CONT), **to establish where the function starts
6. Insert into DirFunc the current quadruple counter (CONT), **to establish where the function starts
7. Release the current VarTable (local)
   Generate an action to end the function (ENDFunc)
   Insert into DirFunc the number of temporal vars used. **to calculate the workspace required for execution

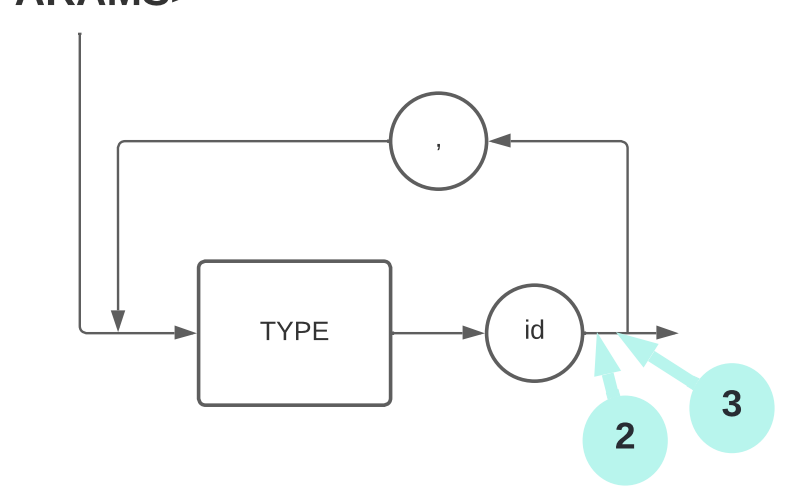
<VOIDFUNC>



<TYPEFUNC>

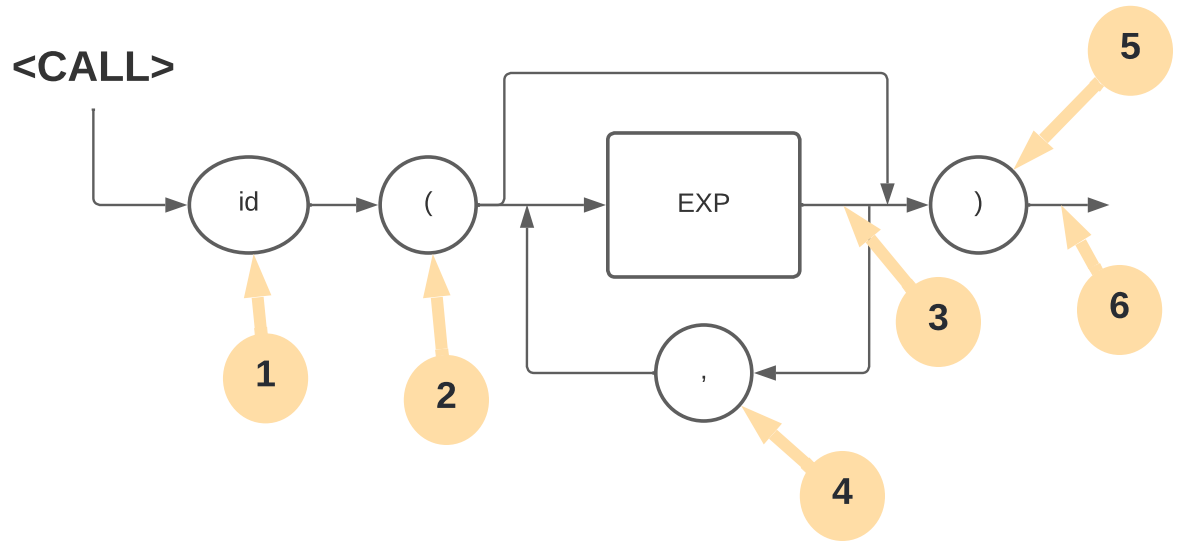


<PARAMS>

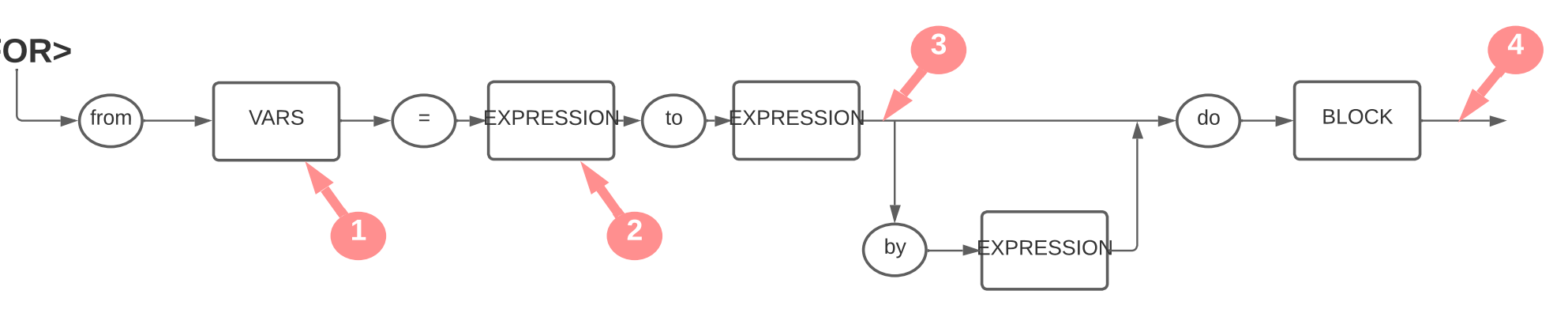


1. Verify that the function exists into the DirFunc
2. Generate action ERA size (Activation Record expansion -NEW- size)
   Start the parameter counter (k) in 1
   Add a pointer to the first parameter tpye in the ParameterTable
3. Argument = PilaO.Pop() ArgumentType = PTypes.Pop()
   Verify ArgumentType against current Parameter (#k) in ParameterTable
   Generate action PARAMETER, Argument, Argument #k
4. K = K + 1, move to next parameter
5. Verify that the last parameter points to null (coherence in number of parameters)
6. Generate action GOSUB, procedure-name, , initial-address

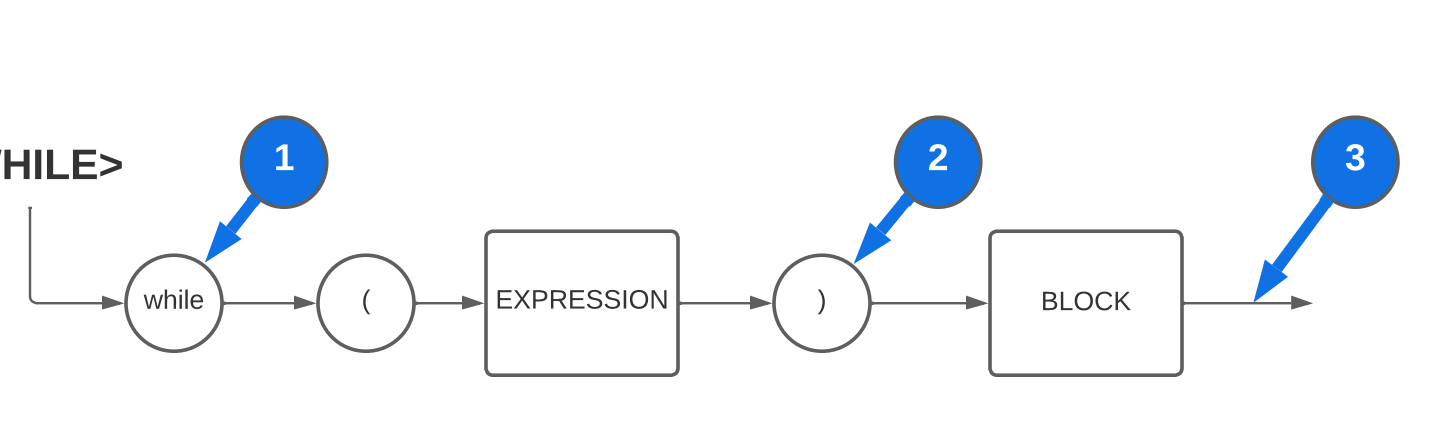
<CALL>



<FOR>



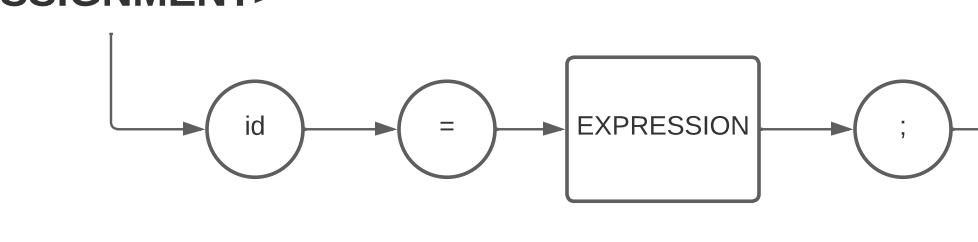
<WHILE>



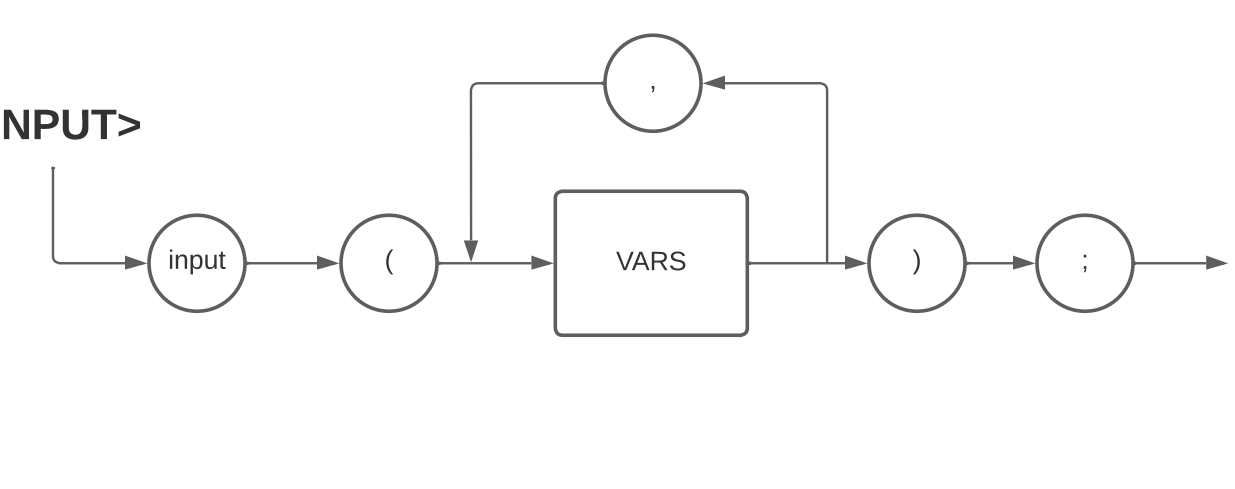
1. PilaO.push(id) PTypes.push(tipo id)
   Validar que tipo sea numerico,
   si no es ERROR (Type-mismatch)
2. Exp_type = PTypes.Pop()
   if(Exp_type != numerico)
   ERROR(Type-mismatch)
   else
      Exp = PilaO.Pop()
      VControl = PilaO.Top();
      Control_type = PType.pop()
      Tipo_res = Semantica[=, Control_type, Exp_type]
   Si Tipo_Res == ERR --> ERROR (Type-mismatch)
   else
      Genera (=, Exp, VControl)
3. Exp_type = PTypes.Pop()
   if (Exp_type != numerico)
   ERROR(Type-mismatch)
   else
      Exp = PilaO.Pop()
      Genera(=, Exp, , VFinal)
      Genera(<, VControl, VFinal, Tx)
      PJumps.push(cont-1);
      Genera(GotoF, Tx, , __)
      PJumps.push(cont-1);
4. Genera (+, VControl, 1, Ty)
   FIN = PJumps, Pop()
   RET = PJumps.Pop()
   Genera (Goto, RET)
   FILL (FIN, cont)
   idOriginal: PilaO.Pop();
   Genera(=, VControl, , idOriginal)

1. PJumps.Push(cont)
2. exp_type = PTypes.Pop()
   if (exp_type != bool) ERROR (Type-mismatch)
   else
      result = PilaO.Pop()
      Generate quad: GotoF, result, , ___
      PJumps.Push(cont-1)
3. end = PJumps.Pop()
   return = PJumps.Pop()
   Generate quad: GOTO return
      FILL(end, cont)

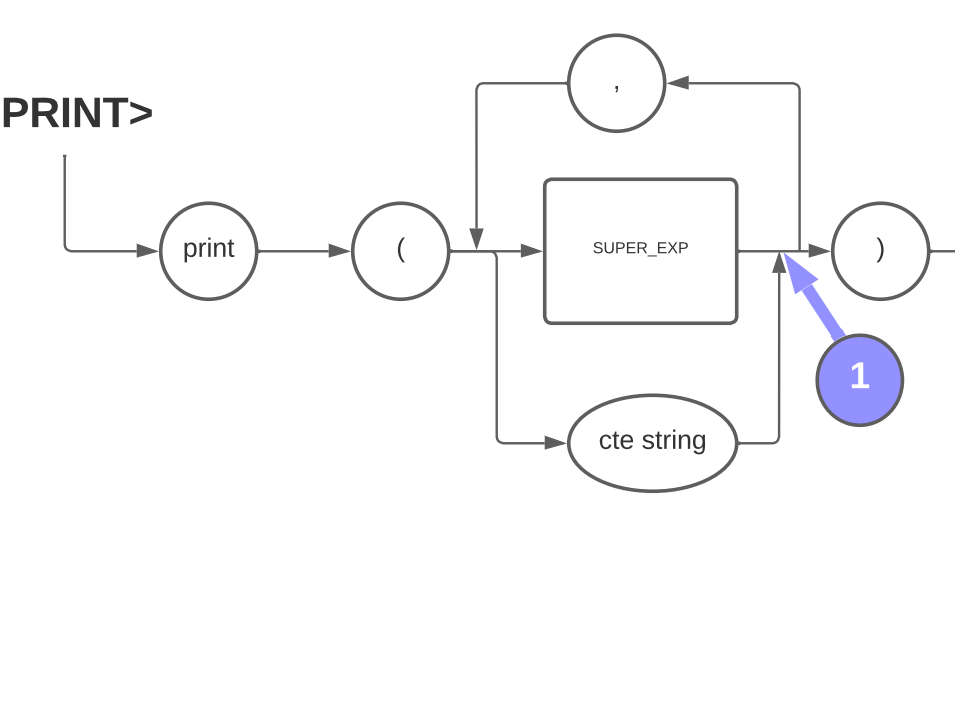
<ASSIGNMENT>



<INPUT>

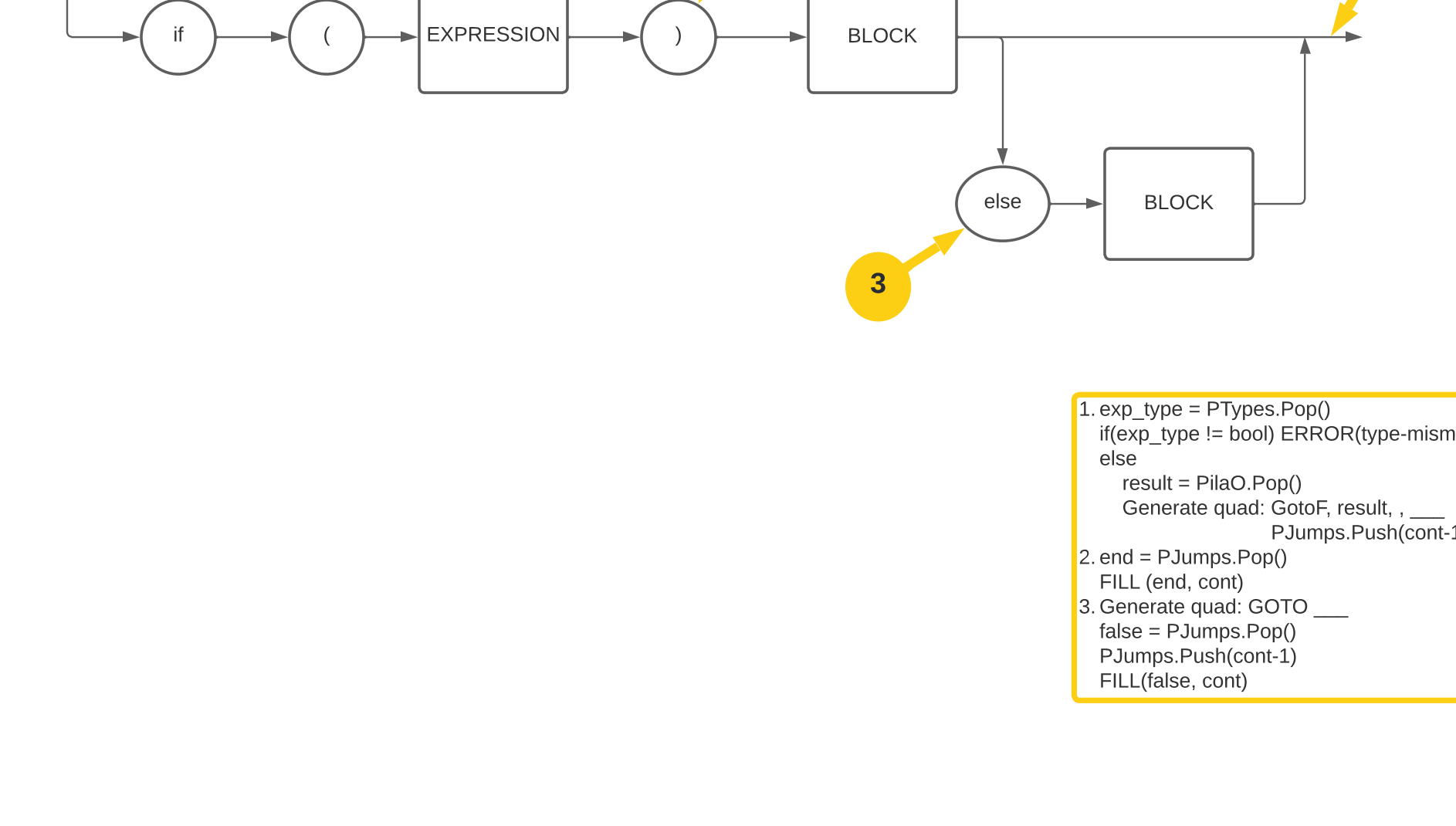


<PRINT>



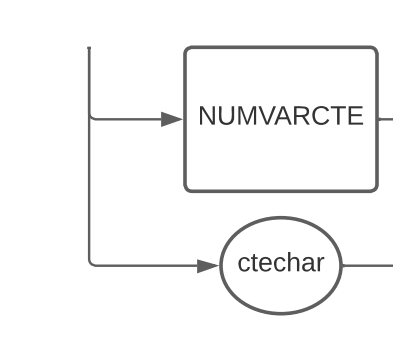
1. Resultado = pop(PilaO)
   a. genera cuadruplo < print, , , resultado >

<CONDITION>

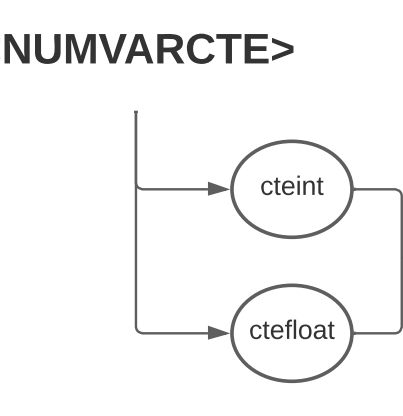


1. exp_type = PTypes.Pop()
   if(exp_type != bool) ERROR(type-mismatch)
   else
      result = PilaO.Pop()
      Generate quad: GotoF, result, , ___
      PJumps.Push(cont-1)
2. end = PJumps.Pop()
   FILL (end, cont)
3. Generate quad: GOTO ___
   false = PJumps.Pop()
   PJumps.Push(cont-1)
   FILL(false, cont)

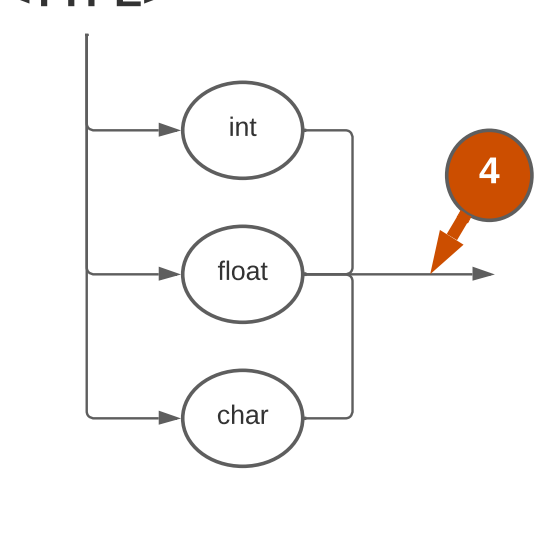
<VARCTE>



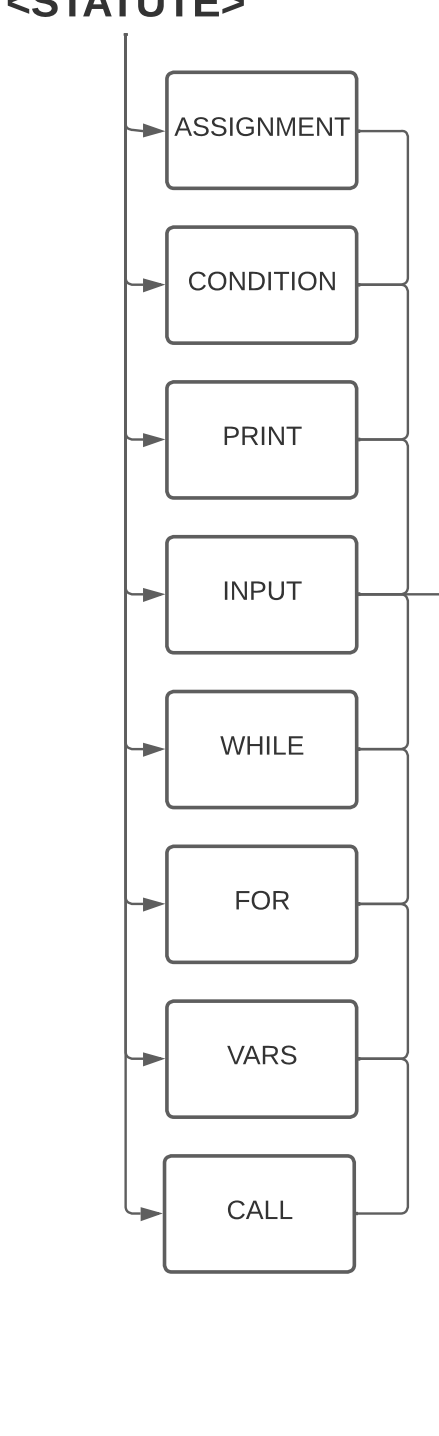
<NUMVARCTE>



<TYPE>



<STATUTE>



<SPECIALFUNCS>