Pseudocódigos para expresiones aritméticas

Con construcción de árbol

```
function exp(): syntaxTree;
var temp, newtemp : syntaxTree;
begin
      temp := term();
      while token == + or token == – do
            match(token);
            newtemp := makeOpNode(token);
            leftChild(newtemp) := temp;
            rightChild(newtemp) := term();
            temp := newtemp;
      end-while;
      return temp;
end-exp;

function term(): syntaxTree;
var temp, newtemp : syntaxTree;
begin
      temp := factor();
      while token == * do
            match(*);
            newtemp := makeOpNode(*);
            leftChild(newtemp) := temp;
            rightChild(newtemp) := factor();
            temp := newtemp;
      end-while;
      return temp;
end-exp;

function factor() : syntaxTree;
var temp : syntaxTree;
begin
      case token of
            ( : match( ( );
                  temp := exp();
                  match( ) );
            number:
                  match(number);
                  temp := makeConstantNode(number);
            else:
                  syntaxError("Unespected token -->");
                  token := getToken();
```

```
      end-case;
      return temp;
end-factor;

function ifStatement() : syntaxTree;
var temp : syntaxTree;
begin
      match(if);
      match( ( );
      temp := makeStmtNode(if);
      testChild(temp) := exp();
      match( ) );
      thenChild(temp) := statement();
      if token == else then
            match(else);
            elseChild(temp) := statement();
      else
            elseChild(temp) := nil;
      end-if;
      return temp;
end-ifStatement;
```