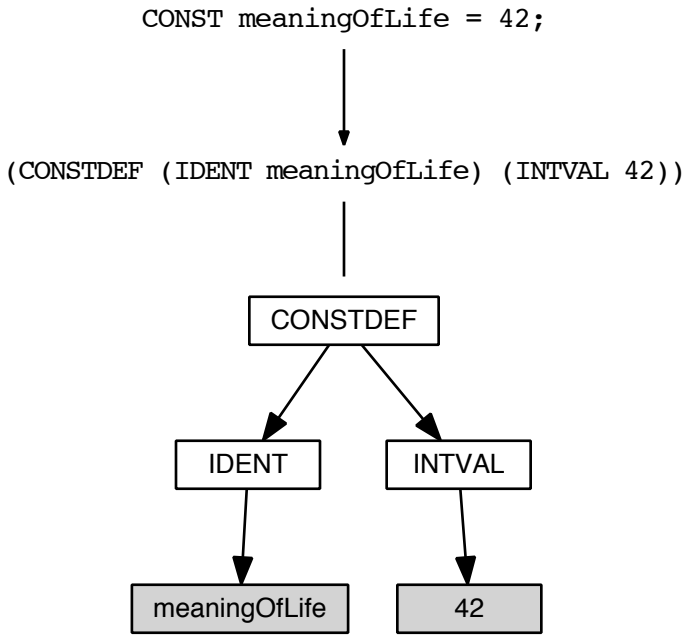
M2C Modula-2 Compiler & Translator

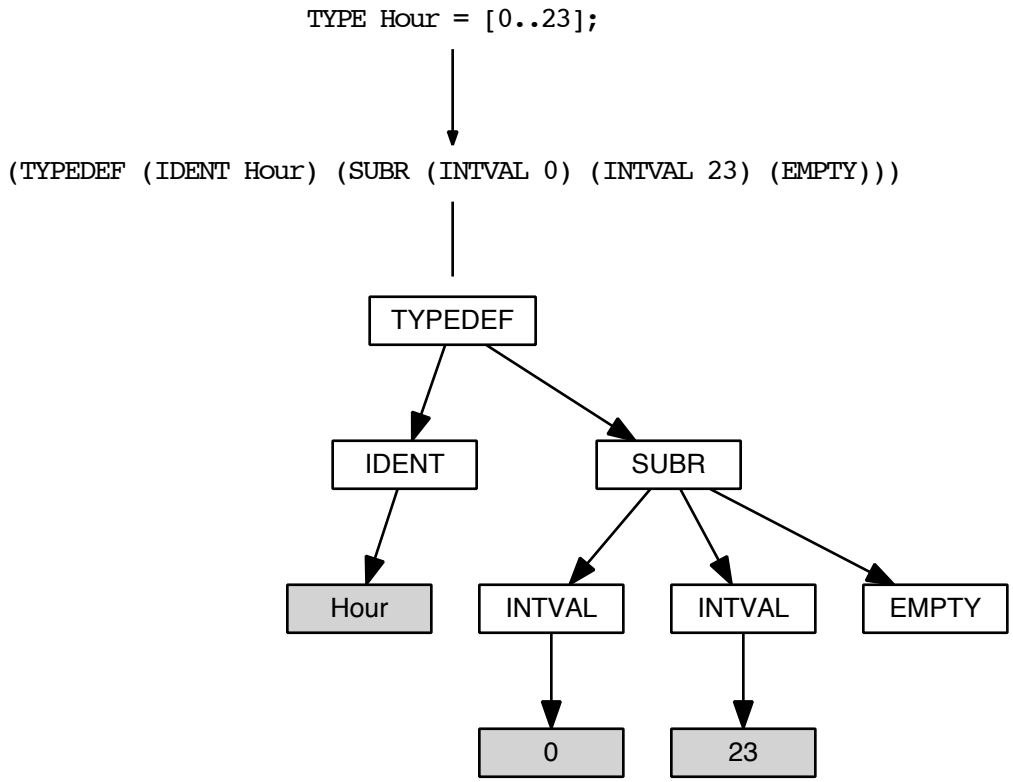
Abstract Syntax Tree Node Example Scenarios

Version 1.01 -- January 12, 2016

Constant Definition AST Node

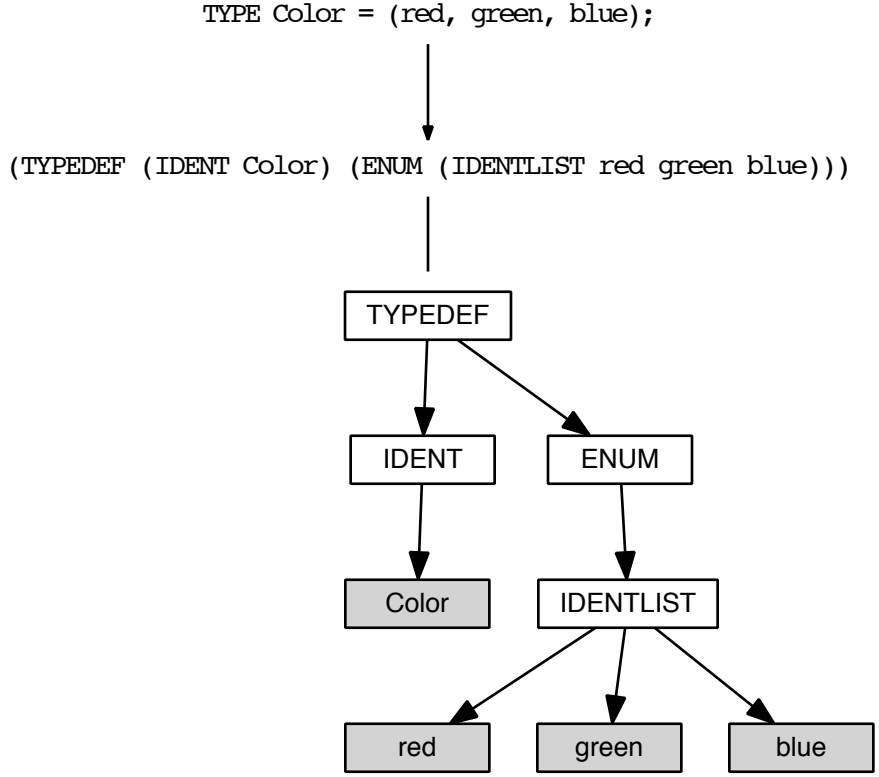


Subrange Type Definition AST Node



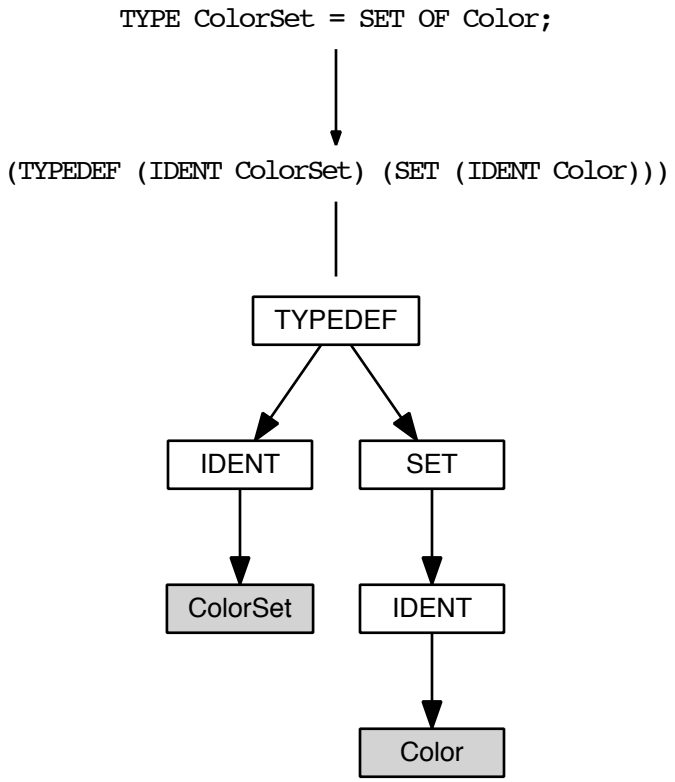
Enumeration Type Definition AST Node

```
TYPE Color = (red, green, blue);
```



```
(TYPEDEF (IDENT Color) (ENUM (IDENTLIST red green blue)))
```

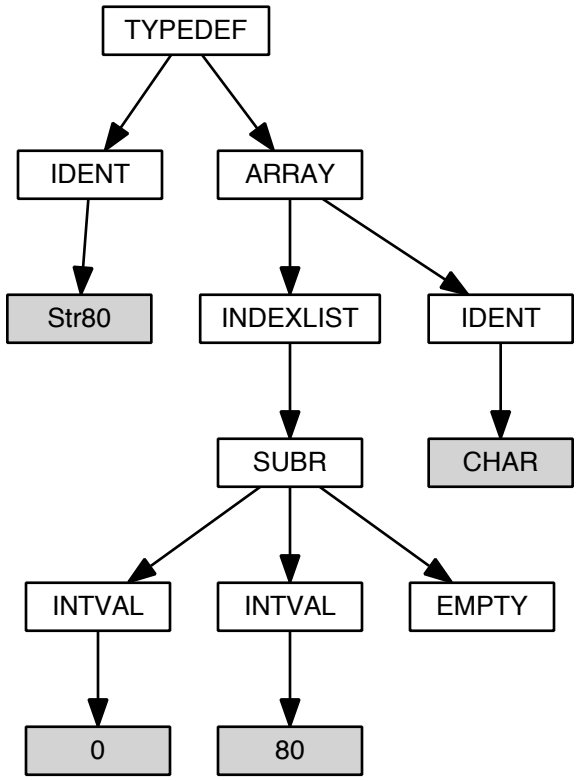
Set Type Definition AST Node



Array Type Definition AST Node

```
TYPE Str80 = ARRAY [0..80] OF CHAR;
```

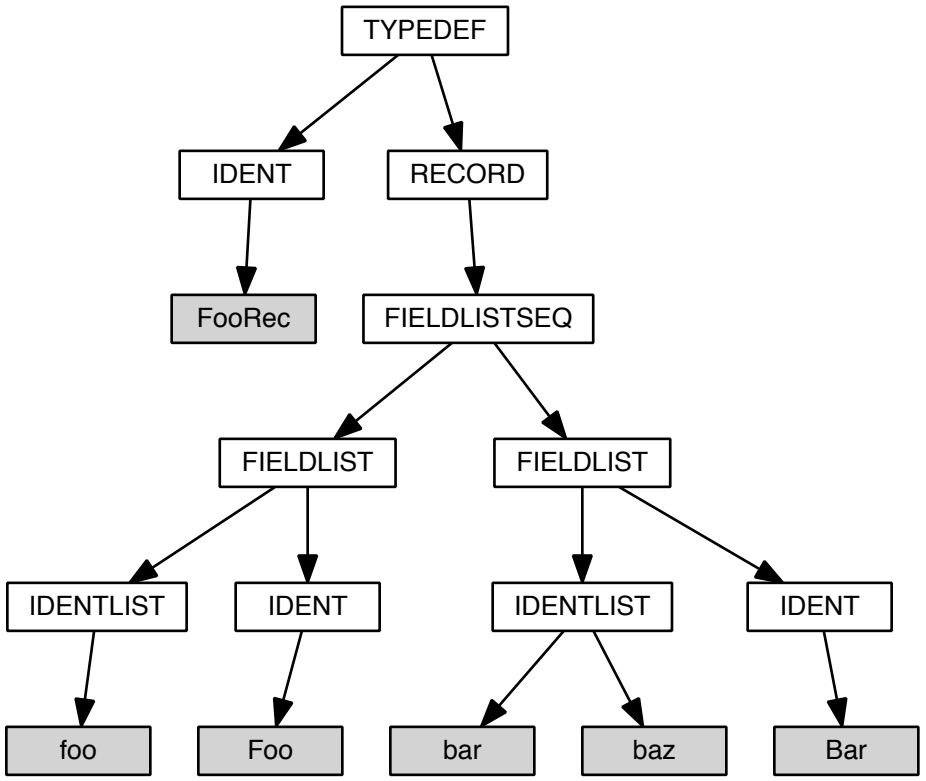
```
(TYPEDEF (IDENT Str80) (ARRAY (INDEXLIST (SUBR (INTVAL 0) (INTVAL 80) (EMPTY))) (IDENT CHAR)))
```



Record Type Definition AST Node

```
TYPE FooRec = RECORD foo : Foo; bar, baz : Bar END;
```

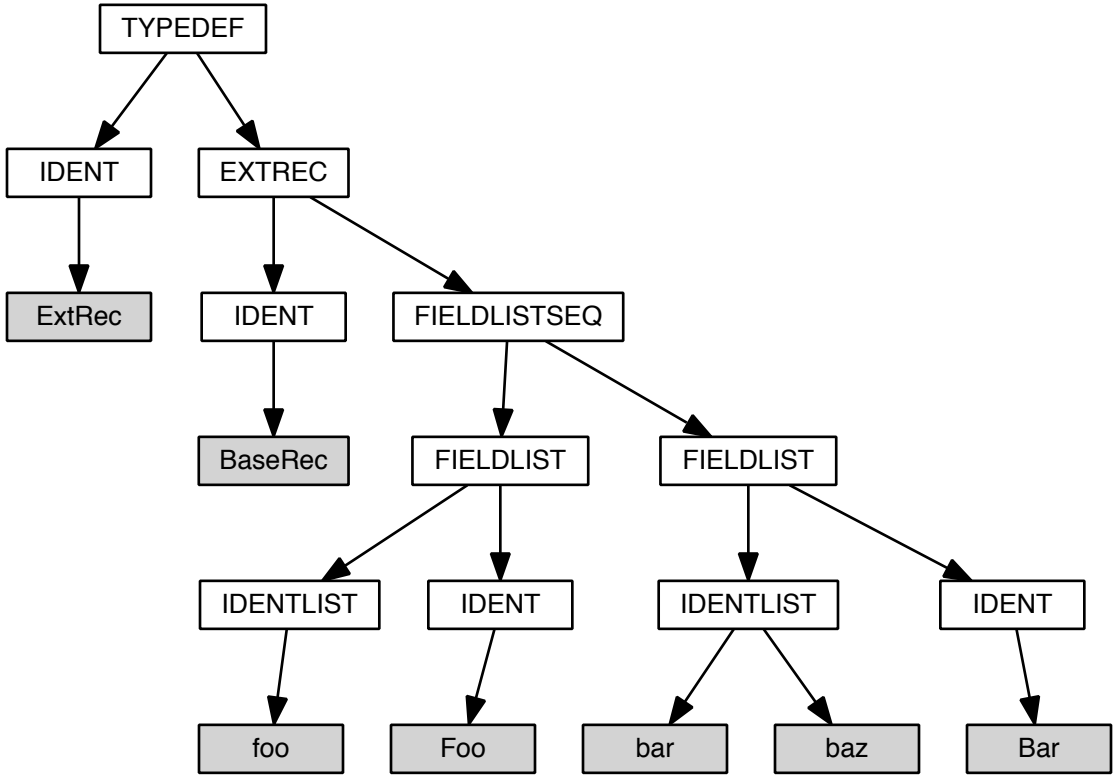
```
(TYPEDEF (IDENT FooRec) (RECORD (FIELDLISTSEQ
     (FIELDLIST (IDENTLIST foo) (IDENT Foo))
   (FIELDLIST (IDENTLIST bar baz) (IDENT Bar)))
```



Extensible Record Definition Type AST Node

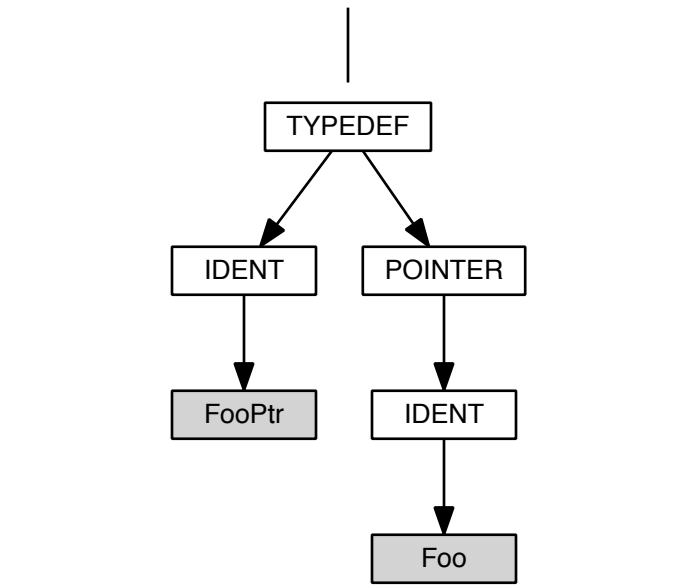
```
TYPE ExtRec = RECORD ( BaseRec ) foo : Foo; bar, baz : Bar END;
```

```
(TYPEDEF (IDENT ExtRec) (EXTREC (IDENT BaseRec) (FIELDLISTSEQ
            (FIELDLIST (IDENTLIST foo) (IDENT Foo))
          (FIELDLIST (IDENTLIST bar baz) (IDENT Bar)))
```



Pointer Type Definition AST Node

```
TYPE FooPtr = POINTER TO Foo;
```

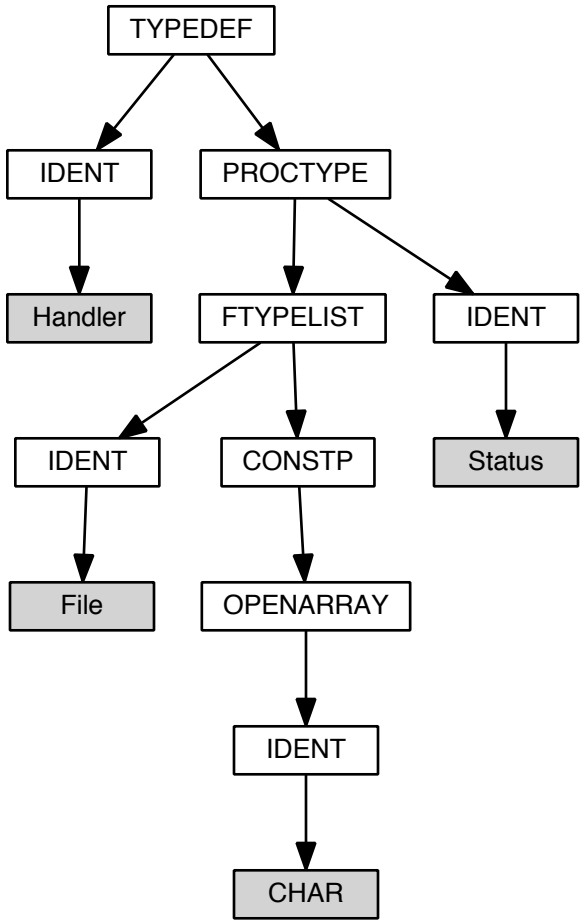


```
(TYPEDEF (IDENT FooPtr) (POINTER (IDENT Foo)))
```

Procedure Type Definition AST Node

```
TYPE Handler = PROCEDURE ( File, CONST ARRAY OF CHAR ) : Status;
```

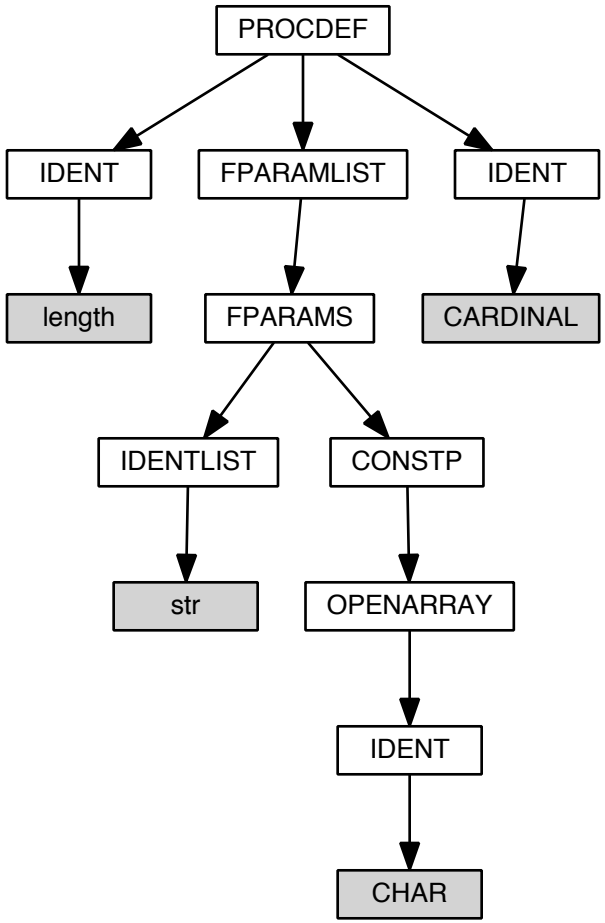
```
(TYPEDEF (IDENT Handler) (PROCTYPE (FTYPELIST (IDENT File)
   (CONSTP (OPENARRAY (IDENT CHAR)))) (IDENT Status)))
```



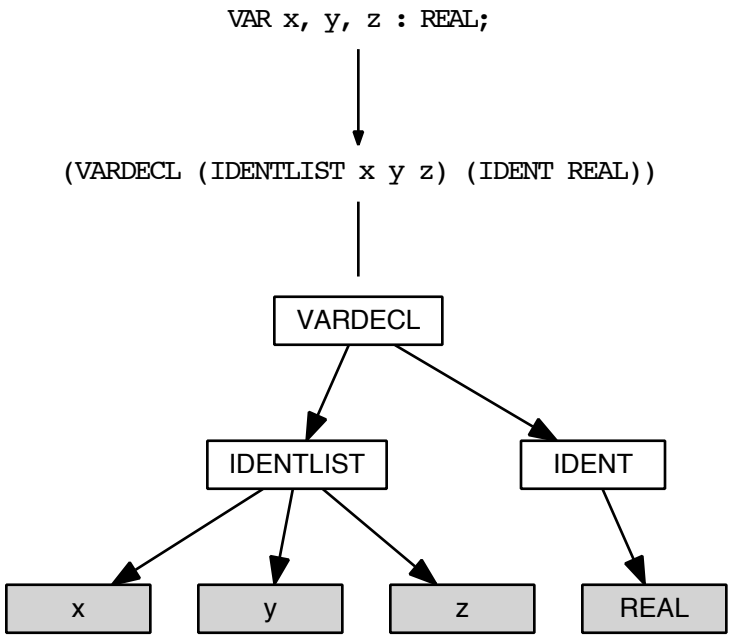
Procedure Definition AST Node

```
PROCEDURE length ( str : CONST ARRAY OF CHAR ) : CARDINAL;
```

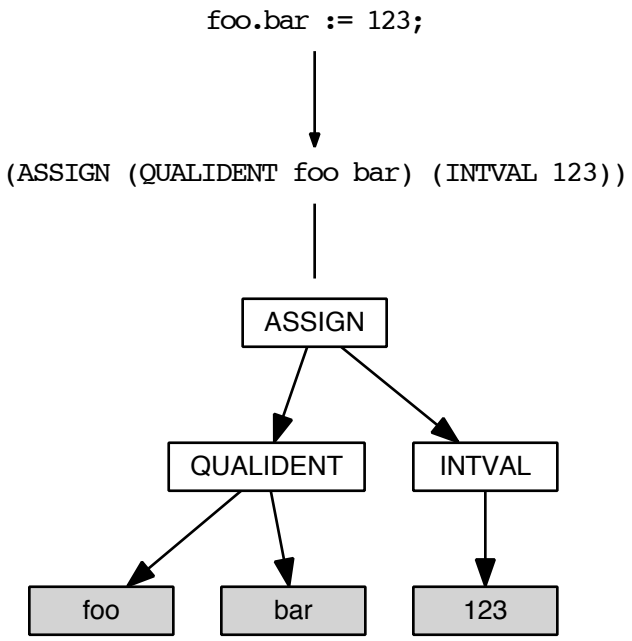
```
(PROCDEF (IDENT length) (FPARAMLIST (FPARAMS (IDENTLIST str)
   (CONSTP (OPENARRAY (IDENT CHAR))))) (IDENT CARDINAL))
```



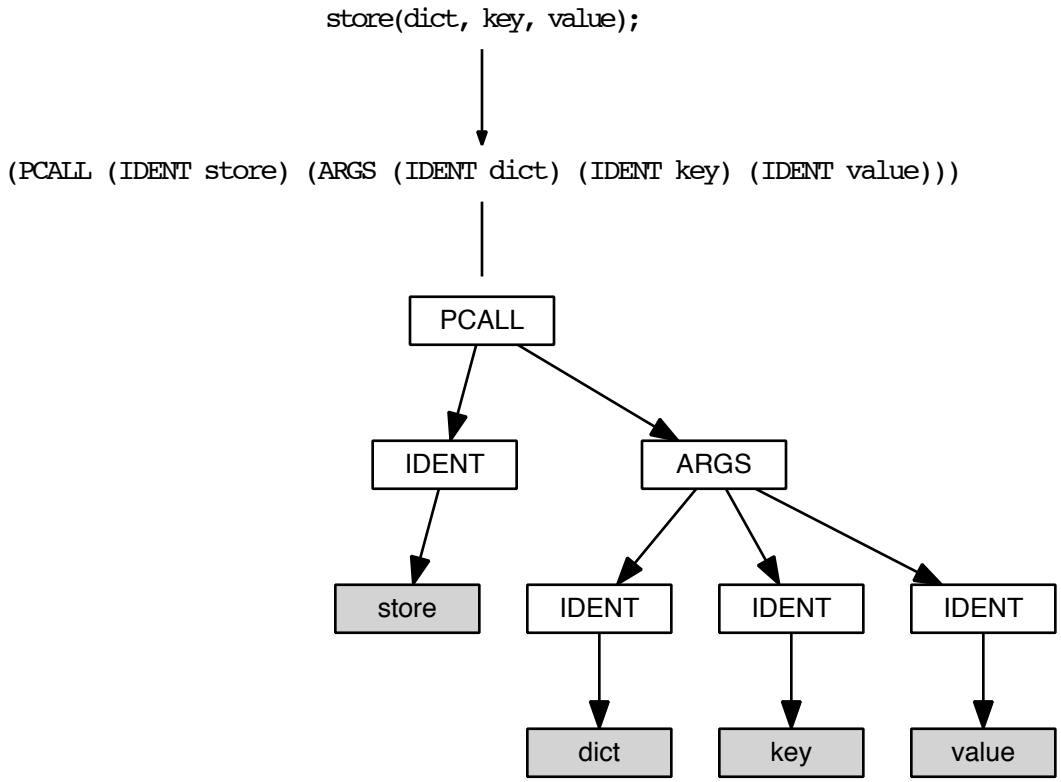
Variable Declaration AST Node



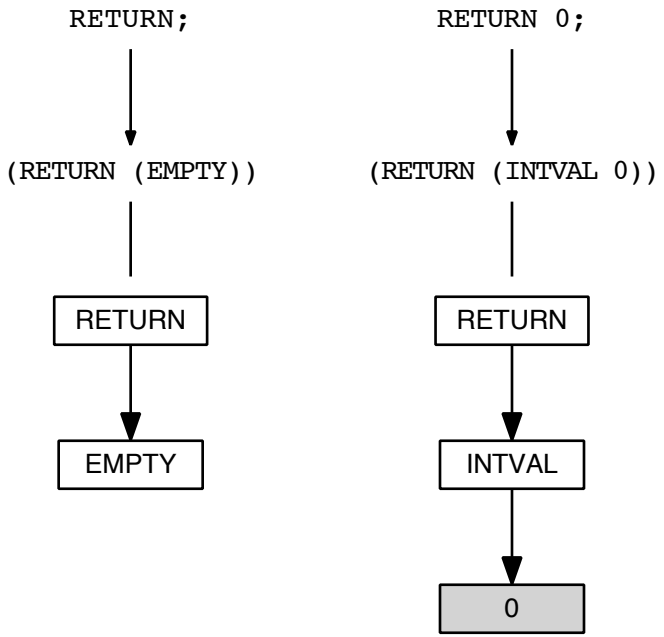
Assignment Statement AST Node



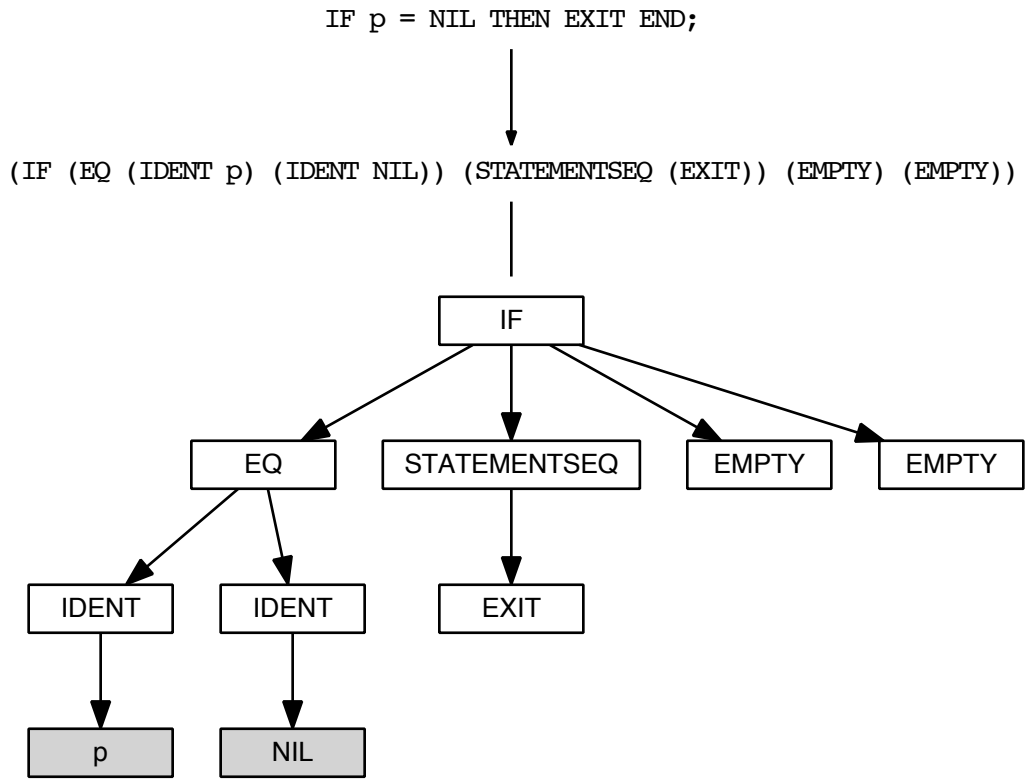
Procedure Call Statement AST Node



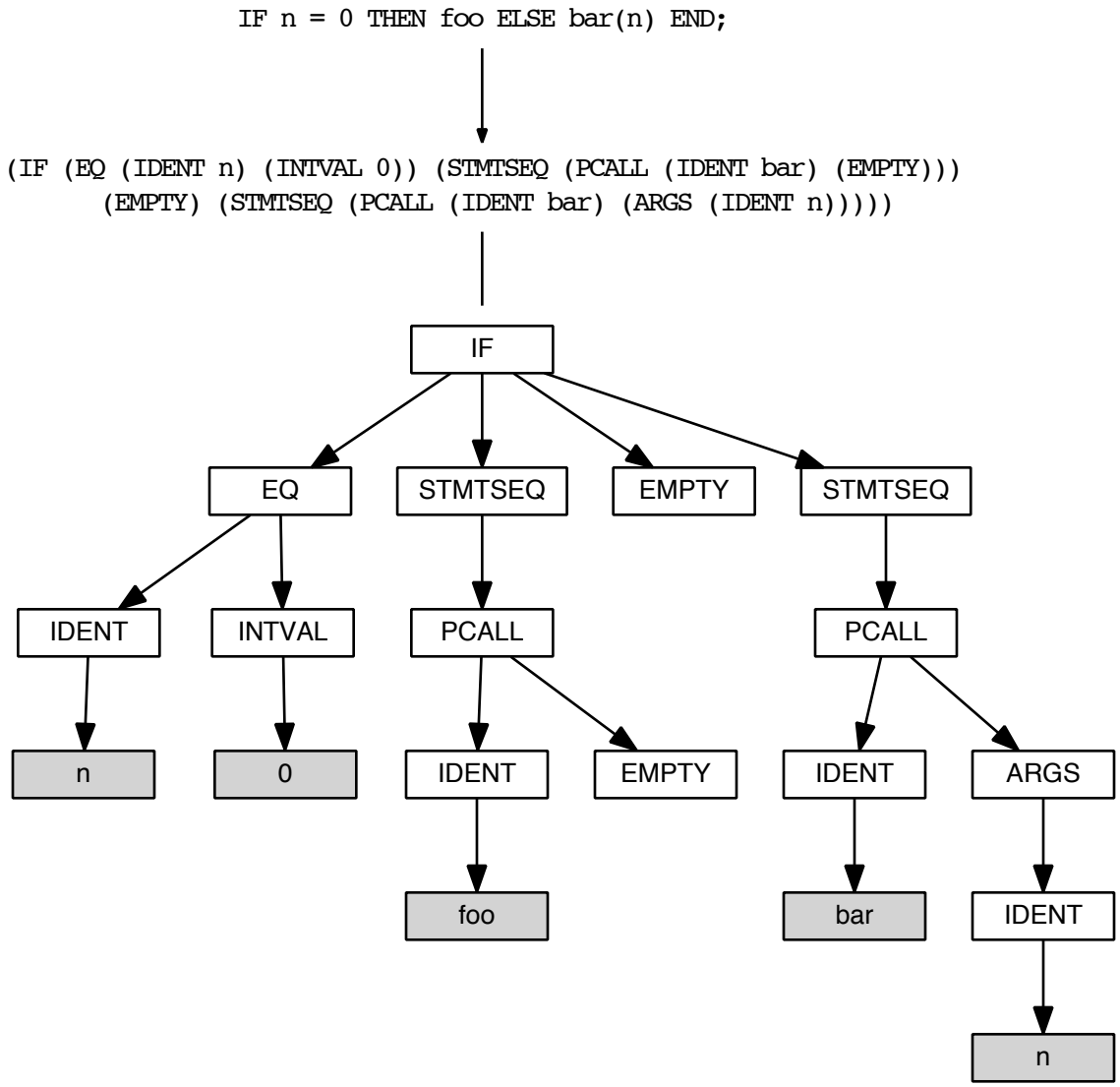
Return Statement AST Node



If Statement AST Node



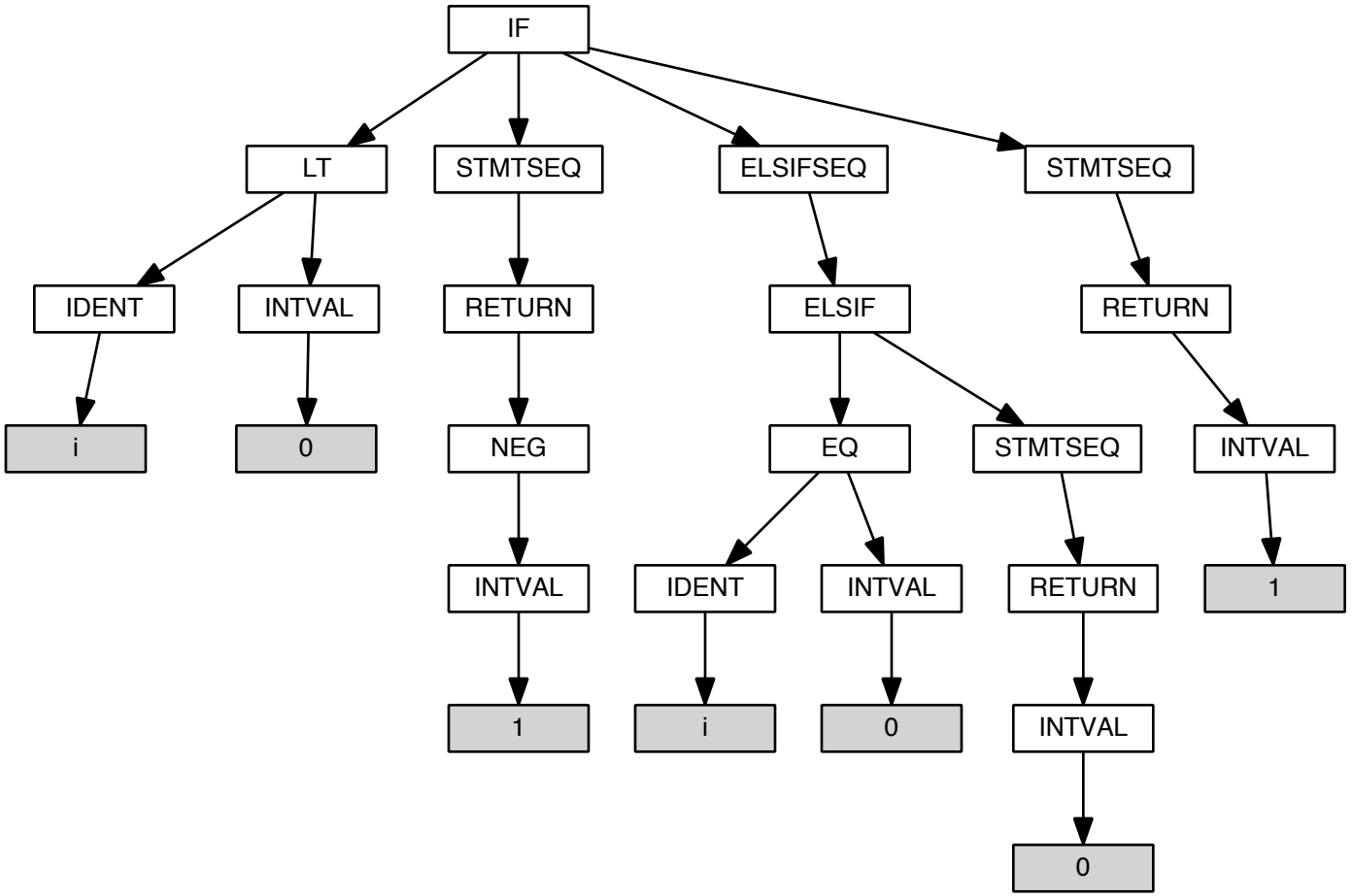
If-Else Statement AST Node



If-Elsif-Else Statement AST Node

```
IF i < 0 THEN RETURN -1 ELSIF i = 0 THEN RETURN 0 ELSE RETURN 1 END;
```

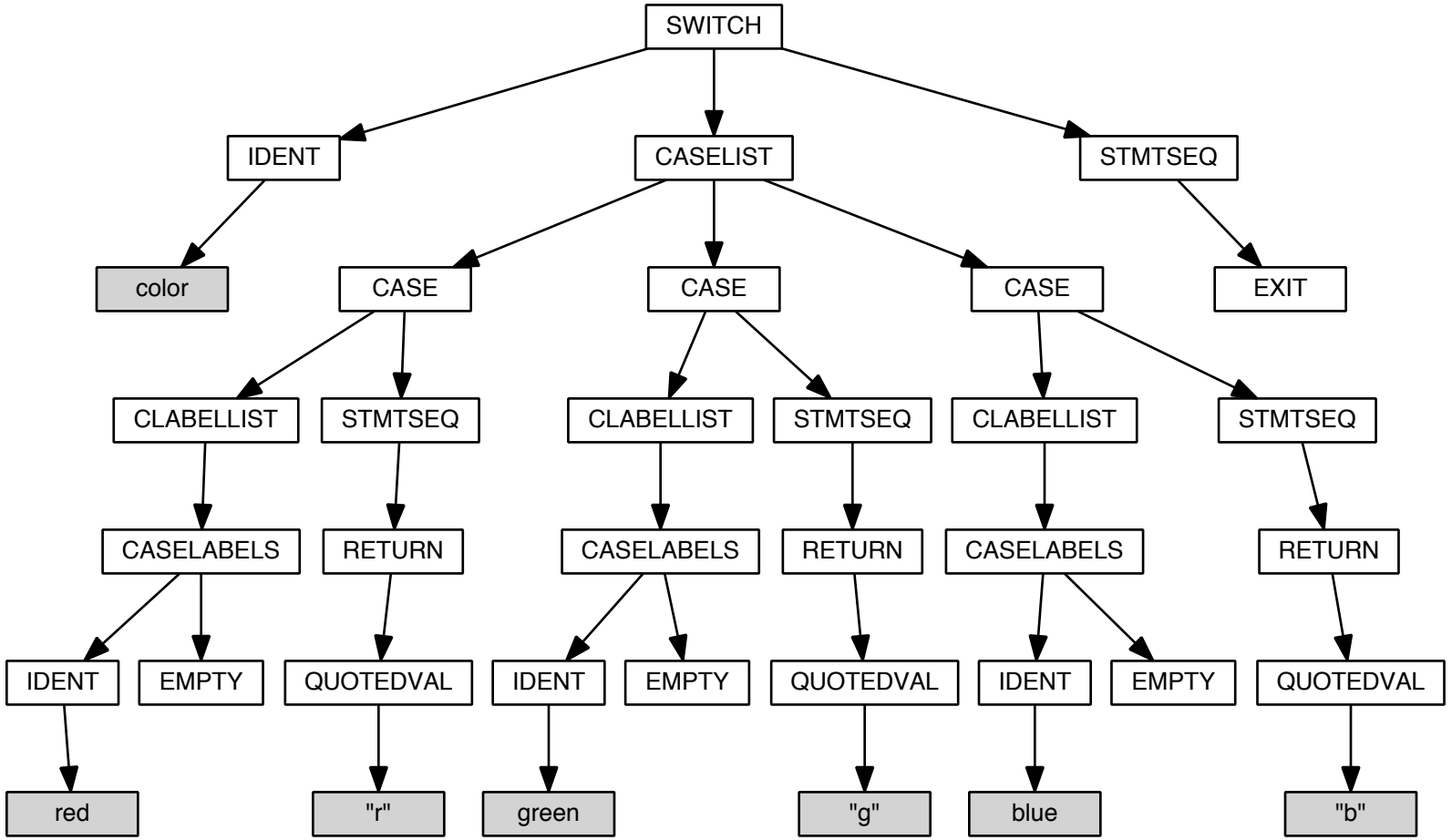
```
(IF (LT (IDENT i) (INTVAL 0)) (STMTSEQ (RETURN (NEG (INTVAL 1))))
(ELSIFSEQ (ELSIF (EQ (IDENT i) (INTVAL 0)) (STMTSEQ (RETURN (INTVAL 0))))) (STMTSEQ (RETURN (INTVAL 1))))
```



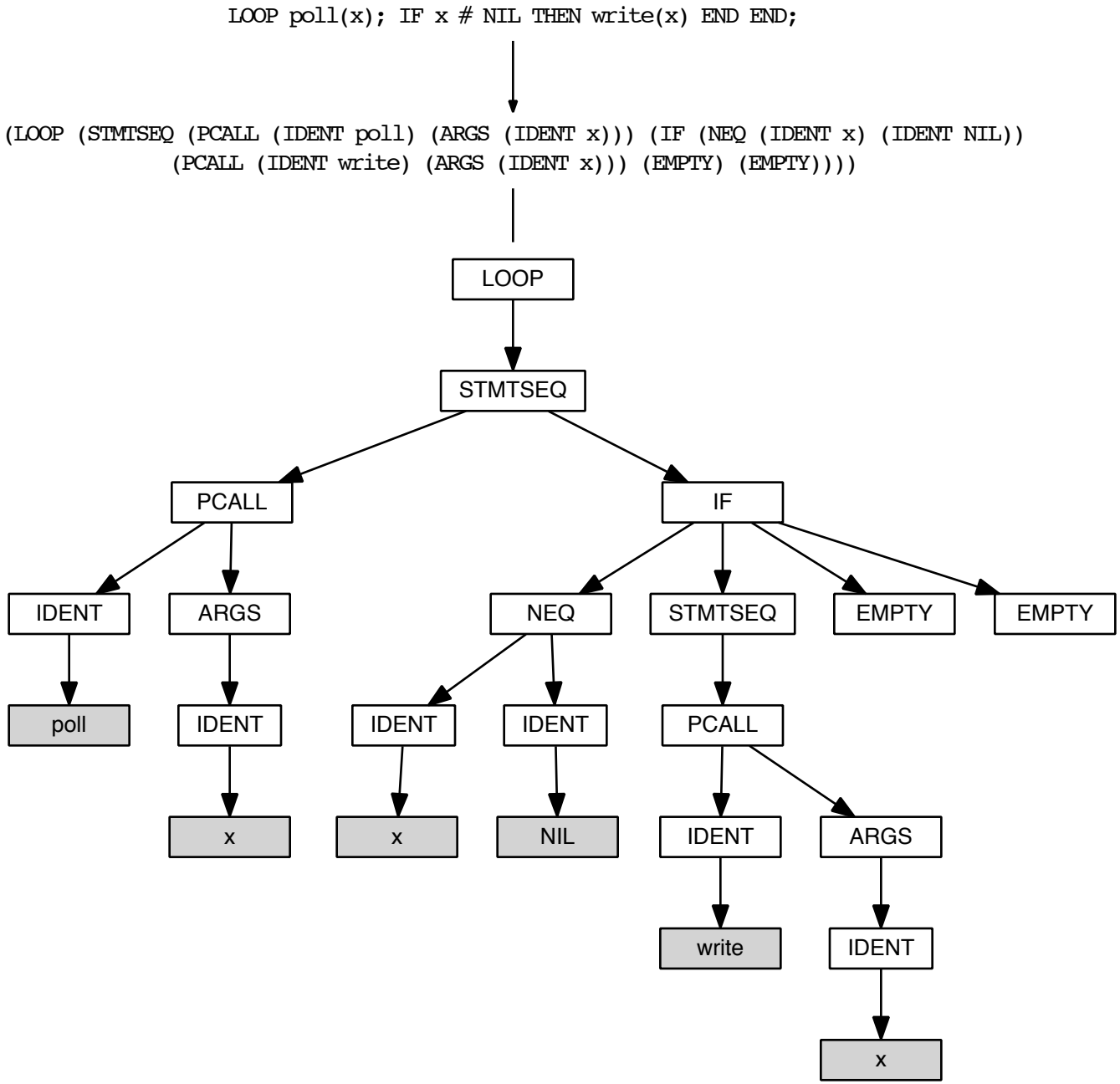
Case Statement AST Node

```
CASE color OF red : RETURN "r" | green : RETURN "g" | blue : RETURN "b" ELSE EXIT END;
```

```
(SWITCH (IDENT color) (CASELIST
  (CASE (CLABELLIST (CASELABELS (IDENT red) (EMPTY))) (STMTSEQ (RETURN (QUOTEDVAL "r"))))
  (CASE (CLABELLIST (CASELABELS (IDENT green) (EMPTY))) (STMTSEQ (RETURN (QUOTEDVAL "g"))))
(CASE (CLABELLIST (CASELABELS (IDENT blue) (EMPTY))) (STMTSEQ (RETURN (QUOTEDVAL "b"))))) (STMTSEQ (EXIT)))
```



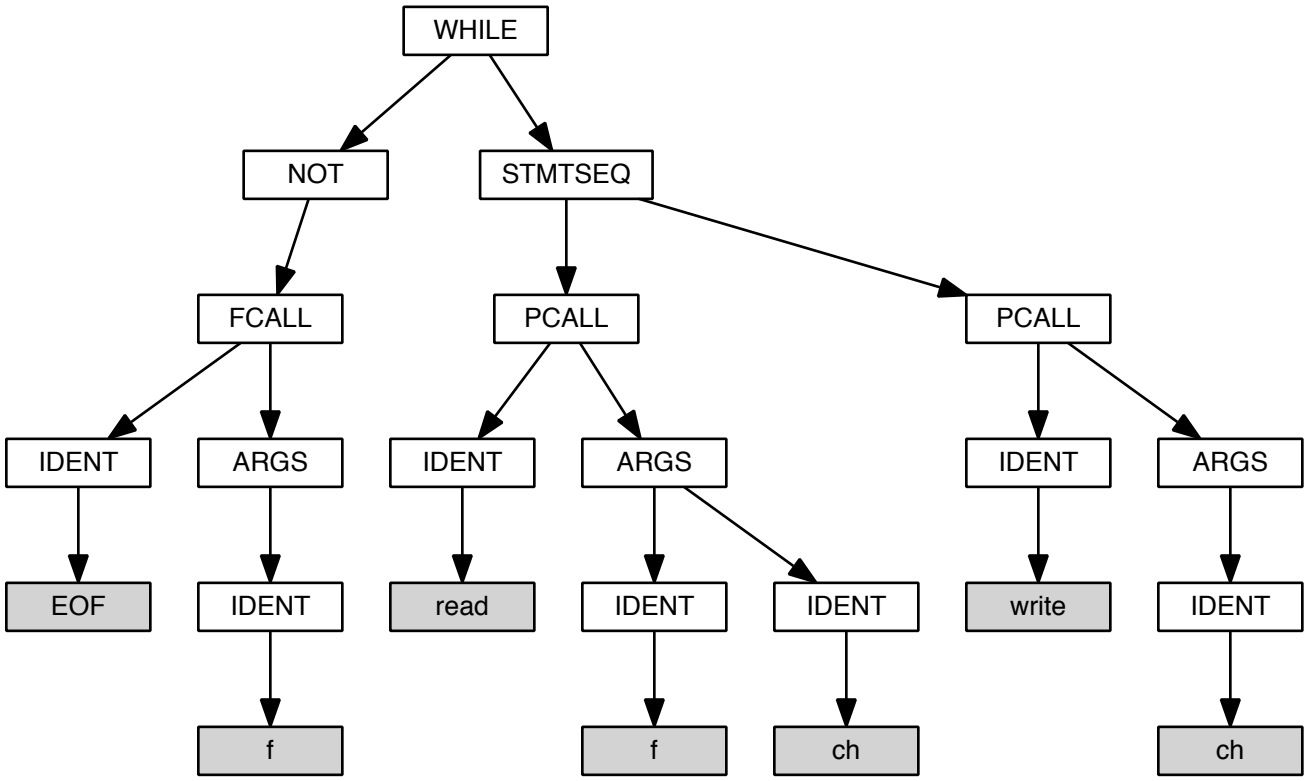
Loop Statement AST Node



While Statement AST Node

```
WHILE NOT EOF(f) DO read(f, ch); write(ch) END;
```

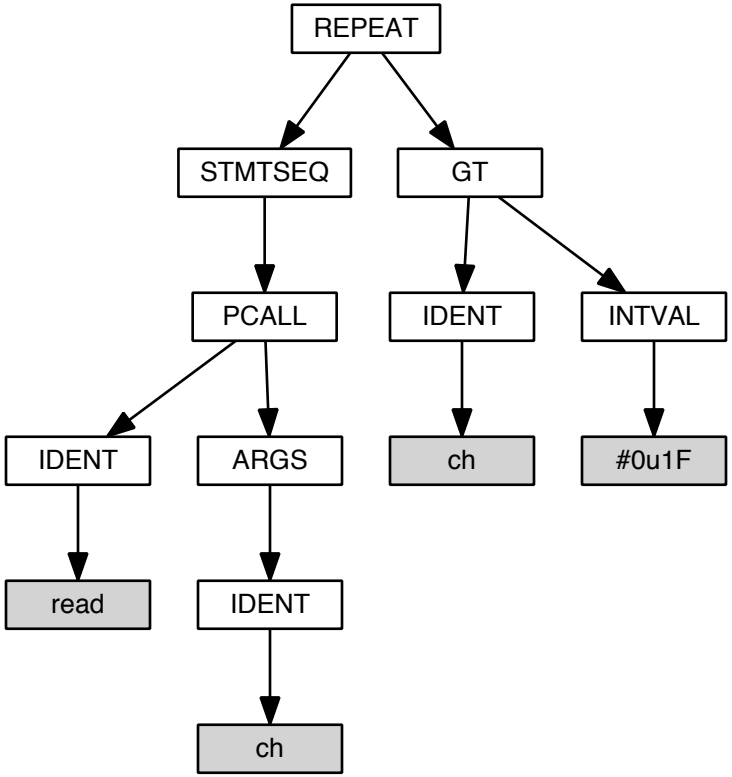
```
(WHILE (NOT (FCALL (IDENT EOF) (ARGS IDENT f)))
(STMTSEQ (PCALL (IDENT read) (ARGS (IDENT f) (IDENT ch)))
         (PCALL (IDENT write) (ARGS (IDENT ch)))))
```



Repeat Statement AST Node

```
REPEAT read(ch) UNTIL ch > 0u1F;
```

```
(REPEAT (STMTSEQ (PCALL (IDENT read) (ARGS (IDENT ch)))) (GT (IDENT ch) (INTVAL #0u1F)))
```



For Statement AST Node