63 *factor* → **id**
64 *factor* → **id [** *expression* **]**
65 *factor* → **id (** *expression_list* **)**
66 *factor* → **(** *expression* **)**
67 *factor* → **not** *factor*
68 *factor* → **intlit**
69 *factor* → **realit**
70 *factor* → **chrlit**

63 *factor* → **id**

1. Find the symbol descriptor Sym* S in the symbol table
2. Determine the kind (symkind sk) of symbol S is.
3. switch (sk) {
   3.1. constant symbol
   3.2. variable symbol
   3.3. function symbol
   3.4. else error

63 *factor* → **id**
id is a constant symbol

1. Obtain the type, T, of the constant symbol.
2. Obtain the value of the constant symbol, cv, a string.
3. Obtain the type character, tc, one of a, b, c, i, r.
4. Generate the P-Code instruction
   ldc tc cv
5. For example, in the program
   ```
   program p;
       var b:boolean;
   begin{p}
       b:=true
   end{p}.
   ```

   factor->ID(true)
   ldc b 1
6. Store the P-Code in an expression e.
7. Return expression e.

63 *factor* → **id**
id is a variable symbol

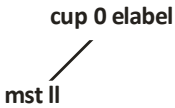
1. Obtain the type, T, of the variable symbol S
2. Create a string, op, that is the concatenation of the string "lv" and the type character, one of a, b, c, i, or r.
3. Find the difference ll, an integer that is the difference between the current lexical level and the lexical level of the variable symbol S.
4. Obtain the address, or offset, a, an integer of the variable symbol S.
5. Create P-Code op ll a,
   For example lvi 0 5
6. Store the P-Code in an expression e.
7. Return expression e.

Example:

```
program p;
    var i,j:integer;
begin{p}
    i:=j
end{p}.
```

factor -> ID(j)
lvi 0 6

63 *factor* → **id**



id is a function symbol

Example:

```
program p;
    var r:real;
    function g:real;
    begin{g}
        g:=9.8
    end{g};
begin{p}
    r:=g
end{p}
```

1. Create an expression E, having the P-Code, P.
   mst ll

   ll is computed as the difference between the current lexical level and the lexical level of the function (g).
2. Create an expression E, having the P-Code, P.
   cup 0 elabel

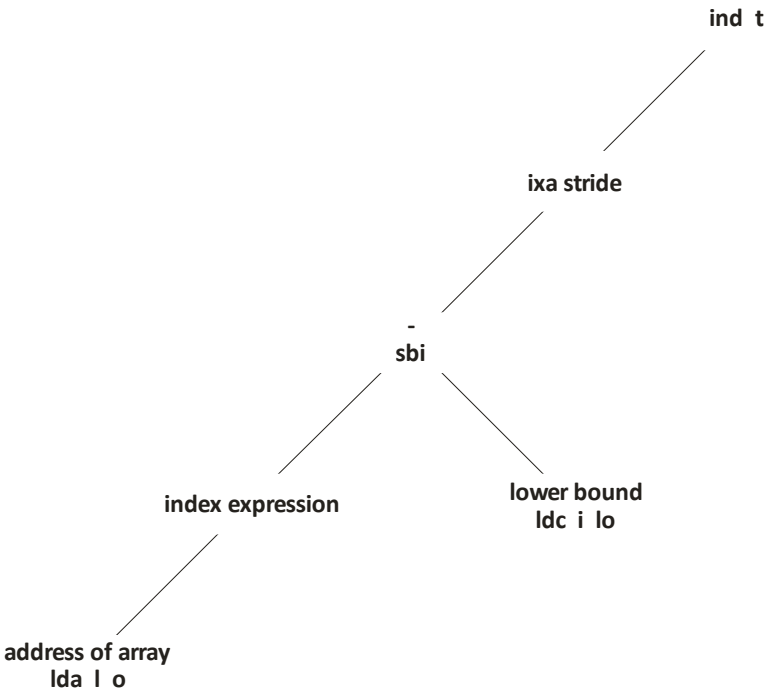
   0 is the number of arguments/parameters accepted by the function

   elabel is the entry label stored in the Symbol Descriptor for this function.

   Make the expression containing the mst-P-Code the left sub expression of the expression containing the cup-P-Code.

3. Return a pointer to the expression containing the cup-P-Code..

64 *factor* → **id [** *expression* **]**



1. Create an expression containing a P-Code the pushes the address of the array **id** on the computation stack.
   lda l a

   l is the difference between the current lexical level and the lexical level stored in the symbol descriptor for **id**.

   a is the relative address stored in the symbol descriptor for **id**.

2. Append the expression created in step to the left-most sub expression of the index expression that is passed as the semantic value for *expression* in the grammar.

3. Create an expression that pushes an integer constant that is the lower bound of the array **id**.
   ldc i lo

   I is the type (integer) of the constant that will be pushed on the computation stack

   lo is the value of the integer constant that will be pushed on the computation stack

4. Create and expression for the unbiased index. The unbiased index is the difference between the index expression and the lower bound of the array.
   sbi

   Make the left sub expression the index expression

   Make the right sub expression the lower bound of the array

5. Create an expression containing the P-Code
   ixa stride

   stride is the distance between successive elements in the array. The ixa stride instruction computes the address of the element whose value is desired.
   Top of stack: unbiased index expression, u
   Second to top: address of array, a
   call the stride s.
   ixa stride computes the address of the element $e = a + u \cdot s$
   and replaces the two items on the stack with the address of the element.

   Make the left sub expression the expression containing the sbi-P-Code.

6. Create an expression containing the P-Code
   ind t

   ind t replaces the address on top of the stack with a corresponding value of type **t**.

   Make the left sub expression the expression containing the ixa-P-Code.

65 *factor* → **id (** *expression_list* **)**

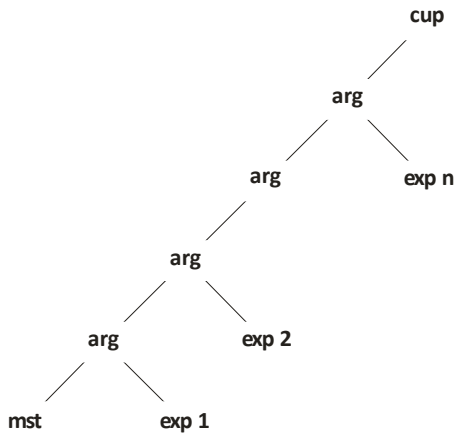


Figure 1. Expression tree for *factor* → **id (** *expression_list* **)**

id is a user function

1. Create an expression M, having the P-Code, P.
   mst ll

   ll is computed as the difference between the current lexical level and the lexical level of the function (**id**).

2. Traverse the list of expression in the *expression_list*. For each expression e_i in the list, create an expression A having the P-Code
   arg

   Make the right sub expression e_i

   Make the left sub expression as follows.
   The first left sub expression is the expression having the mst-P-Code.
   The second and subsequent left sub expressions have the previous argument-expression A.

4. Create an expression E, having the P-Code, P.
   cup c elabel

   c is the number of arguments/parameters accepted by the function

   elabel is the entry label stored in the Symbol Descriptor for this function.

   Make the left sub expression the last argument-expression A

factor → **id (** *expression_list* **)**

id is a standard function

1. Traverse the list of expression in the *expression_list*. For each expression e_i in the list, create an expression A having the P-Code **arg**

   Make the right sub expression e_i

   Make the left sub expression as follows.
   The first left sub expression is the expression having the mst-P-Code.
   The second and subsequent left sub expressions have the previous argument-expression A.

2. Create an expression E having the P-Code
   csp cspid

   cspid is the identifier of the standard function stored in the symbol descriptor for **id**.

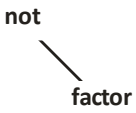
   Make the left sub expression the argument-expression A.

Function	P-Code	Parameter Type	Return Type
succ	**inc**	integer	integer
pred	**dec**	integer	integer
abs	**abi**	integer	integer
abs	**abr**	real	real
sqrt	**sqrt**	real	real
round	**rnd**	real	integer
trunc	**trc**	real	integer
ord	**ord**	char	integer
chr	**chr**	integer	char

Table 1. Subset Pascal Grammar (continued)

66 *factor* → **(** *expression* **)**
Assign the semantic variable for factor, $$, the value of the semantic variable for *factor*, $1.

67 *factor* → **not** *factor*



1. Create an expression E having the P-Code
   not

   make the right sub expression the expression that is the semantic value for the nonterminal symbol *factor*.

   The expression representing the nonterminal *factor* must have type Boolean.

2. Return expression E

68 *factor* → **intlit**
1. Create an expression E having the P-Code
   ldc i v

   i represents the integer type – the type of the value to be pushed on the stack.

   v is a string – the value to be pushed on the stack

2. Return expression E

69 *factor* → **realit**
1. Create an expression E having the P-Code
   ldc r v

   r represents the real type – the type of the value to be pushed on the stack.

   v is a string – the value to be pushed on the stack

2. Return expression E

70 *factor* → **chrlit**

1. Create an expression E having the P-Code
   ldc c v

   c represents the character type – the type of the value to be pushed on the stack.

   v is a string – the value to be pushed on the stack

2. Return expression E