

L-value vs. R-value. LHS vs. RHS of an assignment.

Assignment: $i := i + 1$

LHS vs. RHS

Address of I \leftarrow value of I + 1

$LHS(i) = \text{Address of I} := RHS(i) = (\text{the contents from the address } i) + 1$

LHS (Left Hand Side of an assignment = L-value = address) := RHS
(Right Hand Side of an assignment = R-Value = CONTENTS !!)

The syntax of most programming languages define assignment as:

Symbol := expression_value

Identifier := expression (usually in 99% of the cases)

C++ and ASSEMBLY !!! allow a much more general syntax:

Address_computation_expression := expression (the most general)

(mov [ebx+2*EDX+v-7], a+2)

Dereferencing (extracting the value from a given address) is usually implicit (depending on the context !) in 99% of the cases. Exception: BLISS language, where dereferencing must always be explicitly specified; $i \leftarrow *i + 1$ (also we have some similar situations in Algol68)

C++ reference variables are a special type of variables offering 3 kinds of usages:

- 1) `Int& j = i; // j becomes an ALIAS for i`
- 2) Passing variables **by reference** at subprograms call:
`float f(int&x, y);.....`
- 3) Returning L-values as a result of functions calls :

`Int& f(x,i) {...return v[i];}` – Function f returns a LHS (Left Value)
`F(a,7) = 79;` meaning that `v[7]=79 !!!`

Independently of these, the conditional ternar operator may be used as a L-value:

`(a+2?b:c) = x+y+z ; - correct`
`(a+2?1:c) = x+y+z; - syntax error !!! 1:=n !!!!`
