

Parameter Passing

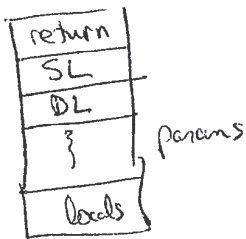
formal parameters

fun f x = x

↑
formal parameter. The name by which the parameter is known inside the function

f 3
↑ actual parameter

~~Pass by value~~
Recall ~ when a function is called an AR is created.
In the AR there is a slot for each formal param.



Pass by value: formal param is bound to the evaluated ~~expression that is actual~~ value in a language with variables and pass by value

function f(x) = { x := x + 1; return x }

ML translation using ML refs to represent variables
fun f (z : int) = let x = ref z in x := !x + 1; !x end

Pass by reference

function f(x) = { x := x + 1; return x } lang w/ vars & PBR

fun f (x: ~~int~~ int ref) = (x := !x + 1; !x)

Pass by name

In pass by name the ~~expression~~ ^{actual arg. expression} is not evaluated until it is used in the called function

~~function f(x) = {x := x + 1; return x} and PBNName.~~ ^{lang. with vars}

~~fun f(th: unit → int) = let ref~~

function g(x) = return x + 7 ; @ PBNName
g(y)

ML translation

fun g(th: unit → int) = (th () + 7)

g(fn () ⇒ y)

Pass by name was introduced in Algol 60 and has now-or-less disappeared from modern languages; however, in lang like ML, LISP and Python it is sometimes a useful idiom to know - avoid evaluating an expensive expression (even non-terminating) until you know you need its value - (or you know it will terminate)

~~Pass~~-by-need: similar to pass-by-name except the value is only computed once.

About static typing:

Two people remarked about ~~static~~ static typing either on
exam or later in e-mail: I hope you are not
going to beat dynamic typing. "I've ~~write~~ written
lots of code in dynamically typed languages and
find myself:

- A) very productive

B) able to find type errors easily

and I find these less true in stat. cally typed languages.
My experience matches this, esp. initially when dealing w/ S.T. languages.

→ Why very productive?

A) First off, note that we're in pretty much agreement on need for

Strong typing: ~~eliminate~~ greatly reduce mysterious errors

2nd: ~~productive~~ less productive in S.T. lang. because less familiar
w/ overall lang. (ie ML is a foreign lang) vs PHP
is a strong regional dialect.

3rd: coercions are helpful, don't violate static typing but
not impl. in ML

4th: Static typing has some overhead - e.g. to ~~create~~ union datatypes
where dynamic typing would allow ~~some~~ type discrimination
only at uses.

5th Pure Static typing ^{input} copes poorly or not at all with constructy
values from data.

B) able to find type errors easily is not, it seems to me the cause ~~of the~~
~~in~~ some settings: want to eliminate as many errors as possible
before deploying the code. This more true of S/W that will
exist in many instances, less true of code in a single instance - e.g.
a particular web site.