CSCI 275: Programming Abstractions Lecture 29: Parameter Passing Spring 2025

Stephen Checkoway Slides from Molly Q Feldman



Parameter-passing mechanisms

Parameter Passing

Parameter passing mechanisms help us understand how values are passed between procedures

In essence: how do we associate the formal parameters with the arguments?

Terminology reminder

(lambda (x y z) ...): x, y, and z are the formal parameters (f 25 (+ 8 36) "hello"): 25, (+ 8 36), and "hello" are the arguments

but we don't)

(We should probably call it argument passing rather than parameter passing,

Two Major Approaches*

Call by value (CBV) Arguments are evaluated in the caller's environment

Values are bound to parameters

Most languages you've used work this way



Call by name (CBN) Arguments are not evaluated, passed "as is"

You can think about it as the "text" of the argument is passed and replaces the parameters in the function's body



((lambda (x) 5) (/ 1 0))

What will the result be in Call by Value? In Call by Name? A. CBV: 5 **CBN: 5**

B. CBV: divide by zero error CBN: divide by zero error

C. CBV: divide by zero error **CBN: 5**

D. Something else

Thanks to "Design Concepts in Programming Languages" by Turbak, Gifford, and Sheldon for this example

12

(let* ([v 0]

(f (+ v 5)))

B. 5

A. 6

- **C**.0
- D. 1
- E. Error

[f (lambda (x) (set! v (+ v 1)) X)]) returns what in Racket?

Call by Value Example in Racket

(let* ([v 0] [f (lambda (x) (set! v (+ v 1)) X)]) (f (+ v 5)))

- f is called with value 5, so x is bound to 5
- v is set to 1
- x equal to 5 is returned

Call by Name Example in Racket (let* ([v 0] [f (lambda (x) (set! v (+ v 1)) X)]) (f (+ v 5)))

x with the text of the argument) (set! v (+ v 1)) (+ v 5)

v is set to 1 and then 6 is returned

The text of f's body becomes the two expressions (by replacing



PL Theory vs. PL Practice

 \bullet we can have a nice conversation

- real and therefore messy
 - is hard

When we talk about Call by Value or Call by Name in theory,

Language implementations make these types of formalisms

This is why having a parameter passing discussion for Java



PL Practice: Variations on CBV Depending on the language at hand, many use a call by

value approach or a related approach

Related approaches:

- Call by sharing
- Call by reference

Basic principle: the callee function gets a copy of what the caller supplies

Most prevalent in languages with objects

These are *subtlety* different, so much so call by sharing tends to not be used as a term



(Classic) Call by Value

Caller



Callee

5	



Call by Sharing

Caller

Callee



Call by Reference

Caller

5



Call by Reference

Caller

Callee



Call by Sharing versus Call by Reference Call by Sharing is when you get your own copy of the pointer

- What's happening in Java and Python Can mutate the object, cannot replace it

Call by Reference is when you get a reference to the object (a "pointer to the value")

- What's happening in C++
- Can entirely replace the object

Call by Reference in C++

https://pythontutor.com/render.html#code=%23include%20%3Ciostream%3E%0A%0Avoid%20change_value%28int%20%26x%29%20%7B%0A%20%20x%20%3D%2010%3B %0A%7D%0A%0Aint%20main%28%29%20%7B%0A%20%20int%20num%20%3D%200%3B%0A%20%20change value%28num%29%3B%0A%20%20std%3A%3Acout%20 %3C%3C%20num%20%3C%3C%20%22%5Cn%22%3B%0A%20%20return%200%3B%0A%7D&cumulative=false&curlnstr=0&heapPrimitives=nevernest&mode=display&origi n=opt-frontend.js&py=cpp_g%2B%2B9.3.0&rawInputLstJSON=%5B%5D&textReferences=false

Call by Sharing in Python

https://pythontutor.com/render.html#code=def%20change_value%28lst%29%3A%0A%20%20%20%20lst.append%2825%29%0A%20%20%20%20%20lst%20%3D%20%5B1,%202 ,%203%5D%0A%0Adef%20main%28%29%3A%0A%20%20%20%20vals%20%3D%20%5B10,%2020,%2030%5D%0A%20%20%20%20change value%28vals%29%0A%20 %20%20%20print%28vals%29%0A%0Amain%28%29&cumulative=false&curlnstr=0&heapPrimitives=nevernest&mode=display&origin=optfrontend.js&py=3&rawInputLstJSON=%5B%5D&textReferences=false



Python Tutor Example

Python Tutor Example

Aside: Prof. Barbara Liskov

Institute Professor of Computer Science at the Massachusetts Institute of Technology

Researcher in programming languages and systems: what are the rules and proofs behind how we communicate with computers

Developed the programming language CLU, defines call by sharing in it's manual

Turing Award Winner



Parameter Passing Is a Design Choice

Different languages have made different choices!



https://pythontutor.com/visualize.html#mode=edit



Call by Name in TeX TeX is a macro language for writing documents

$\det work#1#2{\%}$ All work and no play makes #1 a dull #2.\par} \work{Jack}{boy} $\x \{2+3\}{5}$ \bye

All work and no play makes Jack a dull boy. All work and no play makes 2+3 a dull 5.



Parameter Passing Is a Design Choice

With varying levels of precision and difficulty, we could make MiniScheme work with Call by Value, Call by Name, or Call by Reference

We **chose** classic Call by Value in our implementation when implementing lambdas

Example: if we wanted MiniScheme as CBN

We can make MiniScheme use Call by Name via function rewriting

- Don't evaluate arguments at all!
- In (apply-proc p args), rewrite the procedure's body the parse tree for the corresponding argument

(which is a parse tree) replacing each use of a parameter with

