Programming Paradigms

Control Abstraction (Part 2)

Prof. Dr. Michael Pradel

Software Lab, University of Stuttgart

Summer 2020

Overview

- Calling Sequences
- Parameter Passing 
- Exception Handling
- Coroutines
- Events

Parameter Passing

- **What does it mean if a parameter is passed to a callee?**
- **Different PLs have different parameter passing modes**
  - Call by value
  - Call by reference
  - Call by value/result
  - Call by sharing

Call by Value

- Caller passes copy of value to callee
- Once in callee, formal parameter is independent of the actual parameter

Call by Value

- Caller passes copy of value to callee
- Once in callee, formal parameter is independent of the actual parameter

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Call by Value

- Caller passes copy of value to callee
- Once in callee, formal parameter is independent of the actual parameter

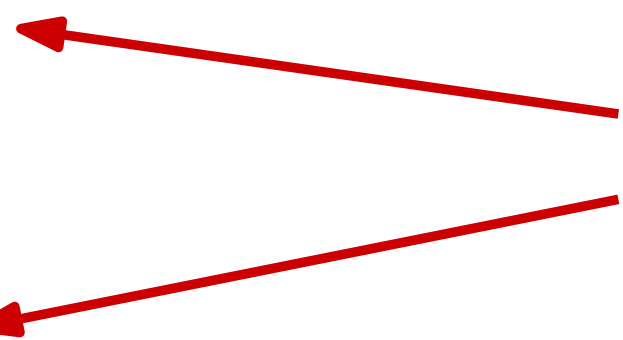
```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Assignment has no visible effect

Call by Value

- Caller passes copy of value to callee
- Once in callee, formal parameter is independent of the actual parameter

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```



Prints 2 twice

Call by Reference

- Formal parameter is new name of actual parameter
- Both names refer to the same value

Call by Reference

- Formal parameter is new name of actual parameter
- Both names refer to the same value

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Call by Reference

- Formal parameter is new name of actual parameter
- Both names refer to the same value

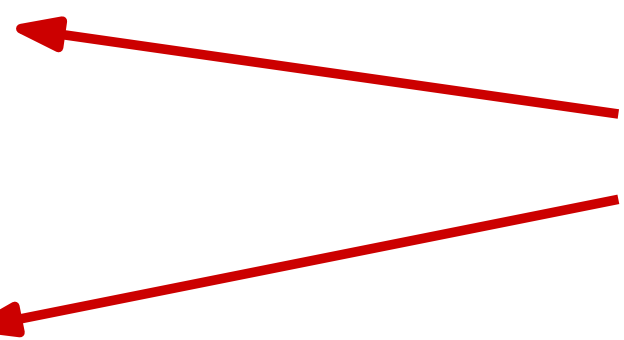
```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Refers to same value as outer variable `x`

Call by Reference

- Formal parameter is new name of actual parameter
- Both names refer to the same value

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```



Prints 3 twice

Call by Value/Result

- Argument is copied on call
- Resulting value is copied back to argument on return

Call by Value/Result

- Argument is copied on call
- Resulting value is copied back to argument on return

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Call by Value/Result

- Argument is copied on call
- Resulting value is copied back to argument on return

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Quiz: What does the code print?

Please vote in Ilias.

Call by Value/Result

- Argument is copied on call
- Resulting value is copied back to argument on return

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Refers to value different from outer variable `x`

Call by Value/Result

- Argument is copied on call
- Resulting value is copied back to argument on return

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```

Prints 2

Prints 3

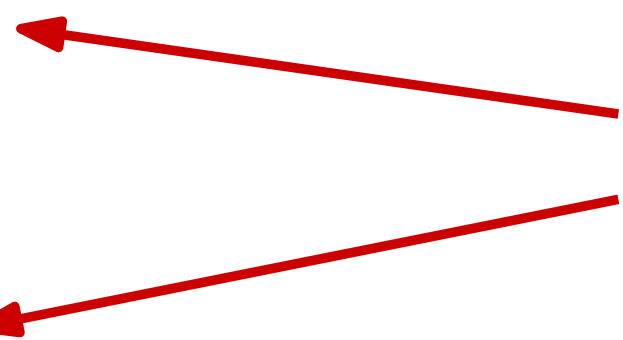
Call by Sharing

- **In PLs with reference model of variables**
  - Arguments are passed as values, but the values are references

Call by Sharing

- **In PLs with reference model of variables**
  - Arguments are passed as values, but the values are references

```
x : integer
procedure foo(y : integer)
  y := 3
  print x
...
x := 2
foo(x)
print(x)
```



Prints 3 twice

Passing Models in Popular PLs

- **C: By value, except arrays are passed by reference**
  - Passing pointers can emulate call by reference
- **Fortran: All arguments are passed by reference**
- **Java: Hybrid passing model**
  - Built-in, primitive types: By value
  - Instances of classes: By sharing