Procedures vs. Functions

Function:

* no side effect
* return a value
* Function call: expression

Procedure:

* side effect, executed for it
* no return value
* Procedure call: statement

No clear distinction made in most languages

* C/C++: void
* Ada/FORTRAN/Pascal: procedure/function

Syntax

Terminology:

* body
* specification interface
  + name
  + type of return value
  + parameters (names and types)

Text

Description automatically generated

Procedure Call

Text

Description automatically generated

Control transferred from caller to callee, at procedure call

Transferred back to caller when execution reaches the end of body

Can return early

Environment

Environment: binding from names to their attributes

Diagram

Description automatically generated

stack is managed by runtime system.

Activation Record for Nested Blocks

Activation record: memory allocated for the local objects of a block

* Entering a block: activation record allocated
* Exit from inner block to surrounding block: activation record released

Activation record is kept on the stack

A picture containing chart

Description automatically generated

Diagram

Description automatically generated with medium confidence

Activation Record for Procedures

A picture containing table

Description automatically generated

Diagram

Description automatically generated

Diagram

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence

Procedure Call

Text

Description automatically generated

Parameter Passing Mechanisms

Pass/Call by Value 🡪 mostly for imperative languages

Pass/Call by Reference

Pass/Call by Value-Result

Pass/Call by Name 🡪 mostly for functional languages

* lazy evaluation / non-strict evaluation

Example:

Graphical user interface, text, application

Description automatically generated

Procedure call because I don’t return a value.

* For call by value 🡪 1, 2
* For call by reference 🡪 2, 1
* For call by value-result 🡪 1, 2
* For call by name 🡪 2, 1

PASS BY VALUE

Diagram, schematic

Description automatically generated

*i could be variable or expression.*

Most common one

Replace formal parameters by the values of actual parameters

Actual parameters: No change

Formal parameters: Local variables (C, C++, Java, Pascal)

Example:

Graphical user interface, text, application

Description automatically generated

Are these Pass-by-Value?

Text, letter

Description automatically generated

Pass-by-Value: Pointers

Diagram

Description automatically generated

Pass-by-Value: Arrays

A picture containing graphical user interface

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Pass-by-Value: Java Objects

Graphical user interface, application

Description automatically generated

Graphical user interface, text, application

Description automatically generated

PASS BY REFERENCE

Graphical user interface, diagram

Description automatically generated

Formal parameters become alias of actual parameters

Actual parameters: changed by changes to formal parameters

Examples:

* Fortran: the only parameter passing mechanism
* C++ (reference type, &) /Pascal (var)

Example:

A picture containing diagram

Description automatically generated

This swap cannot be function (even if it returns a value) because it has a side effect.

Side effect is controlled by me. I know swap can change the value of i and j as a caller.

How to mimic it in C?

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

C++ Constant Reference

Graphical user interface, text, application

Description automatically generated

C++ Reference-to-Pointer

Chart

Description automatically generated

Graphical user interface, application

Description automatically generated

C++ Reference-to-Array

Chart, bar chart

Description automatically generated

PASS BY VALUE-RESULT

Diagram

Description automatically generated

Combination of Pass-by-Value and Pass-by-Reference (Pass-by-Reference without aliasing)

Replace formal parameters by the values of actual parameters

Value of formal parameters are copied back to actual parameters AT THE END

Example:

Diagram

Description automatically generated

If there is an exception in the middle of swap, recovery will be easier than pass-by-reference bc i and j hasn’t been affected.

Unspecified Issues

Diagram

Description automatically generated

PASS BY NAME

Graphical user interface, application

Description automatically generated

Actual parameters only evaluated when they are needed

The same parameter can be evaluated multiple times

Evaluated in calling environment

Callee can change the values of variables used in the argument expression and hence change the expression's value

Essentially equivalent to normal order evaluation

Example:

* Algol 60
* Not adopted by any major languages due to implementation difficulty

**Evaluation Strategey Revisited**

Strict Evaluation

* the arguments to a function are always evaluated completely before the function is applied (called)
* eager evaluation

Non-strict Evaluation

* the arguments to a function are not evaluated unless they are actually used in the evaluation of the function body
* short-circuit & lazy evaluation

Call-by-name differs from call-by-reference in that the parameter is not evaluated before the procedure is called but is instead evaluated lazily. That is, it is evaluated when and only when the parameter is actually used.

**Strict Evaluation Revisited**

Applicative order: the arguments of a function are evaluated from left to right

Call by value: the argument expression is evaluated, and the resulting value is bound to the corresponding variable in the function

Call by reference: a function receives an implicit reference to a variable used as argument

Call by sharing (or object): differing from call-by-reference in that assignments to function arguments within the function are not visible to the caller

Call by value-result (or copy-restore): a special case of call-by-reference where the provided reference is unique to the caller (Fortran & in multiprocessing context)

**Non-Strict Evaluation Revisited**

Normal order (or leftmost outermost): the outermost reducible expression is always reduced, applying functions before evaluating function arguments

Call by name: the arguments to a function are not evaluated before the function is called , they are substituted (capture-avoiding) directly into the function body and then left to be evaluated whenever they appear in the function

Call by need: a memoized version of call-by-name where, if the function argument is evaluated, that value is stored for subsequent uses. In pure functional programming, this produces the same results as call-by-name; when the function argument is used two or more times, call-by-need is almost always faster.

Call by macro extension: similar to call-by-name, but uses textual substitution rather than capture-avoiding substitution



Memoization is an optimization technique used to speed up programs by having function calls avoid repeating the calculation of results for previously processed inputs

*Yellow parts are not important.*

Example:

Graphical user interface, application

Description automatically generated

int temp; 🡪 no evaluate

temp = a; 🡪 i needs to be evaluated

a = b; 🡪 j needs to be evaluated

**Side Effects**

Graphical user interface, application

Description automatically generated

a keeps the name i, b keeps the name p[i]

Whenever you see a, you put i there.  
Whenever you see b, you put p[i] there.

This code changes the value of p[3], not p[1] as you hope.

Language may have said actual parameter has to be variable but p[i] is not a variable.

**What’s the use?**

Graphical user interface, text, application

Description automatically generated

**COMPARISONS**

Call by Value

* Efficient. No additional level of indirection.
* Less flexible and less efficient without pointer.
  + (array, struct, union as parameters)

Call by Reference

* Require one additional level of indirection (explicit dereferencing)
* If a parameter is not variable (e.g., constant), a memory space must be allocated for it with a temporary variable, in order to get a reference.
* Easiest to implement.

Call by Value-Result

* You may not want to change actual parameter values when facing exceptions.

Call by Name

* Lazy evaluation
* Difficult to implement