A Book Chapter

By Narendra Kumar Chahar

on **Subprogram and Parameter Passing Methods**

Index:

Chapter 7

Subprogram and Parameter Passing Methods

Subprogram:

- Each subprogram has a single entry point.
- The calling program is suspended during execution of the called subprogram.
- Control always returns to the caller when the called subprogram‘s execution terminates.

Basic Definitions

- A subprogram definition describes the interface to and the actions of the subprogram abstraction.
- A subprogram call is an explicit request that the subprogram be executed.
- A subprogram header is the first part of the definition, including the name, the kind of subprogram, and the formal parameters.
- The parameter profile (aka signature) of a subprogram is the number, order, and types of its parameters.
- The protocol is a subprogram‘s parameter profile and, if it is a function, its return type.
- Function declarations in C and C++ are often called prototypes.
- A subprogram declaration provides the protocol, but not the body, of the subprogram.
- A formal parameter is a dummy variable listed in the subprogram header and used in the subprogram.
- An actual parameter represents a value or address used in the subprogram call statement.

Actual/Formal Parameter Correspondence

- Positional

–The binding of actual parameters to formal parameters is by position: the first actual parameter is bound to the first formal parameter and so forth

–Safe and effective

- Keyword

–The name of the formal parameter to which an actual parameter is to be bound is specified with the actual parameter.

–Parameters can appear in any order.

Formal Parameter Default Values

- In certain languages (e.g., C++, Ada), formal parameters can have default values (if not actual parameter is passed).

  –In C++, default parameters must appear last because parameters are positional associated.
- C# methods can accept a variable number of parameters as long as they are of the same type.

Procedures and Functions

- There are two categories of subprograms

  –Procedures are collection of statements that define parameterized Computations.

  –Functions structurally resemble procedures but are semantically modelled on mathematical functions.
- They are expected to produce no side effects.
- In practice, program functions have side effects.

Design Issues for Subprograms

- What parameter passing methods are provided?
- Are parameter types checked?
- Are local variables static or dynamic?
- Can subprogram definitions appear in other subprogram definitions?
- Can subprograms be overloaded?
- Can subprogram be generic?

Local Referencing Environments

- Local variables can be stack-dynamic (bound to storage).

Advantages

- Support for recursion.
- Storage for locals is shared among some subprograms.

Disadvantages

•Allocation/de-allocation, initialization time.

•Indirect addressing.

•Subprograms cannot be history sensitive.

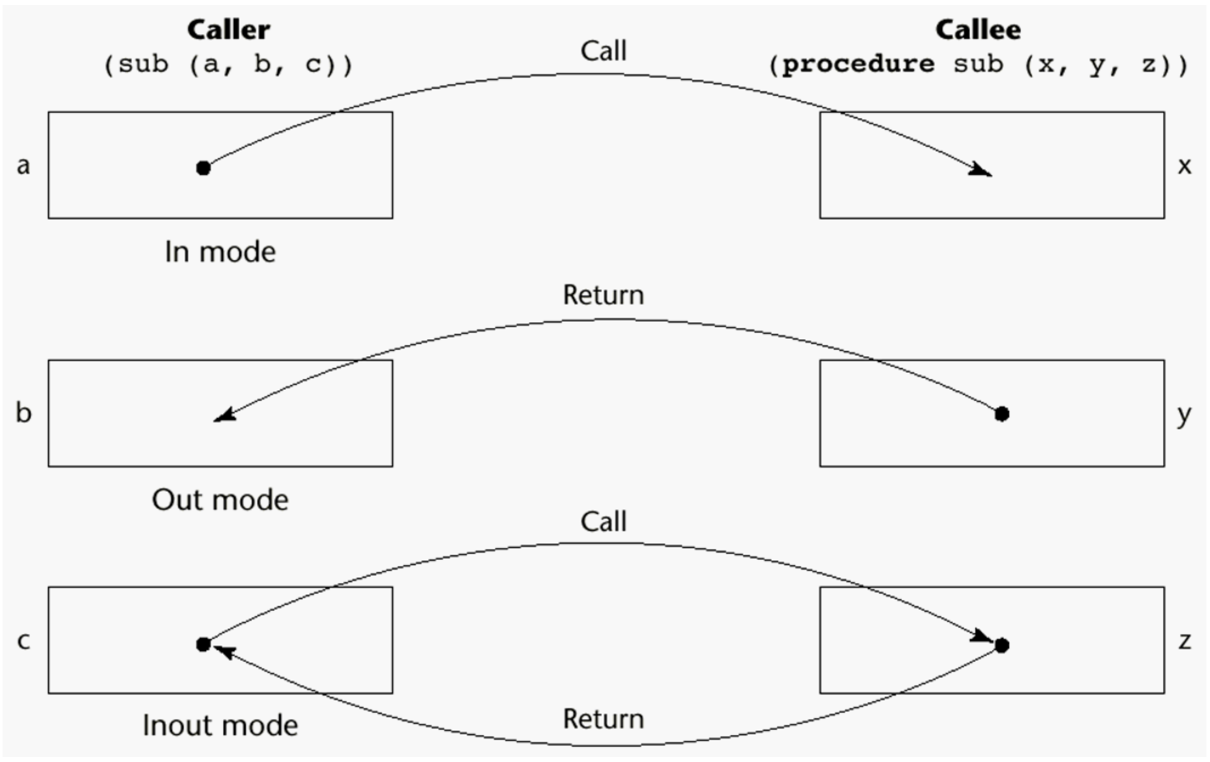
•Local variables can be static.

Parameter Passing Methods

Ways in which parameters are transmitted to and/or from called subprograms

- Pass-by-value
- Pass-by-result
- Pass-by-value-result
- Pass-by-reference
- Pass-by-name

Models of Parameter Passing



Pass-by-Value (In Mode):

The value of the actual parameter is used to initialize the corresponding formal parameter

–Normally implemented by copying

–Can be implemented by transmitting an access path but not recommended (enforcing write protection is not easy).

–When copies are used, additional storage is required

–Storage and copy operations can be costly

Pass-by-Result (Out Mode)

When a parameter is passed by result, no value is transmitted to the subprogram; the corresponding formal parameter acts as a local variable; its value is transmitted to caller‘s actual parameter when control is returned to the caller.

–Require extra storage location and copy operation

Potential problem: sub(p1, p1); whichever formal parameter is copied back will represent the current value of p1.

Pass-by-Value-Result (in out Mode)

•A combination of pass-by-value and pass-by-result

•Sometimes called pass-by-copy

•Formal parameters have local storage

•Disadvantages:

–Those of pass-by-result

–Those of pass-by-value

Pass-by-Reference (Inout Mode)

•Pass an access path

•Also called pass-by-sharing

•Passing process is efficient (no copying and no duplicated storage)

•Disadvantages

–Slower accesses (compared to pass-by-value) to formal parameters

–Potentials for un-wanted side effects

–Un-wanted aliases (access broadened)

Pass-by-Name (Inout Mode)

- By textual substitution
- Formals are bound to an access method at the time of the call, but actual binding to a value or address takes place at the time of a reference or assignment.
- Allows flexibility in late binding.

Implementing Parameter-Passing Methods

- In most language parameter communication takes place thru the run-time Stack.
- Pass-by-reference are the simplest to implement; only an address is placed in the stack.
- A subtle but fatal error can occur with pass-by-reference and pass-by-value result: a formal parameter corresponding to a constant can mistakenly be changed.

Parameter Passing Methods of Major Languages

- Fortran

  –Always used the inout semantics model

  –Before Fortran 77: pass-by-reference

  –Fortran 77 and later: scalar variables are often passed by value-result
- C

  –Pass-by-value

  –Pass-by-reference is achieved by using pointers as parameters
- C++

  –A special pointer type called reference type for pass-by-reference
- Java

  –All parameters are passed are passed by value

  –Object parameters are passed by reference
- Ada

  –Three semantics modes of parameter transmission: in, out, in out; in is the default mode.

  –Formal parameters declared out can be assigned but not referenced; those declared in can be referenced but not assigned; in out parameters can be referenced and assigned.

- C#

  –Default method: pass-by-value

  –Pass-by-reference is specified by preceding both a formal parameter and its actual parameter with ref.
- PHP: very similar to C#
- Perl: all actual parameters are implicitly placed in a predefined array named @_.

References:

[1]. Plaku, Erion, Kostas E. Bekris, and Lydia E. Kavraki. "Oops for motion planning: An online, open-source, programming system." *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007.

[2]. Brameier, Markus, et al. "SYSGP-A C++ library of different GP variants." *Technical Rep. No. CI-98/48, Collaborative Research Center* 531 (1998).

[3]. Tang, L. S. "A Comparison of Ada and C++." *Proceedings of the conference on TRI-Ada'92*. 1992.

[4]. Fox, Geoffrey, et al. "A prototype of Fortran-to-Java converter." *Concurrency: Practice and Experience* 9.11 (1997): 1047-1061.

[5]. Taft, S. Tucker. "Programming the internet in Ada 95." *International Conference on Reliable Software Technologies*. Springer, Berlin, Heidelberg, 1996.

[6]. Brosgol, Benjamin M. "A Comparison of the Object-Oriented Features of Ada 2005 and Java TM." *International Conference on Reliable Software Technologies*. Springer, Berlin, Heidelberg, 2008.

[7]. Benson, David B. "Parameter passing in nondeterministic recursive programs." *Journal of Computer and System Sciences* 19.1 (1979): 50-62.

[8]. Li, Jingke. "Parameter Passing." (2005).

[9]. Fleury, Ann E. "Programming in Java: Student-constructed rules." *Proceedings of the thirty-first SIGCSE technical symposium on Computer science education*. 2000.