# Principles of Programming Languages(CS F301)

BITS Pilani
Hyderabad Campus

Prof.R.Gururaj
CS&IS Dept.

BITS Pilani
Hyderabad Campus

# Subprograms(Ch.9 of T1)

Prof R Gururaj

## Introduction

Programming languages support two sorts of abstractions.

1. Data abstraction.
2. Process abstraction.

Subprograms are the form of process abstraction.

General subprogram characteristics:

A. Each subprogram has single entry point.

B. Caller is suspended when the subprogram is under execution.

C. Control always returns to the caller once the called subprogram execution is completed.

A subprogram definition: Describes the interface to and the actions of the subprogram abstraction.

A subprogram call: Is the explicit request to execute a specific subprogram.

Active subprogram is one which is called - begun to execute - and yet to complete the execution.

A subprogram header: is the first part of the definition, and consist of - the kind (static, public etc); the name; and the list of parameters.

Ex: in C : *void adder(parameter list)*

in Python : *def adder parameter):*

The parameter profile of a subprogram: contains the number, order , and the type of its formal parameters.

## The parameter profile of a subprogram:

Contains-

the number, order , and the type of its formal parameters.

## The protocol of a subprogram:

Is the parameter profile plus the return type of the subprogram.

## 2.3 Parameters

Subprograms typically describe computations.

Two ways a subprogram can gain access to data it is expected to process is-

1. Non-local variables (declared outside and visible in the subprogram).
2. Through parameter passing.

The data passed to the subprogram is accessed through names and are local to subprograms.

Extensive access to non-locals can reduce reliability.

## How methods are different?

Though methods access external data trough non-local references, the primary data to be processed by a method is the members of the object through which the method is invoked.

Pure functional languages such as Haskell do not have mutable data, so functions cannot change memory in anyway. They simply perform computations and return the results.

## Categories of parameters

Formal parameters: parameters appear in the header. They are thought of dummy variables.

They are bound to the storage only when the subprogram is called.

Actual parameters: is the list of parameters to be bound to formal parameters.

Positional parameters: binding between the FP and AP is done by position. Good when the list is small.

Keyword parameters: When the list is long, the programmer may make mistakes. One solution is to provide keyword parameters, in which the name of the formal parameter to which the actual parameter is bound is specified in the function call. Hence order is insignificant.

## Procedures & Functions

We have two distinct categories of subprograms –

Procedure and functions.  Both are collections of statements and have parameters.

Procedure- do not return anything.

Functions – can return a value.

Most languages do not include procedures.

In place of a procedure,  a function defined not to return value, can be used.

The computations /statements of  procedure can be enacted by a single call to the procedure.

In Ada we have procedures, and in Fortran we have Subroutines.

Most languages do not have procedure.

A procedure can produce result in the calling program in two ways.

1. Variables which are not local to subprograms, visible to both caller and the subprogram, both can change them.
2. If the procedure has formal parameters that can transfer the data to the caller.  Ex: OUT parameter.

Faithful functions: will not modify its parameters nor modify outside data items.

Has no side effects.

Only compute some value and return it.

Functions define new user-defined operators.

## 4. Local referencing environments

Local variables: defined inside the subprogram (either static or stack dynamic).

Stack dynamic variables: Allow recursion.

Cost of allocation, deallocation, initialization.

Access is indirect.

Not history sensitive.

Static local variables: Direct access, hence no run time overhead.

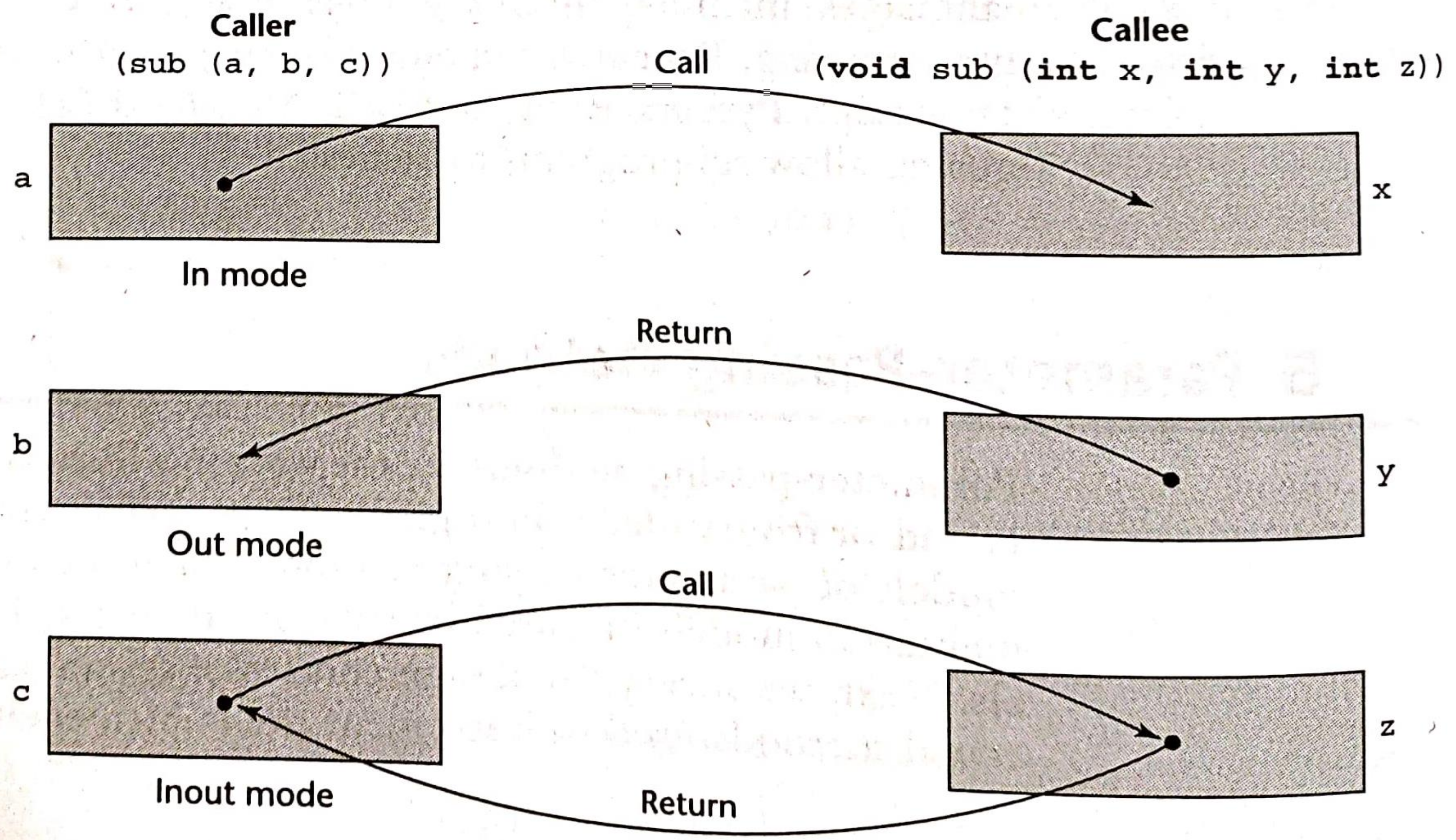
History sensitive. But can’t support recursion.

In many programing languages, they are stack-dynamic, unless declared static.

## 5. Parameter Passing

Semantic models of parameter passing:

1. Receive data from corresponding actual parameter. (IN)
2. Transmit data to actual parameter.(OUT)
3. Both (INOUT)


Caller
(sub (a, b, c))
Callee
(void sub (int x, int y, int z))
Call
a
x
In mode
Return
b
y
Out mode
Call
c
z
Inout mode
Return

Semantic models of parameter passing:

1. Receive data from corresponding actual parameter. (IN)
2. Transmit data to actual parameter.(OUT)
3. Both (INOUT)

Implementations of Semantic models of parameter passing:

A. Pass-by-value (IN mode semantics)
B. Pass-by-result (OUT mode semantics)
C. Pass-by-value-result (INOUT mode semantics)
D. Pass-by-reference (another implementation of INOUT )
E. Pass-by-name (INOUT mode)

## 8. Overloaded subprograms:

An overloaded operator is one that has multiple meanings.

Meaning is determined by the type of operands.

An overloaded subprogram is the one which has same name as another subprogram.

Protocol must be unique.

Name is same.

Must differ in number of parameters, type, and the order.

C++, Java, Ada, C# support method overloading.

C++ , Java, C# also support overloaded constructors.

In java , C# and C++ return type is not important.

## Design issues in functions

1. Are side effects allowed? (pass-by-reference)
2. What types can be returned?

   Most imperative languages restrict this . C can’t return arrays or functions.

   Ada, python, Ruby can return any type.

In some languages, functions are first class objects, they can be passed as parameters or can be returned. Python, Ruby.

Java and C# don’t have functions, only methods.

Any type of object or type can be returned.

3. Number of values that can be returned: mostly single value. But Ruby can support returning multiple values.

# Summary of Ch.9

1. Introduction
2. Parameters- semantic models
3. Categories
4. Procedures & Functions
5. Local referencing environments
6. Parameter passing
7. Overloaded subprograms
8. Design issues in Functions