Chapter 5

Basic Terminologies

Subprogram: describes the interface to and the actions of the subprogram abstraction.

Note: In Python it's executable but for other languages, it's not executable.

Subprogram call: explicit request that the subprogram be executed.

Subprogram header: it's the first part of the definition, including the name, the kind of subprogram, and the formal parameters.

Parameter profile (signature of subprogram): the number, order, and types of its parameters.

The protocol: it's the subprogram's parameter profile and, if is a function, its return type.

Prototypes: function declaration in C and C++

Subprogram declaration: provide the protocol but not the body of the subprogram

Formal parameter: dummy variable listed in the subprogram header and used in the subprogram.

Actual parameter: represents a value or additional used in the subprogram call statement.

Procedures and Functions

categories of subprograms

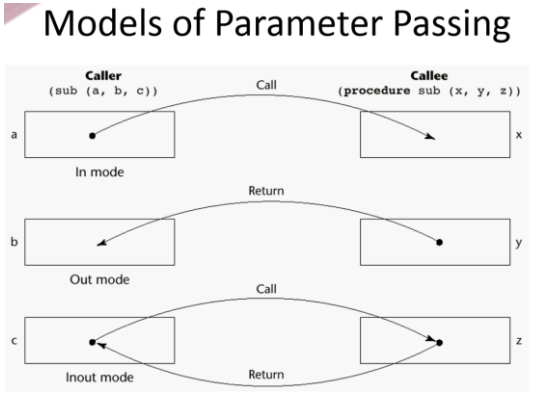
- Procedures:
  collection of statements that define parameterized computations.

- Functions:
  structurally resemble procedures but
  are semantically modeled on mathematical functions.

  They are expected to produce no side effects.
  In practice, program functions have side effects.

Semantic Models of Parameter Passing

- In mode
- Out mode
- Inout mode



Chapter 5

- **Pass-by-value (in mode)**
  value of the actual parameter is used to initialize the corresponding formal parameter
  - Normally implemented by copying.
  - implemented by transmitting an access path but not recommended.

Disadvantages:

- Additional storage is required.
- must write-protect in the called subprogram and accesses cost more.

- **Pass-by-Result (Out Mode)**

No value is transmitted into the subprogram.

Require extra storage.

- **Pass-by-Value-Result**
  Companion of pass-by-value and pass-by-Result.
  And sometimes called **Pass-by-copy**
  Formal parameters have local storage.

Disadvantages: Those of Pass-by-Value and Pass-by-Result

Parameter Passing Methods of Major Languages

C

Pass-by-Value

Pass-by-reference: is achieved by using pointers as parameters.

C++

reference type: special pointer 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.

Chapter 5

multidimensional array as parameters

If a multidimensional array is passed to a subprogram and the subprogram is separately compiled, the compiler needs to know the declared size of that array to build the storage mapping function.

Overloaded subprograms

Is one that has the same name as another subprogram in the same referencing environment.

Every version has unique protocol.

C++, Java, C#, and Ada include predefined overloading subprograms.

In Ada, the return type of an overloaded function can be used to distinguish calls

Ada, Java, C++, and C# allow users to write multiple versions of subprograms with the same name.

Example of parametric polymorphism: C++

```
template <class Type>
Type max(Type first, Type second) {
    return first > second ? first : second;
}
```

- The above template can be instantiated for any type for which operator > is defined

```
int max (int first, int second) {
  return first > second? first : second;
}
```

user-defined overloaded operators

Operators can be overloaded in Ada, C++,

Python, and Ruby

An Ada example

```
function "*" (A,B: in Vec_Type): return Integer
  is
  Sum: Integer := 0;
  begin
  for Index in A'range loop
   Sum := Sum + A(Index) * B(Index)
  end loop
  return sum;
end "*";
…
c = a * b; -- a, b, and c are of type Vec_Type
```

Chapter 5

Corounties

It’s a subprogram that has multiple entries and controls itself

Also called **symmetric control**

Coroutine call: is named a resume.

C program for function with argument and with return value

```
#include<stdio.h>
int sum(int, int);
void main()
{
   int a,b,result;
   printf("\nGoing to calculate the sum of two numbers:");
   printf("\nEnter two numbers:");
   scanf("%d %d",&a,&b);
   result = sum(a,b);
   printf("\nThe sum is : %d",result);
}
int sum(int a, int b)
{
   return a+b;
}
```

Output

```
Going to calculate the sum of two numbers:
Enter two numbers:10
20
The sum is : 30
```