Parameter Passing

Elaine Li (efl9013@nyu.edu)
Office Hours Fridays 1:00pm – 2:00pm

Nisarg Patel(nisarg@nyu.edu)
Office Hours Mondays 11:00am – 12:00pm

Introduction

- Formal and actual parameters
- Parameter Passing
    - Examples
- Deep and shallow binding
    - Examples

Formal and Actual parameters

Formal Parameters: The identifier used in a method to stand for the variable that is passed into the method by a caller

Actual Parameters: The actual variable/value that is passed into the method by a caller

- The values passed into a function

[Formal and Actual Parameters (ccsu.edu)]

Function:

```
int func1 (int a, char b, float& c)
 {
      ..
      ..
}
```

Formal Parameters

Function Call:

```
func1(5 * 3, 'A', z);
```

Actual Parameters

Parameter Passing

- Mechanism which determines what kind of association is in between the formal and actual parameters.
- Techniques used for parameter passing are:
  - Strict evaluation
    - Pass by value
    - Pass by reference
    - Pass by value-result (copy-return)
  - Lazy evaluation
    - Pass by name
    - Pass by need

Parameter Passing Techniques: Strict

- By value: formal is bound to fresh memory location containing copy of actual (C, Java)
  - Formal == Actual.copy()
  - Pros:
    - The function will work exactly how it reads
    - No side effects
- By reference: formal is bound to actual when it is a location (Pascal, C++)
  - Formal == Actual
  - Pros:
    - Less overhead (No need to make copies of objects)
    - Functions can be in-place

Parameter Passing Techniques: Strict

- By value-result (copy-return): formal is bound to copy of actual; upon return from routine, actual gets copy of formal (Fortran, in-out parameter mode in Ada)
  - Inside Function: Formal == Actual.copy(). Return: Actual == Formal.copy()
  - Pros:
    - Attempts to get the readability of “Pass by value” but also allows in-place functions and multiple return values

Parameter Passing Techniques: Lazy

- By name: formal is bound to expression of actual; expression evaluated whenever needed; formal parameter cannot be reassigned (but re-evaluation can change its value) (Algol)
  - You replace the function call with the code of the function
  - Pros:
    - Allows for the passing of an expression as a parameter
    - Very simple/fast to implement
    - Parameters are only evaluated when used (Lazy evaluation)
      - Leads to optimizations
        - Pass-By-Name Parameter Passing (sfu.ca)

Lazy Evaluation

```
fun foo(bool a, int b)
    if(a)
        return b;
    return 0;

foo(false, 1/0);
```

What happens here?

Example 1:

```
program example ;
var
      global : integer := 10;
      another : integer := 2 ;
procedure confuse (var first , second : integer ) ;
begin
     first := first + global ;
     second := first ∗ global ;
end ;
begin
confuse (global , another ) ;
end
```

Pass by value: global:=10; another:=2
Pass by value-result: global:=20 ;another:=200
Pass by reference: global:=20 ; another:=400
Pass by name: global:=20; another:=400

Example 2:

```
begin
    integer n;
    procedure p(k: integer);
    begin
        n := n+1;
        k := k+4;
        print(n);
    end;
    n := 0;
    p(n);
    print(n);
end;
```

Pass by value: 1 1
Pass by value-result: 1 4
Pass by reference: 5 5
Pass by name: 5 5

Example 3

```
begin
    integer n;
    procedure p(k: integer);
    begin
        print(k);
        n := k+10;
        print(k);
        n := k+5;
        print(k);
    end;
    n := 0;
    p(n+1);
end;
```

Pass by value: 1 1 1
Pass by value-result: 1 1 1
Pass by reference: 1 1 1
Pass by name: 1 12 18

Example 4:

```
begin
    array a[1..10] of integer;
    integer n;
    procedure p(b: integer);
    begin
        print(b);
        n := n+1;
        print(b);
        b := b+5;
    end;
    a[1] := 10; a[2] := 20;
    a[3] := 30; a[4] := 40;
    n := 1;
    p(a[n+2]);
    print(a);
end;
```

```
Pass by value: 30 30 [10, 20, 30, 40]
Pass by value-result: 30 30 [10, 20, 35, 40]
Pass by reference: 30 30 [10, 20, 35, 40]
Pass by name: 30 40 [10, 20, 30, 45]
```

Example 5:

```
int i = 1;
void p(int f, int g)
{
      g++;
      f = 5 * i;
}
int main() {
     int a[] = {0, 1, 2};
     p(a[i], i);
     printf("%d %d %d %d\n", i, a[0], a[1], a[2]);
}
```

Pass by value: 1 0 1 2
Pass by value-result: 2 0 5 2
Pass by reference: 2 0 10 2
Pass by name: 2 0 1 10

Example 6:

```
program main;
      i: integer;
     a: array[1..2] of integer;

     procedure f(x:integer, y:integer)
         begin x := x + 1;
         y := y + 1;
         end
begin (* main *)
     i := 1;
     a[1] := 1;
     a[2] := 2;
     f(i,a[i]);
     print(a[1],a[2]);
end;
```

Pass by value: 1 2
Pass by value-result: 2 2
Pass by reference : 2 2
Pass by name : 1 3

Example 7:

```
program one;
    a: array[1..5] of integer;
    i: integer;
    procedure f(x: integer; y: integer)
    begin
        y := y + 1; x := x * 2; a[i] := a[i] + 15;
    end
begin (* main program *)
    for j := 1 to 5 loop
        a[j] := j;
    end loop;
    i := 1;
    f(a[i], i);
    for j := 1 to 5 loop
        print(a[j]);
    end loop;
end (* main program *)
```

Pass by value: 16 2 3 4 5
Pass by reference: 2 17 3 4 5
Pass by value-result: 2 2 3 4 5
Pass by name: 1 19 3 4 5

Deep and Shallow Binding

- Deep Binding:
  - When a subroutine is passed as argument to another subroutine, a *closure* must be created and passed in place of the subroutine.
  - A closure is a pair of a subroutine together with its reference environment.
  - When a subroutine is called through a closure, the reference environment from when the closure was created is restored as part of the calling sequence.
- Shallow Binding:
  - When a subroutine is called, it uses the current reference environment at the call site.

Example 1:

```
function f1()
{
    var x = 10;
    function f2(fx)
    {
        var x;
        x = 6;
        fx();
    };

    function f3()
    {
        print x;
    };

    f2(f3);
};
```

Deep binding: 10
Shallow binding: 6

Example 2:

```
begin
    local x;
    procedure Q;
    begin
        print(x);
    end;
    procedure P;
    begin
        local x;
        x:=1;
        Q;
    end;
    x: = 2;
    P;
end;
```

Deep binding: 2
Shallow binding: 1