School of Electrical Engineering and Computing

SENG2200/6220 PROGRAMMING LANGUAGES & PARADIGMS

(S1, 2020)

Parameter-Passing

Dr Nan Li
Office: ES222
Nan.Li@newcastle.edu.au

THE UNIVERSITY OF NEWCASTLE AUSTRALIA

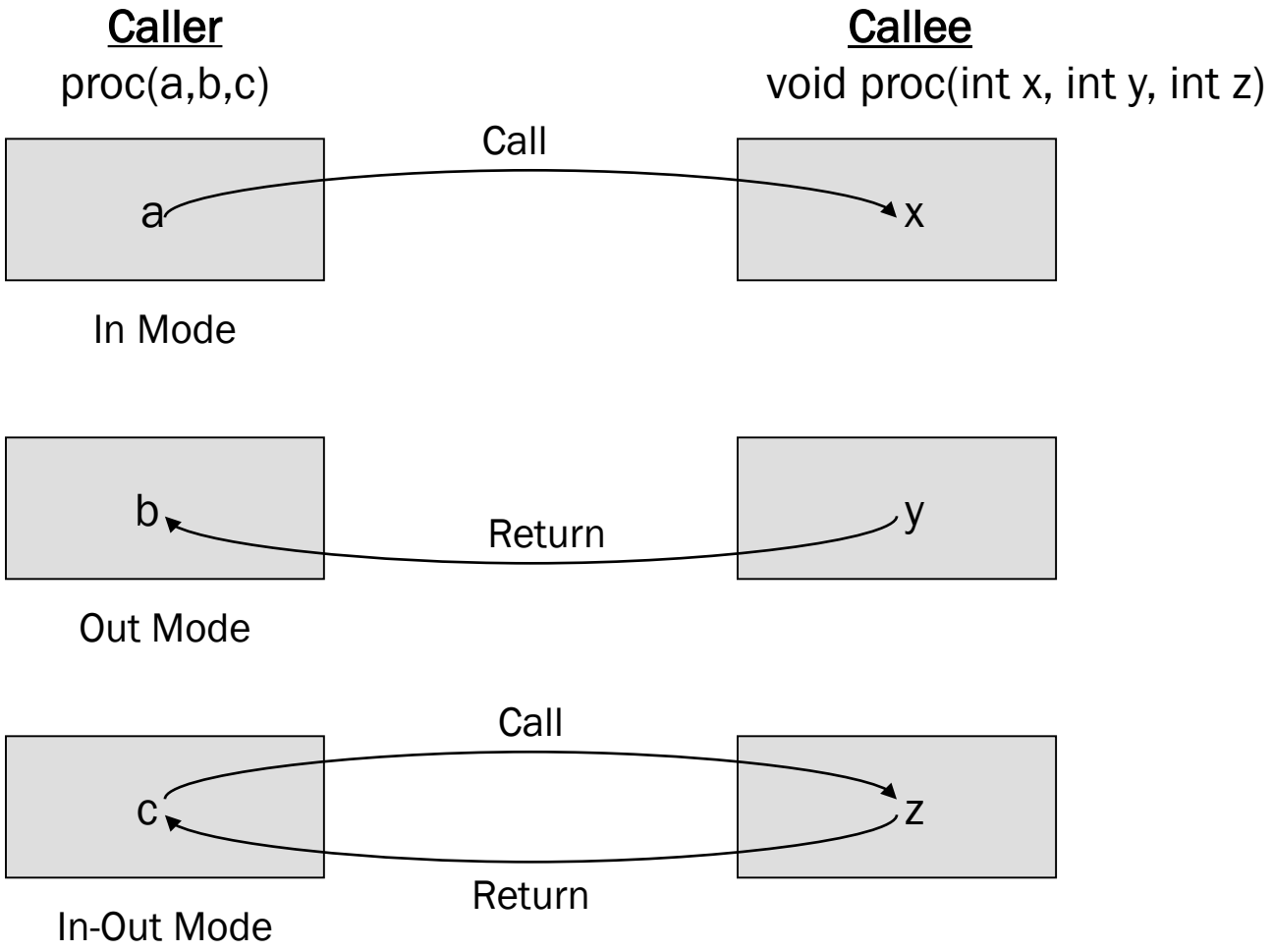
Outline

- In, out and In-out parameters
- Parameter-passing methods
- Implementation of parameter-passing methods
- Common usages

In, Out & In-Out Parameters

- In
  - *Caller provides data (input) to the procedure*
- Out
  - *Procedure provides data (output) back to the caller*
- In-Out
  - *Caller provides input to procedure, and*
  - *Procedure provides output back to the caller*
- Actual Parameter
  - *Specification in the actual procedure call*
- Formal Parameter
  - *Specification within the procedure definition*

In, Out & In-Out Parameters



In, Out & In-Out Parameters

C++ example

```
void method(int i, int& j) {
    i = 55;   // The caller will not see this new value.
    j = 44;   // The caller will see this new value.
}
```

In, Out & In-Out Parameters

C# example

```
void Method(string in, out int i, out string s1) {
    in = "The caller will not see this new value."
    i = 44;   // the caller will see this new value
    s1 = "The caller will see this new value.";
}
```

In C#, both ref and out keywords enable the out mode. Their difference is: An argument that is passed to a ref parameter must be initialised before it is passed, while an argument passed to an out parameter does not have to be explicitly initialised before they are passed.

Pass-by-Value

- The value of the actual parameter is used to initialise the corresponding formal parameter
- The formal parameter then acts as a local variable
- Usually implemented using copy (esp for primitives)
  - *Access path (alias) plus write-protection?*
    - Write-protection is difficult to ensure
    - C++ with “const” does this

Pros:

- Easy to implement.
- Fast for primitive value.

Cons:

- Inefficient for non-primitive values.
- Additional storage is used.

Alias Write-Protection Example

```
// Java
class A {
    private int x = 0;
    public void set(int x) {this.x = x;}
}
class ByValue {
    public static void set(final A a) {
        a.set(5);}
    public static void main (String[] args) {
        set(new A());
    }
} // write protected??
```

```
// C++
class A {
    int x = 0;
    public:
        void set(int x) {this->x = x;}
};
void set(const A a) { a.set(5);}
//compile error
int main () {
    A a;
    set(a);
    return 0;
}
```

Pass-by-Result

- No value is transmitted to the procedure – out mode
- Corresponding formal parameter must be initialised to a default value within the procedure and then acts as a local variable
- Part of the return mechanism is to copy the value of the formal parameter back to the actual parameter
  - *Actual param needs to be a variable, not an expression*
- Similar problems with access path implementation to pass-by-value
- Actual parameter collision can cause difficulty with the semantics, as the order of copy-back is crucial
  - *e.g. proc(p1,p1) -> different formals, same actual*
- What happens if the procedure throws an exception?

Pass-by-Result

```
// C# code
static void Method(out int i, out string s1) {
    i = 44;
    s1 = "I've been returned.";
}

static void Main() {
    int value;
    string str1, str2;
    Method(out value, out str1);
    // value is now 44
    // str1 is now "I've been returned."
}
```

Pass-by-Value-Result

- Often referred to as Pass-by-Copy
- Calling is by Pass-by-Value
- Return is by Pass-by-Result
- Similar disadvantages as Pass-by-Value and Pass-by-Result
  - *Requires multiple storage for parameters*
  - *Requires time for copying values in and out*
  - *Problems associated with copy-out order of parameters*
- What happens if the procedure throws an exception?
  - *No result should be returned for the parameter*

Pass-by-Reference

- Used as an In-Out mechanism
- Access path is established between the actual and formal parameters (usually an address)
- The procedure alters the actual parameter at will, via the access path
- Efficient in both time and space - no duplicate space required
- Disadvantages
  - *Slower access to formal parameters*
  - *Can create aliases via parameter collisions*
- If the procedure throws an exception?
  - *Partial results will be left in the calling environment*

Pass-by-Name

- Used as in-out mode
- Actual parameter is textually substituted for the corresponding formal parameter
- Formal parameter is bound to an access mechanism at the time of the procedure call, but the actual binding to a value or an address is delayed until the formal parameter is actually referenced
- Not as odd or way-out as it may seem at first - can be thought of as an extension of the late-binding mechanism used to implement polymorphism being extended into the parameter passing mechanism

Pass-by-Name

- This was used in early versions of ALGOL, BUT …
- An example that showed how *interesting* it is as a general parameter passing mechanism is as follows:

  ```
  procedure Swap(m, n) begin … end;
  ```

  When this is invoked with the call:

  ```
  Swap(i, A[i]);
  ```

  then it is extremely difficult to come up with meaningful semantics for this procedure call using pass-by-name.

Pass-by-Name

```
A[] = {5, 4, 3, 5, 4, 3}
procedure Swap(var int m, var int n)
var int temp;
begin
    temp := m;
    m := n;
    n := temp;
end;
```

```
Calling Swap(i, A[i])
begin
    temp := i;
    i := A[i];
    A[i] := temp;
end;
```

What is the result of calling `Swap(2, A[2])`?

	i	A[i]	
Before Call	2	3	A[3]=5
After Call	3	3	A[3]=2

Pass-by-Name

- Often used at compile time by the macros in assembly language and for the generic parameters for templates in C++, generic subprograms in Ada, etc.
- Not used as the basic mechanism by any widely used language.

Comparison of Mechanisms

	In	Out
Pass-by-Value	Copy	No out
Pass-by-Reference	Not copy, evaluate actual parameter.	Not copy, actual parameter may be modified.
Pass-by-Name	Substitute formal parameter by the text expression of actual parameter. Re-evaluate whenever appears in procedure.	
Pass-by-Result	No input value	Copy
Pass-by-Value-Result	Copy	Copy

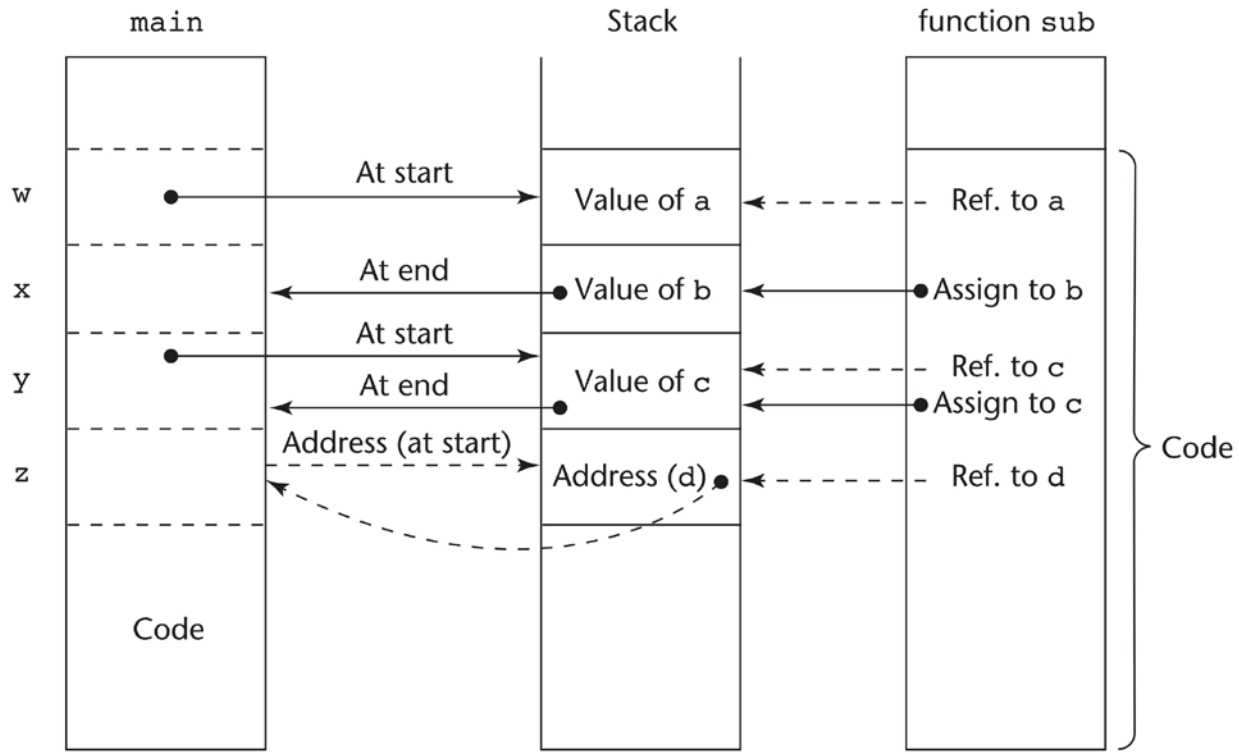
Parameter-Passing Methods Implementation

- In most contemporary languages, parameter communication takes place through the run-time stack.
- Run-time stack: initialised and maintained by the run-time system. It store information about the subprograms, such as,
  - *Return value*
  - *Function parameter(s)*
  - *Return address*
  - *Function's local variable(s)*
- Access to the formal parameters in the called subprogram is by indirect addressing from the (run-time) stack location of the address.

Parameter-Passing Methods Implementation

- Pass-by-value: stack stores the information for the formal parameter
- Pass-by-result: stack stores the information for the actual parameter
- Pass-by-value-result: combination of pass-by-value and pass-by-result
- Pass-by-reference: stack stores the address of the actual parameter.

Parameter-Passing Methods Implementation



Function header: **void** sub (**int** a, **int** b, **int** c, **int** d)
Function call in main: sub (w,x,y,z)
(pass w by value, x by result, y by value-result, z by reference)

Sebesta 10e

Design Considerations

- Two main considerations
  - *Efficiency*
  - *Is one-way or two-way data transfer required?*
- Need to minimize the access by a procedure to data outside the procedure itself
- Use In-mode params when no data needs to be returned
- Use Out-mode params when no data are transferred in
- In-Out mode should be used only when data must move in both directions between the caller and the callee
- Efficiency considerations drive many languages towards providing an access path coupled with read-only or write-only semantics for the one-way (in or out) transfer

Common Usage

- C uses pass-by-value, with pass-by-reference (in out mode) semantics being simulated by using pointer (values) as parameters.
- C++ includes a special type of pointer called a reference type which is automatically de-referenced in the called function or method – giving pass-by-reference.
- Both C and C++ allow formal (pointer and reference) parameters to be specified as constants (const) which means that they can never be assigned in the procedure.
  - *This also means that constant objects can be passed in*
  - *Note the differences between const & normal "in-mode" operation (which allows the local copy to change).*

Common Usage

- All Java parameters are passed by value.
- Because objects can only be passed by supplying a reference to that object, object parameters are effectively passed by reference.
- The object reference passed in cannot be changed, but the object it refers to can be changed provided a method to do such a change is available.
- Scalar (primitive) variables cannot be passed by reference in Java, but if a scalar (attribute) is contained in an object that is passed by reference, then it can be changed.

Distributed Computing

- As soon as procedure calls are required to cross a network to be remotely executed on another machine, the whole situation as to what is efficient and what is in-efficient is changed.
- Copy-in, copy-out (message passing) becomes a preferred method of communication.
- Simulation of explicit pass-by-reference becomes far more costly.