Parameter Passing

- Standard mechanisms
  - Call by value
  - Call by reference
- Other methods
  - Call by value-result
  - Call by name, result

Terms

- Function definition – where the details of the function are presented (type, name, parameters, body) - only one
- Function call – where the function is invoked (name, arguments) – zero or more (not interesting if no calls)
- Parameters (formal parameters) – names of local variables in function that are given values during call
- Local variables – variables in function body that are not parameters
- Arguments (actual parameters) – values provided for parameters

Terms

- Parameter passing – method(s) used to determine how argument values relate to parameters
- Overloading – when the same function name can have more than one set of parameters
- L-value – location of variable
- R-value – contents of variable (or result of expression)

Terms Example

Function definition

```
int func1 (int a, char b, float& c) {
                        Parameters
  int x,
  char y;    Local variables
  …
}
```

Function call

```
  func1(5 * 3, ‘A’, z);
              Arguments
```

Call by Value

- Calling mechanism
  - Arguments are evaluated for their values
  - Local variables created for each parameter
  - Values resulting from arguments copied to new parameter variables
  - When function call ends, parameter variables are discarded
- During function execution, value of parameters may diverge from argument values (function does not affect arguments)

Call by Value

- Call by Value used in
  - C
  - Most C++ parameters
- Variables are changed in functions only indirectly
  - Pointer values are passed to functions
  - Variables that the pointer point at may be changed in a function
- Characteristics:
  - Variables may not directly be changed in function body (but changes in function do not change the values of arguments)
  - Arguments can be complex expressions
  - Mechanism is simple (easy to explain)

Call by Reference

- Calling mechanism
  - Variable locations for arguments determined
  - Parameter names added to the location for each argument
  - When function call ends, extra names are discarded
- During function call, changes to referenced variables persist even after function ends

Call by Reference

- Call by Reference used
  - Pascal (var parameters)
  - C++ (& parameters)
  - Some versions of FORTRAN
- Characteristics:
  - Changes to parameters change corresponding argument variables
  - Arguments must be variables (cannot connect a reference to an expression)

Parameter Passing Example – Call by Value

```
int x = 1; // global x

void func1 (int a) {
  // Location 2
  x = 2;
  // Location 3
  a = 5;
  // Location 4
}

void main () {
  // Location 1
  func1(x);
  // Location 5
}
```

Location 1: a undefined, x is 1

Location 2: a is 1, x is 1

Location 3: a is 1, x is 2

Location 4: a is 5, x is 2

Location 5: x is 2

Parameter Passing Example – Call by Reference

```
int x = 1; // global x

void func1 (int a) {
  // Location 2
  x = 2;
  // Location 3
  a = 5;
  // Location 4
}

void main () {
  // Location 1
  func1(x);
  // Location 5
}
```

Location 1: a undefined, x is 1

Location 2: a is 1, x is 1

Location 3: a is 2, x is 2

Location 4: a is 5, x is 5

Location 5: x is 5

Call by Value-Result

- Calling mechanism
  - Arguments are evaluated for their values
  - Local variables created for each parameter
  - Values resulting from arguments copied to new parameter variables
  - When function call ends, values from parameters copied back to calling variables
- Operates somewhat like Call by Reference but differs under certain circumstances
- Used in some versions of FORTRAN, inout params in CORBA

Parameter Passing – Call by Value-Result

```
int x = 1; // global x

void func1 (int a) {
  // Location 2
  a = 3;
  // Location 3
  x = 4;
  // Location 4
}

void main () {
  // Location 1
  func1(x);
  // Location 5
}
```

Location 1: a undefined, x is 1

Location 2: a is 1, x is 1

Location 3: a is 3, x is 2

Location 4: a is 3, x is 4

Location 5: x is 3

Other Mechanisms

- Call by name
  - Used in Algol
  - Functions are a bit like a complex macro, the argument values replace the corresponding parameter names in the function body and then the body executed
- Call by result
  - Value of parameter copied back to argument at end of function
  - Out params in CORBA

What About Java?

- Everything is Call by Value
- What about when we pass objects?
  - An object variable in Java always holds a reference (pointer) to an instance in Java
  - When called a copy of the reference (or l-value) is made
  - Changes to the object pointed at can be made
  - But the pointer to the object can then also be changed