Tutorial 6

prepared by Anya Tafliovich[1]

July 9, 2003

[1]based on material provided by Diane Horton, Eric Joanis, Alfredo Gabaldon

Parameter passing methods. Trace each of the following.
Example 1.

```
a) by value:  0 0 2
```

```
int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

```
b) by result
(addresses computed at call time):  garbage 0 garbage


int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

```
c) by result
(addresses computed at return time,
 but before any assignments):  garbage garbage 2


int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

```
b) by result
(addresses computed at return time,
 after assignments in left to right order):   garbage 0 2

(unless the first garbage value happens to be 0/1,
 in which case 0/2 gets replaced with garbage)


int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

c) by value-result: 2 0 1

```
int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

d) by reference: 0 0 1

```
int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

e) by name: 0 1 2

```
int i, A[2]
i := 1
procedure foo(int x, int y)
  int temp
  temp := x
  x := y
  i := 0
  y := temp
end

A[0] := 0
A[1] := 2
foo( i, A[i])
write i, A[0], A[1]
```

Example 2:

a) by value-result
 (addresses are computed at call time): 0 0 1

```
main() {
  int value = 0,
  list[0] = 1,
  list[1] = 0

  swap(list[0], list[list[0]]);
  output(value, list[0], list[1]);
}

swap(int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

b) by reference: 0 0 1

```
main() {
  int value = 0,
  list[0] = 1,
  list[1] = 0

  swap(list[0], list[list[0]]);
  output(value, list[0], list[1]);
}

swap(int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

c) by name: 0 1 0

```
main() {
  int value = 0,
  list[0] = 1,
  list[1] = 0

  swap(list[0], list[list[0]]);
  output(value, list[0], list[1]);
}

swap(int a, int b) {
  int temp;
  temp = a;
  a = b;
  b = temp;
}
```

Subroutines

Local offset:
  offset from the beginning of the activation record of the
  local scope; represents a reference to a local variable;
  we assume all variables take 1 position;
  (some sources: first 3 - return address, static link, dynamic link)

Static link:
  (static scope pointer) points to bottom of AR instance of
   the static parent (contains declaration of me)

Dynamic link:
   points to top of activation record of my caller

Static chain:
   a chain of static links in the stack;
   can search static chain to find the correct AR
   instance of a nonlocal variable.

Static depth :
   integer associated with a static scope that indicates how
   deeply it is nested in the outermost scope

Chain offset :
   length of the static chain needed to reach the correct
   activation record instance for a non-local variable X
  ==   static depth of procedure containing the reference to X -
       static depth of procedure containing declaration of X

Reference (static chains method):
(chain offset, local offset)

```
program A;
  procedure B;
    procedure C;
    ...
    end;  { C}
    ...
  end; { B }
...
end; { A }

static depth (A) = 0
static depth (B) = 1
static depth (C) = 2

If procedure C references a variable declared in A,
  chain offset = 2 - 0 = 2.
If procedure C references a variable declared in B,
  chain offset = 2 - 1 = 1.
Reference to a local variable => chain offset = 0.
```

```
Display:
  alternative to static chains;
  static links not stored in the AR;
  instead - array of static links

Contents of display:
  list of addresses of the accessible activation record instances
  in the stack, one for each active scope, in order, in which they
  are nested;

display[k] = ARI for a procedure with a static depth of k

Exactly 2 steps for access to non-local variables:
1) find link to correct AR, using display offset
2) compute local offset within correct AR

Reference = (display offset, local offset)

Call to procedure P, which has static depth k:
1) make new AR; save copy of display[k]
2) display[k] := link to AR for P

Terminate procedure P:
1) display[k]:=saved pointer from AR of P
2) remove AR of P
```

Example:
 trace the state of AR through execusion of the following
 program using both the display method and the static chains method

Using a display:

```
program M;
  procedure P;
    integer x, y, z;
    procedure Q;
       procedure R;
       begin
         ...
         z := P;
         ...
       end R;
    begin /* Q */
      ...
     y := R;
      ...
    end Q;
 begin /* P */
   ...
   Q;
   ...
 end P;
begin /* M */
  ...
  P;
  ...
end M;
```

Using static chains:

```
program M;
  procedure P;
    integer x, y, z;
    procedure Q;
       procedure R;
       begin
        ...
        z := P;
        ...
       end R;
    begin /* Q */
     ...
     y := R;
     ...
    end Q;
 begin /* P */
   ...
   Q;
   ...
 end P;
begin /* M */
  ...
  P;
  ...
end M;
```

The program never terminates (clearly), so just follow it's execution until you have two instances of AR for each of P and Q in the stack. Then assume that the second Q instance terminates, then the second P, to see how previous pointers in the display are restored when a procedure terminates and its AR is removed.