INFORMATION and REMINDERS
WELCOME BACK!

• Homework 5 will be posted by tomorrow.
• Project 1 deadline extension?
• My office hours will be rescheduled; details will be posted on our class webpage
Procedures

• Modularize program structure
  – **Argument**: information passed from caller to callee (actual parameter)
  – **Parameter**: *local* variable whose value (usually) is received from caller (formal parameter)

• Procedure declaration
  – procedure name, formal parameters, procedure body with local declarations and statement lists, optional result type
  example: `void translate(point *p, int dx)`
Parameters

Scott: Chapter 8.3

Parameter Association

- **Positional association**: Arguments associated with formals one-by-one; example: C, Pascal, Scheme, Java.

- **Keyword association**: formal/actual pairs; mix of positional and keyword possible; example: Ada

  procedure plot(x, y: in real; penup: in boolean)

  ... plot (0.0, 0.0, penup => true)

  ... plot (penup => true, x => 0.0, y => 0.0)

Parameter Passing Modes

- **pass-by-value**: C, Pascal, Ada (**in** parameter), Scheme, Algol 68

- **pass-by-result**: Ada (**out** parameter)

- **pass-by-value-result**: Ada (**in** **out** parameter)

- **pass-by-reference**: Fortran, Pascal (**var** parameter)

- **pass-by-name** (not really used any more): Algol60
Review: Stack Frames

- Run-time stack contains frames for main program and each active procedure.
- Each stack frame includes:
  1. Pointer to stack frame of caller (control link)
  2. Return address (within calling procedure)
  3. Mechanism to find non-local variables (access link)
  4. Storage for parameters
  5. Storage for local variables
  6. Storage for final values

![Diagram of stack frame](image-url)
begin
  c: array[1..10] of integer;
  m, n: integer;
  procedure r(k, j: integer)
  begin
    k := k+1;
    j := j+2;
  end r;
...
  m := 5;
  n := 3
  r(m,n);
  r(m,n);
  write m,n;
end

Output?:
Pass-by-value

begin
  c: array[1..10] of integer;
  m, n: integer;
  procedure r(k, j: integer)
  begin
    k := k+1;
    j := j+2;
  end r;
...
  m := 5;
  n := 3
  r(m,n);
  write m,n;
end

Output:
5 3

Advantage: Argument protected from changes in callee
Disadvantage: Copying of values takes execution time and space, especially for aggregate values (e.g.:arrays, structs).
Pass-by-reference

begin
    c: array[1..10] of integer;
    m, n: integer;
    procedure r(k, j: integer)
    begin
        k := k+1;
        j := j+2;
    end r;
    ...
    m := 5;
    n := 3
    r(m,n);
    write m,n;
end

Output:
6 5

Advantage: more efficient than copying
Disadvantage: leads to aliasing: there are two or more names for the same storage location; hard to track side effects
Pass-by-result

begin
  c: array[1..10] of integer;
  m, n: integer;
  procedure r(k, j: integer)
  begin
    k := k+1;          ==> ERROR: CANNOT USE PARAMETERS
    j := j+2;          WHICH ARE UNINITIALIZED
  end r;
...
  m := 5;
  n := 3
  r(m,n);
  write m,n;
end

Output: program doesn’t compile or has runtime error
Pass-by-result

begin
  c: array[1..10] of integer;
  m, n: integer;
  procedure r(k, j: integer)
  begin
    k := 1;            ==> HERE IS ANOTHER PROGRAM
    j := 2;            THAT WORKS
  end r;
...
  m := 5;
  n := 3
  r(m,m);            ==> NOTE: CHANGED THE CALL
  write m,n;
end

Output: 1 or 2?

Problem: order of copy-back makes a difference; implementation dependent.
Pass-by-value-result

begin
  c: array[1..10] of integer;
  m, n: integer;
  procedure r(k, j: integer)
  begin
    k := k+1;
    j := j+2;
  end r;

  m := 5;
  n := 3
  r(m,n);
  write m,n;
end

Output:
6 5

Problem: order of copy-back can make a difference; implementation dependent.
Pass-by-value-result

begin
  c: array[1..10] of integer;
  m, n: integer;
procedure r(k, j: integer)
begin
  k := k+1;
  j := j+2;
end r;
...
/* set c[m] = m */
  m := 2;
  r(m,c[m]); ==> WHAT ELEMENT OF ‘‘c’’ IS ASSIGNED TO?
  write c[1], c[2], ... c[10];
end

Output:
1 4 3 4 5 ... 10 on entry
1 2 4 4 5 ... 10 on exit

Problem: When is the address computed for the copy-back operation? At procedure call (procedure entry), just before procedure exit, somewhere inbetween? (Example: ADA on entry)
More on Aliasing

Aliasing:

More than one way to name the same object within a scope

Even without pointers, you can have aliasing through (global ↔ formal) and (formal ↔ formal) parameter passing.

begin
  j, k, m: integer;
  procedure q(a,b: integer); // call-by-reference
  begin
    b := 3;
    m := m*a;
  end
  ...
  q(m,k);  ==> global/formal <m,a> ALIAS PAIR
  q(j,j);  ==> formal/formal <a,b> ALIAS PAIR
  write y;
end
Comparison: by-value-result vs. by-reference

Actual parameters need to evaluate to L-values (addresses).

```
begin
  y: integer;
  procedure p(x: integer);
  begin
    x := x+1;  ==> ref: x and y are ALIASED
    x := x+y;  ==> val-res: x and y are NOT ALIASED
  end
  ...
  y := 2;
  p(y);
  write y;
end
```

Output:

- pass-by-reference: 6
- pass-by-value-result: 5

Note: by-value-result: Requires copying of parameter values (expensive for aggregate values); does not have aliasing, but copy-back order dependence;
Next Lecture

Functional programming

Please see our website for an online Scheme textbook