Class Information

- Midterm exam: Next Friday, March 14, in class, closed book, closed notes. We will have two exam rooms.
- Homework problem set 5 sample solutions will be posted tonight

Midterm Exam

1. Closed book, closed notes, 80 minutes
2. Jackets, backpacks, all electronic equipment (**includes phones**) have to be stored in the front of the room. Don't bring your own paper.
3. You will need to bring your student ID.
4. Be here 10 minutes early.
5. **Section 1: Tillet 257** - 47 students

   Section 2&3: Tillet 254 - 81 students
6. You are not allowed to leave the room during the exam.

Review: Lexical scoping

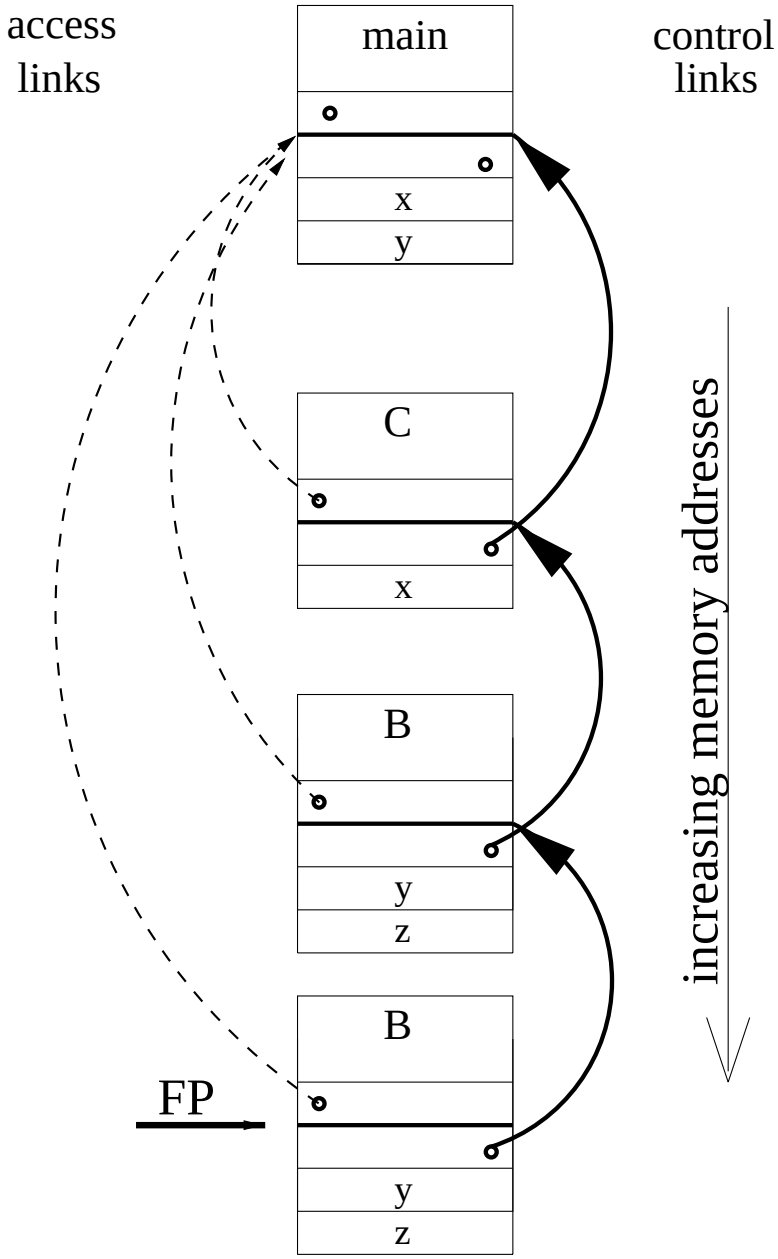
Symbol table matches declarations and occurrences.
⇒ Each variable name can be represented as a pair

(nesting_level, local_index).

```
Program
   (1,1), (1,2):  integer    // declarations of x and y
   begin
      Procedure B          // declaration of B
         (2,1), (2,2):  real   // declaration of y and z
         begin
            ...    // occurrences of y, x, and z
(*)         (2,1) = (1,1) + (2,2)
            if (...)  call B      // occurrence of B
         end
      Procedure C    // declaration of C
         (2,1):  real      // declaration of x
         begin
            ...
            call B    // occurrence of B
         end
      ...
      call C    // occurrence of C
      call B    // occurrence of B
   end
```

Review: Lexical Scoping Example

Calling chain: MAIN ⇒ C ⇒ B ⇒ B



Access to non-local data (lexical scoping)

To find the value specified by (l, o)

- need current procedure level, k
- if $k = l$, is a local value
- if $k > l$, must find l's activation record
  $\Rightarrow$ follow $k - l$ access links
- $k < l$ cannot occur

Maintaining access links:

If procedure p is nested immediately within procedure q, the access link for p points to the activation record of the most recent activation of q.

- calling level $k + 1$ procedure
  1. pass my FP as access link
  2. my backward chain will work for lower levels
- calling procedure at level $l \leq k$
  1. find my link to level $l - 1$ and pass it
  2. its access link will work for lower levels

The display

To improve run-time access costs, use a *display*.

- table of access links for lower levels
- lookup is index from known offset
- takes slight amount of time at call

Access with the display

assume a value described by (l, o)

- find slot as DP$[l]$ in display pointer array
- add offset to pointer from slot

"setting up the activation frame" now includes display manipulation.

Display management

Single global display: *simple method*

on entry to a procedure at level l

save the level l display value

push FP into level l display slot

on return

restore the level l display value

Review: 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)` Xo

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)

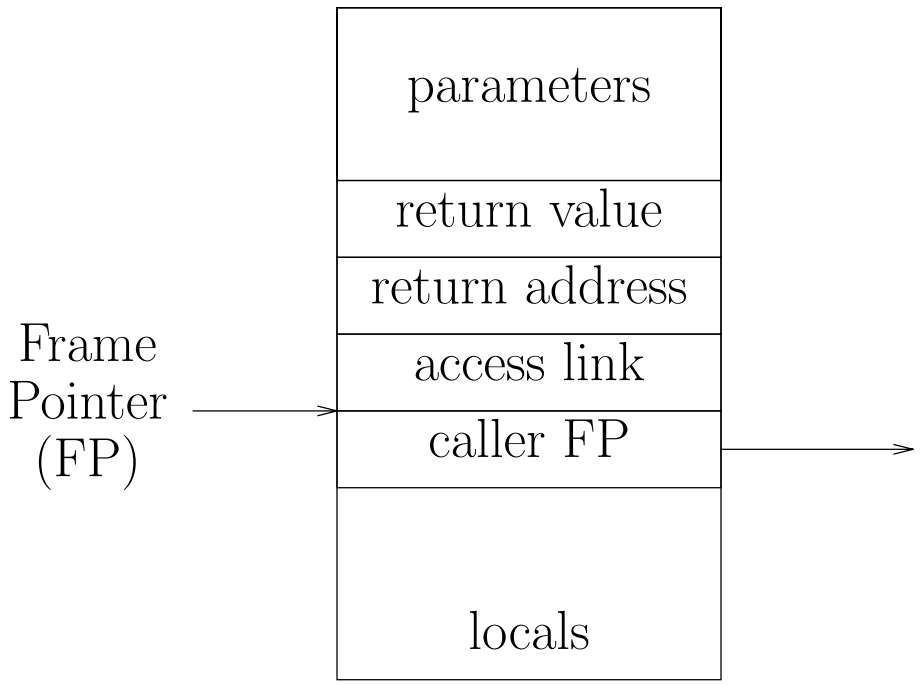
  . . . plot (penup ⇒ true, x ⇒ 0.0, y ⇒ 0.0)

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



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)

Next Lectures Roadmap (after spring break)

- Introduction to functional languages; read Scott Chapter 10
- Lambda calculus