Chapter 9: Referencing Environments

CMSC 124, 1st Semester, AY 2009-10

Chapter 9: Referencing Environments

Prelude

- A referencing environment of a procedure is the set of identifiers (or data objects) accessible to the procedure.
- **Types of Environments**
  1. Local environment
  2. Non-local environment
  3. Global environment
  4. Common environment

Chapter 9: Referencing Environments

Local Environment

- Set of data objects created on entry to the procedure and accessible to the currently executing procedure.

- **Composition**
  - ✓ Parameters
  - ✓ Variables declared in the procedure

Chapter 9: Referencing Environments

Local Environment

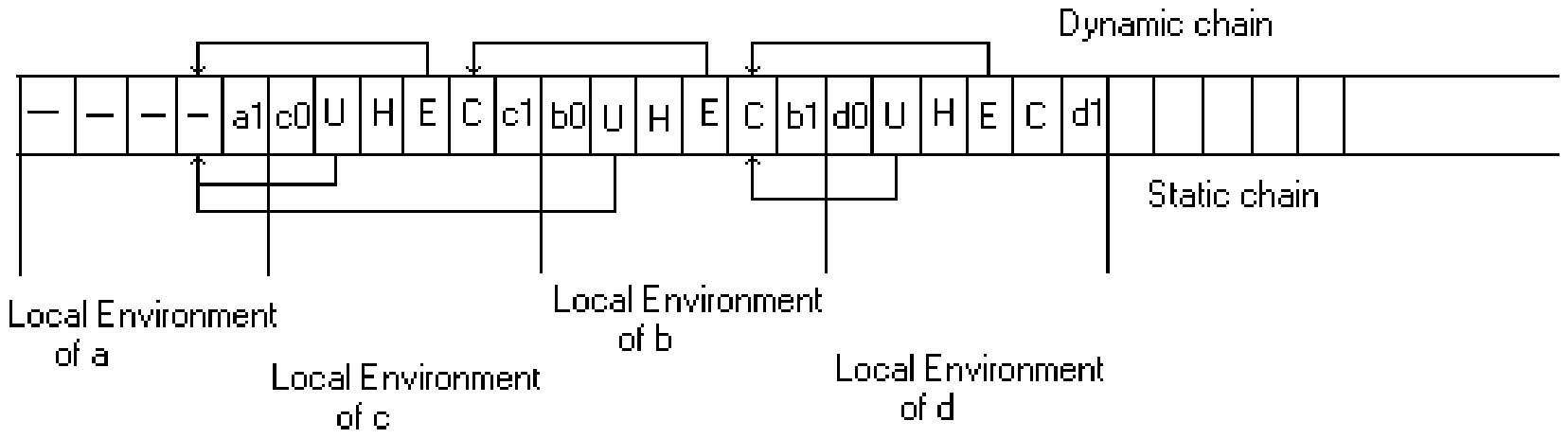
```
program a;
var a1: integer;
  procedure b(b0: integer);
  var b1: integer;
    procedure d(d0: integer);
    var d1: integer;
    begin
    end;
  begin
    d(b1);
  end;
```

```
procedure c(c0: integer);
  var c1: integer;
  begin
    b(c1);
  end;
begin
  c(a1);
end.
```

Procedure	Local Environment
a	a1
b	b0, b1
c	c0, c1
d	d0, d1

Chapter 9: Referencing Environments

Local Environment



where

C - current instruction pointer

E - current environment pointer

H - top of stack

U - address of the nearest non-local environment

Chapter 9: Referencing Environments

Static Variables

- Local environments that persist even though the execution has been completed.
- In C, it is local to a particular function or procedure.
- It is created once (on the first call to the function or procedure) and its value is remembered.
- On the next call to the procedure, the static variable has the same value as the value when procedure is last called.
- In Java, it is stored in the code segment.

Chapter 9: Referencing Environments

Static Variables

```
main() {
 int i;
 for (i=0;i<5;++i)
   count();
}

count(){
 int local_var = 0;
 static int static_var = 0;
 printf( ``local = %d, static
   = %d n'', local_var,
   static_var);
 ++local_var;
 ++static_var;
}
```

Output

local = 0, static = 0

local = 0, static = 1

local = 0, static = 2

local = 0, static = 3

local = 0, static = 4

Initialization is done only once at compile time when memory is allocated for the static variable

Chapter 9: Referencing Environments

Non-Local Environment

- Set of data objects that is accessible to the procedure even though these are created before the procedure is called.
- Applicable to programming languages where nesting of procedure is allowed.
- Data objects accessible to a procedure that is declared at various level of nesting.

Chapter 9: Referencing Environments

Non-Local Environment

```
program a;
var a1: integer;
  procedure b(b0: integer);
  var b1: integer;
    procedure d(d0: integer);
    var d1: integer;
    begin
    end;
  begin
    d(b1);
  end;
```

```
procedure c(c0: integer);
  var c1: integer;
  begin
    b(c1);
  end;
begin
  c(a1);
end.
```

Procedure	Non-Local Envi.
a	--
b	a1
c	a1
d	a1, b0, b1

Chapter 9: Referencing Environments

Non-Local Environment: Implementing Access

Static Link Implementation

• This is done by maintaining a pointer to the nearest nonlocal data frame accessible to the currently executing procedure.

• The pointer to this nearest data frame is stored in the activation record of a procedure.

Display Method

• Array of pointers to activation records is maintained to keep track of the environment of a procedure.

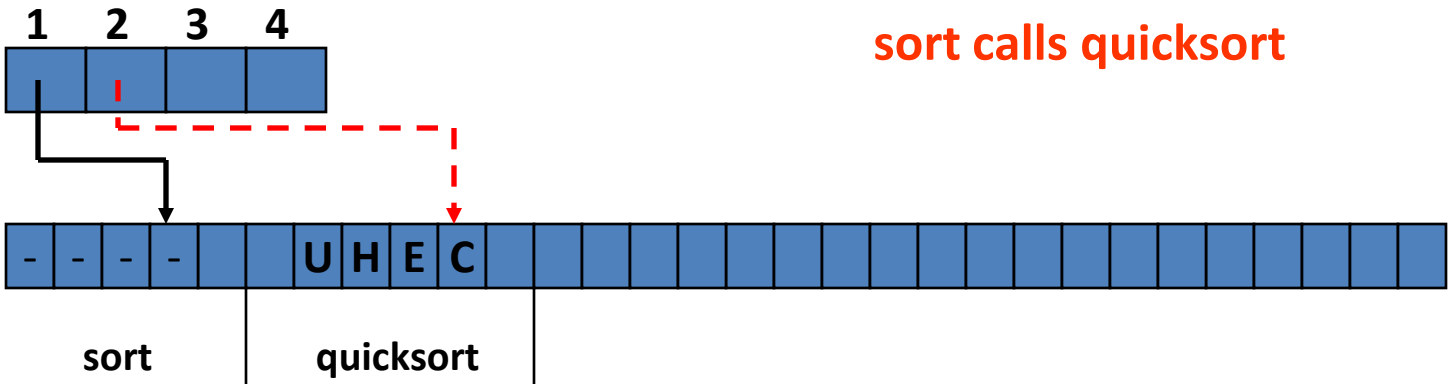
Chapter 9: Referencing Environments

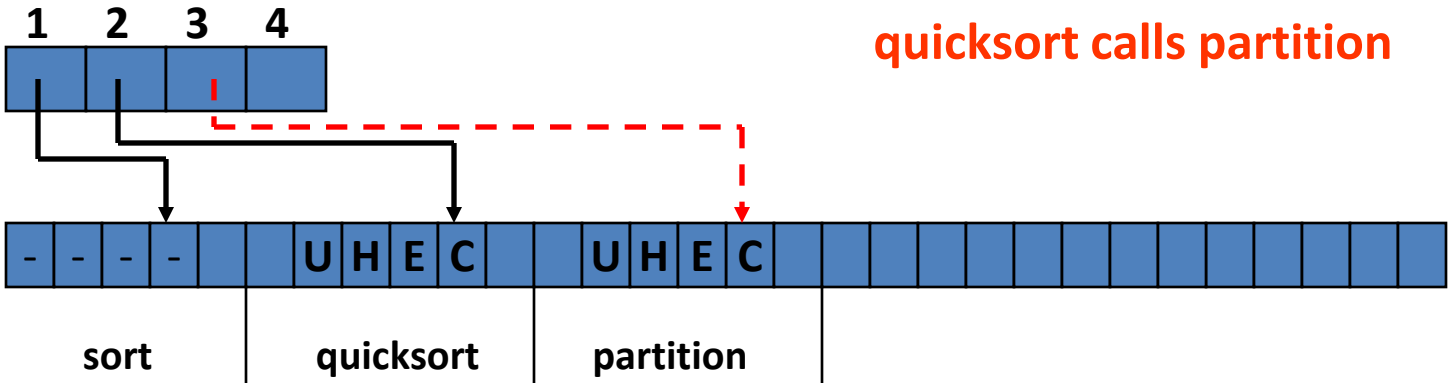
Display Method

```
program sort;
  procedure readarray;
  begin
  end;
  procedure exchange;
  begin
  end;
  procedure quicksort;
    procedure partition;
    begin
      …
      exchange;
      …
   end;
begin
    partition;
    …
    quicksort;
  end;
begin
  quicksort;
end.
```

Chapter 9: Referencing Environments

Display Method





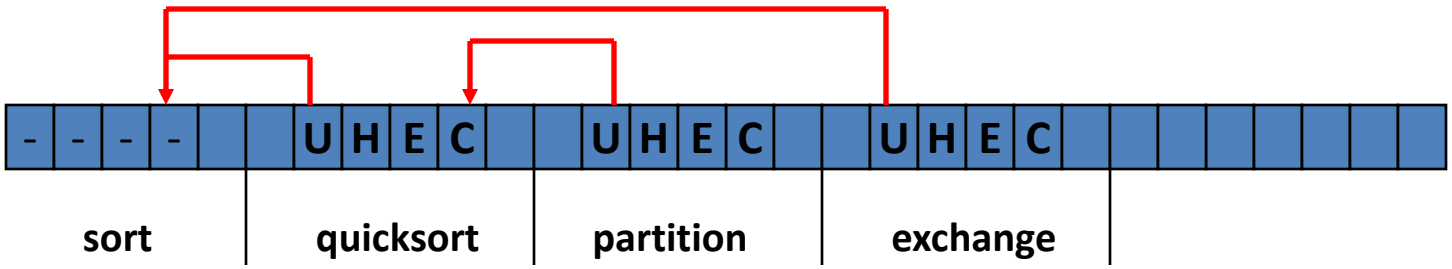
Chapter 9: Referencing Environments

Display Method

- When a new activation record for a procedure at **nesting level i** is set up, the following are executed:
    1. Save the value of display[i] in the new activation record;
    2. Set display[i] to point to the new local environment.
- To access a variable at nesting level i, **display[i]** is used as a reference point.

Chapter 9: Referencing Environments

Static Link Implementation



Chapter 9: Referencing Environments

Global Environment

- Set of data objects created at the start of the execution of a main program.
- Available to any procedure during the execution of the program.
- **Implementation**
Maintain a pointer to the local environment of the main program.

Chapter 9: Referencing Environments

Global Environment

Eg 1 (In Pascal):

```
program programe;
/* global variables */
var i, j,k : integer;
/* procedure declarations */
begin
end.
```

Eg 2 (In C):

```
int i,j,k;
/*global variables */
main() {

}
```

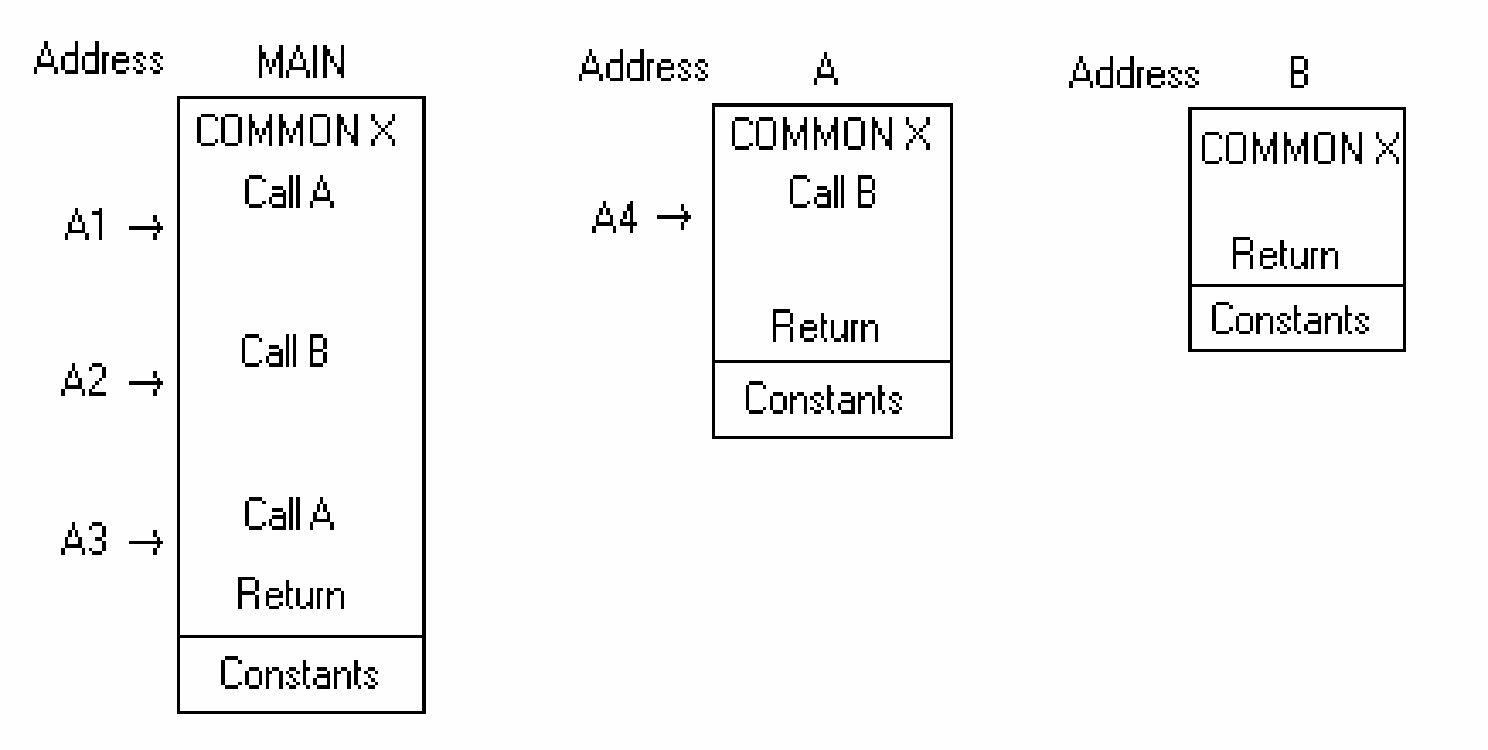
Chapter 9: Referencing Environments

Common Environment

- Usually setup for sharing the data objects to a set of procedures.
- **Implementation**
  - ✓ Allocate a separate block of storage **visible** to a set of procedures.
  - ✓ This block of storage is called:
    - **COMMON** in FORTRAN
    - **package** in ADA
    - **EXTERNAL** in PL/I
    - **compool** in other PL's

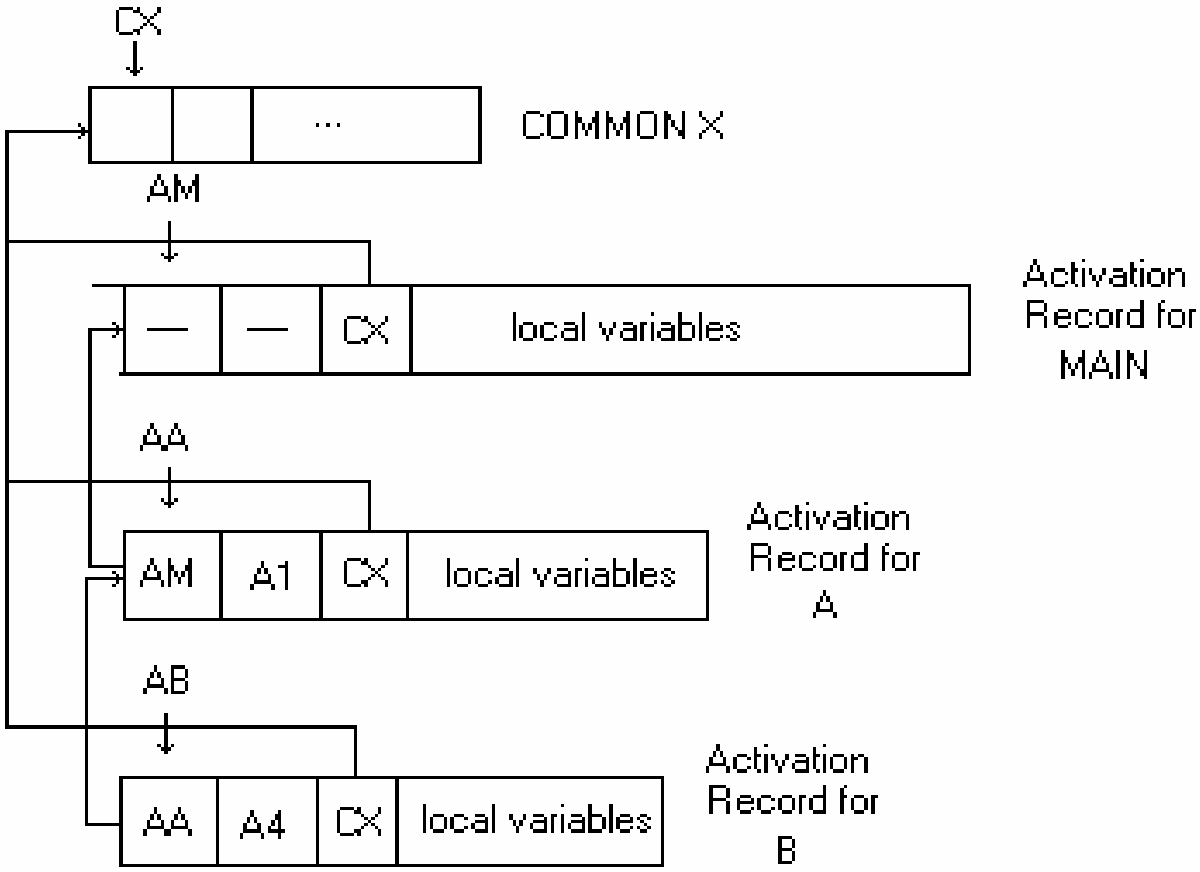
Chapter 9: Referencing Environments

Subroutines and Common Environments



Chapter 9: Referencing Environments

Common Environment



Common Environment: Another Implementation

Import/Export of Shared Variables

- Data object is owned by one procedure.
- Others simply import the data objects.

Implementation

- Allocate a space for the exported variables in the code segment of the procedure.
- The activation record maintain a pointer to the space where the imported variables are allocated space.

```
procedure exporter;
    defines a, b, c:;
    a, b, c: integer;
begin
end;

procedure importer;
    uses exporter.a,
    exporter.b;
    c: integer;
begin
end.
```

Chapter 9: Referencing Environments

Scope Rules

Scope Rules

• The scope of a variable is the set of statements where the variable may be accessed in a program.

Static Scope

• Dependent on the syntax of the program.

Dynamic Scope

• Determined by the execution of the program.

Chapter 9: Referencing Environments

Dynamic Scope

- Dynamic scope of each binding is based on the dynamic course of **program execution**.
- **Common dynamic scope rule:** Includes not only that activation record of the procedure but also any activation of a subprogram called later.

Chapter 9: Referencing Environments

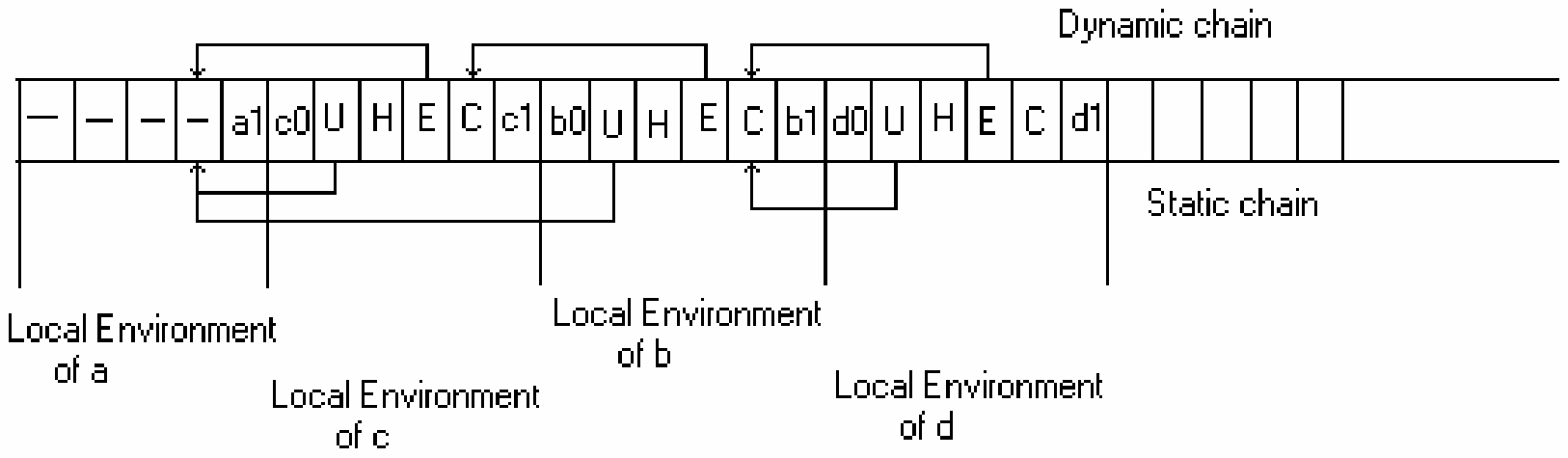
Dynamic Scope

```
program a;
var a1: integer;
  procedure b(b0: integer);
  var b1: integer;
    procedure d(d0: integer);
    var d1: integer;
    begin
    end;
  begin
    d(b1);
  end;
```

```
procedure c(c0: integer);
  var c1: integer;
  begin
    b(c1);
  end;
begin
  c(a1);
end.
```

Chapter 9: Referencing Environments

Dynamic Scope



The dynamic scope of the binding of **a1** to the memory location is the procedure activations of **a, c, b** and **d**, while the dynamic scope of the binding of **d1** is the procedure activation of **d** only.

Chapter 9: Referencing Environments

Static Scope

- The static scope of a declaration is based on a particular declaration (**textual organization**).
- **In the Pascal example (slide 23):**
  - ✓ Static scope of the declaration in program **a**: include the bodies of procedures a, b, c, and d.
  - ✓ Static scope of the declaration in **c**: only the body of procedure c.

Chapter 9: Referencing Environments

Static Scope versus Dynamic Scope

```
program scopes;
var i: integer;
    procedure printi;
    begin
        write(i:3);
    end;
    procedure half;
    var i: integer;
    begin
        i := 5;
        printi;
    end;
begin
    i := 10;
    printi; half; writeln;
    printi; half; writeln;
end.
```

Results

Static Scope

10 10

10 10

Dynamic Scope

10 5

10 5