Concepts Introduced in Chapter 7

- Storage Allocation Strategies
  - Static
  - Stack
  - Heap
- Activation Records
- Access to Nonlocal Names
  - Access links

followed by Fig. 7.1
Activation Records

- An activation record is usually a contiguous block of storage that holds information relevant to one activation (execution) of a routine.

- Activation records can be placed in different storage areas depending on the run-time environment of the programming language.
  - Static data: FORTRAN
  - Stack: ALGOL, C, Pascal, Ada, C++
  - Heap: Functional Languages
    - LISP, ML, Haskell, Erlang
Call Graphs

- Call graphs represent the possible sequence of function calls in a program.
Typical Actions During Call / Return

• Calling sequence actions
  – caller assigns the actual arguments.
  – caller stores return address.
  – caller jumps to callee.
  – callee adjusts stack pointer for the activation record.
  – callee saves nonscratch register values it will use.

• Return sequence actions
  – callee assigns the return value.
  – callee restores nonscratch register values and stack pointer address.
  – callee jumps to the return address.
  – caller accesses the return value.
Storage Allocation Strategies

- Static Allocation - lays out storage for all data objects at compile time
  - Restrictions
    - size of object must be known and alignment requirements must be known at compile-time
    - no recursion
    - no dynamic data structures
Storage Allocation Strategies (cont.)

- Stack allocation - manages run-time storage as a stack (LIFO model)
  - Activation record is pushed on as function is entered.
  - Activation record is popped off as function is exited.
  - Restrictions
    - Values of locals cannot be retained when an activation ends.
    - A called activation cannot outlive a caller.

followed by Fig. 7.6, 7.7
Issues to Address in Calling Conventions

- Responsibility of the caller versus the callee.
- Identifying registers for special purposes.
  - stack pointer, frame pointer, return address
- Preserving the value of registers across calls.
  - callee save, caller save
- Passing arguments.
  - through registers, on the stack
Dangling Reference

```c
main ( )
{
    int *p;
    p = dangle ( );
}

int *dangle ( )
{
    int i = 23;
    return &i;
}
```
Storage Allocation Strategies (cont.)

• Heap Allocation - allocates and deallocates storage as needed at run-time from a data area known as a heap
  – Most flexible
  – Most inefficient
Heap Storage Reclamation Strategies

- explicit reclamation
  - used in Pascal, Ada, C, C++
  - complicates programming
- implicit reclamation
  - used in Lisp and Java
  - reference counts
  - mark and sweep (garbage collection)
Access to Nonlocal Names

• The scope of a declaration in a block-structured language is given by the most closely nested rule:
  – The scope of a declaration in a block B includes B.
  – If a name X is not declared in a block B, then an occurrence of X in B is in the scope of a declaration of X in an enclosing block B´ such that
    • B´ has a declaration of X, and
    • B´ is more closely nested around B than any other block with a declaration of X.
Lexical Scope without Nested Procedures

- The lexical-scope rules for a language without nested procedures, such as C, are simpler than those of a block-structured language, such as Pascal.
- The scope of a declaration outside a function consists of the function bodies that follow the declaration.
Lexical Scope without Nested Procedures (cont.)

- C functions accessing a nonlocal variable `a`.

```c
int a[11];
readarray() { ... a ... }
int partition(int y, int z) { ... a ... }
quicksort(int m, int n) { ... a ... }
main() { ... a ... }
```

- Note that all functions access the same `a`.

- Functions in C can be easily passed as parameters and can be returned as a function result.
Lexical Scope with Nested Procedures

• In a block structured language, a procedure can access nonlocal names that are local variables in other procedures.

• Since such variables are local variables in other procedures, they are placed in activation records on the run-time stack.

• How can they be accessed at run-time?

followed by Fig. 7.10
Referencing Variables with Access Links

- An access link points to most recent activation of the procedure that contains the current procedure.

- Suppose
  - $N_p$ is the nesting depth of procedure $p$ that refers to a nonlocal variable $a$.
  - $N_a$ is the nesting depth of procedure that contains $a$.

- $N_p - N_a$ access links would have to be traversed when in procedure $p$ to get to the activation record that contains $a$.

followed by Fig. 7.11, 7.12, 7.13
Establishing Access Links

- Suppose \( p \) calls \( x \).
- Two cases for setting up the access link.
  - If \( N_p < N_x \)
    - Procedure \( x \) is nested more deeply than procedure \( p \). \( x \) must be declared within \( p \). The access link should point to the caller.
  - If \( N_p \geq N_x \)
    - Follow \( N_p - N_x + 1 \) access links from the caller to reach the activation record of the procedure that encloses the called procedure \( x \) most closely.

followed by Fig. 7.14