Figure 7.1: Typical subdivision of run-time memory into code and data areas.
int a[11];

void readArray() { /* Reads 9 integers into a[1], ..., a[9]. */
    int i;

    ...}

int partition(int m, int n) {
    /* Picks a separator value v, and partitions a[m..n] so that
       a[m..p-1] are less than v, a[p] = v, and a[p+1..n] are
       equal to or greater than v. Returns p. */

    ...}

void quicksort(int m, int n) {
    int i;
    if (n > m) {
        i = partition(m, n);
        quicksort(m, i-1);
        quicksort(i+1, n);
    }
}

main() {
    readArray();
a[0] = -9999;
a[10] = 9999;
    quicksort(1,9);
}

Figure 7.2: Sketch of a quicksort program
enter main()
  enter readArray()
leave readArray()
enter quicksort(1,9)
  enter partition(1,9)
leave partition(1,9)
enter quicksort(1,3)
  ...
leave quicksort(1,3)
enter quicksort(5,9)
  ...
leave quicksort(5,9)
leave quicksort(1,9)
leave main()

Figure 7.3: Possible activations for the program of Fig. 7.2
Figure 7.4: Activation tree representing calls during an execution of quicksort
Figure 7.5: A general activation record
(a) Frame for main

(b) $r$ is activated

(c) $r$ has been popped and $q(1, 9)$ pushed

(d) Control returns to $q(1, 3)$

Figure 7.6: Downward-growing stack of activation records
Figure 7.7: Division of tasks between caller and callee
Figure 7.8: Access to dynamically allocated arrays
1) fun sort(inputFile, outputFile) =
   let
     val a = array(11,0);
   2) fun readArray(inputFile) = ... ;
   3)     ... a ... ;
   4) fun exchange(i,j) =
   5)     ... a ... ;
   6) fun quicksort(m,n) =
       let
   7)         val v = ... ;
   8)     fun partition(y,z) =
       9)         ... a ... v ... exchange ...
           in
   10)         ... a ... v ... partition ... quicksort
           end
   11) in ... a ... readArray ... quicksort ...
   12) end;

Figure 7.10: A version of quicksort, in ML style, using nested functions
Figure 7.11: Access links for finding nonlocal data
fun a(x) =
  let
    fun b(f) =
      ... f ...
    ;
    fun c(y) =
      let
        fun d(z) = ...
      in
        ... b(d) ...
      end
    in
      ... c(1) ...
  end;

Figure 7.12: Sketch of ML program that uses function-parameters
Figure 7.13: Actual parameters carry their access link with them
Figure 7.14: Maintaining the display