Storage Areas

Abstract machines usually offer the following storage areas:

- **Code:** Instructions for the program. (Usually static: does not change after the program is loaded into memory).
- **Static Area:** Space for variables with global scope (lifetime is same as that of the program).
- **Stack:** Space for variables/data structures that are created at run-time (whose lifetime is **same as** the procedure where they were created).
- **Heap:** Space for data structures that can be dynamically allocated or deallocated (whose lifetime is different from the procedure where they were created).

Variables and Storage

Consider the following program to compute the **factorial** function:

```
int fact(int n) {
  if (n == 0)
     return 1;
  else
     return n * fact(n-1);
}
```

How many different variables are used in the evaluation of `fact(10)`?

Stack Storage

- Stack space is used for variables that are *local* to a procedure (including formal parameters).
- Stack variables allocated for a procedure are visible during the execution of the procedure
  and can be safely **de-allocated** upon exit from the procedure
- Advantages:
  - Same space can be used for two procedure calls whose durations do not overlap in time
  - Local variables can be accessed at a "fixed offset" from a point on stack (use of relative addressing)
  - Non-local variables can also be accessed using relative addressing (more on this later).

Nesting of Procedure Calls: Quicksort

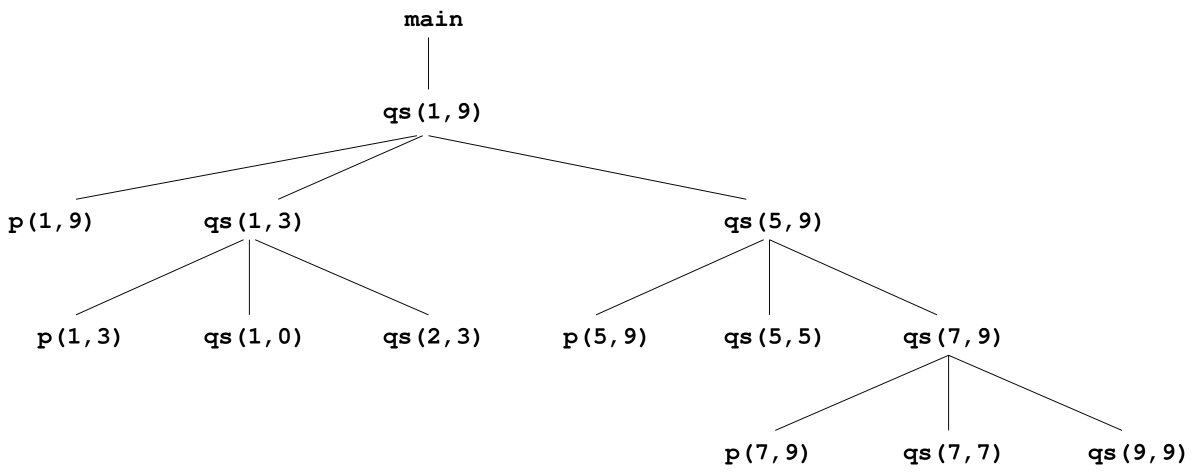
```
int a[10];

int partition(int m, int n) { //Choose a pivot value "v", and
   // reorder sub-array a[m..n] such that a[m..p-1] are all < v,
   // a[p] = v, and a[p+1...n] are all >= v;      return p
...}

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

Activation Tree

- Nodes of the tree are all procedure calls done in one execution of a program
- Root of the tree is the call to `main`
- *q* is a descendant of *p* if a call to *p* results in a call to *q*.
- The sequence of procedure calls $\equiv$ pre-order traversal of activation tree.
- The sequence of returns $\equiv$ post-order traversal of activation tree.

Activation Tree: QuickSort

Activation tree for one execution of `quicksort`:



Activation Stack

Stack of current activations

- If a procedure p has been called, but has not yet returned, then that activation of p is “live”.
- If control is in one activation, say N of a procedure, then all activations on the path from root to N of the activation tree are “live”.
- The information regarding the live activations are kept on a stack, with the most recent call on top of the stack.

Activation Record

Information about a single activation of a procedure:

- Actual parameters of the procedure
- Space for return value

- Book-keeping information (more on this later)
- Saved machine status (volatile registers)
- Local data (space for local variables)
- Temporaries (data not placed in variables/registers)

Book-keeping in Activation Records

- **Control Link:** a link to the previous activation record on stack. Example: in x86, the following code is commonly found at the beginning of every procedure:

  ```
  pushl %ebp
  movl  %esp, %ebp
  ```

  The *Base Pointer* `ebp` points to the current activation record. When a call is made, the pointer to old activation record is first saved on stack (forming the control link). Then `ebp` is made to point to the current activation record (which is on top of stack)
- **Access Link:** a link used to access non-local variables (more on this later).

Space for variables

- Space for formal parameters and local variables of a procedure is allocated in the activation record.
- These variables are accessed using relative addressing (relative to the "base" of the activation record)
- Space for global variables is allocated in *Static Area*.

Nested Procedure Definitions

- In languages like C, procedures are all defined at the “global” level.
- In languages like Pascal and SML, procedure definitions can themselves be nested.
  For example:

```
proc p(int i) {
   int x;
   proc q(int j) {
      int y;
      ... some expression with x ...
   }
   ....
   call q
}
```

Access Link

```
proc p(int i) {
   int x;
   proc q(int j) {
      int y;
      ... use of x ...
   }
   ....
   call q
}
```

- Note that `q` can access `x`, which is in `p`’s activation record.
- `q`’s activation record contains a link to `p`’s activation record, called its *access link*.
- This link is used as the base address for `x`.