1. (10 pts)
Show the run-time stack with all activation record instances, including static and dynamic chains, when execution reaches position 1 in the following skeletal program (in C-style syntax). Assume that the programming language is statically scoped but does not allow functions as return values.

```c
void Foo () {
    void B (int);

    void A () {
        void C () {
            ...
            // position 1
        }
        ...
        C ();
    }

    void B (int flag) {
        void D () {
        ...
            B (1);
        }
        if (flag)
            A ();
        else
            D ();
        ...
    }

    B (0);
    ...
}
```

The calling sequence for this program for execution to reach C is: main calls Foo, Foo calls B, B calls D, D calls B, B calls A, and A calls C.
2. (10 pts)
Consider the following program fragment:

```c
void test () {
    int i;
    int[5] a;

    void f (int x) {
        a[i] = 5;
        i = i - 1;
        i = x;
    }

    i = 2;
    a[1] = 3;
    a[2] = 7;
    f (a[i]);
    // ...
}
```

What are the values of `i` and `a[2]` after function `f` returns if the parameter is passed by

<table>
<thead>
<tr>
<th>i</th>
<th>a[2]</th>
</tr>
</thead>
<tbody>
<tr>
<td>value</td>
<td></td>
</tr>
<tr>
<td>copy-in-copy-out</td>
<td></td>
</tr>
<tr>
<td>reference</td>
<td></td>
</tr>
<tr>
<td>need</td>
<td></td>
</tr>
</tbody>
</table>

3. (10 pts)
Construct a small piece of code in C-style syntax like in the previous question that behaves differently for call-by-name and call-by-need.

4. (10 pts)
Explain why an implementation of a functional language such as Scheme or ML requires a garbage collector. Why does a C implementation not require a garbage collector? Which garbage collection mechanism would you choose for a functional language and why?