This exercise is about a very simple imperative language which allocates all memory on the stack and supports two datatypes: ints and pointers, along with function calls, basic arithmetic and if statements.

```c
f(beg, end) {
    if (beg == end) {
        return 0;
    } else {
        return *beg + f(beg + 1, end);
    }
}

a() {
    x = allocate(3);
    *(x + 0) = 1;
    *(x + 1) = 2;
    *(x + 2) = 3;
    return f(x, x + 3);
}

b() {
    x = allocate(1);
    *x = 5;
    return f(x, x + 1);
}
```

The language is going to be evaluated in the context of \( \langle e, \Gamma, s, S, P \rangle \), where:

- \( e \) is the current expression
- \( \Gamma \) is a mapping from names to locals
- \( s \) is the address of the top of the stack
- \( S \) is the stack, represented as a map from locations to any required value
- \( P \) is the program, mapping names to arguments, function bodies and return types

Provide the small-step semantics of all the constructs you can identify the the code sample. Unfold 10 steps using the rules you have defined, starting from \( \langle a(), 0, \emptyset, \{ f \mapsto \ldots, a \mapsto \ldots, b \mapsto \ldots \} \rangle \). Same for \( b \).

Comment on the safety of the language.