Exercises 9 (Variables and storage)

9A. *(Lifetimes of global and local variables)*

Consider the following C program:

```
extern int r;
void main () {
  r = fac(3);
}
int fac (int n) {
  int f, i;
  f = 1;
  for (i = 2; i <= n; i++)
    f *= i;
  return f;
}
```

(a) Draw a diagram showing the lifetimes of the global and local variables in this program. (Note that the formal parameter `n` is, in effect, a local variable of the `fac` function.)

(b) Repeat with the above version of the `fac` function replaced by the following recursive version:

```
int fac (int n) {
  if (n > 1) return n * fac(n-1);
  else return 1;
}
```

9B. *(Lifetimes of global and heap variables)*

Consider the following outline of a Java program:

```
class IntList {
  // An IntList object is a sorted list of integers.
  private int elem;
  private IntList succ;
  public IntList () {
  // Create an empty sorted list.
    ...
  }
  public void add (int i) {
  // Insert a node containing i into this sorted list.
    ...
  }
  public void rem (int i) {
  // Remove the first node containing i from this sorted list.
  // Do nothing if there is no such node.
    ...
  }
  private static IntList ns;
  public static void main () {
    ns = new IntList();
    ns.add(6);  ns.add(2);  ns.add(9);  ns.add(5);
    ns.rem(9);  ns.rem(6);
  }
}
```

Make a diagram showing the lifetime of the variable `ns`, and the lifetimes of the list nodes containing 6, 2, 9, and 5.

9C. *(Equivalent commands)*

We can gain insight into the nature of commands by stating equivalences between different commands. For example, in Java:

```
C ;              ≡ C                (The skip command ";" has no effect.)
if (E) C         ≡ if (E) C else ;
while (E) C      ≡ if (E) {
                     C
                     while (E) C
                   }
```

where *C* is any command and *E* is any expression.

On the other hand, the sequential command "C_1 C_2" is *not* equivalent to "C_2 C_1" (since the order of execution matters).

Write down as many equivalences between Java commands as you can discover.

9D. *(Expressions with side effects)*

(a) Some programmers assert that side effects in expressions are a useful feature of a programming language. Develop arguments both *for* and *against* this assertion.

(b) Non-void functions in C may have side effects (as well as returning results). Formulate restrictions on such functions that would eliminate side effects. Bear in mind that any such restrictions should, ideally, be enforceable by the compiler. Do your restrictions significantly reduce the language's expressive power?

Exercises 10 (Bindings and scope)

10A. *(Environments)*

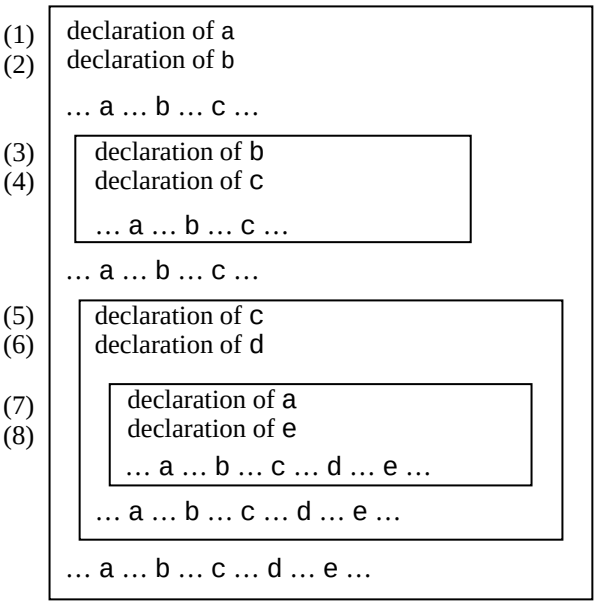
What is the environment at each numbered point in the following C program?

```
        extern int n;
        void zero () {
(1)       n = 0;
        }
        void inc (int d) {
(2)       n += d;
        }
        void main (int argc, char** argv) {
(3)         ...
        }
```

10B. *(Block structure)*

The following is an outline of a program in a block-structured language. The outline shows binding occurrences (‘declaration of *x*’) and applied occurrences (‘… *x* …’) of some identifiers. The boxes represent blocks.

For each applied occurrence, state whether it is legal, and if so identify its associated binding occurrence.



10C. *(Static vs dynamic scoping)*

Consider the following outline of a program in a C-like language:

```
        int add (int i) {
          return i + d;
        }
        void p () {
          const int d = 1;
(1)       print(add(20));
        }
        void q () {
          const int d = 2;
(2)       print(add(20));
        }
```

(a) If the language is *dynamically* scoped, what will be printed at points (1) and (2)?

(b) If the language is *statically* scoped, what will happen?

10D. *(Initializing variable declarations)*

In some languages all variables must be initialized in their declarations, perhaps using a default initial value for each type (such as *false* for booleans, 0 for integers, *null* for pointers). What are the advantages and disadvantages of compulsory initialization of variables?

10E. *(C type declarations)*

(a) List all the forms of type definition in C.

(b) Redesign this part of C's syntax so that there is only one form of type definition, say "**`type`** *I* **`=`** … **`;`**".