Quiz Section 7: ADT Correctness

In the problems, we will implement the following interface:

```
/**
 * A **mutable** list of integers that can only be modified by adding
 * and removing elements at the front or emptying the entire list.
 */
public interface MutableIntStack {
    /**
     * Returns the length of the list.
     * @returns len(obj)
     */
    int length();

    /**
     * Adds the given number to the front of the list.
     * @param n The number to add to the list.
     * @modifies obj
     * @effects obj = n :: obj_0
     */
    void push(int n);

    /**
     * Removes and returns the first element in the list.
     * @requires len(obj) != 0
     * @modifies obj
     * @effects obj_0 = n :: obj
     * @returns n
     */
    int pop();

    /**
     * Removes all elements in the list.
     * @modifies obj
     * @effects obj = nil
     */
    void clear();
}
```

We will implement this interface by storing the list in a compressed manner, where multiple consecutive, identical values are stored as a pair that records the value and how many times it occurs.

To formalize this, we first need a type that represents a list of *pairs* of integers. We can define such a type inductively as follows:

$$\textbf{type}\ \mathsf{PList} := \mathsf{pnil} \mid \mathsf{pcons}((\mathbb{Z}, \mathbb{N}), \mathsf{PList})$$

We will use the shorthand "::", when applied to a PList, to refer to a pcons operation. For example, the expression "$(1,2) :: (3,4) :: \mathsf{pnil}$" is shorthand for $\mathsf{pcons}((1,2), \mathsf{pcons}((3,4), \mathsf{pnil}))$. Hopefully, this will not cause confusion with cons on List.

With that definition in hand, our concrete representation will store the list $1 :: 1 :: 1 :: 2 :: 3 :: 3 :: \mathsf{nil}$, which contains runs of 1s and 3s, as the shorter list $(1,3) :: (2,1) :: (3,2) :: \mathsf{pnil}$. The following class uses this concrete representation:

```
public class CompressedIntStack implements MutableIntStack {
    // AF: obj = expand(this.pairs)
    private PairList pairs;
```

This uses the `PairList` class, defined below, to store a PList, and the function expand : (PList) → List, which is defined as follows:

$$\begin{aligned} \mathsf{expand}(\mathsf{pnil}) &:= \mathsf{nil} \\ \mathsf{expand}((n,0) :: L) &:= \mathsf{expand}(L) \\ \mathsf{expand}((n,c+1) :: L) &:= n :: \mathsf{expand}((n,c) :: L) \end{aligned}$$

The abstraction function of `CompressedIntStack` says that the abstract state is the list that you would get by expanding the PList stored in the field `pairs`.

Finally, the `PairList`, which stores a PList directly as a linked list, is defined as follows:

```
/** Represents a list of pairs. */
private static class PairList {
    public final int value;
    public final int count;
    public final PairList next;

    public PairList(int value, int count, PairList next) {
        this.value = value;
        this.count = count;
        this.next = next;
    }
}
```

Task 1 – Hold My Clear [6 pts]

The `clear` method in `CompressedIntStack` is implemented as follows:

```
public void clear() {
  this.pairs = null;
  {{ P: ________________________________ }}
}
```

a) Use forward reasoning to fill in the blank assertion above.

Remember that, since we are using `PairList`, not implementing it, we should describe the value of `this.pair` in terms abstract states (PLists).

b) Prove that the specifications claim, in `@effects`, that "obj = nil" holds at this point.

c) Given that this is a mutator, what other fact do we need to prove to know that the implementation is correct? How do we know that it holds?

Task 2 – Stack-a-doodle **[12 pts]**

The `push` method in `CompressedIntStack` is implemented as follows:

```
public void push(int n) {
  if (this.pairs == null) {
    this.pairs = new PairList(n, 1, null);
    {{ P1: ________________________________ }}
  } else {
    {{ this.pairs_0 = (m, c) :: L }}
    if (this.pairs.value == n) {
      this.pairs = new PairList(n, this.pairs.count+1, this.pairs.next);
      {{ P2: ________________________________ }}
    } else {
      this.pairs = new PairList(n, 1, this.pairs);
      {{ P3: ________________________________ }}
    }
  }
}
```

a) Use forward reasoning to fill in the blank assertion P_1 above. Then, prove that P_1 implies that the spec's claim that obj $= n ::$ obj$_0$ holds.

b) Use forward reasoning to fill in the blank assertion P_2 above. Then, prove that P_2 implies that the spec's claim that obj $= n ::$ obj$_0$ holds.

Note that, if this.pairs $\neq$ pnil, then it must be $(m, c) :: L$ for some integers m and c and some PList L. That fact is already filled in above.

c) Use forward reasoning to fill in the blank assertion P_3 above. Then, prove that P_3 implies that the spec's claim that $\text{obj} = n :: \text{obj}_0$ holds.