C+- Formal Specification Language Conventions

1. The `return` and `throw` keywords are used as values, not as a control constructs.
   a. E.g., `(return == x)` in a postcondition states that the return value of a function is equal to `x`.
   b. `throw == ExceptionX` in a postcondition states that thrown exception value of the function is equal to `ExceptionX`.

2. The `if`, `then` and `else` keywords are used as expression operators not as control constructs. To be precise, the following C+- if-then-else expression

   ```
   if X then Y else Z
   ```

   is semantically equivalent to the following standard C++ expression

   ```
   X ? Y : Z
   ```

   The following if expression

   ```
   if X then Y
   ```

   is semantically equivalent to *no* standard C++ expression, since C++ always requires the "else" part of and if-then-else expression. Therefore, the elseless if-then expression above is equivalent to the following C++

   ```
   X ? Y : NULL
   ```

   for all types for which NULL is a legitimate value.

3. The "prime" notation is added, where *x* means call-state value of *x* and *x'* means return-state value of *x*. The prime suffix can only be appended to function parameters or to accessible data members within parameters.

4. Single-assignment bindings are provided in the following form:

   `name` `:= ` *`expression`*

   The following C+- binding

   ```
   X := Y
   ```

   is semantically equivalent to the following standard C++ macro definition:

   ```
   #define X Y
   ```

   where X must be a syntactically legal C+- identifier and Y must be a syntactically and semantically legal expression within the scope where the binding appears.

   Given the restriction noted above for the prime suffix, a bound name cannot be appended with a prime.

5. Universal and existential quantifiers are supplied, with the following syntax:

   ```
   forall ( simple-declarator-list [ | expression1 ] ) expression2
   exists ( simple-declarator-list [ | expression1 ] ) expression2
   ```

   The universal quantification form is read "for all variables declared in the simple-declarator-list such that expression1 is true, expression2 is true", and the "such that expression1 clause is optional. The existential form is read similarly.

6. We add the `in` operator for testing if a value is the value of at least one element of an array.

7. We add the `freed` predicate to test if an object's storage has been freed

8. Finally we add the basic syntax for declaring preconditions and postconditions for functions.
   a. The general syntax is:

```
function-declarator;
    let: bindings;
    pre: boolean-valued-expression;
    post: boolean-valued-expression;
```

   where multiple bindings are separated by semicolons.

   b. For example

```
int F(int x, int y);
    pre: (x <= 0) && (y >= 100);
    post: return == x+y;

int G(int& i, int& j);
    let:  x := i + j;
          y := i - j;
          z := x * y;
          w := i' + j';
    pre:  x < y;
    post: (x < y) &&
          (return == z + w);

int H(int& i, int& j,int k) {
    i = i+j;
    return i * k;
};
    post: (i' == i+j) &&
          (j' == j) &&
          (return == i' * k);
```