LPL compared with Java

The semantics of LPL can be readily understood by comparison with the procedural core of Java (no classes). Making that comparison is the purpose of this document. This document does not attempt to provide complete coverage of LPL's syntax (consult the LPL grammar document for that).

Compare the Java program on the left with the LPL program on the right:

```
class Main {

  public static void main(String[] a) {
    int x = Integer.parseInt(a[0]);
    int[] b = new int[] { x, x };
    inc(b);
    System.out.println(sumTwo(b));
  }

  static void inc(int[] a) {
    for(int i = 0; i < a.length; ++i) {
      a[i] = a[i] + 1;
    }
  }

  static int sumTwo(int[] a) {
    return a[0] + a[1];
  }

}
```

```
def unit test(int x) {
  int[] b;
  b = new int[2];
  b[0] = x; b[1] = x;
  inc(b);
  output sumTwo(b);
  outchar 13; outchar 10;
}

def unit inc(int[] a) {
  int i; i = 0;
  while (i < (a.length)) {
    a[i] = (a[i]) + 1;
    i = i + 1;
  }
}

def int sumTwo(int[] a) {
  return (a[0]) + (a[1]);
}
```

- An LPL program starts with an initial *function definition*, which plays a similar role to a `main` method in Java. In this example the initial function is called `test` and it has a return type of `unit`; the `unit` type is similar to `void` in Java (but see below). In contrast with a `main` method in Java, which must have a single parameter of type `String[]`, the initial function in LPL can have multiple parameters of various types (there is no `String` type in LPL, however).
- The rest of an LPL program is a sequence of zero or more function definitions. All function definitions in LPL are like static methods in Java. An LPL function with `unit` return type corresponds to a Java method with `void` return type but, in contrast to `void` in Java, `unit` is a first-class type in LPL; for example you can declare a variable with type `unit`, though there is nothing useful that can be done with such a variable.
- As in Java, an LPL `return` statement is used to return a value from a function. In contrast with Java, return statements in LPL functions are optional; if execution of an LPL function reaches the end of the function body without executing an explicit return, then it returns the default value for the function's return type (0 for `int`, false for `bool`, `null` for array types).

- The LPL command `output` behaves the same way as `System.out.print` in Java, but it can *only* be applied to expressions of type `int`.
- LPL has an `outchar` command for printing single characters, where the character is specified by its Unicode code (which coincides with ASCII for ASCII characters). So `outchar 13; outchar 10;` prints a newline (a CR followed by a LF).
- Because there is no class structure, LPL doesn't have visibility modifiers.

Additional features of LPL

- Numbers and operators: **`int`** is the only numeric type in LPL. **`/`** is integer division. **`&&`** and **`||`** are Boolean conjunction and disjunction, respectively, both with short-circuit semantics (as in Java, if the first argument of **`&&`** evaluates to false, or if the first argument of **`||`** evaluates to true, these operators return a result immediately, without evaluating their second argument).
- Arrays: array creation works in essentially the same way as in Java. Array elements are initialized to the default value for their element type. For example, **`new`** `bool[5]` creates a new array of Booleans, with five elements (all initially containing the value `false`).

  Unlike Java, there is no special syntax for initialising multi-dimensional arrays. For example, in Java you could write

  ```
  int[][] a = new int[2][5];
  a[1][4] = 77;
  ```

  but in LPL you would have to do something like:

  ```
  int[][] a;
  a = new int[][2];
  a[0] = new int[5]; a[1] = new int[5];
  (a[1])[4] = 77;
  ```

- Null pointers: **`isnull`** tests if a reference is null; it can only be applied to expressions of array type. The Java equivalent of (**`isnull`** *e*) would be (*e* **== null**) but there is no explicit literal for null in LPL. As in Java, all elements in a newly created array of arrays will contain null (see above).