Notes on Implementing Arrays

Arrays

The simplest way to implement arrays is to provide three built-in functions, `newArray`, `getArray`, and `setArray`. Let’s assume `newArray` takes a single argument, the size of the array to be constructed. Here is an evaluator function for `newArray`:

```javascript
function evalNewArray(evaluatedArgList) {
    assert(length(evaluatedArgList) == 1) // ensure only one argument
    var size = car(evaluatedArgList);
    assert(size.type == INTEGER); // ensure an integer argument
    var a = new Lexeme(ARRAY);
    a.aval = new Lexeme[size.ival]; // allocate the array
    assert(a.aval != NULL); // ensure a good allocation
    return a;
}
```

This creates a lexeme of type `ARRAY` whose `aval` field points to an array of lexeme pointers. The size of the `aval` array was specified by the first (and only) argument to the `newArray` built-in function. In C, the type of the `aval` would be:

```c
struct Lexeme {
    ....
    Lexeme **aval;
    ....
};
```

In C, the allocation of the `aval` field would be:

```c
a->aval = malloc(sizeof(struct Lexeme *) * size.ival);
```

The next built-in, `getArray`, is used to retrieve a value from an array. It takes two arguments, the array itself and the index of the value to be retrieved:

```javascript
function evalGetArray(evaluatedArgList) {
    assert(length(evaluatedArgList) == 2);
    var a = car(evaluatedArgList);
    var i = cadr(evaluatedArgList);
    // check for valid types here
    return a.aval[i.ival];
    return v;
}
```

The last built-in, `setArray`, is used to set a particular slot in an array. It takes three arguments, the array itself, the index to be set, and the value to be stored at that index:

```javascript
function evalSetArray(evaluatedArgList) {
    assert(length(evaluatedArgList) == 3);
    var a = car(evaluatedArgList);
    var i = cadr(evaluatedArgList);
    var v = caddr(evaluatedArgList);
    // check for valid types here
    a.aval[i.ival] = v;
    return v; // could also return the previous value
}
```

If you have syntactic support of arrays in your language, you would build parse trees tagged appropriately (say `NEW_ARRAY`, `GET_ARRAY`, and `SET_ARRAY`) and provide evaluators for those tags. These parse trees would have the same number of components as arguments to the built-in functions described above.