Crisp Language Reference

Note: A bunch of the functionality from the C is not a part of my language yet. This is for a reason. I want to get a small subset of C working from top to bottom and then add from there. Better to get a good understanding of the entire process and from there I can expand at a much faster rate.

Types and Variables:

- Variables can be defined anywhere including the global scope.
- Pointers are not explicitly supported, but they are implicitly supported in that arrays are passed to functions by address.
- int will be treated as 32-bit signed integer.
- char will be treated as 8-bit unsigned integer.
- double will have a size of 8 bytes.

Allowed:

- char, int, double, and arrays of said types

Disallowed:

- Comma separated declarations and inits: int x = 5, y = 10;
- User-defined types (typedef, struct, enum, union)
- Pointers
- Type specifiers such as registers, static, volatile, signed, or unsigned

---

Functions:

- Functions can be forward declared. They must return either void, int, char, double or a variation as an array. The same if for function parameters.
- As in Standard C, arrays are passed by address while the base types are passed by value.
- User-defined functions do not support variable arguments or default values.
- There must be an entry function named “main” that returns an integer. No parameters are allowed for the main function.

Operators and Expressions:

- There are some restrictions on the operators – for example, pre-increment/decrement must be performed directly on an identifier, so code such as ------(x) is not allowed. Most other operators can be performed on expressions.
- For logical operators, Standard C rules are followed where any non-zero value is considered "true." However, one restriction is that any such operators on a string/array are not allowed. Logical || and && are short-circuited as they are in Standard C.
- One important note is that assignment in crisp is not treated as an expression. This greatly simplifies expression evaluation, since expressions such as (x = 5) * 5 are not possible.

Allowed:

- Logical Operators: &&, ||, !
- Comparison Operators: ==, !=, &&, <, >, >=, <=
- Assignment Operators: =, +=, -=
- Arithmetic Operators: +, -, *, /, %, ++, --

Disallowed:

- Assignment Operators: *=, /=, %=, &=, |=, ^=, >>=, <<=

Comments:

Single line // comments are supported. Multiline comments are not supported.

Statements:

There are only a few types of statements supported in crisp, but they still allow for complex logic. Compound statements (multiple statements surrounded by braces) are allowed, as are assignment statements, return statements, expression statements, and null statements.

Allowed:

- if (with or without else) is, for

Disallowed:

- break, continue, goto, switch, elseif, do while

Memory:

As there are no explicit pointers types, there is no dynamic memory allocation support in crisp. This means that all variables are local to the function they are declared in. Of course, arrays are implicitly passed by address. This means that it is possible to create arrays with a single element to pass it by address.

Preprocessor/Multiple Source Files:

There is no preprocessor whatsoever, and all code must be contained in a single source file. Though certainly, it would be possible to use an external preprocessor tool to preprocess the source file. However, in practice I'd suspect very few header files from a Standard C program would work within the confines of this subset.

Standard Library:

In order to allow for basic output, crisp allows for calls to the printf function from the Standard C Library. This is the only Standard Library function that is currently supported. The compiler will automatically add a declaration of printf to the IR if it detects use of the function, as there would otherwise be no way to forward declare the function (since crisp does not support variable arguments). No validation of the parameters to printf is performed.

Example program:

```
int partition(char array[], int left, int right, int pivotIdx) {
    char pivotVal = array[pivotIdx];
    int storeIdx = left;
    int i = left;
    char temp;

    // Move pivot to end
    temp = array[pivotIdx];
    array[pivotIdx] = array[right];
    array[right] = temp;

    while (i < right) {
        if (array[i] < pivotVal) {
            // Swap array[i] and array[storeIdx]
            temp = array[i];
            array[i] = array[storeIdx];
```

```
            array[storeIdx] = temp;
            ++storeIdx;
        }
        ++i;
    }

    // Swap array[storeIdx] and array[right]
    temp = array[storeIdx];
    array[storeIdx] = array[right];
    array[right] = temp;

    return storeIdx;
}

void quicksort(char array[], int left, int right) {
    int pivotIdx;

    if (left < right) {
        // Pick the middle point
        pivotIdx = left + (right - left) / 2;

        pivotIdx = partition(array, left, right, pivotIdx);
        quicksort(array, left, pivotIdx - 1);
        quicksort(array, pivotIdx + 1, right);
    }
}

int main() {
```

```
    char letters[] = "thequickbrownfoxjumpsoverthelazydog";
    quicksort(letters, 0, 34);

    printf("%s\n", letters);

    return 0;
}
```