Final Project Details

For the final project you will be working to develop a compiler for a simplified version of the Xi language. The various simplifications are listed below.

General

- `LD/ST` pairs
- No dynamic memory allocation or heap allocation
- C-style semicolons at the end of statement are compulsory
- No strings/characters (`'a'` or `"hello"` not allowed)
- Every program will have a `main` function which acts as the entry point
- The original Xi language specification allows for identifiers to contain `'` (single quote) in their name. We are disallowing such identifiers.
- braces `{ }` compulsory for `if\else\while` blocks (`if (a<b) c=2;` is invalid, `if (a<b) {c=2;}` is valid)

Declarations

- No initialization during declaration (`x: int = 5` not allowed)
- All declarations at the beginning of the program/function
- Single declaration per line (`x: int, y: int` not allowed)

Arrays

- Only 1-dimensional arrays allowed
- `a: int[5];` Array size will be declared using a constant (`a: int[n];` not allowed)
- Array indexing can contain expressions e.g. `a[c[i]]` or `a[b-2]`

Functions

- The maximum stack depth will be 2 i.e. no function calls in the definition of a function other than `main` . `main` is stack depth 1 and all the remaining functions are stack depth 2 (this restriction doesn't apply to recursive function calls like fibonacci/factorial)
- Note that this does not disallow `f(g(h))` or `f(b-2)`
- Only two scopes: Global scope and Current Function scope (`if/else/while` blocks don't count as scopes)
- The last line of the `main` function is a `print` statement which is to be implemented using `syscall` (the `print` statement doesn't appear anywhere else)
- The `main` function neither accepts any parameters nor returns any values

- The last line of every function (except `main`) is a `return` statement
- Every function (except `main`) returns exactly one value
- Every function (except `main`) accepts either one or two parameters

Sample Code

1. Loops and Arrays

```
// sum of elements of an array
// (while loops)

main() {
    n: int;     // length of array
    i: int;     // looping variable
    sum: int;
    a: int[5];  // array
    n = 5;      // length of array is 5

    // set array a to be {-2,3,1,5,-4}
    a[0] = -2;
    a[1] = 3;
    a[2] = 1;
    a[3] = 5;
    a[4] = -4;

    i = 0;
    sum = 0;
    while (i < n) {
        sum = sum + a[i];
        i = i + 1;
    }
    print sum;  // implement using syscall
}
```

2. Non-recursive function calls

```
// compute gcd of two numbers
// (non-recursive functions)

// computes gcd of two integers
gcd(a: int, b: int): int {
    while (a != 0) {
        if (a<b) {
            b = b-a;
        }
        else {
            a = a-b;
        }
    }
    return b;
}

// adds one to input
addone(a: int): int {
    b: int;
    b = a+1;
    return b;
}

// main function
main() {
    a: int;
    b: int;
    c: int;
    a = 174;
    b = 250;
    c = gcd(a, addone(b+1)); // => 6
    print c;                 // implement using syscall
}
```

3. Recursive function calls

```
// compute the ith fibonacci number
// (recursive functions)

// compute the ith fibonacci number
// (assume i >= 0)
fib(i: int): int {
    res: int;
    res = -1;
    if (i <= 1) {
        res = i;
    } else {
        res = fib(i-1) + fib(i-2);
    }
    return res;
}

// main function
main() {
    i: int;
    f: int;
    i = 7;
    f = fib(i);  // => 13
    print f;     // implement using syscall
}
```