Arrays

Lecture #15

Charles Averill

Practical Compiler Design
The University of Texas at Dallas

Spring 2023

The Current State of the Compiler

Previously, we added support for variables that are local to function scopes (but not other scopes):

```
(base) charles@nostromo:~/Desktop/ecco$ cat examples/test_factorial
int iterative_fact(int x) {
        int y;
        y = x - 1;

        while (y > 0) {
                x = x * y;
                y = y - 1;
        }

        return x;
}

int recursive_fact(int x) {
        if (x <= 0) {
                return 1;
        }

        return x * recursive_fact(x - 1);
}

int main() {
        print recursive_fact(5);
        print iterative_fact(5);
(base) charles@nostromo:~/Desktop/ecco$ ./scripts run examples/test_factorial
 && clang test_factorial.ll && ./a.out
-----RUN-----
120
120
```

Arrays

This update has been a long time coming. Today, we will add support for statically-allocated arrays, and parentheses in expressions along the way:

```
int main() {
    int a[10];
    int i;
    int x;

    for(i = 0; i < 10; i = i + 1) {
        a[i] = 2 * i;
        x = a[i];

        printint(i);
        printint(x);
    }

    printint(a[8] = 2 * (3 + 5));
    printint(a[8]);
}
```

Arrays in LLVM

- Unlike real assembly, LLVM has a dedicated "array" data type
- Downside: We have to account for this
- Upside: Way more secure (clang will catch when our users try to do weird stuff)

```
; int a[10];
%a = alloca [10 x i32], align 4
; a[i] = 99;
%5 = load i32, i32* %i
%6 = zext i32 %5 to i64
%7 = getelementptr inbounds [10 x i32], [10 x i32]* %a, i64 0, i64 %6
store i32 99, i32* %7
; a[i]
%9 = load i32, i32* %7
```

ACWJ's Array Approach

ACWJ treats arrays as pointers, and vice versa. Therefore, array accesses with bracket notation are essentially just a parsing problem ($array[i] = *(\&array + i)$).

- Nice because we can reuse our existing addressing and dereferencing code
- Nice because it automatically treats arrays like pointers
- Not nice because it isn't conducive to LLVM's array representation

ECCO's Array Approach

We will treat arrays as their own data type, **not** pointers to their root data type. Additionally, we will define a new meta-TokenType for array accesses.

- Nice because it's conducive to LLVM's array representation, therefore we implicitly get the security of LLVM
- Sort of nice that we can reuse *some* of our addressing and dereferencing code, but still requires custom logic
- Not nice because we can't treat arrays as pointers at all unless we extend our implementation later

The Plan

- Update our expression parser to respect parentheses
- Create a new `Array` type in addition to `Number` and `Function`
- Add parsing, generation for array declarations
- Add parsing, generation for array accesses (doubles as parsing for array assignments thanks to our lvalue revitalization)
- Update the `LLVMValue` class to distinguish array and number values stored in virtual registers
- In the meantime, condense LLVMValue representations so we don't have so much code reuse in `llvm.py`
- Add the `long` type, as all array access offsets are `i64`s

Parentheses

This is a super easy update: when we parse a terminal token, if we see a left parenthesis, parse a binary expression, then match a right parenthesis. That's it!

I've also updated the arithmetic tester so that it uses parentheses, as well as the division operator that we left out earlier.

Array Type

The `Array` type will keep track of an array's storage type, array length, contents (currently unused), and dimension (currently unused).

Additionally, `LLVMValue` now takes in an optional `Array` object to handle its LLVM representation.