Finishing Local Variables

Lecture #14

Charles Averill

Practical Compiler Design
The University of Texas at Dallas

Spring 2023

The Current State of the Compiler

We added support for usable function arguments in the last lecture. We had an issue however: assigning to those variables was not functional!

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

        while (y > 0) {
                // This has a bug!
                // x was not operated on before the loop,
                // so our compiler will use the original %x
                // In a future update we will fix this bug
                // by manually allocating stack space for
                // function arguments
                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);
}
(base) charles@nostromo:~/Desktop/ecco$ ./scripts run examples/test_fa
-----RUN-----
120
```

An Overview of the Issue

LLVM gives us function headers with parameters for free. The only issues is that those parameters come as virtual registers, which we know are immutable.

Therefore, this code would end up printing "5", even though an equivalent C program should print "10":

```
int fred(int x) {
    x = x + 5;
    return x;
}

int main() {
    print fred(5);
}
```

The Solution

This is not an unsolvable problem. In fact, it has a solution that we should be fairly comfortable with: when we generate the function preamble, also generate `alloc` and `store` statements to immediately copy the data from those virtual registers into stack pointers:

```
int bob(int x, int y) {
    ...
}
```

```
define dso_local i32 @bob(i32 %0, i32 %1) #0 {
    %x = alloca i32, align 8
    store i32 %0, i32* %x
    %y = alloca i32, align 8
    store i32 %1, i32* %y
    ; ...
}
```

While We're At It

While we're fixing function parameters, it makes sense to add support for non-global variables as well.

We can't yet support scopes yet due to how we've structured the relationship between the parser and generator, but we will at least have variables local to their respective functions.

Fixing Parameters

The meat of the solution here is in `llvm_function_preamble`. Allocating space on the stack is no issue, we've done that before.

However, to store into this new stack space, we have to make our `llvm_store_local` a bit more complicated. We now allow the passing of LLVMValues into this function rather than just SymbolTableEntries. This is going to signify that we want to actually store data into that variable, rather than just updating the latest value that variable contains.

Hopefully you should see how this development leads into local variables as a whole!

Local Variables

The first big change we're making is in the parser for declaration statements: I've added a new TokenType called "VAR_DECL". Remember that previously, global variables would be generated into a separate file because the order of generation was getting complicated.

When our generator encounters VAR_DECL tokens, it's going to call the new `llvm_declare_local` function for us. This allocates stack memory for the variable, allowing us local access to a rewritable data space.

At this point, we've essentially replaced all of our dealings with global variables. This has an interesting side effect: all of our dereference operations now have to be called *twice*. This also fixes the issues with dereference assignment we had earlier.

Bug Fixes

1. `ensure_registers_loaded` actually loads to `load_level` now
2. Function calls load their arguments to the pointer depth expected
3. Comparisons load their operands to the smaller pointer depth of the two