Part 56: Local Arrays

Well, colour me surprised. It wasn't hard to get local arrays implemented at all. It turns out that we had all the pieces in the compiler already, we just had to wire them together.

Local Array Parsing

Let's start on the parsing side. I want to allow local array declaration but only with a number of elements, no assignment of values.

The declaration side is easy, we just add these lines to `array_declaration()` in `decl.c`:

```
// Add this as a known array. We treat the
// array as a pointer to its elements' type
switch (class) {
  ...
  case C_LOCAL:
    sym = addlocl(varname, pointer_to(type), ctype, S_ARRAY, 0);
    break;
  ...
}
```

Now, we must prevent assignment to local arrays:

```
// Array initialisation
if (Token.token == T_ASSIGN) {
```

```
  if (class != C_GLOBAL && class != C_STATIC)
    fatals("Variable can not be initialised", varname);
```

I also added some more error checking:

```
// Set the size of the array and the number of elements
// Only externs can have no elements.
if (class != C_EXTERN && nelems<=0)
  fatals("Array must have non-zero elements", sym->name);
```

And that's it on the declaration side for local arrays.

Code Generation

In `cg.c`, we have a function `newlocaloffset()` that calculates the offset of a local variable relative to the top of the stack frame. Its argument was a primitive type because the compiler only allowed int and pointer types as local variables.

Now that each symbol has its size (which `sizeof()` uses), we can change the code in this function to use the symbol's size:

```
// Create the position of a new local variable.
static int newlocaloffset(int size) {
  // Decrement the offset by a minimum of 4 bytes
  // and allocate on the stack
  localOffset += (size > 4) ? size : 4;
  return (-localOffset);
}
```

And in the code that generates the function's preamble, `cgfuncpreamble()`, we only have to make these changes:

```
// Copy any in-register parameters to the stack, up to six of them
// The remaining parameters are already on the stack
for (parm = sym->member, cnt = 1; parm != NULL; parm = parm->next, cnt++) {
  if (cnt > 6) {
    parm->st_posn = paramOffset;
    paramOffset += 8;
  } else {
    parm->st_posn = newlocaloffset(parm->size);       // Here
    cgstorlocal(paramReg--, parm);
  }
}
```

```
  // For the remainder, if they are a parameter then they are
  // already on the stack. If only a local, make a stack position.
  for (locvar = Loclhead; locvar != NULL; locvar = locvar->next) {
    locvar->st_posn = newlocaloffset(locvar->size);     // Here
  }
```

That's it! It possibly means that we can also allow structs and unions as local variables. I haven't worried about this yet, but it is something to explore later.

Testing the Changes

`test/input140.c` declares:

```
int main() {
  int  i;
  int  ary[5];
  char z;
  ...
```

The array is filled with a FOR loop, `i` being the index. The `z` local is also initialised. This checks to see if any of the variables will tromp over the other variables. It also checks that we can assign all elements of the array and get their values back.

Files `test/input141.c` and `test/input142.c` check that the compiler spots and rejects arrays as parameters and array declarations with no elements.

Conclusion and What's Next

In the next part of our compiler writing journey, I'll return to mopping up duties. Next step