Prof. Jingke Li (FAB120-06, lij@pdx.edu), Tue & Thu 12:00-13:15 @ ASRC 230, Lab: Fri 10:00-11:15/11:30-12:45 @ FAB 170

# Lab 9: Nested and High-Order Functions

(Adapted from Profs. Jones and Tolmach's earlier version)

In this lab, you are going to practice writing programs with nested and high-order functions. Download and unzip the file `lab9.zip`. You'll see a set of programs, `exampleX.c`.

*Note:* These exercises require gcc extensions to C. They are enabled by default on linux lab machines, but may not be available on other plaftforms, such as Macs.

### Exercise 1

Consider the program `example1.c`:

```
int f(int a, int b, int c) {
  c = c + 1;
  return a + b * c;
}
int k(int z) {
  return z+42;
}
int g(int p, int q) {
  return k(p) - f(p,q,q) + f(p,p,q);
}
```

```
int h(int m) {
  int n = (m>0) ? 0 : 42;
  int r = k(m);
  return g(n,r) + g(m,r);
}

int main() {
  int w = 10;
  printf("%d\n",h(w));
}
```

1. Compile and run it.
2. Rewrite it to use nested functions as much as possible, but without changing the parameters to any function. Name this program `nested1.c`. Test this new program by compiling and running it.
3. Now attempt to drop as many parameters as possible from each nested function, but without changing the sequence of calls made. Name this program `dropped1.c`. Test this new program by compiling and running it.

### Exercise 2

Consider the program `example2.c`:

1. Compile it to a `.s` file using

   ```
   linux> gcc -O1 -S example2.c
   ```

2. Walk step-by-step through the behavior of the assembly code, annotating it.
3. Show the contents of the stack and key registers at each point.

```
int f(int x,int y) {
  int h(int u) {
    return y*u;
  }
  int g (int z) {
    return h(x+h(z));
  }
  return g(x+y) + g(0);
}

int main() {
  printf("%d\n",f(1,2));
}
```

**Exercise 3**

*Lexical depth* is a term used to represent a function's nesting level. A function at the top level (*i.e.*, with no enclosing function) has lexical depth 0. A function that appears inside the definition of a function with depth $n$ has depth $n + 1$.

Consider the program `example3.c`:

```
int f(int x,int y) {
  int g (int z) {
    int h(int u) {
      return y*u-z;
    }
    return h(x+z) + h(0);
  }
  return g(x+y) + g(0);
}

int main() {
  printf("%d\n",f(1,2));
}
```

1. Repeat the same steps for it. Annotate the assembly code.
2. Identify the lexical depth of each function.
3. Clearly identify the code that implements a variable reference which spans a lexical depth difference of 2.

**Exercise 4**

1. Fill in the code for these three functions in `example4.ir`. For simplicity, assume that they are always called directly, rather than via closures, so they have no closure argument.

```
(int)int compose(int(int) f, int(int) g) {
  return \x -> f (g x);
}
(int)int h(int x) {
  return compose(\z -> x + z,
                 \x -> x + 1);
}
int top() {
  return (h(42))(0);
}
```

2. Test your code using the IR interpreter:

```
linux> java -jar IRInterp.jar example4.ir
```