1 Introduction

In this lab assignment, we are in the heart of the middle-end. The objective in this phase has two parts, including conversion to *functional* intermediate code (FIR), and optimization. FIR has no `SetVar`, `LetAddrOfVar`, or `LetCopy` instructions. Our version also has the following properties:

- all functions are top-level,
- all type definitions are top-level,
- all string definitions are top-level,
- no `Return` or `LetApply` instructions.

Conversion to FIR has three parts:

- Continuation-passing style (CPS) conversion,
- Escaping variable (ESC) conversion,
- Closure conversion

For full credit on this lab, you must implement CPS and closure conversion in your lab group.

In addition, you may implement the following optimizations for extra-credit. For these optimizations, you must work individually, although you may discuss the problems freely with your group, or anyone who is willing to listen. Each optimization has credit that is a fraction of a full-credit lab. Optimizations are not subject to the lab4 due date; they may be turned in at any time until the final project (lab5) is due. No credit will be given to optimizations turned in after that time. In addition, there will be a performance runoff at the end of lab5. The winning team will receive 20% extra-credit, and second and third place will each receive 10%. Benchmark programs will be included with lab5.

- Deadcode elimination (10%)
- Function inlining and constant folding (10%)
- Common-subexpression elimination (10%)
- Strength reduction (10%)
- Hoisting (20%)
- Alias analysis (20%)

2 Your task

Implement CPS, ESC, and Closure conversion. These stages have been discussed in class. The optimizations will to be discussed in class Feb 12-26. As usual, you can get lab4 code with `cvs update -d`. The base code is in the `fc_fir` directory. The parts you need to implement can be found with `grep NotImplemented *.ml`.

3 Getting started

Please Note!

1. Lab4 does not include a toploop anymore, because the toploop is getting harder to implement (still possible, though).
2. The `-stage` argument specifies how many stages in the compiler to run. You can get the list of stages from `fcc -help`. Stages are inclusive: stage $i+1$ includes stage i; `-stage 1` is useless.

```
<kenai 2993>./fcc -help
usage: compile [options] [files...] [-- <program arguments>]
-stage <num>: numbers are (default is 100):
        1: AST only
        2: IR conversion
        3: CPS conversion
        4: Optimization: LetAtom elimination
        5: ESC conversion
        6: Closure conversion
        7: Optimization: deadcode elimination
        8: Optimization: function inlining
        9: Optimization: commone-subexpression elimination
        10: Optimization: deadcode elimination (again)
  -print_ast print expressions
  -print_ir print intermediate code
  -check_ir typecheck intermediate code
  -eval evaluate intermediate code
  -debug_eval use the debugger during evaluation
  -debug_pos print position information
  -debug_closure print closure equations
  -stage set compiler stages
  -- program arguments
Exit 2
```

4 What to turn in

You should turn in your entire compiler in your `submit/lab4` directory on `mojave` (you should probably keep a copy in your CS account too). IF you are working in a group, only one of you should do the submission, and you should send mail to `cs134-admin` to tell us of your submission.

In addition, you should include the following.

- A `README` file explaining what you did, how it works, and whether you had any problems.
- A `DIFF` file generated using the command `cvs diff` in the `fc_fir` directory. If you like, you can insert this into the `README` file, with brief explanations of what you changed.
- The files `test1.ir`, `test2.ir`, `test3.ir`, and `test4.ir` generated using your compiler with the `-print_ir` option.

```
% ./fcc -check_ir -print_ir -stage 6 test1.c > test1.ir
% ./fcc -check_ir -print_ir -stage 6 test2.c > test2.ir
% ./fcc -check_ir -print_ir -stage 6 test3.c > test3.ir
% ./fcc -check_ir -print_ir -stage 6 test4.c > test4.ir
```

- Program output of the test runs. Use the following commands.

```
% ./fcc -eval -stage 6 test1.c -- test1 10 > test1.out
% ./fcc -eval -stage 6 test2.c -- test2 > test2.out
% ./fcc -eval -stage 6 test3.c -- test3 > test3.out
% ./fcc -eval -stage 6 test4.c -- test4 > test4.out
```