```
ben23480@LAPTOP-KTR9FAIG:~$ opt -version
Ubuntu LLVM version 17.0.6
  Optimized build.
  Default target: x86_64-pc-linux-gnu
  Host CPU: skylake
```

Installed LLVM 17.0.6 on Window WSL Ubuntu

Date: 1/16/2024
Objective: Finish the full project.
Tasks Completed: All tasks required
Challenges Faced: I had an issue getting LLVM working on windows natively. There weren't any challenges looking up the LLVM API for each function and how to use them. However, finding out how to use opt to generate the .dot files was an issue for 30 minutes because seemingly there was an update since the last time this course was offered, and the help given was not useful.
Solutions/Workarounds: Then I gave up after an hour trying windows, installed WSL Ubuntu on Windows, and finished the installation in 10 minutes. I found this link for the working command to obtain the .dot files for example_function(int x) and main().
Code Snippets:
`test.c - Code`

```c
#include <stdio.h>
void example_function(int x) {
    if (x > 0) {
        printf("Positive\n");
    } else if (x < 0) {
        printf("Negative\n");
    } else {
        printf("Zero\n");
    }
}
int main() {
    int a = 5;
    int b = -2;
    if (a > b) {
        example_function(a);
    } else {
        example_function(b);
    }
    return 0;
}
```

Testing: Summarize any testing done if applicable, including results and issues found.
There were some typos for some commands, but overall this lab was very straightforward to complete and clear in the objectives to meet.

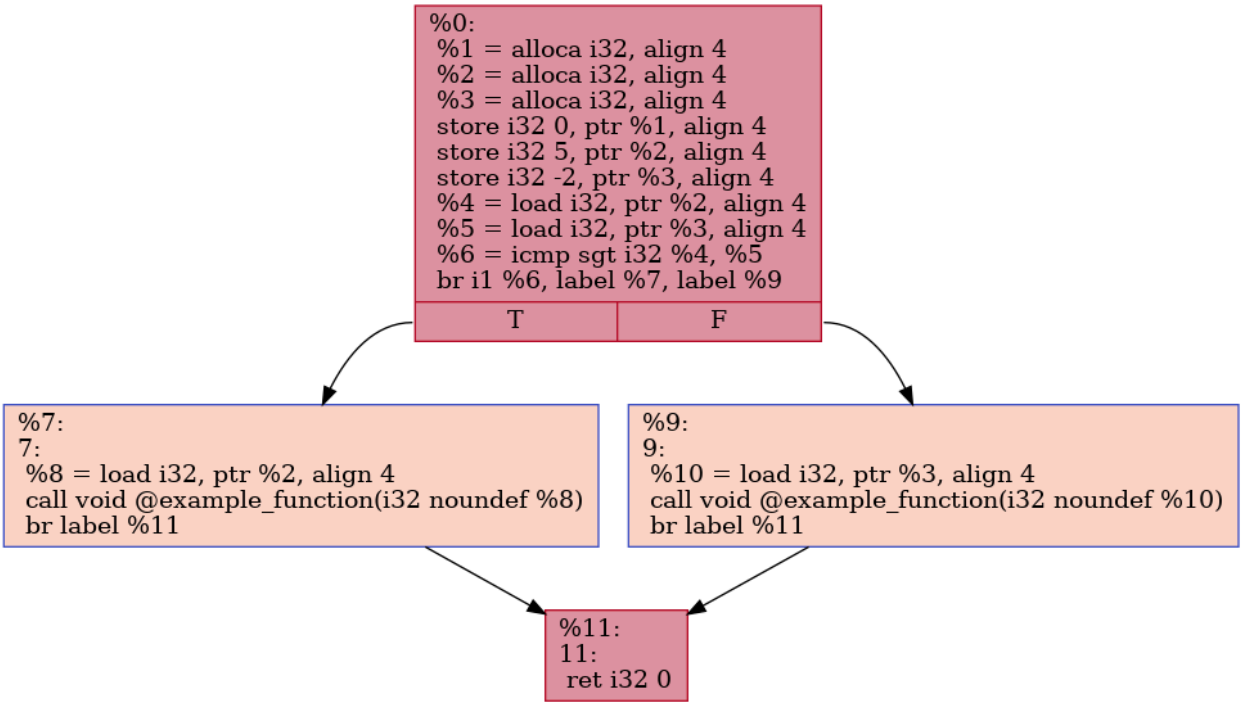
Learnings:

a) Learn the basic uses of the following commands:
   i) `clang - compile c code`
   ii) `opt - applies optimizations to target files`
   iii) `llc - compile LLVM IR file into machine assembly`
   iv) `lli - print output of program out`
   v) `llvm-link - links .bc files together into one .bc file.`
   `ex. llvm-link -o out.bc in1.bc in2.bc`
   vi) `llvm-as - compile LLVM IR file into LLVM bitcode.`
   vii) `llvm-dis - compile LLVM bitcode into LLVM IR file.`

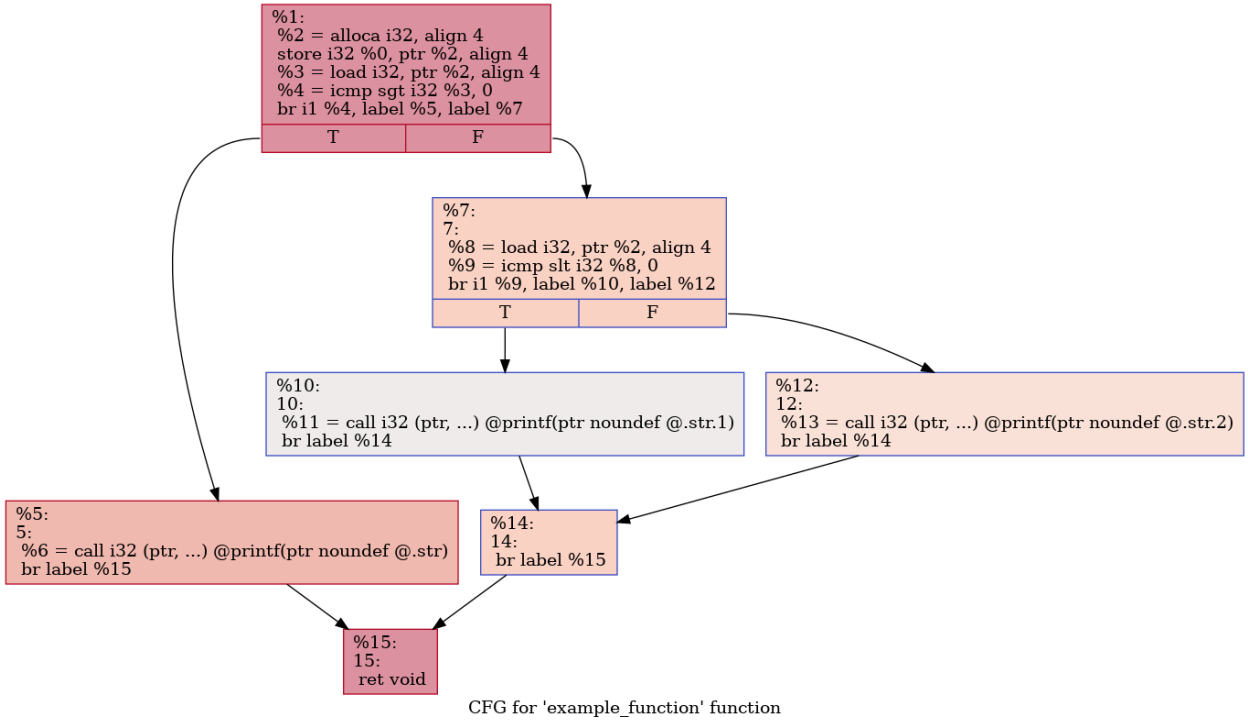
b) Translate between different code formats using the above commands.
   i) source (.c) to binary (executable)
      (1) clang test.c -o test
   ii) source (.c) to objective (.o) - Use -c to produce (.o) file
      (1) clang -c test.c -o test.o
   iii) source (.c) to machine assembly (.s)
      (1) clang test.c -S -o test.s
   iv) source (.c) to LLVM bitcode (.bc)
      (1) clang -c -emit-llvm -o test.bc test.c
   v) source (.c) to LLVM IR (.ll)
      (1) clang -S -emit-llvm -o test.ll test.c
   vi) LLVM IR (.ll) to LLVM bitcode (.bc)
      (1) llvm-as -o test.bc test.ll
   vii) LLVM bitcode (.bc) to LLVM IR (.ll)
      (1) llvm-dis -o test.ll test.bc
   viii) LLVM IR (.ll) to machine assembly (.s)
      (1) llc -o test.s test.ll
   ix) interpret the LLVM IR (which directly prints the output without compilation)
      (1) lli test.ll

c) Generate the CFG(s) of `test.c - .c -> .ll`
   i) `clang -S -emit-llvm -o test.ll test.c`
   ii) `opt -passes="dot-cfg" -debug-pass-manager test.ll`
   iii) `dot -Tpng .main.dot -o main.png`
   iv) `dot -Tpng .example_function.dot -o example_function.png`

`These two CFGs for both functions are shown on the next page.`



CFG for 'main' function



CFG for 'example_function' function

Next Steps: The project is finished.