Instructions: Quiz #1, on 08/26, will consist of questions taken or inspired from Part I of this homework ("Short Questions") and will also serve to assess your understanding of Part II ("Longer Problems").

1 Short Questions

Disclaimer: some of this first batch of questions was strongly inspired by Compilers: Principles, Techniques, and Tools *by A. V. Aho, R. Sethi & J. D. Ullman.*

1. What is the difference between a compiler and an interpreter?
2. What are the advantages of (a) a compiler over an interpreter (b) an interpreter over a compiler?
3. What is a *source-to-source* translator?
4. Indicate which of the following "imperative", "functional", "object-oriented", "interpreted", "compiled", "statically typed", "open-source", "multi-paradigm", "with manual memory management", "procedural", applies to the following languages: Cobol, C#, Lisp, C, Perl, Fortran, Python, Java, Rust, Go. Make sure you understand the definition of those terms, and indicate at least one thing the language is good at, and one thing the language is bad at.
5. Define what a "cross compiler" is.
6. Define what a "native compiler" is.
7. Briefly explain what a disassembler is.
8. Consider each of the following (hypothetical) translators. Do you think the translator might be useful in practice? Explain your answer, and give a possible usage example. Also, what difficulties could be anticipated in making the translator generate good-quality object code? (a) a Java → C translator; (b) a C → Java translator; (c) a machine code → C decompiler.
9. For the block-structured C code below, indicate the values assigned to w, x, y, and z.

```
int w, x, y, z;
int i = 4; int j = 5;
{
    int j = 7;
    i = 6;
    w = i + j;
}
x = i + j;
{
    int i = 8;
    y = i + j;
}
z = i + j;
```

Use `printf("%d",w );` and similar statements to check your answers once you have installed a C compiler.

---

2 Longer Problems

Problem 1 Read the "Three-stage compiler structure" Section of the Wikipedia page on Compiler.

Problem 2

1. You will need to access a (mostly) POSIX-compliant operating system for this class. You can install a virtual machine manager and debian in it, use Cygwin or the Windows Subsystem for Linux on your windows machine, or simply your macOS (possibly improved with brew) or linux machine.

2. To test your set-up, install gcc and try to compile and execute the program presented in my *Very Short Intro to C*.
3. Using `gcc -v`, identify the target of your compiler.
4. Save the following C program:

```
#include <stdio.h>
    int main() {
        printf("Hello, World!");
        return 0;
    }
```

   into a file, compile and execute it.
5. Execute `xxd -b a.out` on the file you obtained at the previous step. Try to make a guess on xxd's role.
6. Type `man xxd` and read the description to confirm or infirm your guess.

☛ **Problem 1** Read the (very short) paper *Formally Verifying a Compiler: What Does It Mean, Exactly?* and the report on this talk (starting at the end of p. 1), and answer the following questions:

- Is there, to this day, any other verified compiler than CompCert?
- Look for definitions of "choice refinement" and "behavior refinement", and rephrase them in your own terms.
- Explains what it means to write that "Coq is used both as a proof assistant and as a programming language".
- What is a "labelled transition systems"?

☛ **Problem 2** Read the second Section of the (drafty) paper *Of What Use is a Verified Compiler Specification?*, "*Why Compiler Verification is Important?*", as well as the Second Figure (p. 7) and answer the following questions:

- What is a "mnemonic assembly language"?
- Why is code generated by optimising compilers harder to verify?
- What could be the benefits of not needing a front-end?
- According to this paper, a "Semantics" is a function between what and what?