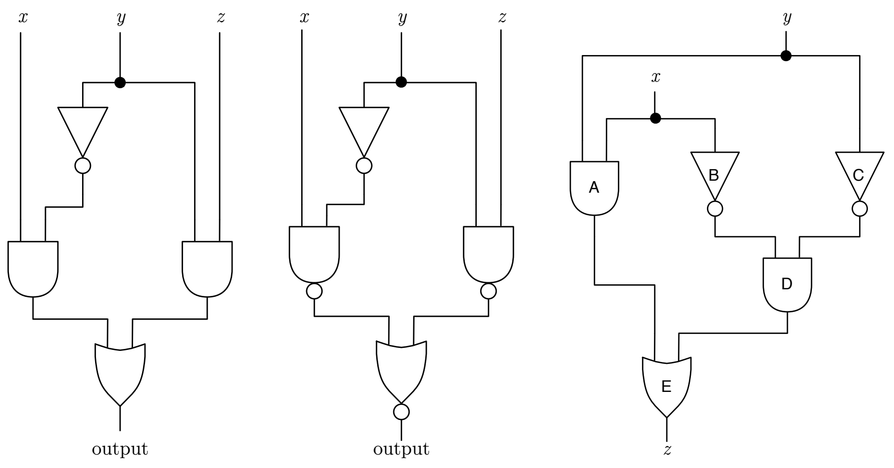
CSC173: Project 5
Boolean Logic and Boolean Circuits

We're going to keep it simple for this last, end-of-term project.

We have provided code for a simple Boolean circuit simulator written in C. This includes a small test program that demonstrates the use of the simulator API to setup a circuit and test it on some input(s). Note that this code may be different from previous terms.

Your assignment is to create and test the following circuits using the simulator framework:



(a) (b) (c)

For each circuit, your program must have an appropriately-named function that creates and returns the circuit.

Note that the simulator we provide does not implement NOR and NAND gates. You are welcome to extend our code and implement the other gates, or use the relevant equivalences (algebraic identities) to transform the circuit into an equivalent one. Explain how you handled this in your README.

Your program must also have a *single* function that takes a circuit and tests it on *all possible combinations* of its inputs. Our sample code runs a few test cases. Your program must generalize this into a clean, well-designed, well-implemented function. Here's a hint: cut-and-paste bad.

Your main function should create each of the circuits and test them on all possible combinations of their inputs, printing informative messages. You might find it helpful to figure out what the right answers are so that you can test your program automatically. . .

Extra Credit (max 20% total extra)

Also implement the one-bit adder circuit (FOCS Fig. 13.10) and test it on all combinations of inputs [up to 10%].

What would really make the project great would be to automate the synthesis of circuits from Boolean expressions. To do this, you would need a parser that reads a Boolean expression from the input and produces an expression tree (parse tree) representation. You then walk the expression tree bottom-up and create the circuit corresponding to the expression. A recursive descent parser for fully-parenthesized, prefix-operator Boolean expressions is straightforward. Doing a post-order tree traversal is easy. The main challenge is the bookeeping necessary to keep track of inputs, outputs, and gates as you create them using the simulator API. Still, if you want a challenge, go for it. It really pulls together several aspects of the course and you'll learn a ton. [up to 20%]

Project Submission

Your project submission MUST include the following:

1. A `README.txt` file or `README.pdf` document describing:
   (a) Any collaborators (see below)
   (b) How to build your project
   (c) How to run your project's program(s) to demonstrate that it/they meet the requirements
2. All source code for your project. Eclipse projects must include the project settings from the project folder (`.project`, `.cproject`, and `.settings`). Non-Eclipse projects must include a `Makefile` or shell script that will build the program per your instructions, or at least have those instructions in your README.txt.

We must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better grade your will be.** It is your job to make both the building and the running of programs easy and informative for your users.

Programming Policies

You must write your programs using the "C99" dialect of C. This means using the "`-std=c99`" option with `gcc` or `clang`. For more information, see Wikipedia.

You must also use the options "`-Wall -Werror`". These cause the compiler to report all warnings, and to make any warnings into errors that prevent your program from compiling. You should be able to write code without warnings in this course.

With these settings, your program should compile and run consistently on any platform. We will deal with any platform-specific discrepancies as they arise.

If you submit an Eclipse project, it must have these settings associated with the project. Projects with that compile with warnings will be considered incomplete.

Furthermore, your program should pass `valgrind` with no error messages. If you don't know what this means or why it is A Good Thing, look at the C for Java Programmers document which has a short section about it. Programs that do not receive a clean report from `valgrind` have problems that **should be fixed** whether or not they run properly. If you are developing on Windows, you will need to look for alternative memory-checking tools.

Late Policy

Late projects will **not** be accepted. Submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

Collaboration Policy

You will learn the most if you do the projects YOURSELF.

That said, collaboration on projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC173.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit on the group's behalf and the grade will be shared with other members of the group. Other group members should submit a short comment naming the other collaborators.
- All members of a collaborative group will get the same grade on the project.

Academic Honesty

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.