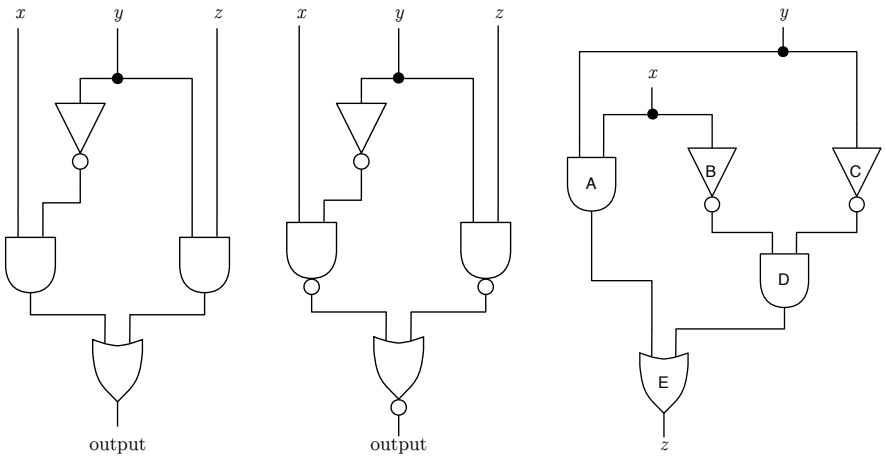
CSC173: Project 5
Boolean Logic and Boolean Circuits

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).

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-design, 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 PDF document describing:
   (a) How to build your project
   (b) How to run your project's program(s) to demonstrate that it/they meet the requirements
   (c) Any collaborators (see below)
   (d) Acknowledgements for anything you did not code yourself (you should avoid this, other than the code we've given you, which you don't have to acknowledge)
2. All source code for your project, including a `Makefile` or shell script that will build the program per your instructions. Do not just submit Eclipse projects without this additional information.

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.**

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, check 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.

Furthermore, your program must pass `valgrind` with no error messages. Of course this might differ for different executions, but it should be clean for any fixed examples requested in the project description. Any questions, ask the TAs *early*.

Projects in this course will be evaluated using `gcc` on Fedora Linux (recent version, like 25 or 26). Here are several ways you can setup this environment for yourself:

- Visit `getfedora.org` and download and install Fedora on a spare computer or separate partition of your main computer.
- Don't have a spare computer? No problem. Visit `virtualbox.org` and download VirtualBox. Then setup a new virtual machine running Fedora Linux (you may need to download the installer as above). VMWare is a commercial virtualization app if you want to buy something.
- Not confident setting up a VM or don't have space? No problem. Visit `docker.com` and download and install Docker for your computer. You can easily setup an image with Fedora Linux and `gcc`, or see the appendix below for more.
- Don't want to install anything? No problem. You need an account on the Computer Science Department's undergraduate cycle servers. Ask the TAs about it. Do not wait until the last minute. This will not happen overnight.

Mac and Windows users especially beware: You must test your code under Fedora Linux, or expect problems (and low grades).

Late Policy

Don't be late. But if you are: 5% penalty for the first hour or part thereof, 10% penalty per hour or part thereof after the first.

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.

Appendix: Using Docker

"Docker provides operating-system-level virtualization [...] to allow independent "containers" to run within a single Linux instance, avoiding the overhead of starting and maintaining virtual machines." (Wikipedia)

Getting Started with Docker

I have built a docker image based on Fedora linux and containing `gcc`, `make`, `gdb`, `valgrind`, and `emacs`, as well as manpages. You can download it from here:

`http://www.cs.rochester.edu/u/ferguson/csc/docker/`

Once you have installed Docker, load the image to create the `fedora-gcc` image in your Docker installation:

```
docker load -i fedora-gcc-docker.tgz
```

Then you can do, for example:

```
docker run -it --rm -v "`pwd`":/home -w /home fedora-gcc bash
```

This runs `bash` (the command shell) using the `fedora-gcc` image. The `-it` option means to run interactively; `--rm` tells Docker to remove the container when the process exits; `-v` and the bit with the colon says to make your current working directory (outside docker) available on `/home` within docker; and `-w` tells docker to make `/home` the initial working directory for the `bash` process. Whew!

You can do a lot with Docker. You'll have to read the docs but it's a great option. Please report any problems with the image ASAP.