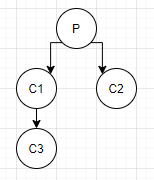
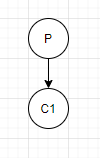
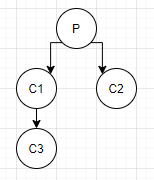
1. The program sets up two variables, id and ret, and then assigns the fork() to ret splitting it up into a child process and a parent process. After that, the id gets assigned the id of either the child, or the parent, depending on which one happens to go first, and then the system prints out the id. After the first print is handled, it then hits a wait, which makes that process fork stop and allows the other one to run. It does the same thing, prints out the id of the current fork, and then waits. The program never ends because neither fork ever terminates, they just sit there waiting.
2. The program will follow the same initial process again with setting up two variables id and ret. After that it will fork once, and then both the parent and the child will fork again. At this point, there will be 4 lines printed out with 4 different ids, and again the process wont end because there is only a wait statement.



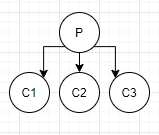
1. The program sets up a ret variable, and then forks. Depending on which forked path goes first, two lines will be printed out. If it’s the parent first then the else will be ran and the output will have the “Hello from parent”, and on the next line it will have “Hello from child”. If the child goes first, then the lines will be flipped.



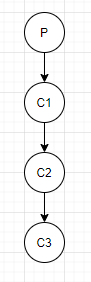
1. The program will first print out an id followed by L0. On the next line it will print out L1 twice, with an id before each and both will be on their own lines. Finally, Bye will be printed out 4 times, each time on its own line and with a unique id. There will be 3 lines with the id of the first parent and lines of L0 L1 and Bye; 2 lines with the id of the first child and lines L1 and Bye, and 1 id for each of the two new children made in the final forks and the line Bye for each.



1. The program will print out L0 with an id once. Next it checks to see if the parent process is currently running, and if it isn’t then the child process prints out its id and Bye. If the parent is the current process, then it prints its id with L1 and forks again. The same check is done and the child prints Bye and the parent forks again, printing L2. Finally, the Parent prints out Bye.

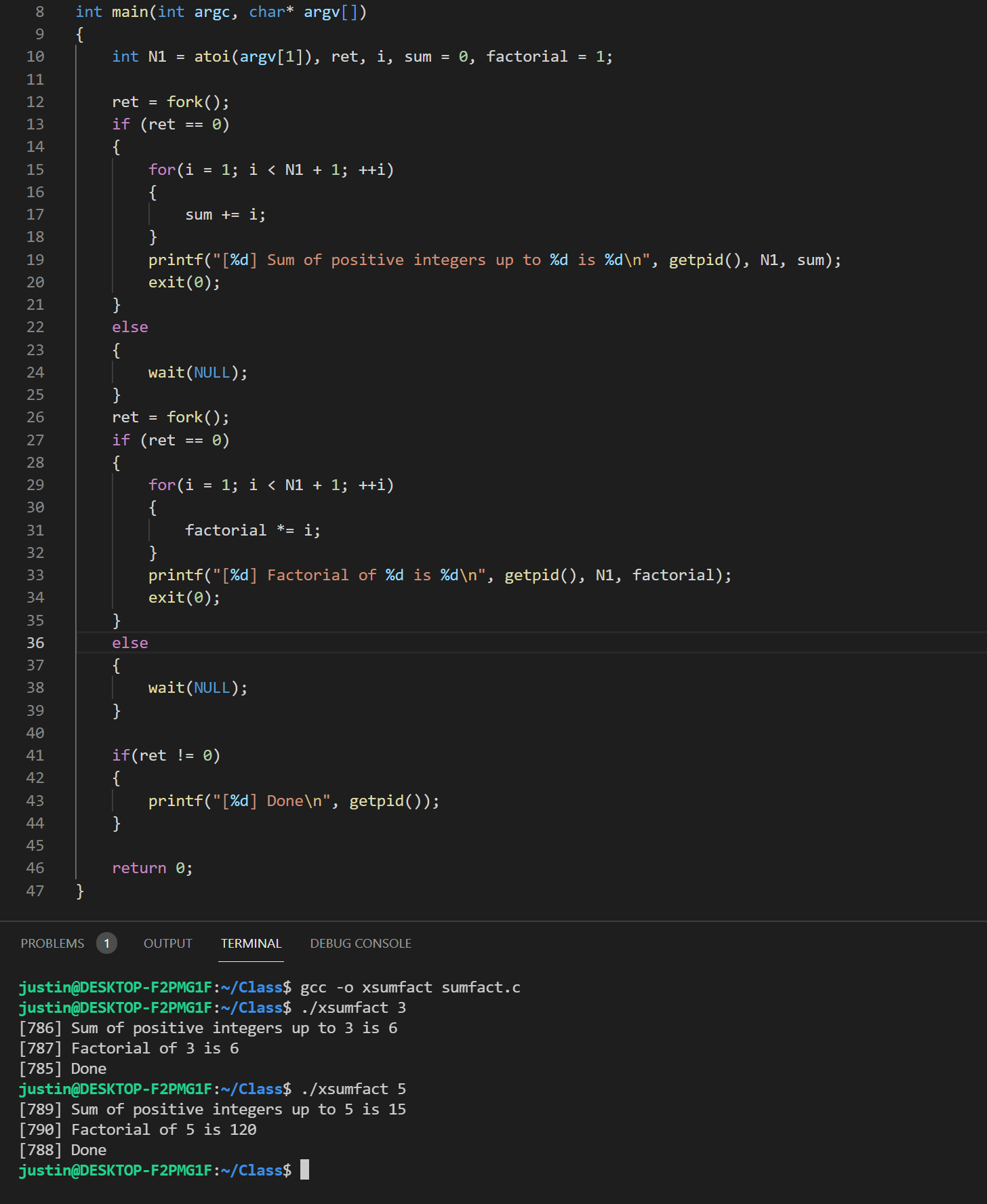


1. This program is the inverse of 5. The first line that is printed is an id with L0. Then when the child process is the active one, it prints out L1 and forks again. The parent process only prints out bye. Then in that new fork, the child prints out L1 and the parent prints bye again. This same process happens again with L2 and bye. So instead of the parent forking and printing L0, L1, L2, and Bye, the child of the newest fork is the one that prints out the L statement and forks again, and the parents always print Bye.



1. This program will check to see if the parent or the child fork is running, and if it is the parent then it will print out the id and “Waiting Parent”, and then it will wait for the child to run, and then print out “Ending Parent” once the child is done. The Child process would print out “Running Child” and then “Ending Child” after the sleep(2) line is finished making the program sleep.



1. 
2. 