CSC209H Worksheet: Pipes (workers & master)

In the last worksheet we wrote a program that forked one child for each commandline argument. The child computed the length of the commandline argument and exits with that integer as the exit status. The parent sums these return codes and reports the total length of all the commandline arguments together. For this worksheet, we will do the same program, except that each child will communicate the string's length to the parent through a pipe.

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <stdlib.h>
#include <string.h>


int main(int argc, char **argv) {
    // Declare any variables you need
    int pipe_fd[argc][2]; // we will need one pipe for each child process

    // write the code to loop over the command line arguments (remember to skip the executable name)
    for (int i = 1; i < argc; i++) {
        // call pipe before we fork
        if ((pipe(pipe_fd[i])) == -1) {
            perror("pipe");
            exit(1);
        }
        // call fork
        int result = fork();
        if (result < 0) {   // case: a system call error
            // handle the error
            perror("fork");
            exit(1);
        } else if (result == 0) {  // case: a child process
            // child does their work here
            // child only writes to the pipe so close reading end
            if (close(pipe_fd[i][0]) == -1) {
                perror("close reading end from inside child");
                exit(1);
            }
            // before we forked the parent had open the reading ends to
            // all previously forked children -- so close those
            int child_no;
            for (child_no = 1; child_no < i; child_no++) {
                if (close(pipe_fd[child_no][0]) == -1) {
                    perror("close reading ends of previously forked children");
                    exit(1);
                }
            }
            int len = strlen(argv[i]);
            // write len to the pipe as an integer
            if (write(pipe_fd[i][1], &len, sizeof(int)) != sizeof(int)) {
                perror("write from child to pipe");
                exit(1);
            }
            // I'm done with the pipe so close it
            if (close(pipe_fd[i][1]) == -1) {
                perror("close pipe after writing");
                exit(1);
            }
            // exit so I don't fork my own children on next loop iteration
            exit(0);
```

```
        } else {
            // in the parent but before doing the next loop iteration
            // close the end of the pipe that I don't want open
            if (close(pipe_fd[i][1]) == -1) {
                perror("close writing end of pipe in parent");
                exit(1);
            }
        }
    }

    // Only the parent gets here

    int sum = 0;
    // read one integer from each child
    // for now print it out as well as adding it in
    int contribution;
    for (int i = 1; i < argc; i++) {
        if (read(pipe_fd[i][0], &contribution, sizeof(int)) == -1) {
            perror("reading from pipe from a child");
            exit(1);
        }
        printf("I just read a %d from child %d\n", contribution, i);
        sum += contribution;
    }

    printf("The length of all the args is %d\n", sum);
    return 0;
}
```