Your assignment is to do problem 3.21 from the textbook (copied in the last page of this document) in C, **with the following changes**.

- Your program (which is the parent process) will fork **two** processes to print their respective sequence for the Collatz conjecture.
- The first child process will produce the sequence which is indicated by the number on the command line and the second child process will produce the sequence from the command line number plus 3.
- The number entered on the command line will be an integer between 1 and 35.
- The variable n will hold the number entered in the command line (see the solution to 3.21 posted in module 7). You **must** increment this variable by 3 (n = n + 3;) **in the parent process** – *not in a child process*. You should **not** create another variable to hold this new value (such as in: m = n + 3;).
- The main program (parent process) should print out the value of the argument passed to each child.
- The children should print the child's "name" (1 or 2), with each number output, - see sample output.
- Each process should also print its own process ID (use getpid()) twice: as soon as it starts and just before it ends execution (see sample output).
- Make sure your program allows the forked processes to run concurrently. If you do the extra credit (below), you will likely see concurrency illustrated in your ouput.
- You will need to call `wait` twice so that the main program finishes after the children (no cascading termination). The child processes will not always finish in the order in which they are forked.

Be extremely careful that a child process does not itself fork a process or you can fill the process table and lock up the machine. Testing of this work must only be done on your own Linux machine. If you lock up another machine (such as circe or a lab machine) trying this assignment out, it is a 0 for this assignment! Make sure you know how to terminate a program that is stuck in an infinite loop: ctrl+c is one way.

You must upload your source code on Canvas. Name your source code file a4.c, and make sure it compiles using gcc a4.c on your linux machine. To test your code, we will use gcc a4.c to compile and ./a.out ## to run, where ## is a number between 1 and 35.

Extra Credit (5 points)

Make the above program work with any number of child processes.

Allow the user to enter the number of child processes (between 2 and 100). You may get this number as a command line argument or prompt the user as soon as your program starts. Before forking each child process (except the first), increment n by 3. Each child will print the Collatz sequence starting at the new n, for example: child 1 starts the sequence from n, child 2 starts from (n + 3), child 3 starts from (n + 6), child 4 starts from (n + 9)….

Sample Output (with 2 children)

```
oscreader@OSC:~/test$ gcc a4.c
oscreader@OSC:~/test$ ./a.out 20

Main program's process ID: 1921

Parent says: About to fork child, starting value = 20

Parent says: About to fork child, starting value = 23

Child 2 (ID: 1923) Start sequence at: 23
(Child 2) 70
(Child 2) 35
(Child 2) 106
(Child 2) 53
(Child 2) 160
(Child 2) 80
(Child 2) 40
(Child 2) 20
(Child 2) 10
(Child 2) 5
(Child 2) 16
(Child 2) 8
(Child 2) 4
(Child 2) 2
(Child 2) 1

About to end execution (I'm process 1923).

Parent says: Done waiting for a child.

Child 1 (ID: 1922) Start sequence at: 20
(Child 1) 10
(Child 1) 5
(Child 1) 16
(Child 1) 8
(Child 1) 4
(Child 1) 2
(Child 1) 1

About to end execution (I'm process 1922).

Parent says: Done waiting for a child.

About to end execution (I'm process 1921).

oscreader@OSC:~/test$
```

3.21 The Collatz conjecture concerns what happens when we take any positive integer n and apply the following algorithm:

$$n = \begin{cases} n/2, & \text{if n is even} \\ 3 \times n + 1, & \text{if n is odd} \end{cases}$$

The conjecture states that when this algorithm is continually applied, all positive integers will eventually reach 1. For example, if $n = 35$, the sequence is

35, 106, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1

Write a C program using the `fork()` system call that generates this sequence in the child process. The starting number will be provided from the command line. For example, if 8 is passed as a parameter on the command line, the child process will output 8, 4, 2, 1. Because the parent and child processes have their own copies of the data, it will be necessary for the child to output the sequence. Have the parent invoke the `wait()` call to wait for the child process to complete before exiting the program. Perform necessary error checking to ensure that a positive integer is passed on the command line.