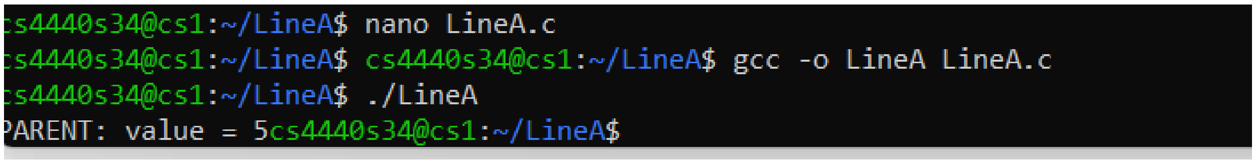
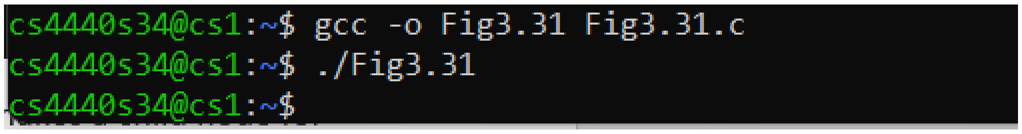
Question 1) 3.1 Using the program shown in Figure 3.30, explain the output at LINE A.



```
:s4440s34@cs1:~/LineA$ nano LineA.c
:s4440s34@cs1:~/LineA$ cs4440s34@cs1:~/LineA$ gcc -o LineA LineA.c
:s4440s34@cs1:~/LineA$ ./LineA
PARENT: value = 5cs4440s34@cs1:~/LineA$
```

The parent output will be 5 due to the OS finishing the parent process first in the term modifications instead of the child one. The wait() gave the time for the child process to finalize the operating system and found the parent process to be more quickly modified than the child process.

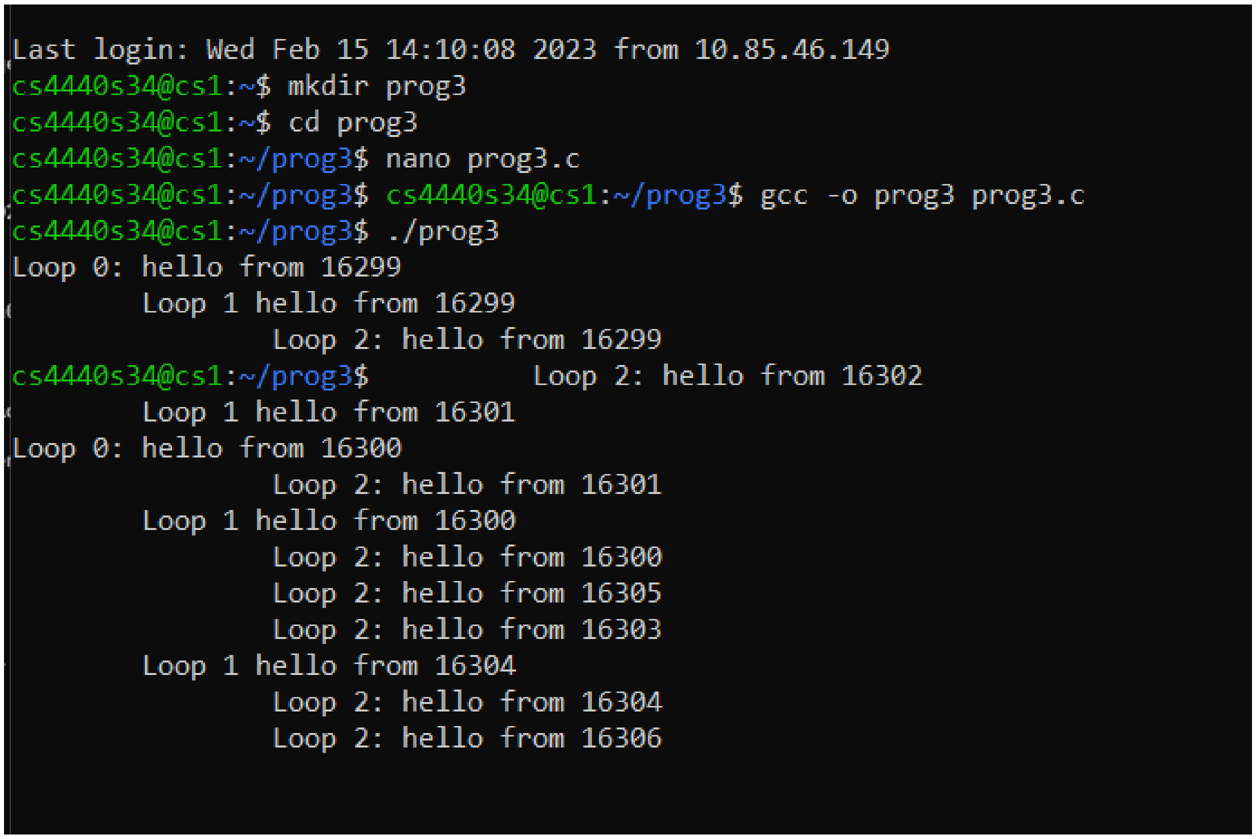
Question 2) 3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?



```
cs4440s34@cs1:~$ gcc -o Fig3.31 Fig3.31.c
cs4440s34@cs1:~$ ./Fig3.31
cs4440s34@cs1:~$
```

There are 8 processes from this, first fork operation creates 2 child processes. It first makes a child node then one for the parent. Each process executes the next fork operation call, thus another child process for each of them, resulting in 4 processes being made. Lastly, all 4 processes execute a third fork operator, which will make another child process for each one of those eventually having 8 total processes.

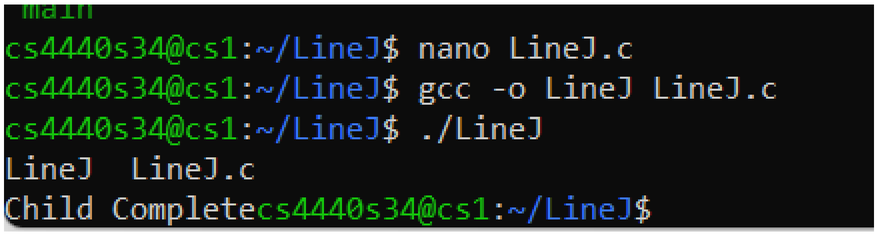
Question 3) 3.11 Including the initial parent process, how many processes are created by the program shown: Fig 3.32 (Fig 3.21)



```
Last login: Wed Feb 15 14:10:08 2023 from 10.85.46.149
cs4440s34@cs1:~$ mkdir prog3
cs4440s34@cs1:~$ cd prog3
cs4440s34@cs1:~/prog3$ nano prog3.c
cs4440s34@cs1:~/prog3$ cs4440s34@cs1:~/prog3$ gcc -o prog3 prog3.c
cs4440s34@cs1:~/prog3$ ./prog3
Loop 0: hello from 16299
        Loop 1 hello from 16299
                Loop 2: hello from 16299
cs4440s34@cs1:~/prog3$          Loop 2: hello from 16302
        Loop 1 hello from 16301
Loop 0: hello from 16300
                Loop 2: hello from 16301
        Loop 1 hello from 16300
                Loop 2: hello from 16300
                Loop 2: hello from 16305
                Loop 2: hello from 16303
        Loop 1 hello from 16304
                Loop 2: hello from 16304
                Loop 2: hello from 16306
```

This program creates 8 processes including the original parent process. The first loop creates one loop child which makes it 2 processes the parent and the child. The second loop creates 2 child processes which in turn makes 4 processes, the parent and 3 children. The last loop makes creates 4 child processes, which at the end of the program creates a total of 8 processes, one parent and 7 child processes.

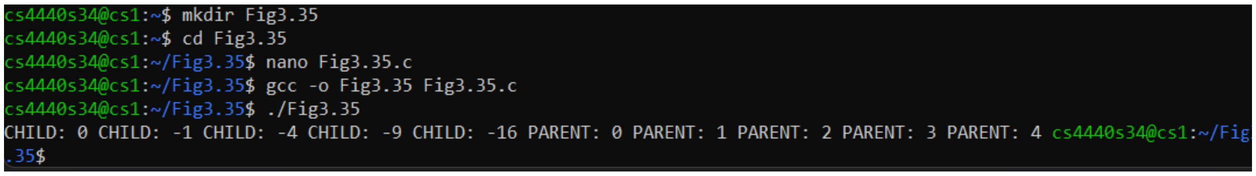
Question 4) 3.12 Explain the circumstances under which the line of code marked printf("LINE J") in Figure 3.33 (Fig 3.22) will be reached.



```
 main
cs4440s34@cs1:~/LineJ$ nano LineJ.c
cs4440s34@cs1:~/LineJ$ gcc -o LineJ LineJ.c
cs4440s34@cs1:~/LineJ$ ./LineJ
LineJ  LineJ.c
Child Completecs4440s34@cs1:~/LineJ$
```

Only if the execlp() function fails to execute the /bin/ls command in the child process will the line of code tagged printf("LINE J") be run. By invoking an executable program, the execlp() function replaces the current process with a new process image. The execlp() function in this program is used to run the ls command, which displays a list of the contents of the current directory. The printf("LINE J") line of code will be run if the execlp() method returns an error after failing to execute the ls command.

Question 5)3.16 Using the program shown in Figure 3.35 (Fig 3.24), explain the output at lines X and Y.



```
cs4440s34@cs1:~$ mkdir Fig3.35
cs4440s34@cs1:~$ cd Fig3.35
cs4440s34@cs1:~/Fig3.35$ nano Fig3.35.c
cs4440s34@cs1:~/Fig3.35$ gcc -o Fig3.35 Fig3.35.c
cs4440s34@cs1:~/Fig3.35$ ./Fig3.35
CHILD: 0 CHILD: -1 CHILD: -4 CHILD: -9 CHILD: -16 PARENT: 0 PARENT: 1 PARENT: 2 PARENT: 3 PARENT: 4 cs4440s34@cs1:~/Fig
.35$
```

We anticipate seeing the output of the array's updated values produced by the child process at line X. As a result, the array causes us to view the CHILD values five times first. We anticipate seeing the output of the updated array values displayed by the parent process at line Y. This is so because the parent process, which ran first and returned the identical values as the first array, departed. ('int nums[SIZE] = 0, 1, 2, 3, 4').