Reference C++ Code A

```
if (fork() == 0) {
    // do stuff
} else if (fork() == 0) {
    // do more stuff..
} else if (fork() == 0) {
    // do more "stuff"..
} else if (fork() == 0) {
    // noooo
}
```

Reference C++ Code B

```
if (fork() == 0) {
    if (fork() == 0) {
        if (fork() == 0) {
            // if ...
            // aaaahhhh
        }
    }
}
```

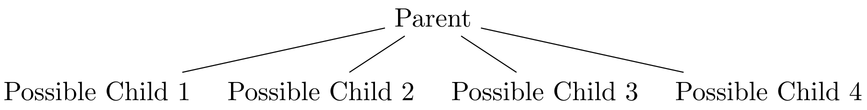
Reference C++ Code C

```
int main(int argc, char* argv[]) {
    // just use execv to run ANY program
    if (execv("/usr/bin/gedit", argv) == -1) {
        // error
        cout << "Error. Booooo!" << endl;
    }
    cout << "Will this line still be printed?" << endl;
}
```

These code snippets were obtained from the Lab 6 assignment page.

1. What should be the resulting process *FAMILY TREES* from these two (Code A and Code B) code snippets? Illustrate.

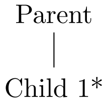
   In reference code A all `fork()` calls performed at the same ”hierarchy level” in the if-else tree. The entry point starts at `if (fork() == 0)`, which produces the first child process. However, should this `fork()` fail, the succeeding `fork()` calls in the following `else if (fork() == 0)` statement produces a child process accordingly. Once a `fork()` successfully produces a child, the program enters the respective instruction block and stops forking. **The resulting process family tree from code A looks like:**



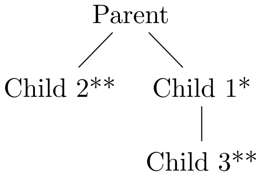
   In this diagram, Possible Child n for $n \geq 2$, only gets instantiated if the first $n - 1$ fork(s) fail.

   In reference code B `fork()` calls are performed sequentially, with further calls only being called if the previous one succeeds. If the first `fork()` succeeds, then the second one is called, then if this succeeds the third one is called; so on and so forth. At the first `fork()` the parent produces a child, in the second one both parent and child produce a child, at the third call the parent, child, and the *grandchild* all produce a child—effectively at each call the amount of total processes *doubles*, demonstrating exponential growth. **The resulting process family tree from code B, can be illustrated per stage (up to the third fork):**

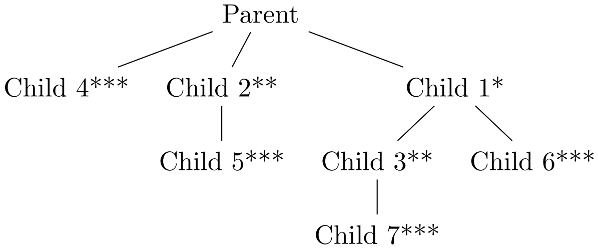
1st Fork



2nd Fork



3rd Fork



*In this diagram each * denotes the nth fork at which the child was instantiated, where n represents the amount of asterisks trailing a child label.*

Effectively at the third fork there will be a total of **8** processes, since at each fork in this structure the amount of processes double, we can express this as 2^n where n is the amount of times the program calls `fork()`.

2. Will the last line in the sample code below still be printed? How about when using `execl()`? Why or why not? Explain.