

```

graph TD
    main["main  
i=1  
j=2"] --> fork1["fork"]
    fork1 --> p1_1["printf  
i"]
    p1_1 --> p1_2["printf  
j"]
    p1_2 --> p1_3["printf  
i"]
    p1_3 --> fork2["fork"]
    fork2 --> p2_1["printf  
i"]
    p2_1 --> p2_2["printf  
j"]
    p2_2 --> p2_3["printf  
i"]
    p2_3 --> r1["return"]
    fork1 --> p3_1["printf  
i"]
    p3_1 --> p3_2["printf  
j"]
    p3_2 --> p3_3["printf  
i"]
    p3_3 --> fork3["fork"]
    fork3 --> p4_1["printf  
i"]
    p4_1 --> p4_2["printf  
j"]
    p4_2 --> p4_3["printf  
i"]
    p4_3 --> r2["return"]
    fork3 --> p5_1["printf  
i"]
    p5_1 --> p5_2["printf  
j"]
    p5_2 --> p5_3["printf  
i"]
    p5_3 --> r3["return"]
    fork2 --> p6_1["printf  
i"]
    p6_1 --> p6_2["printf  
j"]
    p6_2 --> p6_3["printf  
i"]
    p6_3 --> r4["return"]
  
```

The graph illustrates the execution flow of the provided C code. It starts at the 'main' function with initial values $i=1$ and $j=2$. The code then enters a loop structure with nested 'fork' and 'return' statements. The flow is as follows:

- main** ($i=1, j=2$) branches to a **fork** node.
- The top **fork** node leads to a sequence of three **printf** statements (printing i, j, i) followed by a **fork** node.
- The bottom **fork** node leads to a sequence of three **printf** statements (printing i, j, i) followed by a **fork** node.
- Each of these four **fork** nodes leads to another sequence of three **printf** statements (printing i, j, i) followed by a **return** statement.

The graph shows the recursive calls and returns, with the final output being the result of the recursive calls.

| | | | |
|-------|-------|-------|-------|
| A & C | A & D | B & C | C & D |
|-------|-------|-------|-------|

No. For the **Non-terminating Child Example**, the **Trut** process cannot terminate that child process, so the child process will always remain in the loop and stays in the system.

← will never terminate.