**Question 1: Breath First**

For every, node visited, push all of its children to the tail of a queue.

That way, nodes at the same level as the current node will be visited first Breath-First Search.

Order: State of queue after visit node

**A**: [B, F] (appended children of **A**: B, F)

**B**: [F, C, D, E] (appended children of **B**: C, D, E)

**F**: [C, D, E, G] (appended children of **F**: G)

**C**: [D, E, G] (**C** does not have children)

**D**: [E, G] (**D** no children)

**E**: [G] (**E** no children)

**G**: [H, J, K] (appended children of **G**: H, J, K)

**H**: [J, K] (**H** no children)

**J**: [K] (**J** no children)

**K**: [] (**K** no children)

Queue empty Done!

**Question 2: Depth First**

Have a stack. For each node until the stack is empty: If the node has a sibling, add the sibling on top of the stack.

Then move on to the next child.

Stack [Top, Top-1, Top-2…]

Order:

Before A: Stack [A]

**A**: A has no sibling: Stack[] Set r = A.firstChild = B

**B**: Since r null, Add next sibling of **B** to Stack: [F] set r = **B**.firstChild = C

**C**: Since r null, add next sibling of **C** to stack: [*D*, F] set r = C.firstChild = null.

**D**: Since r = null, set r = [D,F].pop = D, stack: [F]. Add next sibling of D (E) to stack: [*E*, F] set r = D.firstChild = null

E: Since r = null, set r = [E,F].pop = E, stack: [F]. Since next sibling of E = null, do not add set r = E.firstChild = null.

**F**: Since r = null, set r = [F].pop = F, stack: []. Since next sibling of F = null, do not add set r = F,firstChild = G.

**G**: Since r null, G does not have sibling set r = G.firstChild = H. Stack: []

**H**: Since r null, add sibling of H to stack: [H] set r = H.firstChild = null

**J**: Since r = null, pop from [H]. Add sibling of H to stack: [J]. Visit J set r = J.firstChild = null

**K**: Since r = null, pop from [J]. Add sibling of J to stack: [K]. Visit K set r = K.firstChild = null.

After the inner loop, the stack is still empty Terminate.

**Question 3: Cat Pedigree Exercise**

Consider at least 4 cases:

* **Valid Cases**, not including valid boundary cases
* **Valid Boundary Cases**
* **Invalid Cases**, not including invalid boundary cases
* **Invalid Boundary Cases**

Tests for Cat.IsValidName

|  |  |  |  |
| --- | --- | --- | --- |
| Test Case | Input Data | Expected Output Data | Actual Output Data |
| Cat.IsValidName(incorrect data type) should result in compilation error | 13 | Cannot compile | Cannot compile |
| Cat.IsValidName(null) should return false | null | False | False |
| Cat.IsValidName(empty string) should return false | “” | False | False |
| Cat.IsValidName(name of length 9) should return false | “Fluffiest” | False |  |
| Cat.IsValidName(name of > 9) should return false | “Woolloongabba” | False |  |
| Cat.IsValidName(name of length 1) should return true | “A” | True |  |
| Cat.IsValidName(name of length 8) should return true | “Fluffier” | True |  |
| Cat.IsValidName(name of length between 2 and 7) should return true | “Cat” | True |  |

Tests for Cat.IsValidAge

|  |  |  |  |
| --- | --- | --- | --- |
| Cat.IsValidAge() should cause compile error | Nothing | Compile Error | Compile Error |
| Cat.IsValidAge(null) should cause compile error | null | Compile Error | Compile Error |
| Cat.IsValidAge(-1) should return false | -1 | False |  |
| Cat.IsValidAge(<-1) should return false | -5 | False |  |
| Cat.IsValidAge(31) should return false | 31 | False |  |
| Cat.IsValidAge(>31) should return false | 35 | False |  |
| Cat.IsValidAge(0) should return true | 0 | True |  |
| Cat.IsValidAge(2 <= age <= 29) returns true | 15 | True |  |
| Cat.IsValidAge(30) should return true | 30 | True |  |