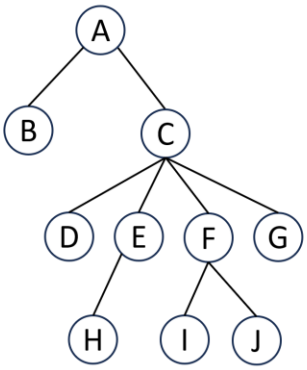
Left Child Right Sibling

(1 sec, 512mb)

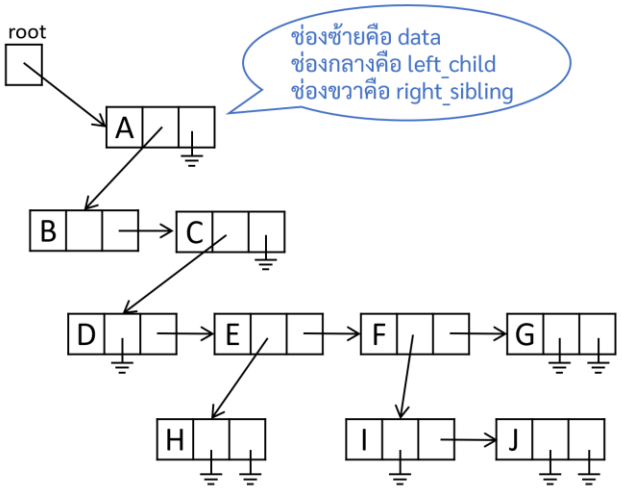
เราสามารถสร้างโครงสร้างข้อมูลประเภทต้นไม้ได้หลายแบบ ในวิชานี้ เราใช้คลาส node เพื่อเก็บแต่ละปมของต้นไม้ โดยที่ใน node จะมีตัวแปรชื่อ left, right เพื่อชี้ไปยัง ลูกซ้าย และ ลูกขวา ตามลำดับ หากเราต้องการให้ปมเรามีลูกมากกว่า 2 ปม เราก็มักจะใช้วิธีสร้าง array ของ node ขึ้นมา (เช่น vector<node*> children) แต่มีอีกวิธีหนึ่งที่เป็นที่นิยมกันคือ การสร้าง node แบบ Left Child Right Sibling (LCRS)

ใน node แบบ LCRS นั้นจะมีตัวแปรสองตัวคือ left_child (เรียกว่า “ลูกซ้าย”) เพื่อเก็บลูกซ้ายสุดของปมดังกล่าว และ right_sibling ซึ่งใช้ชี้ไปยังปมที่เป็นลูกคนถัดไปของพ่อของเรา (เรียกว่า “น้อง”) เช่น หากเราเป็นลูกคนที่ 2 ของปมพ่อของเรา ตัวแปร right_sibling ก็จะชี้ไปยังลูกคนที่ 3 ของพ่อของเรา หาก right_sibling เป็น NULL แล้ว ก็แปลว่าปมเราเป็นลูกคนสุดท้ายของพ่อของเรานั่นเอง code ที่อธิบายถึงคลาส node แบบ LCRS เป็นดังด้านขวามือนี้ และรูปด้านล่างนี้แสดงถึงตัวอย่างการเก็บข้อมูลต้นไม้โดยใช้ node แบบ LCRS

```
class node {
 public:
  int data;
  node *left_child;
  node *right_sibling;
}
```



ต้นไม้ที่ต้องการเก็บ



ตัวอย่างการเก็บข้อมูลแบบ Left Child Right Sibling

จงเขียนฟังก์ชัน int depth(node *n) เพื่อหาว่าต้นไม้ที่มีปมรากเป็น n นั้นมีความสูงเท่าใด โดยให้ node เป็นแบบ LCRS และนิยามให้ต้นไม้ที่ไม่มีปมเลยมีความสูงเป็น -1 และให้ต้นไม้ที่มีปมเดียวมีความสูงเป็น 0

ข้อบังคับ

- โจทย์ข้อนี้จะมีไฟล์ตั้งต้นมาให้ ประกอบด้วยไฟล์ main.cpp และ student.h อยู่ ให้นิสิตเขียน code เพิ่มเติมลงในไฟล์ student.h เท่านั้น และการส่งไฟล์เข้าสู่ระบบ grader ให้ส่งเฉพาะไฟล์ student.h เท่านั้น
  - ในไฟล์ student.h จะมีโครงของฟังก์ชัน depth ให้แล้ว
  - ไฟล์ student.h จะต้องไม่ทำการอ่านเขียนข้อมูลใด ๆ ไปยังหน้าจอหรือคีย์บอร์ดหรือไฟล์ใด ๆ
- หากใช้ VS Code ให้ทำการ compile ที่ไฟล์ main.cpp

**** main ที่ใช้จริงใน grader นั้นจะแตกต่างจาก main ที่ได้รับในไฟล์ตั้งต้นแต่จะทำการทดสอบในลักษณะเดียวกัน ****

คำอธิบายฟังก์ชัน main

main() จะสร้างตัวแปร node *root = nullptr ซึ่งเป็นปมรากของต้นไม้ของเรา หลังจากนั้น main จะรับคำสั่งมาจาก keyboard คำสั่งละ 1 บรรทัด โดยตัวอักษรแรกของแต่ละบรรทัดจะระบุการทำงานที่ต้องการ

- “q” หมายถึงหยุดการทำงาน
- “d” หมายถึงให้การเรียก depth(root) และพิมพ์ค่าที่คืนมา
- “a” หมายถึงการเพิ่มปมลงไปในต้นไม้ของเรา โดย a จะตามด้วยจำนวนเต็ม 1 ค่าที่จะใส่เข้าไปในต้นไม้ และตามด้วย “คำสั่งระบุตำแหน่ง” ซึ่งเป็นจำนวนเต็มอีกหลาย ๆ ค่าที่ระบุตำแหน่งที่ต้องการเพิ่มปม กล่าวคือให้ c1,c2,..,ck เป็น “คำสั่งระบุตำแหน่ง” เราสามารถระบุตำแหน่งได้ดังนี้

- ck (ตัวสุดท้ายของคำสั่ง) จะมีค่าเป็น -1 หรือ -2 เสมอ เพื่อบอกว่าเราต้องการเพิ่มปมไปเป็น “ลูกซ้าย” หรือ “น้อง” ตามลำดับ
- ci แต่ละตัวจะบอกว่าเราต้องการเดินทางไปทางไหนจากปมปัจจุบัน โดยเริ่มต้นให้เราอยู่ที่ปมราก หาก ci มีค่าเป็น 0 แปลว่าเราต้องการไปยังปม “ลูกซ้าย” แต่ถ้า ci มีค่ามากกว่า 0 แปลว่าเราต้องการไปยังปม “น้อง” ลำดับที่ ci
  - ตัวอย่างเช่น หากคำสั่งเป็น a 99 0 1 0 2 -1 หมายถึง เราต้องการเพิ่มปมที่มีค่าเป็น 99 ไปยังปม ลูกซ้ายของลูกลำดับที่ 3 ของ ลูกลำดับที่สอง ของ ปมราก เป็นต้น
- คำสั่ง a ครั้งแรกสุดที่เรียกใช้งาน จะมีค่า c1 เป็น -3 เสมอเพื่อบอกว่าเราต้องการตั้งค่าของปม root

ชุดข้อมูลทดสอบ

รับประกันว่าจำนวนคำสั่งที่เรียกใน main ไม่เกิน 50,000 ครั้ง

- 10% ต้นไม้จะไม่มีน้องเลย
- 15% ต้นไม้ที่เรียก depth มีความลึกไม่เกิน 2
- 15% ต้นไม้ที่เรียก depth มีความลึกไม่เกิน 3
- 20% ต้นไม้เป็น binary tree แน่นอน
- 40% ไม่มีเงื่อนไขอื่น ๆ

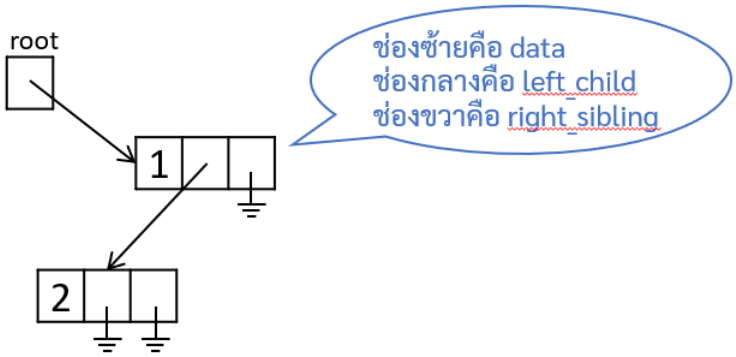
ตัวอย่าง

ข้อมูลนำเข้า	ข้อมูลส่งออก
a 1 -3 a 2 -1 d a 3 0 -1 a 4 0 0 -2 a 5 0 -2 d q	set root to 1 add left child at 1 depth = 1 add left child at 2 add right sibling at 3 add right sibling at 2 depth = 2

(คำอธิบายตัวอย่างอยู่ในหน้าถัดไป)

คำอธิบายตัวอย่าง

ภาพของ Tree ขณะที่ถามคำถาม Depth ครั้งที่ 1



ภาพของ Tree ขณะที่ถามคำถาม Depth ครั้งที่ 2