Programming Exercise 1: โครงสร้างข้อมูล: ต้นไม้

(โอลิมปิกวิชาการ ค่ายสอง ม. ศิลปากร 19 มีนาคม 2556, โดย ดร.ภิญโญ แท้ประสาทสิทธิ์)

Problem 1: Basics of Binary Search Tree

Subtask 1: Master Tree Traversal

จากโค้ดที่เตรียมไว้ให้ จะมีโค้ดที่แสดงไว้ในห้องเรียนแล้ว
และสำหรับการแวะผ่านต้นไม้ก็จะมีแต่การทำงานแบบ in-order เท่านั้น

*** จงเพิ่มฟังก์ชันสำหรับการแวะผ่านแบบ pre-order และ post-order เข้าไป โดยให้มีการประกาศฟังก์ชันดังนี้

```
void preorder(TreeNode* root);
void postorder(TreeNode* root);
```

*** การทดสอบโปรแกรม
โค้ดที่เตรียมไว้ให้จะทำการอ่านไฟล์เข้ามา โดยในแต่ละบรรทัดจะมีการดำเนินการบนต้นไม้หนึ่งอย่าง
ซึ่งที่เป็นไปได้มีดังนี้ Insert, Remove, Find, และ Print tree ด้วยวิธี in-order, pre-order หรือ post-order
ซึ่งคำสั่งมีรูปแบบดังนี้

1. Insert จะมีรูปแบบเป็น I key (ตัวไอใหญ่ ตามด้วย key ที่จะใส่เข้าไป) เช่น I 5
2. Remove จะมีรูปแบบเป็น R key (ตัวอาร์ใหญ่ ตามด้วย key ที่จะลบออก) เช่น R 5
3. Find จะมีรูปแบบคำสั่งเป็น F key (ตัวเอฟใหญ่ ตามด้วย key ที่จะหา) เช่น F 5
   ถ้ามีคีย์นั้นอยู่ในต้นไม้ก็ใช้พิมพ์คำว่า “\nY” (ขึ้นบรรทัดใหม่ก่อนพิมพ์ Y) ถ้าไม่มีคีย์นั้นให้พิมพ์ “\nN”
4. Print tree จะมีรูปแบบคำสั่งเป็น P method (ตัวพีใหญ่ ตามด้วยวิธีที่จะทำ) หาก method เป็น 1
   วิธีที่จะพิมพ์ก็คือ in-order หาก method เป็น 2 และ 3 วิธีที่พิมพ์ก็คือ pre-order และ post-order
   ตามลำดับ โดยจะพิมพ์ค่าเฉพาะคีย์ออกมา และก่อนพิมพ์ตัวแรกในแต่ละคำสั่งพิมพ์ให้ขึ้นบรรทัดใหม่
   ส่วนคีย์ที่ตามมาให้คั่นด้วยช่องว่างหนึ่งช่อง (ดูตัวอย่าง input และ output ในหน้าถัดไป)
   หมายเหตุ รหัสคำสั่งจะคั่นด้วยช่องว่างหนึ่งช่องก่อนที่จะตามด้วย key หรือ method

ในไฟล์ input_1.txt จะทำการใส่คีย์เข้าไป 8 ค่า ซึ่งโค้ดที่เตรียมไว้ทำให้เราแล้ว
แต่เราจะต้องพิมพ์ต้นไม้ออกมาให้ถูกต้อง

ตัวอย่าง input และ output (input_1.txt)

Input	Output
11	
I 5	1 2 3 4 5 6 7 8
I 7	5 3 2 1 4 7 6 8
I 3	1 2 4 3 6 8 7 5
I 2	
I 1	
I 4	
I 6	
I 8	
P 1	
P 2	
P 3	

อย่าลืมว่าการแวะผ่านต้นไม้ช่วยตรวจความถูกต้องของ pointer ในต้นไม้ได้ดีมาก

Subtask 2: The Pain of a Non-Recursive Method

ถึงแม้ว่าการทำงานแบบ non-recursive จะมีแนวโน้มว่าจะทำงานเร็วกว่า
แต่ความเร็วที่เพิ่มขึ้นก็ไม่ได้ทำให้ระดับความซับซ้อนของปัญหาลดลง (order of problem complexity remains the same) การเขียนโปรแกรมแบบ recursive
จึงมีความเหมาะสมมากกว่าในงานที่เวลาในการเขียนโค้ดมีน้อย (เช่น การแข่งขันต่าง ๆ)
เพราะโค้ดจะสั้นและตรวจสอบได้ง่ายกว่า

อย่างไรก็ตามฟังก์ชันแบบ recursive จะมีปัญหาตรงที่ว่า การเรียกซ้ำไปเรื่อย ๆ มาก ๆ
อาจจะทำให้ระบบไม่สามารถรองรับการทำงานและแครชได้
ปัญหานี้อาจจะพบได้ในปัญหาที่ใหญ่ในระดับที่ต้องรันหลายชั่วโมงจึงแล้วเสร็จ
(และไม่น่าจะพบได้ในการแข่งขันโอลิมปิก)

ในแบบฝึกหัดนี้นักเรียนจะได้ทดลองให้เห็นด้วยตัวเองว่า การเขียนโค้ดแบบ non-recursive อาจจะนำไปสู่โค้ดที่ยุ่งยากและเต็มไปด้วยที่ผิด (สำหรับมือใหม่) และการแก้ปัญหาด้วยการเขียนโค้ดแบบ recursive นับว่าเป็นวิธีที่เหมาะสมกว่ามากในการแข่งขันที่มีเวลาน้อย

*** จงเขียนฟังก์ชัน void painful_remove(int key, TreeNode*& root);
ซึ่งทำการลบโหนดออกจากต้นไม้โดยที่ไม่ทำการ recursive และให้เรียกใช้ฟังก์ชันนี้แทนการใช้ฟังก์ชัน remove ที่เตรียมไว้ให้

ตัวอย่าง input และ output (input_2.txt)

Input	Output
14	
I 5	N
I 7	Y
I 3	1 2 3 9 8 6 4
I 2	
I 1	
I 4	
F 8	
I 6	
I 8	
R 7	
R 5	
I 9	
F 8	
P 3	