รายงาน : Lab09 - Tree

จัดทำโดย

นายกิตติพศ หนูทอง รหัสนักศึกษา : 6135512003

นายปฏิภาณ วรรณโก รหัสนักศึกษา : 6135512059

Section : 01

240-207 Programing and Data Structures

“งานทั้งหมดนี้ในรายงานฉบับนี้ล้วนเป็นผลงานของข้าพเจ้า มิได้ลอกหรือสำเนามาจากที่อื่นใดในกรณีที่พบว่าเกิดสำเนาด้วยวิธีใดก็ตาม ข้าพเจ้ายินดีไม่ขอรับคะแนนจากรายงานฉบับนี้”

คะแนนที่ได้

ลงชื่อ

กิตติพศ หนูทอง ปฏิภาณ วรรณโก

(นายกิตติพศ หนูทอง) (นายปฏิภาณ วรรณโก)

Lab09 - Tree

ข้อที่ 1 : Perfect Tree

Perfect Tree

จงรับข้อมูลจำนวนบวกจากผู้ใช้ จนกว่าผู้ใช้จะใส่ 0 หรือ เลขติดลบ นำตัวเลขที่ได้ไปสร้าง tree โดยใช้ฟังก์ชัน `insert_node` ตามลำดับการป้อนข้อมูลของผู้ใช้

นิยามและเรียกใช้ฟังก์ชัน `is_perfect_tree` เพื่อตรวจสอบว่า tree ดังกล่าวสมดุลอย่างสมบูรณ์แบบหรือไม่

โดย tree จะสมดุลอย่างสมบูรณ์แบบ เมื่อ size ด้านซ้าย เท่ากับจำนวน size ด้านขวา สำหรับทุกๆ sub-tree

รูปแบบการแสดงผล

```
N01: <5>
N02: <2>
N03: <7>
N04: <1>
N05: <3>
N06: <6>
N07: <8>
N08: <0>
=Yes
```

Code

lab09-1.c

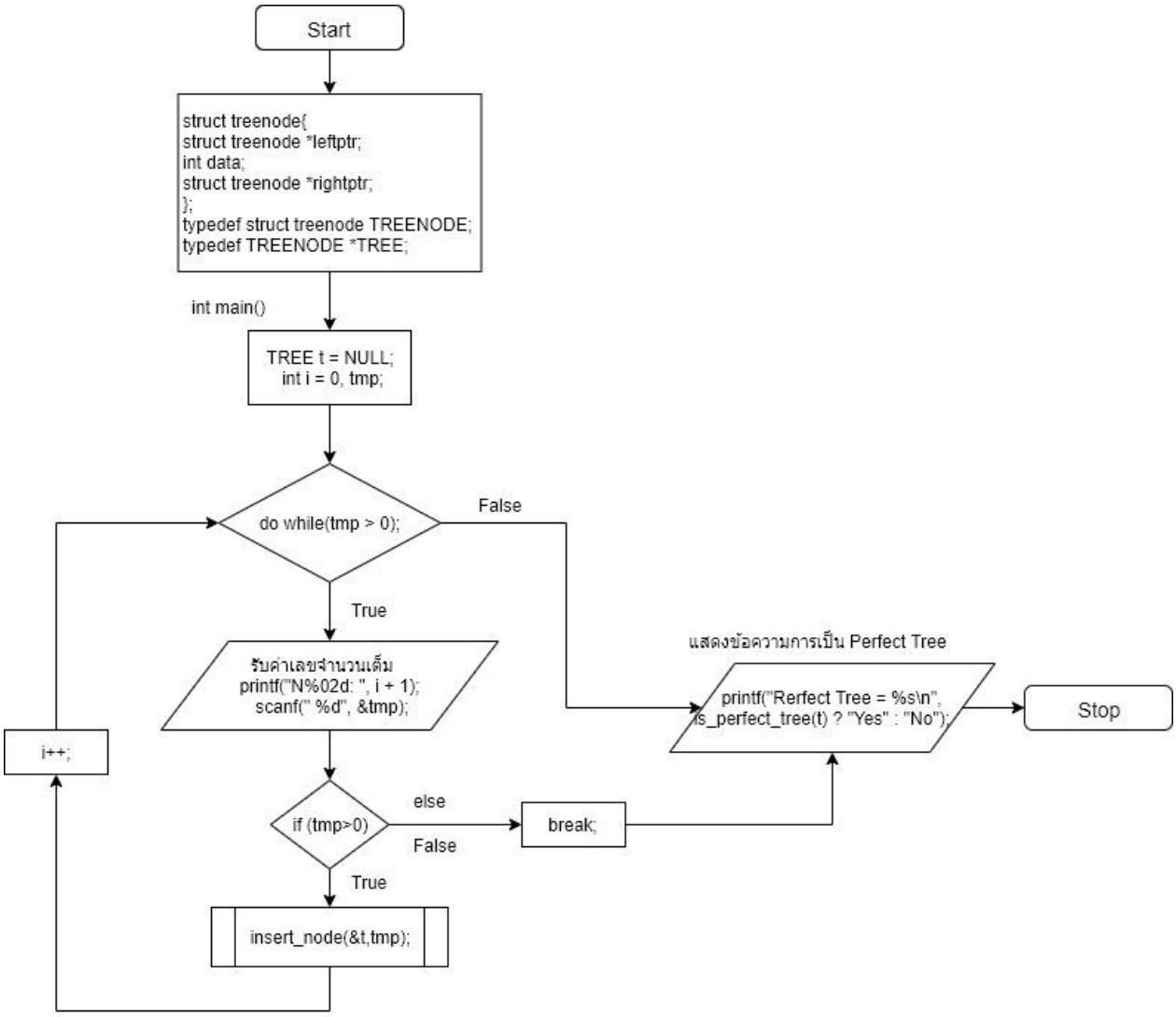
```c
#include <stdio.h>
#include<stdlib.h>
struct treenode{
struct treenode *leftptr;
int data;
struct treenode *rightptr;
};
typedef struct treenode TREENODE;
typedef TREENODE *TREE;

int is_perfect_tree(TREE t);
void insert_node(TREE *tp, int value);

int main()
{
 TREE t = NULL;
 int i = 0, tmp;
 do
 {
    printf("N%02d: ", i + 1);
    scanf(" %d", &tmp);
    if (tmp>0)
     {
        insert_node(&t,tmp);
     }
    else
    {
        break;
    }
    i++;
 }while(tmp > 0);

    printf("Rerfect Tree = %s\n", is_perfect_tree(t) ? "Yes" : "No");

 return 0;
}

```

```c
 void insert_node(TREE *tp, int value) {
/* tp is a pointer to a BST */
if (*tp == NULL)
{
    *tp = malloc(sizeof(TREENODE));
    (*tp)->data = value;
    (*tp)->leftptr = NULL;
    (*tp)->rightptr = NULL;
}
else if (value < (*tp)->data )
    insert_node(&((*tp)->leftptr), value);
else if (value > (*tp)->data )
    insert_node(&((*tp)->rightptr), value);
else
    printf("duplicate node\n");
}

```

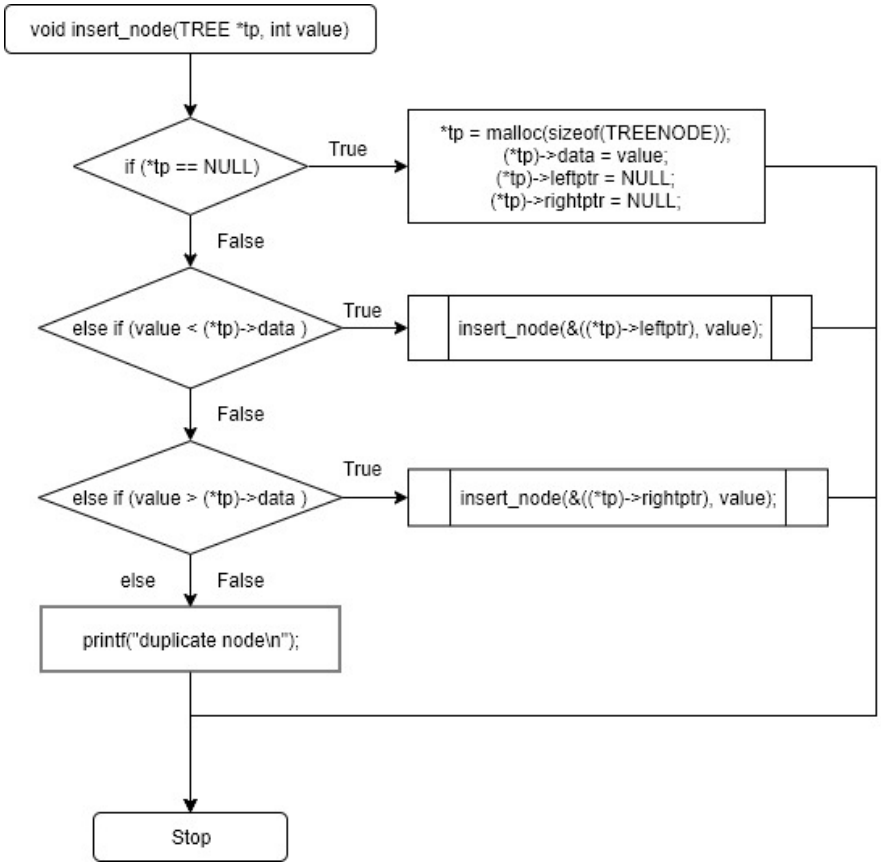
```c
int is_perfect_tree(TREE t)
{
    if(t==NULL)
    {
        return 1;
    }
    else if(t->leftptr == NULL && t->rightptr == NULL)
    {
        return 1;
    }
    else if(t->leftptr== NULL || t->rightptr == NULL)
    {
        return 0;
    }
    else
    {
        return is_perfect_tree(t->leftptr) && is_perfect_tree(t->rightptr);
    }
}

```

Flowchart

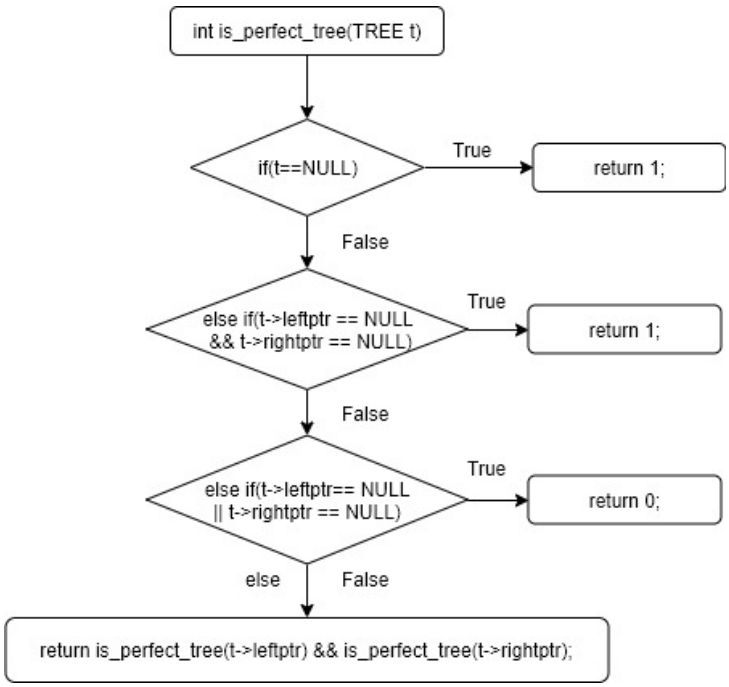
ฟังก์ชัน main()



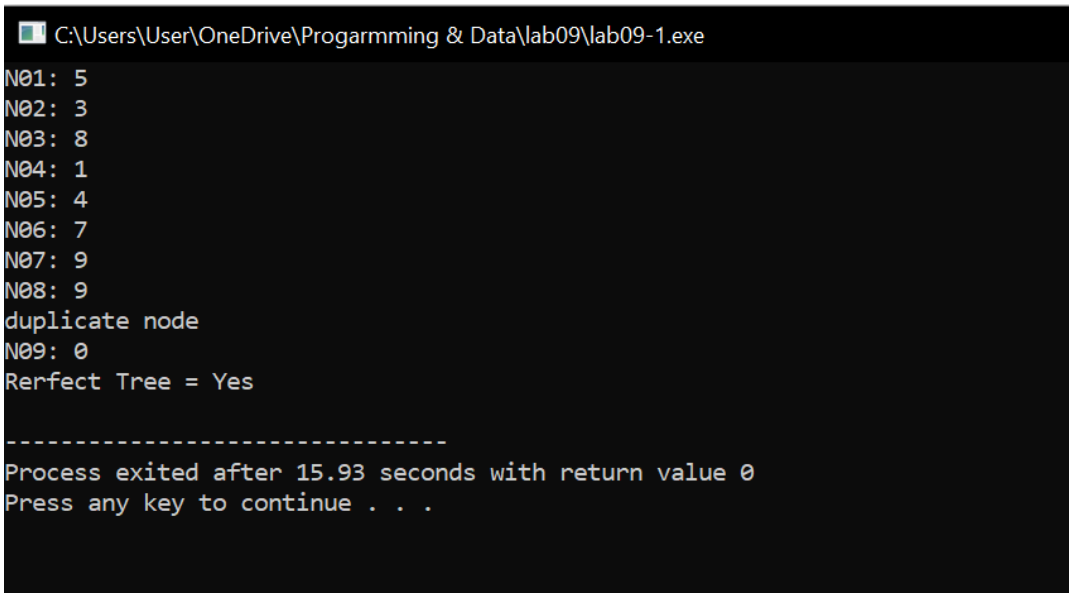
ฟังก์ชัน void insert_node(TREE *tp, int value)



ฟังก์ชัน int is_perfect_tree(TREE t)



ผลการรันโปรแกรม



อธิบายหลักการทำงาน : การทำงานของโปรแกรมจะทำงานโดยการรับค่าตัวเลขจำนวนเต็มมาทำการสร้าง tree ซึ่งจากการเขียน code โปรแกรมนี้จะเป็นการเช็คว่า tree ที่ได้สร้างมาเป็น perfect tree หรือไม่ โดยการตรวจสอบ Level ต่าง ๆ ว่าตรงกับเงื่อนไขของ perfect tree หรือไม่ ถ้าใช่จะตอบ yes ถ้าไม่จะตอบ No ซึ่งจะขึ้นอยู่กับการส่งค่าคืนกลับมาจากฟังก์ชันที่ใช้ในการตรวจ perfect tree โดยที่ code เบื้องต้นนี้ (ไม่รวมฟังก์ชันตรวจการเป็น perfect tree) จะเป็น codeที่โจทย์ให้มาอยู่แล้ว พวกผมจึงจะอธิบายแบบไม่ละเอียดมาก

ก่อนจะขึ้นฟังก์ชันจะมีการประกาศตัวแปรโครงสร้าง tree ทำการเปลี่ยนชื่อให้เป็น pointer *TREE เพื่อให้สะดวกกับการใช้งาน

ฟังก์ชันเมนเป็นฟังก์ชันที่มีหน้าที่ในการรับค่าตัวเลขจำนวนเต็มและใช้ if else ตรวจสอบค่าที่มีมากกว่าหรือเท่ากับ 0 ถ้าตรงเงื่อนไขจะทำการเรียกฟังก์ชัน insert_node tree ซึ่งจะส่งค่าที่รับเข้ามาเพื่อใช้ในการสร้าง tree หลังจากนั้นจะทำการเรียกใช้ฟังก์ชันตรวจสอบ is_perfect_tree เพื่อแสดงค่าคำตอบที่ได้กล่าวมาข้างต้น ซึ่งจะมีการวน loop do while ไปรับค่าเรื่อย ๆ จนกว่าค่าที่รับมาจะน้อยกว่าหรือเท่ากับ 0

ฟังก์ชัน insert_node tree เป็นฟังก์ชันที่ใช้สร้าง tree โดยให้ค่าตัวแรกเป็น root ของ tree จากนั้นใช้เงื่อนไขค่าถัดไปถ้ามีค่ามากกว่า root จะเพิ่มไป right sub tree และถ้าน้อยกว่า root จะเพิ่มไป left sub tree หลังจากนั้นจะทำไปเรื่อย ๆ จนกว่าจะครบค่าที่รับมาทุกตัวและถ้ามีค่าซ้ำกันจะมีการเตือนว่า node ซ้ำ (duplicate node) แล้วให้รับค่าใหม่

ฟังก์ชันตรวจสอบ is_perfect_tree การจะเป็น perfect tree ต้องตรงตามเงื่อนไขต่างๆดังนี้

1.จำนวน level ทางด้านซ้ายและขวาจะต้องเท่ากัน

2.มี size และ height ทางด้านซ้ายและข้างขวาที่เท่ากัน

3.ถ้าจะมี leaves หรือ มี child จะต้องมีเหมือนกันทั้งหมดใน level เดียวกัน

โดยการเขียน code เพื่อตรวจสอบเงื่อนไขดังกล่าวจะมีเงื่อนไขแรกคือ if(t==NULL) ถ้าใช่จะคืนค่า 1 ถ้าไม่ใช่จะตรวจสอบเงื่อนไข else if(t->leftptr == NULL && t->rightptr == NULL) ถ้าใช่จะคืนค่า 1 ถ้าไม่ใช่จะตรวจสอบเงื่อนไขถัดไปคือ else if(t->leftptr== NULL || t->rightptr == NULL) ถ้าใช่จะคืนค่า 0 เพราะใน level นั้นมี leaves หรือ มี child ไม่ครบทุก node และถ้าไม่ใช่จะทำการเรียกใช้ฟังก์ชันนี้ซึ่งเป็นการเรียกใช้ฟังก์ชันตัวมันเอง (recursive function) เพื่อตรวจสอบๆ ทุกใน level โดยส่งค่าของ t->leftprt และ t->rightprt

ความรู้จากการทำ Lab Perfect Tree : Perfect Tree คือ tree ที่มี leaves หรือ child ที่เท่ากันในระดับ level เดียวกันและมีขนาดและความลึกเท่ากัน ในการตรวจสอบ Perfect Tree จะต้องมีการตรวจสอบทั้งซ้ายและขวาของ root และทุกๆ level ของ tree นั้น

recursive function เป็นฟังก์ชันที่ทำการเรียกใช้ฟังก์ชันตัวมันเองซึ่งนำมาใช้ประโยชน์ในการตรวจ leaves และ child ในระดับ level ถัด ๆ ไป

ในทำการใช้ logic AND ระหว่างค่าที่คืนมากจากทั้งสองตัวเพื่อทำการคืนค่า logic ไปให้ฟังก์ชัน main () ซึ่ง logic ที่นี้หมายถึง 1 และ 0 แทนเงื่อนไขถ้าจะมี leaves หรือ มี child โดยต้องมีเหมือนกันทั้งหมดใน level เดียวกัน โดยที่ถ้ามีไม่เหมือนกันจะคืนค่า 1 และ 0 และเมื่อมาทำกระบวนการทาง logic จะได้ค่าเป็น 0 นั้นคือไม่ใช่ Perfect Tree