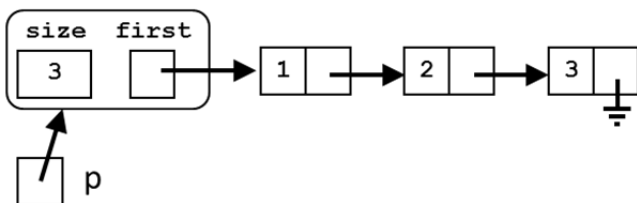
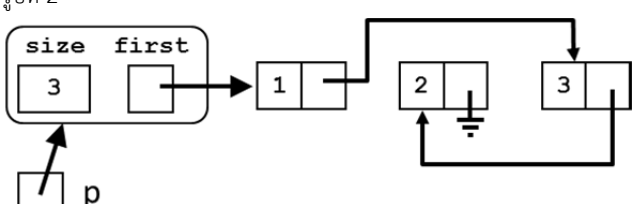
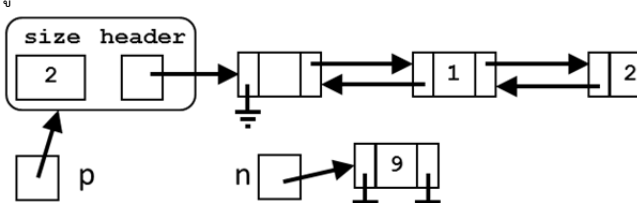
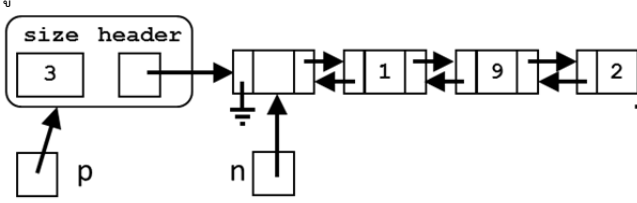


4. (10 คะแนน) ในข้อย่อยต่อไปนี้มี แต่ละข้อมีรูปที่แสดงโครงสร้างของข้อมูลประเภทหนึ่งอยู่ 2 รูป จงเติมส่วนของโปรแกรมลงในช่องว่างที่กำหนดให้เท่านั้น เพื่อให้ส่วนของโปรแกรมห้กล่าวเปลี่ยนแปลงโครงสร้างข้อมูลจากรูปที่ 1 ไปเป็นรูปที่ 2

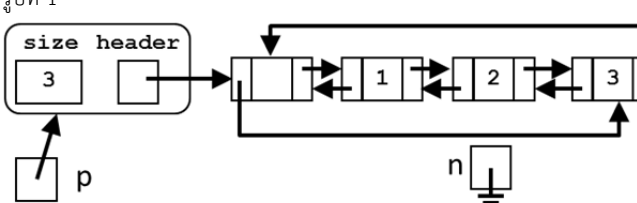
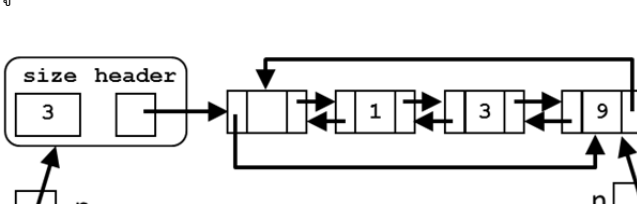
ก) โครงสร้างข้อมูลประเภท Singly Linked List (no header, not circular)

<p>รูปที่ 1</p> 	<p>ส่วนของโปรแกรม</p> <pre>int main(int argc, char *argv[]) { LList p = newLList(); addLList(p,1,0); addLList(p,2,1); addLList(p,3,2); // แปลงเป็นรูปที่ 2 ด้วยคำสั่งต่อไปนี้ p->first->next->next->next = p->first->next = p->first->next->next->next = NULL }</pre>
<p>รูปที่ 2</p> 	

ข) โครงสร้างข้อมูลประเภท doubly linked list with header (not circular)

<p>รูปที่ 1</p> 	<p>ส่วนของโปรแกรม</p> <pre>int main(int argc, char *argv[]) { DLList p = newDLList(); addDLList(p,1,0); addDLList(p,2,1); DLNode n = newDLNode(9,NULL,NULL); // แปลงเป็นรูปที่ 2 ด้วยคำสั่งต่อไปนี้ n->next = p->header-> n->prev = n-> p->header-> = n; n-> = n; p->size = n = p->header; }</pre>
<p>รูปที่ 2</p> 	

ค) โครงสร้างข้อมูลประเภท circular doubly linked list with header

<p>รูปที่ 1</p> 	<p>ส่วนของโปรแกรม</p> <pre>int main(int argc, char *argv[]) { DLList p = newDLList(); DLNode n = NULL; addDLList(p,1,0); addDLList(p,2,1); addDLList(p,3,2); // แปลงเป็นรูปที่ 2 ด้วยคำสั่งต่อไปนี้ n = p->header->prev->prev; n-> = n->next; n-> = n->prev; freeDLNode(n); n = newDLNode(9, p->header->prev,); p->header-> = n; p->header->prev = }</pre>
<p>รูปที่ 2</p> 	

5. (6 คะแนน) ส่วนของโปรแกรมต่อไปนี้เป็นการกระทำกับโครงสร้างข้อมูลประเภท AVL Tree จงวาดรูปที่แสดงถึงโครงสร้างและค่าต่าง ๆ ของ AVL Tree ดังกล่าว “หลังจาก” ที่ได้กระทำการตามบรรทัดที่ได้กำหนดให้เสร็จเรียบร้อยแล้ว

<pre>int main(int argc, char *argv[]) { AVL t = newAVL(); addAVL(t,1); addAVL(t,2); addAVL(t,3); // วาดรูปที่ (1) หลังทำคำสั่งนี้ addAVL(t,5); }</pre>	<pre>addAVL(t,4); // วาดรูปที่ (2) หลังทำคำสั่งนี้ addAVL(t,6); // วาดรูปที่ (3) หลังทำคำสั่งนี้ removeAVL(t,2); // วาดรูปที่ (4) หลังทำคำสั่งนี้ }</pre>
<p>วาดรูปที่ (1)</p>	<p>วาดรูปที่ (2)</p>
<p>วาดรูปที่ (3)</p>	<p>วาดรูปที่ (4)</p>

6. (6 คะแนน) จงระบุ (โดยการ วงรอบ คำว่า “ได้” หรือ “ไม่ได้”) ว่าการใส่ข้อมูลใน hash table ดังต่อไปนี้จะสามารถทำได้โดยไม่มีการ rehash หรือไม่ เมื่อเราใช้ linear probing, quadratic probing และ double hashing ที่มี $g(x) = 1 + (x \% 4)$

ทุกข้อใช้ hash function $h(x) = x \% m$ โดยที่ m คือขนาดของตารางในแต่ละข้อ

a) ตาราง $m = 6$,

Index	0	1	2	3	4	5
ข้อมูล	6		8		16	

เมื่อใส่ $x=12$ Linear Probing: **ได้** **ไม่ได้**
 Quadratic Probing: **ได้** **ไม่ได้**
 Double Hashing: **ได้** **ไม่ได้**

b) ตาราง $m = 6$,

Index	0	1	2	3	4	5
ข้อมูล	6	7	8		16	

เมื่อใส่ $x=13$ Linear Probing: **ได้** **ไม่ได้**
 Quadratic Probing: **ได้** **ไม่ได้**
 Double Hashing: **ได้** **ไม่ได้**

c) ตาราง $m = 8$,

Index	0	1	2	3	4	5	6	7
ข้อมูล		9		11		13	14	15

เมื่อใส่ $x=29$ Linear Probing: **ได้** **ไม่ได้**
 Quadratic Probing: **ได้** **ไม่ได้**
 Double Hashing: **ได้** **ไม่ได้**

d) ตาราง $m = 8$,

Index	0	1	2	3	4	5	6	7
ข้อมูล		1		19		21		7

เมื่อใส่ $x=17$ Linear Probing: **ได้** **ไม่ได้**
 Quadratic Probing: **ได้** **ไม่ได้**
 Double Hashing: **ได้** **ไม่ได้**

7. (10 คะแนน) จงเขียนฟังก์ชัน DLList expandList(DLList source, int rep) ซึ่งจะทำการ “ขยาย” ข้อมูลใน source ซึ่งเป็น circular doubly linked list with header ที่รับเข้ามา โดยฟังก์ชันนี้จะคืนค่า circular doubly linked list with header ใหม่ที่เกิดจากการ “ขยาย” ข้อมูลในแต่ละตำแหน่งของ source เป็นจำนวน rep ครั้ง โดยที่ source ยังคงเหมือนเดิม ตัวอย่างเช่น ถ้าให้ source เป็น list ที่มีข้อมูลเป็น <10,20,30,40> ผลลัพธ์ของการเรียก expandList(source,3) จะคืนค่ารายการใหม่ที่มีข้อมูลเป็น <10,10,10,20,20,20,30,30,30,40,40,40> เพื่อความสะดวกขอกำหนดให้ DType ของ list นั้นเป็น float และฟังก์ชันดังกล่าวควรมีประสิทธิภาพเป็น $O(n * rep)$ เมื่อ n คือขนาดของ source

```
DLList expandList(DLList source, int rep) {
    DLList list2 = newDLList();

    return list2;
}
```

8. (10 คะแนน) สมมุติว่า ลิงค์ลิสต์หนึ่ง มีโครงสร้างของปม เหมือนกับ circular doubly linked list with header ที่เรียนมา คือ มี data, prev, และ next แต่เราไม่รู้ว่าแต่ละปมที่ต่อกันนั้นเชื่อมต่อกันเป็นแบบ doubly linked list จริงหรือเปล่า และไม่รู้ว่าต่อกันจนเป็นวงวนหรือไม่ และไม่รู้ว่ามันมีปม header อยู่จริงหรือไม่ นอกจากนี้ เราก็ไม่รู้ว่าจำนวนปมที่มีอยู่นั้นเท่ากับค่า size หรือไม่ จงเขียนฟังก์ชัน int isCircularDoubly(DLList list) ที่คืนค่า 1 ถ้าลิงค์ลิสต์นี้มีโครงสร้างเชื่อมต่อกันเป็นแบบ circular doubly linked list with header ที่ถูกต้องและมีค่า size ที่ตรงกับความเป็นจริงเท่านั้น มิฉะนั้นฟังก์ชันนี้จะคืนค่า 0

```
int isCircularDoubly(DLList list) {

}
```

9. (10 คะแนน) ในข้อนี้เราจะอธิบายถึงโครงสร้างข้อมูลชนิดหนึ่ง ซึ่งมีชื่อว่า Quad Tree โดย Quad Tree เป็นโครงสร้างข้อมูลแบบต้นไม้ที่ใช้เก็บจุดในสองมิติ โดยแต่ละปมในต้นไม้มีนิยามดังนี้

```
typedef struct SQNode* QNode;
struct SQNode {
    float x, y;
    QNode bottomLeft, bottomRight, topLeft, topRight;
};
```

และต้นไม้มีนิยามดังนี้

```

struct SQTree {
    QNode root;    // รากของต้นไม้
    int size;     // จำนวนจุดในต้นไม้
};
typedef struct SQTree * QTree;
    
```

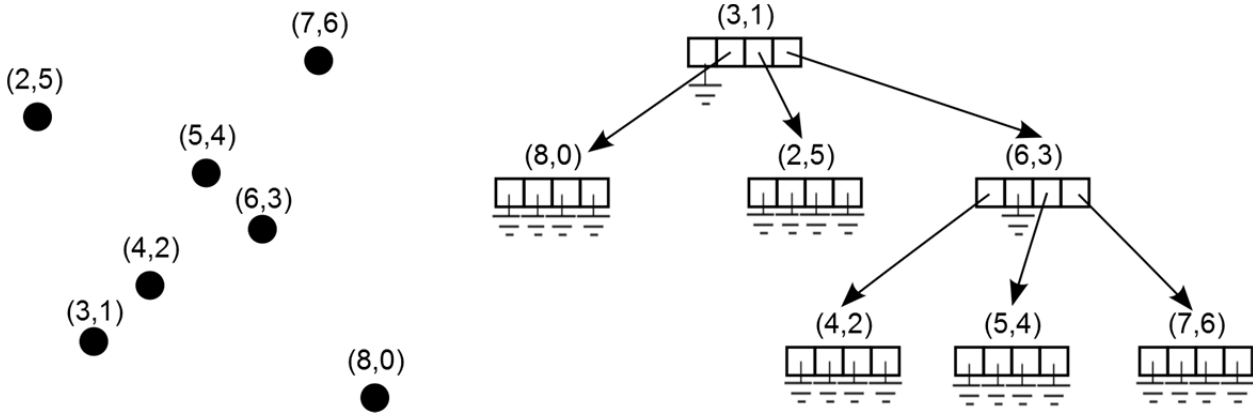
โดยแต่ละปมอาจจะเป็น NULL หรือไม่มีก็เก็บ พิกัด x,y ของจุด และจะมีลูก 4 ปม โดยลูกแต่ละตัวจะต้องมีคุณสมบัติดังนี้

- ลูก bottomLeft จะต้องเป็น NULL หรือมีพิกัด (x_1, y_1) โดยที่ $x_1 < x$ และ $y_1 < y$
- ลูก bottomRight จะต้องเป็น NULL หรือมีพิกัด (x_2, y_2) โดยที่ $x_2 > x$ และ $y_2 < y$
- ลูก topLeft จะต้องเป็น NULL หรือมีพิกัด (x_3, y_3) โดยที่ $x_3 < x$ และ $y_3 > y$
- ลูก topRight จะต้องเป็น NULL หรือมีพิกัด (x_4, y_4) โดยที่ $x_4 > x$ และ $y_4 > y$

ตัวอย่างเช่น

เมื่อเรามีจุด (3,1), (8,0), (2,5), (6,3), (4,2), (5,4), (7, 6)

ตัวอย่าง Quad Tree หนึ่งที่เป็นไปได้



ในข้อนี้เรารับรองว่าจะไม่มี สองจุดใด ๆ ที่มีค่าพิกัดแกน x หรือค่าพิกัดแกน y เท่ากันเลย ตัวอย่างเช่น รับรองว่าถ้ามีจุด (1,3) แล้ว จะไม่มีจุด (1,4) เป็นต้น
 จงเติมส่วนของโปรแกรมของฟังก์ชัน addPoint() ซึ่งทำการเพิ่ม จุดใน 2 มิติเข้าไปในโครงสร้างข้อมูลประเภท QuadTree ให้สมบูรณ์

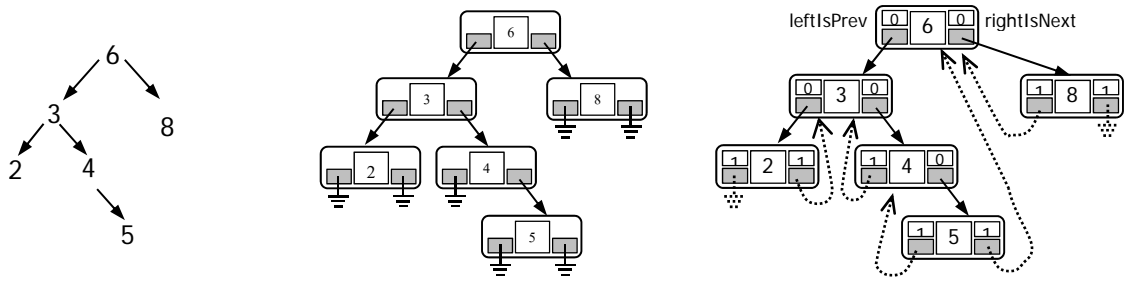
```

QNode newQNode(float x, float y, QNode bl, QNode br, QNode tl, QNode tr) {
    QNode node = (QNode)malloc(sizeof(SQNode));
    node->x = x; node->y = y; node->bottomLeft = bl; node->bottomRight = br;
    node->topLeft = tl; node->topRight = tr;
}

QNode addPoint(QNode node, float x, float y) {
    // เติม Code ของคุณอยู่ที่นี่
}

void addPoint(QTree tree, float x, float y) {
    tree->root = addPoint(tree->root, x, y);
    tree->size++;
}
    
```

10. (10 คะแนน) การแวะผ่านปมแบบตามลำดับ (inorder traversal) ทำได้ไม่ยากด้วยฟังก์ชันแบบ recursive หากเราต้องทำ inorder traversal บ่อย ๆ และต้องการหลีกเลี่ยงภาวะของ recursive ก็สามารปรับปรุงโครงสร้างเล็กน้อย รูปตรงกลางข้างล่างนี้แทนโครงสร้างของปมและการเชื่อมโยงปมต่างๆ ที่แทนการเก็บข้อมูลของต้นไม้ทางซ้าย จะเห็นได้ว่า ถ้าเก็บข้อมูล n ตัว จะมีตัวชี้จำนวน n+1 ตัวที่มีค่าเป็น NULL นำจะนำตัวชี้เหล่านี้มาใช้ให้เป็นประโยชน์ โดยปรับให้ตัวชี้ left ที่เป็น NULL ให้ชี้ไปยังปมที่อยู่ก่อนหน้าในลำดับแบบ inorder และ ปรับให้ตัวชี้ right ที่เป็น NULL ให้ชี้ไปยังปมที่อยู่ตามหลังใน ลำดับแบบ inorder ดังแสดงในรูปขวาข้างล่างนี้ ด้วยเส้นประ (ในกรณีที่ไม่มีปมก่อนหน้าและปมตามหลัง ก็ให้มีค่า NULL) แต่เนื่องจากเราต้องรู้ว่า ตัวชี้ left และ right เป็นแบบใด เราก็มักตัวแปรอีกสองตัวเก็บในปม ตัวหนึ่งชื่อ leftIsPrev มีค่าจริง เมื่อ left ชี้ปมก่อนหน้าในลำดับ inorder (แต่ถ้าเป็นเท็จ แสดงว่าชี้ปมลูกซ้าย) และอีกตัวแปรชื่อ rightIsNext มีค่าจริง เมื่อ right ชี้ปมตามหลังในลำดับ inorder (แต่ถ้าเป็นเท็จ แสดงว่าชี้ปมลูกขวา) รูปขวาข้างล่างนี้ จริงแทนด้วย 1 เท็จแทนด้วย 0



ก) ด้วยโครงสร้างที่นำเสนอข้างต้น เราสามารถเขียน printInorder ด้วยวงวนแบบ while ดังข้างล่างนี้ ซึ่งมีการเรียกฟังก์ชัน nextInorder จงเขียนฟังก์ชัน nextInorder(BNode n) ที่รับปม n และคืนปมที่ถัดจาก n ในลำดับแบบ inorder

```
#include <stdio.h>
#include <stdlib.h>

typedef struct SBNode *BNode;
struct SBNode {
    DType data;
    int leftIsPrev; // เก็บ 1 (จริง) ถ้า left ชี้ปมก่อนหน้าในลำดับแบบ inorder, เก็บ 0 (เท็จ) เมื่อ left ชี้ปมลูกซ้าย
    int rightIsNext; // เก็บ 1 (จริง) ถ้า right ชี้ปมถัดไปในลำดับแบบ inorder, เก็บ 0 (เท็จ) เมื่อ right ชี้ปมลูกขวา
    BNode left;
    BNode right;
};
struct SBST {
    BNode root;
    int size;
};
typedef struct SBST *BST;
//-----
BNode nextInorder(BNode n){
    }

void printInorder(BST t) {
    char buf[100];
    BNode n = t->root;
    if (n == NULL) return;
    while(n->left != NULL) n = n->left;
    while(n != NULL) {
        printf("%s ", toString(buf, n->data));
        n = nextInorder(n); //
    }
}
```

ต้นไม้ที่ปรับโครงสร้างตัวชี้ที่นำเสนอในข้อนี้เรียกว่า Threaded Binary Search Tree

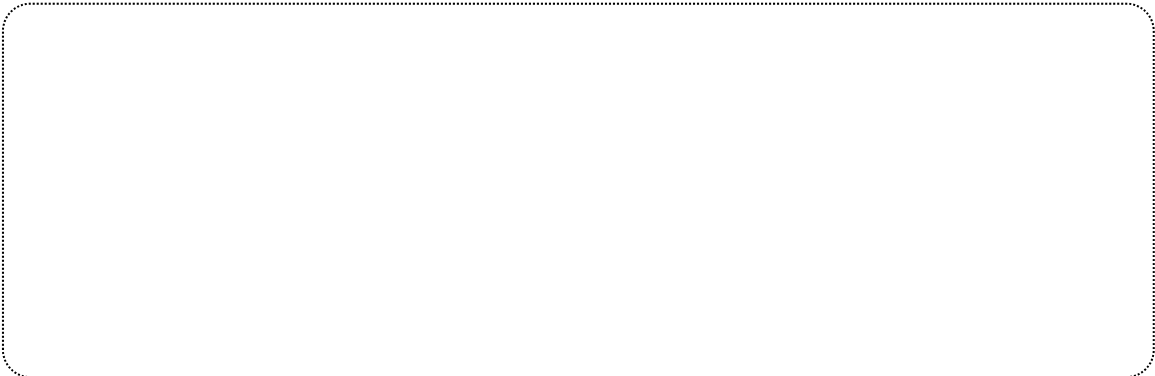
ข) จงเขียนคำสั่งเพิ่มในฟังก์ชัน addBST ในช่องว่างเว้นไว้ เพื่อเชื่อมโยงตัวชี้ left และ right พร้อมทั้งกำหนดค่าของ leftIsPrev และ rightIsNext ในถูกต้องตามโครงสร้างที่เสนอข้างต้น

```

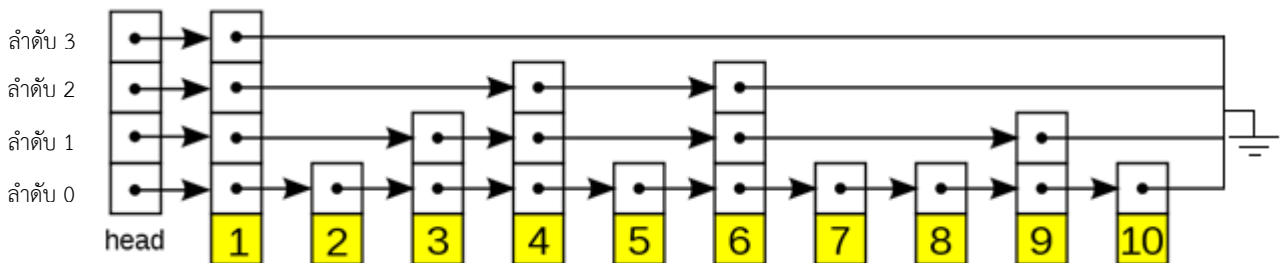
void addBST(BST t, DType x) {
    BNode r = t->root;
    BNode p = NULL; // parent of r
    int c;
    while (r != NULL) {
        p = r;
        c = cmp(x, r->data);
        if (c == 0) {
            r->data = x; // dup -> replace
            return;
        }
        if (c < 0 && r->leftIsPrev) break; // leftIsPrev เก็บ 1 (จริง) แสดงว่าไม่มีลูกซ้ายแล้ว
        if (c > 0 && r->rightIsNext) break; // rightIsNext เก็บ 1 (จริง) แสดงว่าไม่มีลูกขวาแล้ว
        r = (c < 0) ? r->left : r->right;
    }
    BNode n = newBNode(x, NULL, NULL);
    if (t->root == NULL) {
        t->root = n;
    } else {
        if (c < 0) { // เพิ่มปม n ทางซ้ายของปม p
            // ...
        } else { // เพิ่มปม n ทางขวาของปม p
            // ...
        }
    }
    t->size++;
}
    
```

เพิ่มสองคำสั่งนี้กรณีเป็น threaded BST

if (c < 0 && r->leftIsPrev) break; // leftIsPrev เก็บ 1 (จริง) แสดงว่าไม่มีลูกซ้ายแล้ว
 if (c > 0 && r->rightIsNext) break; // rightIsNext เก็บ 1 (จริง) แสดงว่าไม่มีลูกขวาแล้ว



11. (10 คะแนน) SkipList เป็นโครงสร้างข้อมูลทีพัฒนาต่อมาจาก LinkedList โดยจะเก็บข้อมูลเรียงลำดับจากค่าน้อยไปค่ามาก SkipList มีรูปแบบการเก็บข้อมูลดังรูปด้านล่างนี้ โดยในแต่ละปมจะมีตัวโยง k ตัว (แต่ละปมอาจมีค่า k ไม่เท่ากัน) ตัวโยงที่ 0 จะโยงไปยังปมที่เก็บข้อมูลตัวต่อไป, ตัวโยงที่ 1 จะโยงข้ามข้อมูลไปไกลขึ้น, ตัวโยงที่ 2 จะโยงข้ามข้อมูลไปไกลขึ้นไปอีก,.. จนตัวโยงที่ k-1 จะไปไกลสุด โดยตัวอย่างด้านล่างเป็น SkipList ที่มีข้อมูลเก็บอยู่ 10 ตัวคือ 1 ถึง 10 ในข้อนี้สิตจะต้องเติมส่วนของโปรแกรมสำหรับการค้นข้อมูลใน SkipList โดยนิตจะต้องใช้ประโยชน์จากการโยงข้ามข้อมูลของ SkipList มากที่สุดเท่าที่จะทำได้ เพื่อให้การค้นข้อมูลทำได้เร็วขึ้น (มีฉะนั้นจะได้คะแนน)



```
#define MAX_K 30
struct SSkipListNode;
typedef struct SSkipListNode* SkipListNode;
struct SSkipListNode {
    SkipListNode nextNodes[MAX_K];    // nextNodes[i] จะเก็บตัวโยงไปยังปมถัดไป และจะเป็น NULL หากเป็นปมสุดท้าย
    int k;                            // จำนวนตัวโยงที่ปมนี้มี
    DType data;                       // เก็บข้อมูลในปมนี้ ยกเว้นที่ ปมหัว(header) จะไม่มีข้อมูลเก็บอยู่
};
struct SSkipList;
typedef struct SSkipList* SkipList;
struct SSkipList {
    SkipListNode header;
};
..... // คำสั่งอื่นๆ ที่จำเป็นของ SkipList
int isInSkipList(SkipList list, DType x) {
    // เดิม Code ที่นี้ให้คืน 1 ถ้า x อยู่ใน list, คืน 0 ถ้า x ไม่อยู่ใน list
}
}
```

12. (15 คะแนน) ข้อนี้โจทย์ยาว ถ้ายังทำข้ออื่นไม่เสร็จ ควรจะทำข้ออื่นก่อน อีกไม่นานก็จะถึงวันเลือกตั้งผู้ว่า กทม. สมมติให้มีผู้มีสิทธิ์ออกเสียงจำนวน 5,000,000 คน แต่ละคนมีหมายเลขประชาชน 13 หลักที่ไม่ซ้ำกันเลย ผู้มีสิทธิ์แต่ละคนจะถูกกำหนดหน่วยเลือกตั้งที่ต้องไปลงคะแนน กำหนดให้หน่วยเลือกตั้งนั้นกำกับด้วยหมายเลขตั้งแต่ 1 ถึง M และมีผู้สมัครรับเลือกตั้ง N คน ซึ่งมีหมายเลขเป็น 1 ถึง N เราอยากพัฒนาระบบเลือกตั้งให้มีความทันสมัยโดยใช้คอมพิวเตอร์เข้าช่วยในการเก็บข้อมูลการลงคะแนน และตรวจสอบการมาใช้สิทธิ์ของผู้มีสิทธิ์ออกเสียง ให้นิสิตออกแบบโครงสร้างข้อมูลชื่อ **VoteCounter** ที่สามารถทำงานได้ตามฟังก์ชันที่ระบุไว้ข้างล่างนี้
- **VoteCounter newStructure(int candidate, int numUnit, int unit[], char *ID[])** ฟังก์ชันนี้จะสร้างโครงสร้างข้อมูล **VoteCounter** ขึ้นมา โดย **candidate** คือจำนวนผู้สมัครรับเลือกตั้ง, **numUnit** คือจำนวนหน่วยเลือกตั้งที่มีอยู่ ส่วน **unit** และ **ID** นั้นเป็นอาร์เรย์ขนาด 5,000,000 ช่องที่เก็บข้อมูลหน่วยเลือกตั้งของผู้มีสิทธิ์เลือกตั้งแต่ละคน โดย **unit[i]** คือหน่วยเลือกตั้งของคนที่ มีหมายเลขประชาชนเป็น **ID[i]**
 - **int vote(VoteCounter v, char *ID, int unit, int voting)** ฟังก์ชันนี้จะเป็นการลงคะแนน โดย **ID** คือหมายเลขประชาชนของผู้ลงคะแนน ที่มาทำการลงคะแนนที่หน่วยเลือกตั้งหมายเลข **unit** และลงคะแนนให้ผู้สมัครหมายเลข **voting** ฟังก์ชันนี้จะต้องตรวจสอบว่าการลงคะแนนนั้นถูกต้องหรือไม่ โดยการลงคะแนนที่ถูกต้องคือผู้ลงคะแนนนั้นมาลงคะแนนในหน่วยที่ถูกต้องเพียงครั้งเดียว และเลือกผู้สมัครที่มีอยู่จริง โดยจะคืนค่า 1 ก็ต่อเมื่อการลงคะแนนนี้เป็นการลงคะแนนที่ถูกต้องเท่านั้น และคืนค่า 0 ในกรณีอื่น ๆ (มันเป็นไปไม่ได้ที่จะมีการเรียกฟังก์ชันนี้ โดยระบุ **ID** กับ **unit** ที่ไม่ถูกต้อง หรือมีการเรียกฟังก์ชันนี้หลายครั้งโดยมี **ID** ที่ซ้ำกัน ในกรณีตัวอย่างเหล่านี้ ฟังก์ชันนี้จะต้องคืนค่า 0 และไม่ทำการนับคะแนนในการเรียกที่ไม่ถูกต้อง)
 - **void getVotingRate(VoteCounter v, float *rate)** ฟังก์ชันนี้จะคำนวณอัตราส่วนผู้มาใช้สิทธิ์ในแต่ละหน่วย โดยฟังก์ชันนี้จะต้องกำหนดค่าใน **rate[i]** ให้ถูกต้อง กำหนดให้ค่า **rate[i]** คืออัตราส่วนของผู้ที่มาลงคะแนนของหน่วยเลือกตั้งที่ **i** โดยนับเฉพาะการมาลงคะแนนผ่านการเรียกฟังก์ชัน **vote** ที่ถูกต้องเท่านั้น

- `void countVote(VoteCounter v, int *vote);` ฟังก์ชันนี้จะนับคะแนนที่ผู้สมัครแต่ละคนได้ โดยฟังก์ชันนี้จะกำหนดค่าให้ตัวแปร `vote` โดย `vote[i]` จะต้องระบุถึงจำนวนคนที่ได้ลงคะแนนเลือกผู้สมัครหมายเลข `i` อย่างถูกต้อง (นับเฉพาะการลงคะแนนที่ถูกต้อง)
การใช้โครงสร้างข้อมูล `VoteCounter` นั้น จะต้องเรียกคำสั่ง `newStructure` ก่อนเป็นอย่างแรกเสมอ แล้วหลังจากนั้นผู้ใช้อาจจะเรียกใช้ฟังก์ชันที่เหลือตามลำดับใดเป็นจำนวนกี่ครั้งก็ได้ (เช่น อาจจะเรียก `vote` 10 ครั้ง แล้วเรียก `getVotingRate` กับ `countVote` แล้ววนกลับไปเรียก `vote` อีก 20 ครั้ง แล้วเรียก `countVote` อีกครั้ง เป็นต้น) ให้นิสิตตอบคำถามต่อไปนี้ในพื้นที่ด้านล่างนี้
 - 1) อธิบายหลักการทำงานของโครงสร้างข้อมูลที่ออกแบบขึ้น พร้อมทั้งวาดรูปประกอบ
 - 2) เขียน code ที่อธิบายถึง struct ของโครงสร้างข้อมูลดังกล่าว และ struct อื่น ๆ ที่สร้างขึ้นใหม่ (ไม่จำเป็นต้องเขียน struct ของของที่เรียนมาแล้วในห้องเรียน ถ้าไม่มีการเปลี่ยนแปลง)
 - 3) อธิบายการทำงานของฟังก์ชัน 4 ฟังก์ชันข้างต้น โดยนิสิตต้องอธิบายรายละเอียดมากพอที่จะนำไปเขียนโปรแกรมได้ โดยไม่จำเป็นต้องเขียนโปรแกรม นิสิตสามารถเรียกใช้โครงสร้างข้อมูลอื่น ๆ ที่ได้เรียนมาในชั้นเรียนได้ แต่ต้องอธิบาย DType และฟังก์ชัน `cmp` ที่ใช้สำหรับ DType นั้น ๆ