

บทที่ 7

Linked list

สุนทรี คุ่มไพโรจน์

ทำไมถึงต้องใช้ linked list

Linked list vs. array

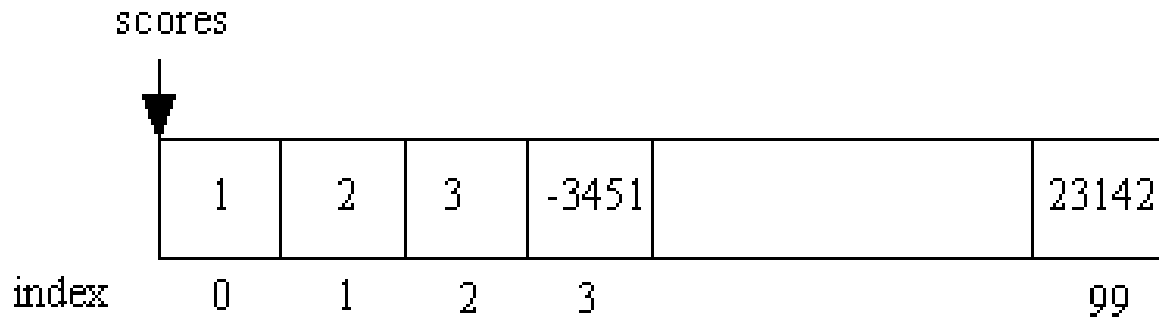
- เป็นโครงสร้างข้อมูลที่เก็บ collections of data เหมือนกัน
- สามารถเก็บข้อมูลได้ทุกชนิดเหมือนกับโครงสร้างข้อมูลอื่น ๆ
- แต่ linked list และ array มีข้อดี ข้อเสียต่างกัน

ทบทวน array

- Array เป็นโครงสร้างที่ง่ายที่สุดและพื้นฐานที่สุดในบรรดาโครงสร้างข้อมูลด้วยกัน
- โครงสร้าง array ประกอบไปด้วยโดเมน (domain) และ เรนจ์ (range)
- แต่ที่เราใช้กันมาก ก็มักจะให้โดเมนเป็นหมายเลขของสมาชิกใน array
- และเรนจ์เป็นค่าสมาชิกใน array ตัวอย่างของ array code เป็นดังนี้

```
void ArrayTest ( ){  
    int scores[100];  
    // operate on the elements of the scores array.....  
    scores[0]=1;  
    scores[1]=2;  
    scores[2]=3;  
}
```

ทบทวน array (ต่อ)



รูป 1 โครงสร้างของ array scores

ข้อดีของ array

- อ้างถึงข้อมูลง่ายและรวดเร็ว โดยใช้เครื่องหมาย []
- เช่นกำหนดให้ค่าสมาชิกของ array ตำแหน่งที่ 0 มีค่าเป็น 1
ทำได้โดยใช้คำสั่ง `scores[0] = 1;`
- ส่วนการเข้าถึงสมาชิกของ array ตำแหน่งอื่น ๆ ก็สามารถทำได้ดังนี้
`scores[i]` เมื่อ `i` คือตำแหน่งของ array

ทบทวน array (ต่อ)

ข้อเสียของ array

1. ขนาดของ array จะกำหนดตายตัว

- * ถ้าใช้งานจริงแค่ 20 หรือ 30 ตัว ที่เหลือจะกลายเป็นขยะ

- * ถ้าต้องการใช้งานมากกว่า 100 ตัว code ที่เขียนไว้ข้างต้นก็ใช้ไม่ได้

2. การเพิ่มสมาชิกใหม่เข้าไปใน array ต้องทำการย้ายสมาชิกเดิมที่มีอยู่
แล้วเป็นจำนวนมาก

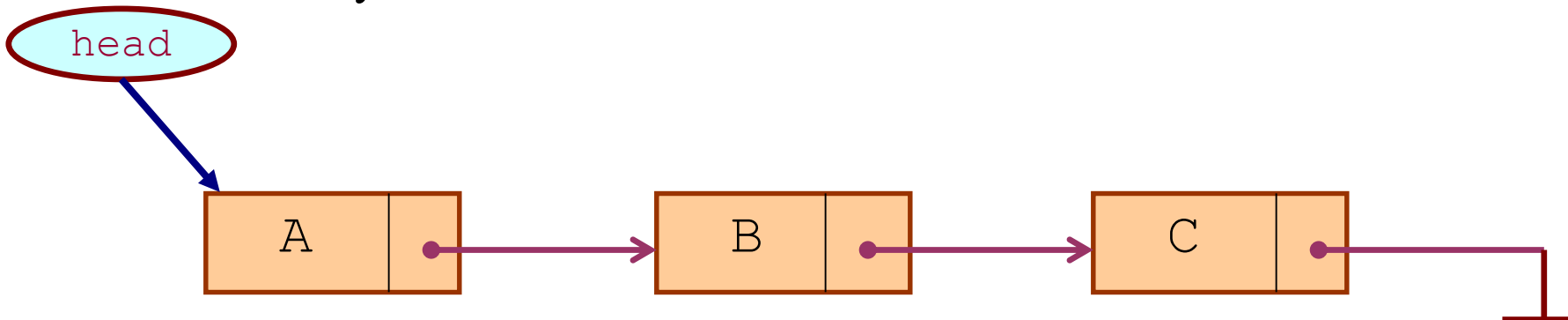
3. Array จองพื้นที่ในหน่วยความจำสำหรับทุกสมาชิกติดกันเป็นบล็อก
(block)

Linked list

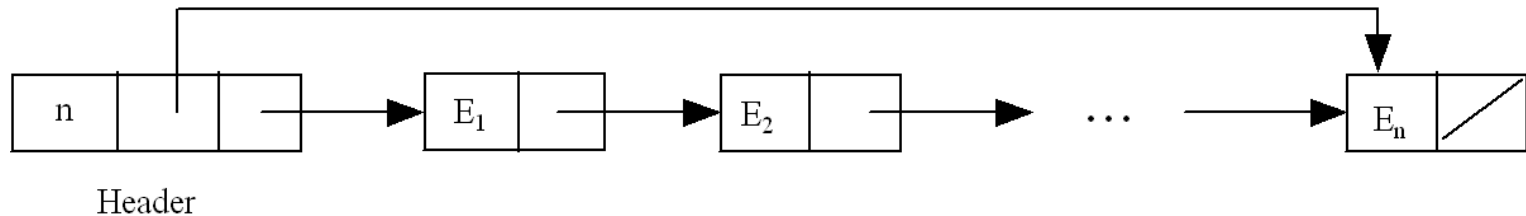
Linked list มีโครงสร้างสำคัญ 2 ส่วนคือ

- ❑ ส่วนพอยน์เตอร์ (pointer) มีไว้สำหรับชี้สมาชิกต่าง ๆ ใน linked list
- ❑ ส่วนข้อมูล (information)

- แต่ละสมาชิกของ linked list ไม่จำเป็นต้องเก็บเรียงเป็นบล็อกติดกัน
เหมือนใน array

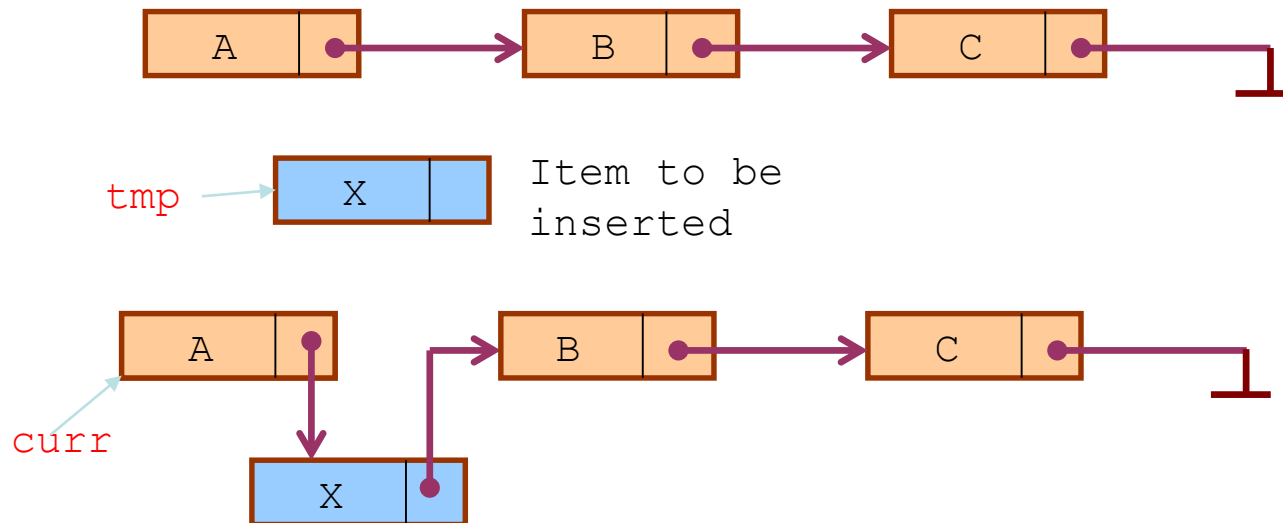


Linked list



- สามารถจัดเรียงตามค่าของพอยน์เตอร์ที่อยู่ในสมาชิกนั้น
- ใช้พอยน์เตอร์เป็นตัวเชื่อมโยง (link) แต่ละสมาชิกของ linked list ให้อยู่รวมกันเป็นสายของข้อมูล
- สามารถจัดการกับข้อมูลในสายของข้อมูลนี้ได้อย่างมีประสิทธิภาพ
- สามารถ เพิ่ม ตัดทอน ข้อมูลในส่วนใด ๆ ได้สะดวกกว่า array
- เช่น ถ้าหากต้องการแทรกข้อมูลใด ๆ ลงไปใน linked list ก็ทำได้ โดยการสร้างสมาชิกใหม่ใน linked list ที่มีค่าตามต้องการ

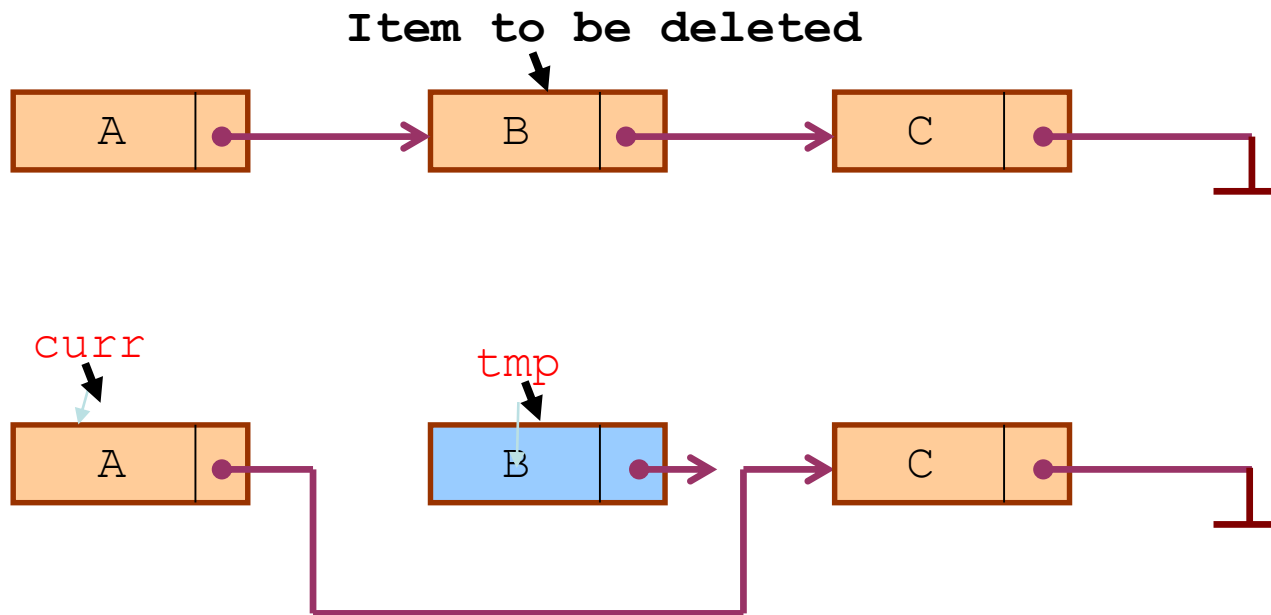
- การแทรกสมาชิก(insert):
 - ทำโดยให้พอยน์เตอร์ของสมาชิกตัวใหม่นี้ชี้ไปที่สมาชิกเดิมที่จะอยู่ตำแหน่งถัดจากสมาชิกใหม่
 - แล้วย้ายพอยน์เตอร์ของสมาชิกที่ต้องการให้สมาชิกใหม่นี้ต่อท้ายไปชี้ที่สมาชิกใหม่



ตัวอย่างโปรแกรมการ insert

```
typedef struct nd {  
    struct item data;  
    struct nd * next;  
} node;  
  
void insert(node *curr) {  
    node * tmp;  
  
    tmp=(node *) malloc(sizeof(node));  
    tmp->next=curr->next;  
    curr->next=tmp;  
}
```

- การลบสมาชิก:
 - ทำโดยให้พอยน์เตอร์ของสมาชิกตัวที่อยู่ก่อนหน้าตัวที่จะลบชี้ไปที่สมาชิกเดิมที่จะอยู่ตำแหน่งถัดจากสมาชิกที่จะถูกลบ
 - ทำการ **free** สมาชิกตัวที่ต้องการลบ(tmp) (คืนค่าให้หน่วยความจำ)



ตัวอย่างโปรแกรมการ delete

```
typedef struct nd {  
    struct item data;  
    struct nd * next;  
} node;  
  
void delete(node *curr) {  
    node * tmp;  
  
    tmp=curr->next;  
    curr->next=tmp->next;  
    free(tmp);  
}
```

กรณีที่ใช้ array

- ต้องทำการย้ายข้อมูลที่อยู่ก่อนหน้าหรืออยู่หลังสมาชิกใหม่ทุกตัวออกไปเสียก่อน เพื่อสร้างที่ว่างให้กับสมาชิกใหม่
- ต้องกำหนดจำนวนสมาชิกที่แน่นอนที่ต้องการใช้ และมีอยู่บ่อยครั้งที่เราใช้เนื้อที่ที่จองนี้ไม่หมด ทำให้สิ้นเปลืองเนื้อที่ในการเก็บข้อมูลโดยใช่เหตุ

ข้อดีอีกประการหนึ่งของ linked list คือ



- สามารถใช้เนื้อที่ในการเก็บข้อมูลได้อย่างมีประสิทธิภาพในกรณีที่เราไม่แน่นอนจำนวนข้อมูลไม่แน่นอน
- สามารถสร้างสมาชิกใหม่ขึ้นมาเพื่อทำการเพิ่มการเก็บข้อมูลได้ ตามต้องการ

การสร้างและการจัดการกับ linked list ในภาษา C

โครงสร้างของ โหนด(node) หรือสมาชิกของ linked list ประกอบด้วย 2 ส่วนคือ

- ส่วนข้อมูลอาจเป็นจำนวนเต็ม ทศนิยม หรือโครงสร้างข้อมูลอื่น ๆ
- ส่วนพอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวถัดไป

// ส่วนโหนดที่เป็นสมาชิก

```
struct node{  
    int item;  ส่วนข้อมูล  
    struct node *next;  ส่วนพอยน์เตอร์ที่ชี้ไปยัง node ตัวถัดไป  
};  
typedef struct node node_t;
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  struct Node {
5      int      num;
6      struct   Node  *next;
7  };
8
9  void main() {
10     struct Node* first  = NULL;
11     struct Node* last   = NULL;
12
13     // allocate 2 nodes in the heap
14     first      = (struct Node*) malloc(sizeof(struct Node));
15     last       = (struct Node*) malloc(sizeof(struct Node));
16
17     first->num = 1;
18     first->next = last;
19
20     last->num = 5;
21     last->next = NULL;
22
23     return 0;
24 }

```

ตัวอย่างโปรแกรม 1

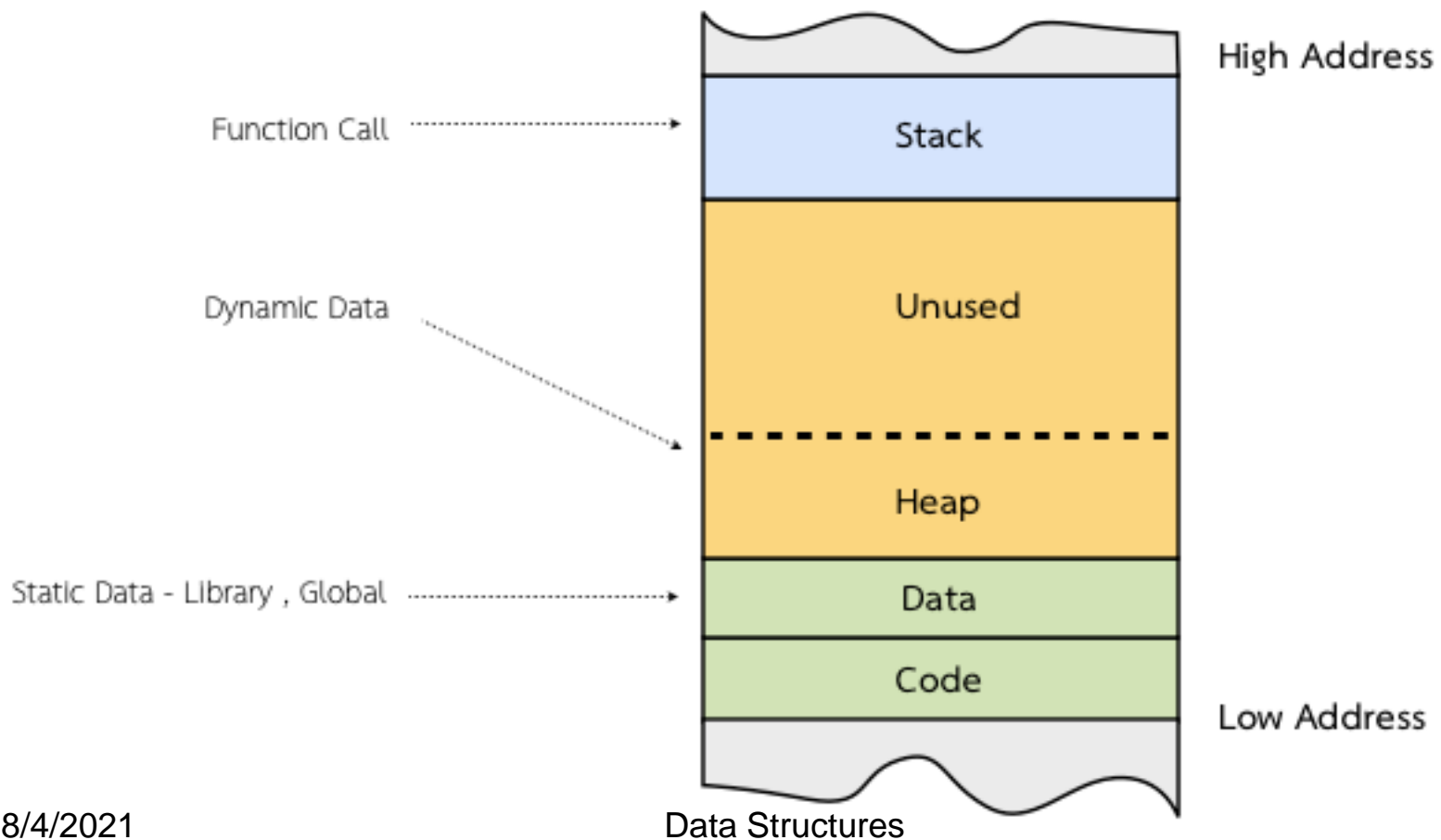
- ให้นิสิตเขียนแผนภาพ linked list จากโปรแกรม
- ให้นิสิตลองเขียนโปรแกรมเพิ่ม node ที่มีค่า num = 3 เข้าไป

Heap

- dynamic storage
- พื้นที่ของ memory ในคอมพิวเตอร์ ที่จะถูกจัดสรร/จัดการด้วย CPU อย่างคร่าวๆ ไม่รัดกุม
- โดยพื้นที่ของ heap ส่วนใหญ่มักจะเป็นพื้นที่ว่างขนาดกว้างๆบน memory
- ในการจองหน่วยความจำแบบ heap
เรียกใช้คำสั่ง malloc() หรือ calloc()
- เมื่อใช้งานเสร็จ ให้เรียก free() เพื่อเลิกจอง
หรือเลิกใช้งานพื้นที่บน heap เมื่อไม่ต้องการใช้งานอีกต่อไป
- ถ้าลืมที่จะ free() หลังจากเลิกใช้งานแล้ว จะเกิด **memory leak**
ทำให้ memory บน heap ยังถูกเก็บข้อมูลขยะไว้
(และ process อื่นๆ ก็จะใช้งานมันไม่ได้)

ที่มา <http://computer2know.blogspot.com/2016/04/stack-heap.html>

การจัดสรร memory



การจองและคืนพื้นที่ในหน่วยความจำ

ฟังก์ชัน `malloc()` เป็นฟังก์ชันที่ทำหน้าที่ allocates a block of memory และ return พอยน์เตอร์ที่ชี้ไปยังค่า block of memory ใหม่ หรือ return ค่า NULL ถ้ามีเนื้อที่ของ memory ไม่พอ

Syntax of malloc()

```
ptr = (castType*) malloc(size);
```

Example

```
ptr = (float*) malloc(100 * sizeof(float));
```

ที่มา <https://www.programiz.com/c-programming/c-dynamic-memory-allocation>

การคืนพื้นที่ในหน่วยความจำ

ฟังก์ชัน `free()` เป็นฟังก์ชันที่ทำหน้าที่คืนเนื้อที่ในหน่วยความจำที่จองไว้

Syntax of free()

```
free(ptr);
```

การใช้งานฟังก์ชัน `malloc()` และ `free()` จะต้องทำการ include header `stdlib.h` เข้ามาด้วย ตัวอย่างการใช้งานฟังก์ชัน `malloc()` และ `free()` เป็นดังนี้

ที่มา <https://www.programiz.com/c-programming/c-dynamic-memory-allocation>

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

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

ตัวอย่างโปรแกรม malloc() & free()

```
void main( void ) {  
    char *string;    /* Allocate space for a path name */  
    string = malloc( sizeof(char) );  
    if( string == NULL )  
        printf( "Insufficient memory available\n" );  
    else {  
        printf( "Memory space allocated for path name\n" );  
        free( string );  
        printf( "Memory freed\n" );  
    }  
}
```

```

1  #include <stdio.h>
2  #include <stdlib.h>
3
4  typedef struct node {
5      int      num;
6      struct   node  *next;
7  } Node;
8
9  Node *newNode () {
10     Node  *p;
11     p = (Node *) malloc(sizeof (Node));
12     p->next = NULL;
13     return p;
14 }
15
16 void main() {
17     Node *first      = newNode ();
18     Node *last       = newNode ();
19     first->num        = 1;
20     first->next       = last;
21     last->num         = 5;
22     last->next        = NULL;
23     return 0;

```

ตัวอย่างโปรแกรม 2

ที่มีการกำหนด typedef และ
สร้าง function newNode();

- ให้นิสิตเขียนแผนภาพ linked list จากโปรแกรม
- ให้เปรียบเทียบ โปรแกรม1&2

การสร้างและการจัดการกับ linked list

เราสามารถกำหนดให้ส่วนหัว(head)ของ linked list ประกอบไปด้วยโครงสร้าง 3 ส่วนคือ

- ความยาวของ linked list
- พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวแรก
- พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวสุดท้าย

ดังตัวอย่างในหน้าถัดไปนี้

// กำหนดส่วนหัวของ linked list

```
struct head{
```

```
    int length;
```

```
    node_t *first,*last;
```

```
};
```

```
typedef struct head head_t;
```

8/4/2021

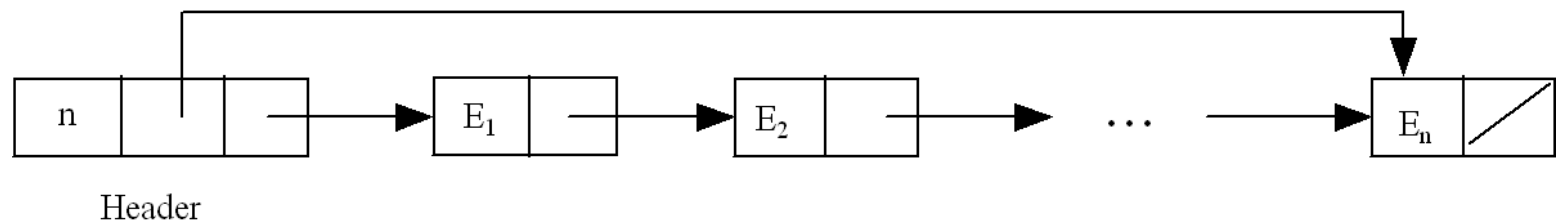
Data Structures

ความยาวของ linked list

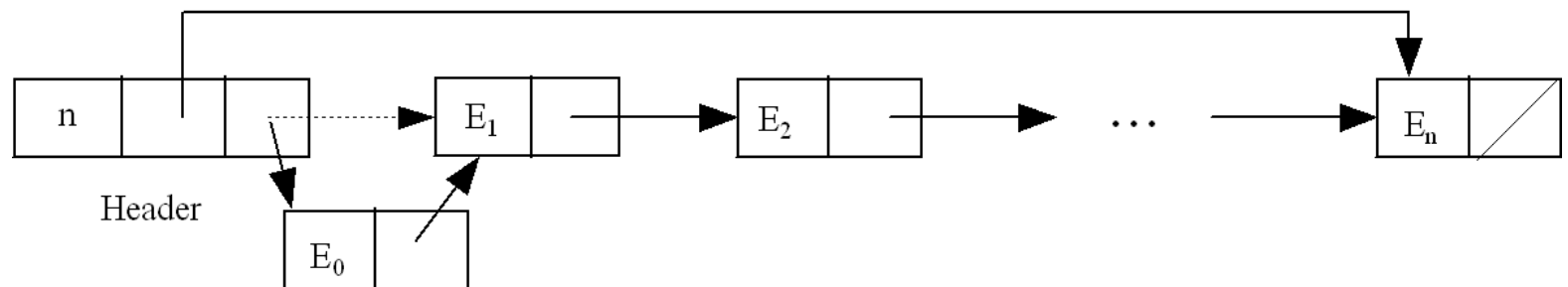
พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวแรก

พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวสุดท้าย

การเพิ่มสมาชิกใหม่เข้าไปในส่วนหน้าของ linked list



ก) ก่อนการเพิ่ม



ข) หลังการเพิ่ม

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

```
typedef struct node {
    int      num;
    struct   node *next;
} Node;
```

```
typedef struct linked_list {
    int      length;
    Node     *head;
} NumList;
```

```
Node *newNode() {
    Node *p;
    p = (Node *) malloc(sizeof (Node));
    p->next = NULL;
    return p;
}
```

```
void initList(NumList *n) {
    n->head = NULL;
}
```

```
25 int appendHeadList(NumList *s, Node *p) {
26     if (s->head == NULL) {
27         s->head = p;
28     } else { // s->head != NULL
29         p->next = s->head;
30         s->head = p;
31     }
32 }
33
34 void printNumList(NumList L) {
35     Node *pCurr = L.head;
36
37     while (pCurr != NULL) {
38         printf("Num: %d\n", pCurr->num);
39         pCurr = pCurr->next;
40     }
41 }
42
43 void main() {
44     NumList NL;
45     Node *N;
46     initList(&NL);
47     for (int i = 0; i < 3; ++i) {
48         N = newNode();
49         N->num = i;
50         appendHeadList(&NL, N);
51     }
52
53     printNumList(NL);
54     return 0;
55 }
```

- ผลลัพธ์การพิมพ์ตัวเลขจะเป็นอย่างไร?
- ให้นิสิตเขียนโปรแกรมคำนวณค่า length

อีกตัวอย่างของการ insert front

// ส่วนไหนที่เป็นสมาชิก

```
struct node{
```

```
    int item; ← ส่วนข้อมูล
```

```
    struct node *next; ← ส่วนพอยน์เตอร์ที่ชี้ไปยัง node ตัวถัดไป
```

```
};
```

```
typedef struct node node_t;
```

// กำหนดส่วนหัวของ linked list

```
struct head{
```

```
    int length; ← ความยาวของ linked list
```

```
    node_t *first, *last; ← พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวแรก
```

```
}; ← พอยน์เตอร์ที่ชี้ไปยังสมาชิกตัวสุดท้าย
```

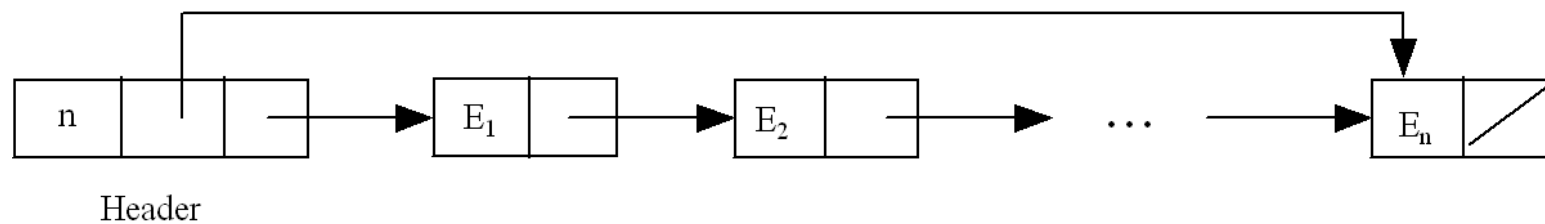
```
typedef struct head head_t;
```


อีกตัวอย่างของการ insert front (ต่อ)

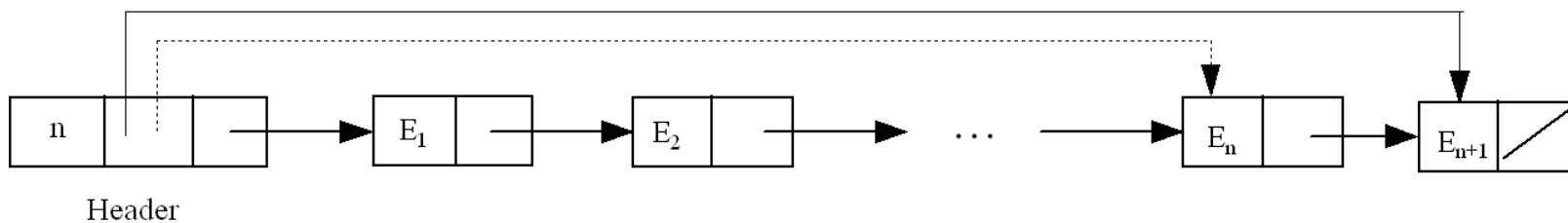
```
void insertFront(int data, head_t * head) {  
    node_t *new;  
    new=inst_node(data, head->first);  
    head->first = new;  
    if (head->length == 0)  
        head->last = new;  
    head->length++;  
}
```

```
node_t * inst_node(int val, node_t *ptr) {  
    node_t *new;  
    if (new = malloc(sizeof(node_t))) {  
        new->item = val;  
        new->next = ptr;  
    }  
    return (new);  
}
```

การเพิ่มสมาชิกใหม่เข้าไปในส่วนท้ายของ linked list



ก) ก่อนการเพิ่ม



ข) หลังการเพิ่ม

การ insert ส่วนท้าย

```
void insertEnd(int data, head_t * head){  
    node_t *new;  
    new=inst_node(data, NULL);  
    if (head->length)  
        head->last->next = new;  
    else  
        head->first = new;  
    head->last = new;  
    head->length++;  
}
```

การแสดงผลข้อมูลใน linked list

```
void printList(head_t * head){
```

```
    node_t * current;
```

```
    int i;
```

```
    current=head->first;
```

```
    if (head->length == 0)
```

```
        printf("Empty list\n");
```

```
    else{
```

```
        printf("link list\n");
```

```
        for(i=1;i<=head->length;i++){
```

```
            printf("%d\t",current->item);
```

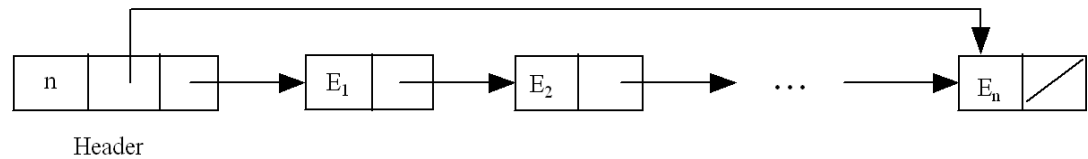
```
            current = current->next;
```

```
        }
```

```
        printf("\n");
```

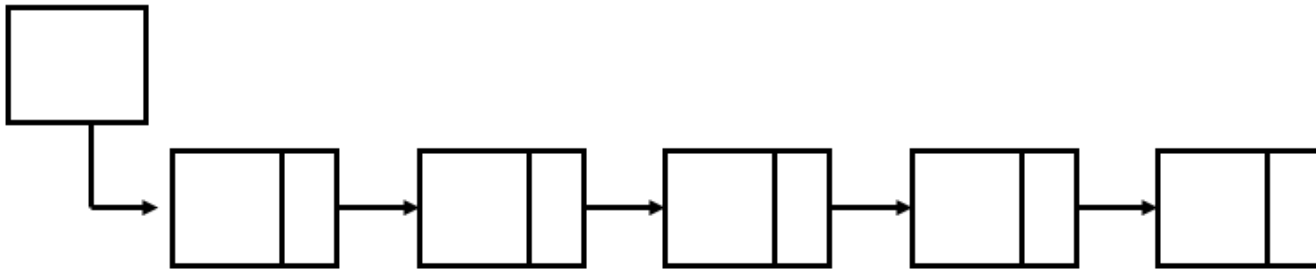
```
    }
```

```
}
```

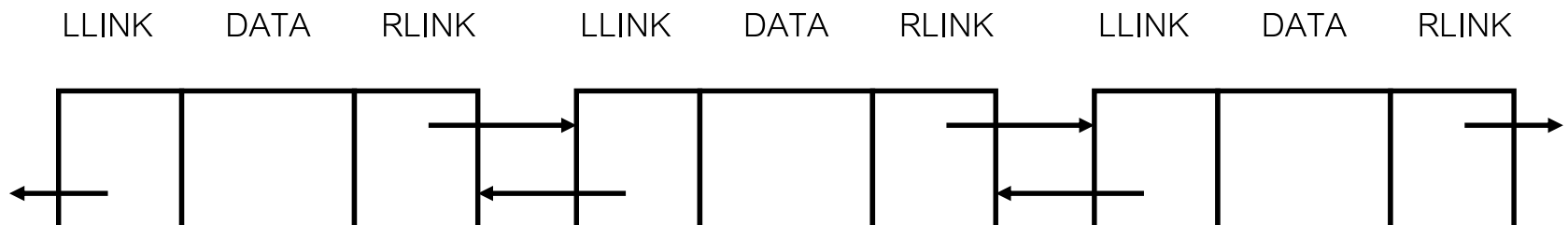


ประเภทของ linked list

- โครงสร้างข้อมูลลิงค์ลิสต์เดี่ยว (Singly Linked List : SLL)

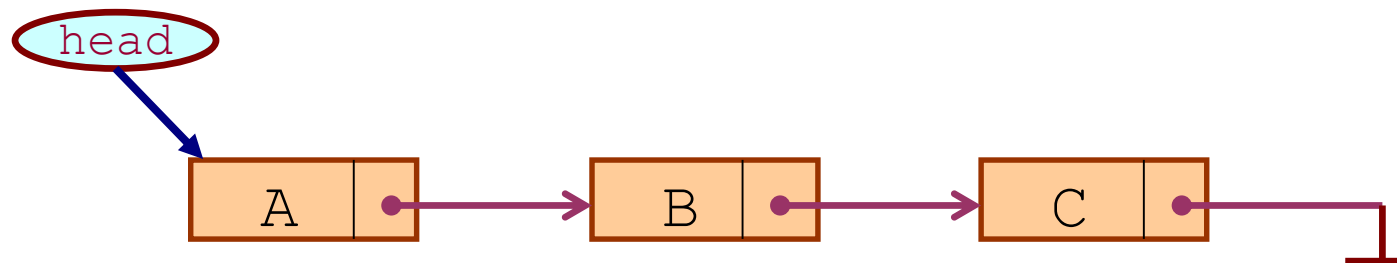


- โครงสร้างข้อมูลลิงค์ลิสต์คู่ (Doubly Linked List : DLL)

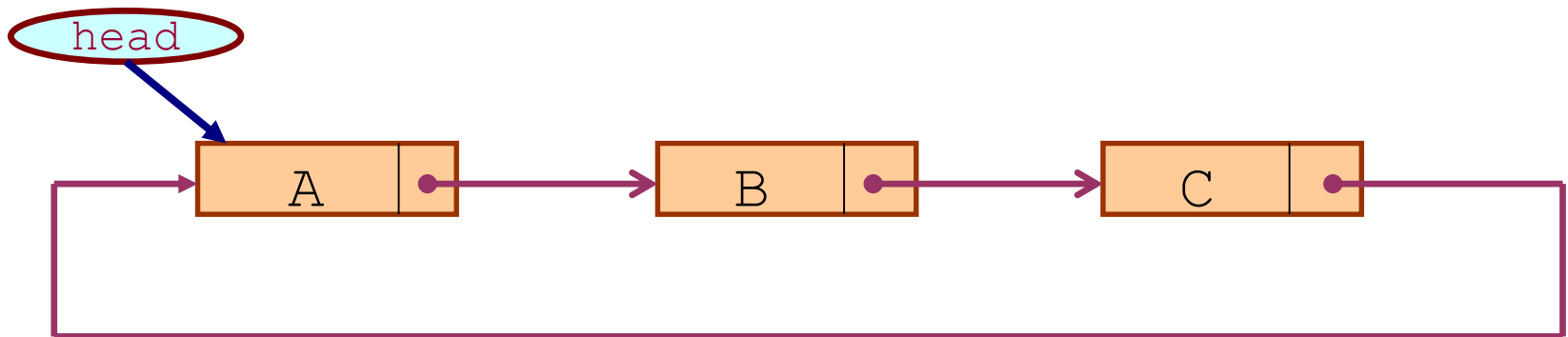


โครงสร้างข้อมูลลิ่งคี่ลิสต์เดี่ยว (SLL)

- โครงสร้างข้อมูลลิ่งคี่ลิสต์แบบ Ordinary Singly Linked List



- โครงสร้างข้อมูลลิ่งคี่ลิสต์แบบ Circular Singly Linked List (CLL)



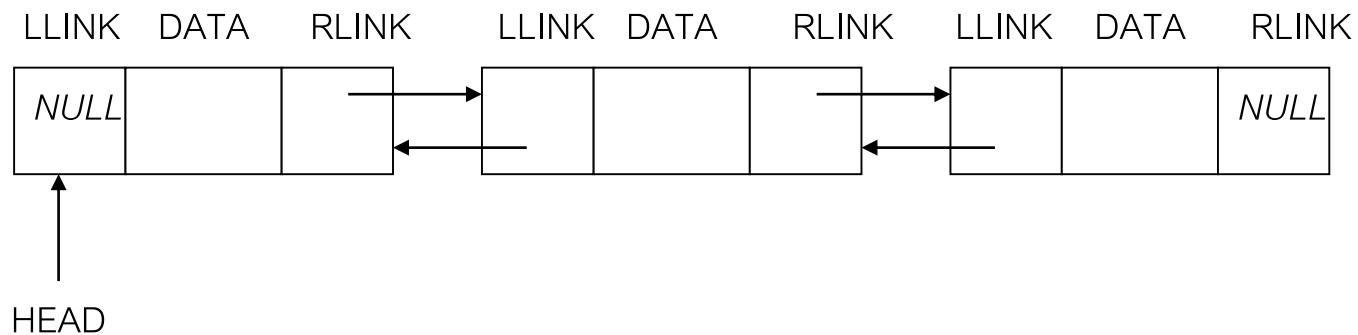
โครงสร้างข้อมูลลิงค์ลิสต์คู่ (Doubly Linked List)

- เป็นโครงสร้างที่แต่ละโหนดข้อมูลสามารถชี้ตำแหน่งโหนดข้อมูลถัดไปได้ 2 ทิศทาง (มีพอยน์เตอร์ชี้ตำแหน่งอยู่สองทิศทาง)
- โดยมีพอยน์เตอร์อยู่ 2 ตัว
 - พอยน์เตอร์ LLINK ทำหน้าที่ชี้ไปยังโหนดด้านซ้ายของโหนดข้อมูลนั้น ๆ
 - พอยน์เตอร์ RLINK ทำหน้าที่ชี้ไปยังโหนดด้านขวาของโหนดข้อมูลนั้น ๆ



ประเภทของลิ่งคี่ลิสต์แบบ Doubly Linked List

- โครงสร้างข้อมูลลิ่งคี่ลิสต์แบบ Ordinary Doubly Linked List (ODLL)



- โครงสร้างข้อมูลลิ่งคี่ลิสต์แบบ Circularly Doubly Linked List (CDLL)

