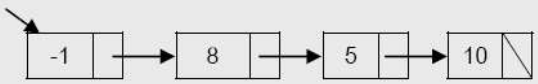
โครงสร้างข้อมูล
Data Structure

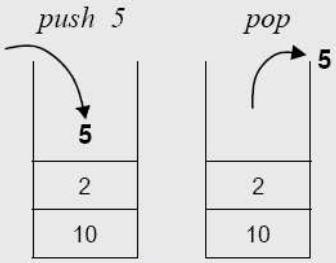
โครงสร้างข้อมูล

- การรวมประเภทข้อมูล (Data Type) เข้าไว้ด้วยกัน จนกลายเป็นกลุ่มประเภทข้อมูล และมีนิยามความสัมพันธ์ภายในกลุ่มข้อมูลอย่างชัดเจน
- การรวมกลุ่มนี้อาจเป็นการรวมกลุ่มกันระหว่างข้อมูลประเภทเดียวกัน ต่างประเภทกัน หรือต่างโครงสร้างข้อมูลกันก็ได้

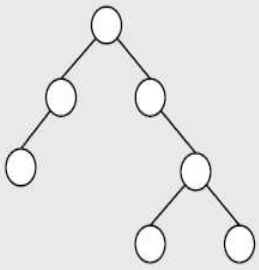


Linked list

Queue



Stack

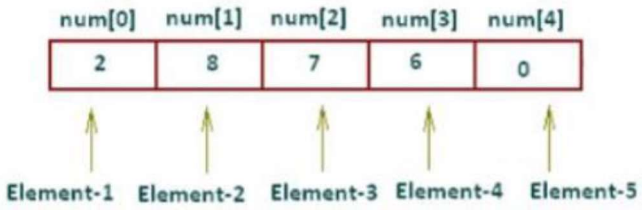

Tree

รายการเชื่อมโยง (linked list)

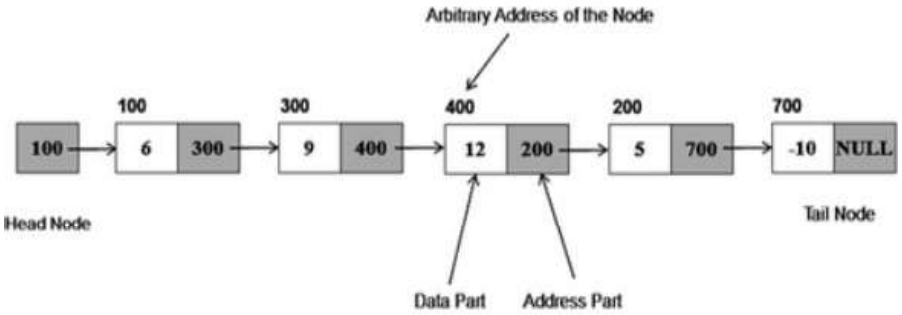
- เป็นวิธีการจัดการข้อมูลที่มีประสิทธิภาพมากกว่าการใช้อาร์เรย์ (array)
- รายการเชื่อมโยงจะใช้ประโยชน์จากพอยน์เตอร์ในการจัดเก็บข้อมูลแบบ dynamic

ข้อดีของการใช้รายการเชื่อมโยง

1. ไม่จำเป็นต้องกำหนดค่าเริ่มต้น
2. สามารถเพิ่มและลบข้อมูลได้อย่างรวดเร็ว



1D- Array



Single
Linked List

โครงสร้างของรายการเชื่อมโยง

ประกอบไปด้วยรายการโหนด (node) ซึ่งแต่ละโหนดจะมี 2 ส่วน คือ

1. info (เก็บข้อมูล)
2. next (เก็บแอดเดรสของโหนดถัดไป) ใช้ตัวแปรพอยน์เตอร์

เพื่อที่จะแสดงสถานะของโหนดสุดท้ายในรายการเชื่อมโยง next จะเก็บค่าพิเศษซึ่งเรียกว่า NULL

โครงสร้างของรายการเชื่อมโยง

ประกอบไปด้วยรายการโหนด (node) ซึ่งแต่ละโหนดจะมี 2 ส่วน คือ

1. info (เก็บข้อมูล)
2. next (เก็บแอดเดรสของโหนดถัดไป) ใช้ตัวแปรพอยน์เตอร์


```
struct  node {
        int info;
        struct node *next;
};
```

sizeof()

- คืนค่าขนาดของตัวแปร หรือ ชนิดของข้อมูล ในหน่วยไบต์

```
char i; char *p; char a[5];
printf("%d ", sizeof(char));
printf("%d ", sizeof(i));
printf("%d ", sizeof(p));
printf("%d ", sizeof(a));
```

malloc()

- void *malloc(size_t size);
- จัดสรรหน่วยความจำตามขนาดที่กำหนด โดยคืนค่าตัวชี้ตำแหน่งของไบต์แรก

```
int *a = (int *)malloc(5 * sizeof (int));
int *a = (int *) malloc(5 * sizeof (a[0]));
```

calloc()

- void *calloc(size_t num, size_t size);
- จัดสรรหน่วยความจำตามจำนวนและขนาดที่กำหนด โดยคืนค่าตัวชี้ตำแหน่งของไบต์แรก

```
int *a = calloc(5 , size of (a[0]));
```

realloc()

- void *realloc(void *ptr, size_t size);
- เปลี่ยนขนาดของหน่วยความจำที่ได้จัดสรรไว้ก่อนหน้านี้ของตัวชี้และขนาดที่กำหนด

```
int *a;
...
a = realloc(a, 5*sizeof(a[0]));
```

free()

- void free(void *ptr);
- คืนพื้นที่ในหน่วยความจำที่จัดสรรไว้ตามตัวชี้ที่กำหนด

```
int *a = malloc(5 * size of (a[0]));
...
free(a);
```

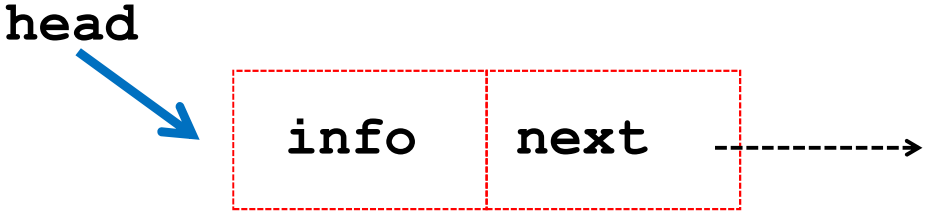
การประกาศรายการเชื่อมโยง

```
struct  node {
     int info;
     struct node *next;
};
```

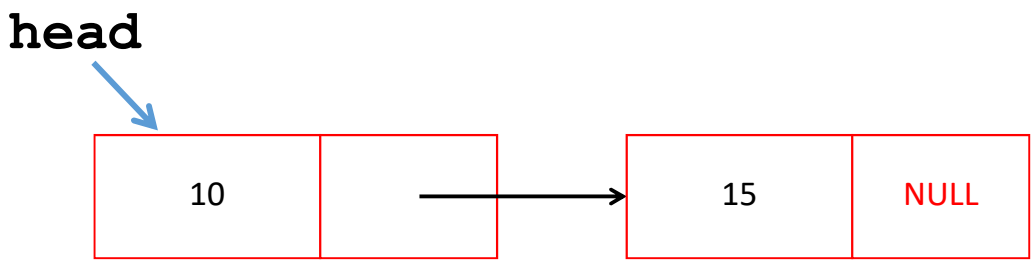
ประกาศโครงสร้างข้อมูล
สำหรับรายการเชื่อมโยง

```
struct node *first;
first = (struct node *) malloc( sizeof (struct node) );
first -> info = 10;        // เข้าถึงตัวแปร info
first -> next = NULL;      //  เข้าถึงพอยน์เตอร์ next
```

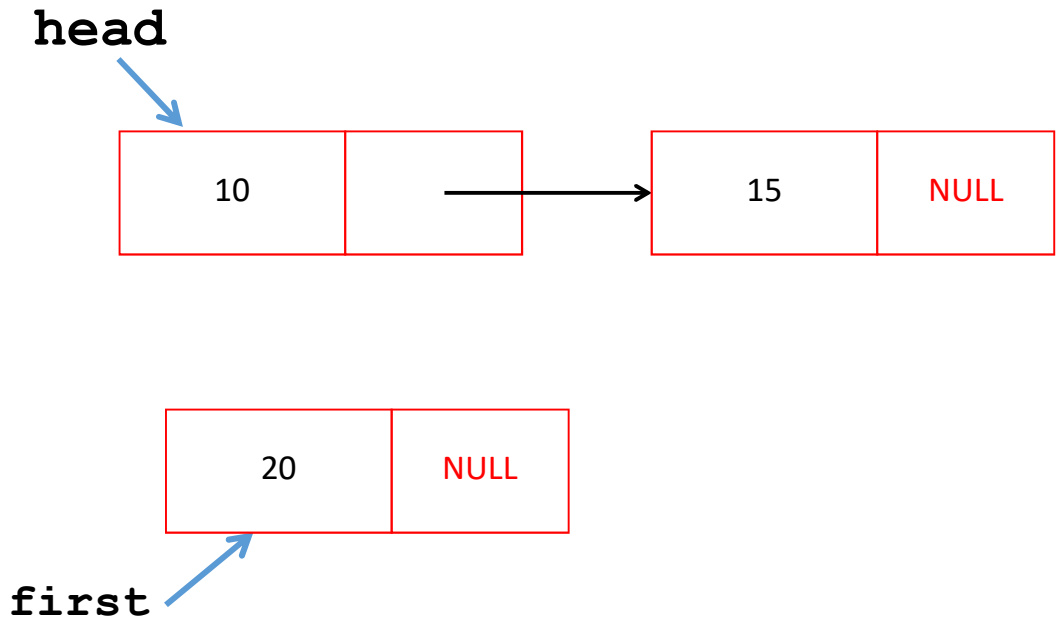

```
#include <stdio.h>
#include <stdlib.h>

struct  node {
     int info;
     struct node *next;
};
struct node *head = NULL;
```



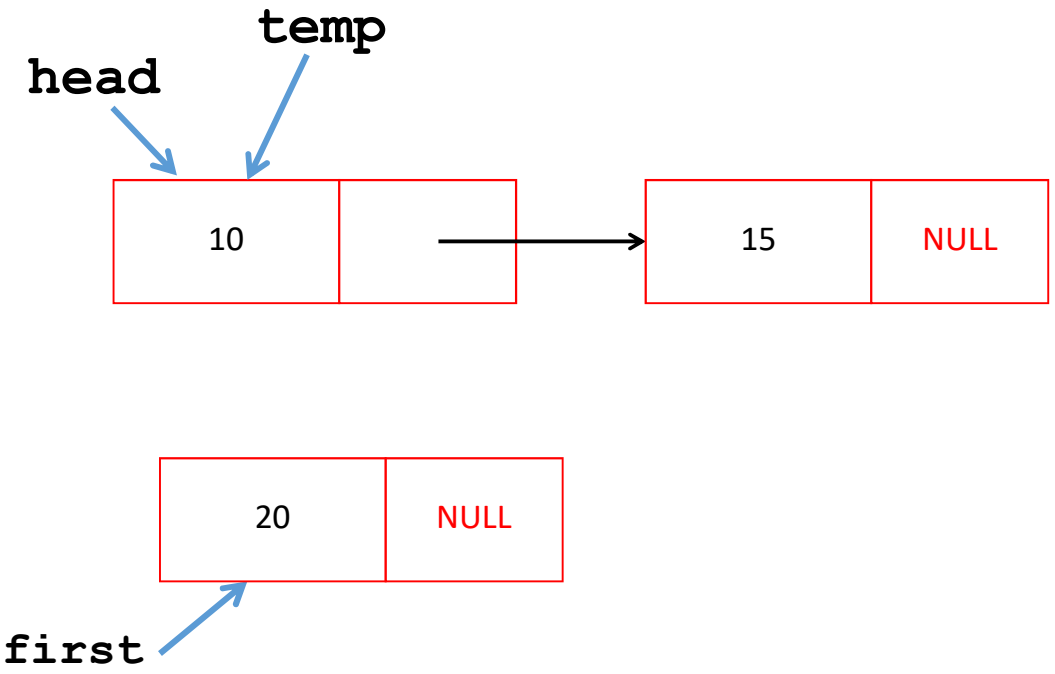
การเพิ่มสมาชิกต่อท้าย



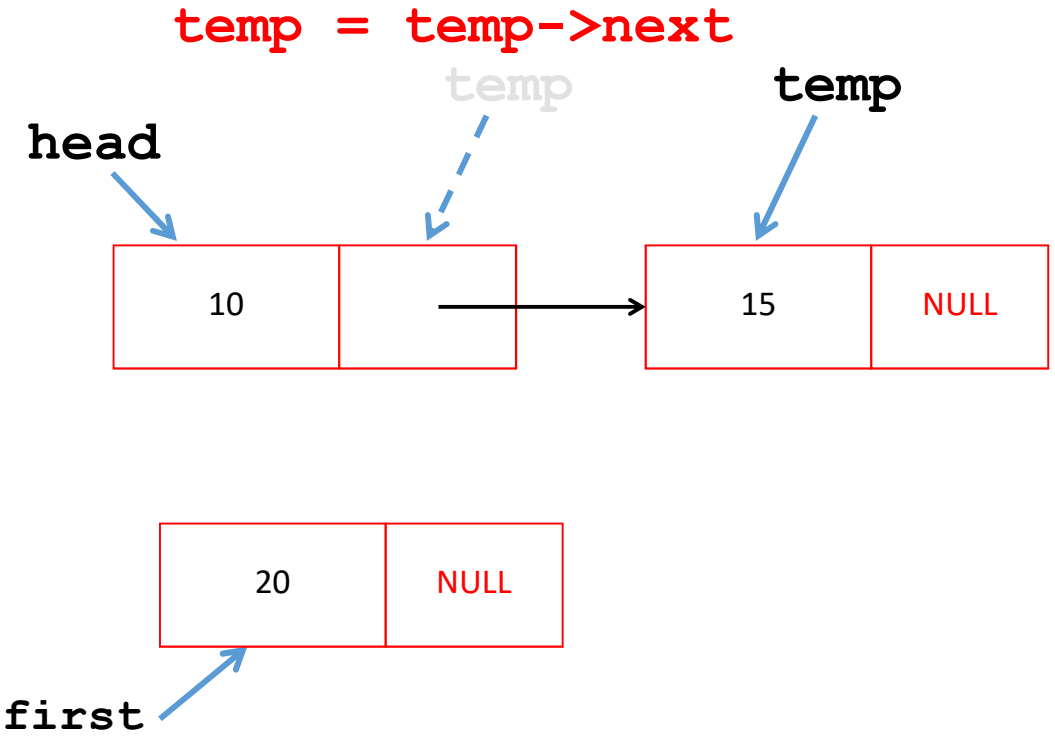
การเพิ่มสมาชิกต่อท้าย



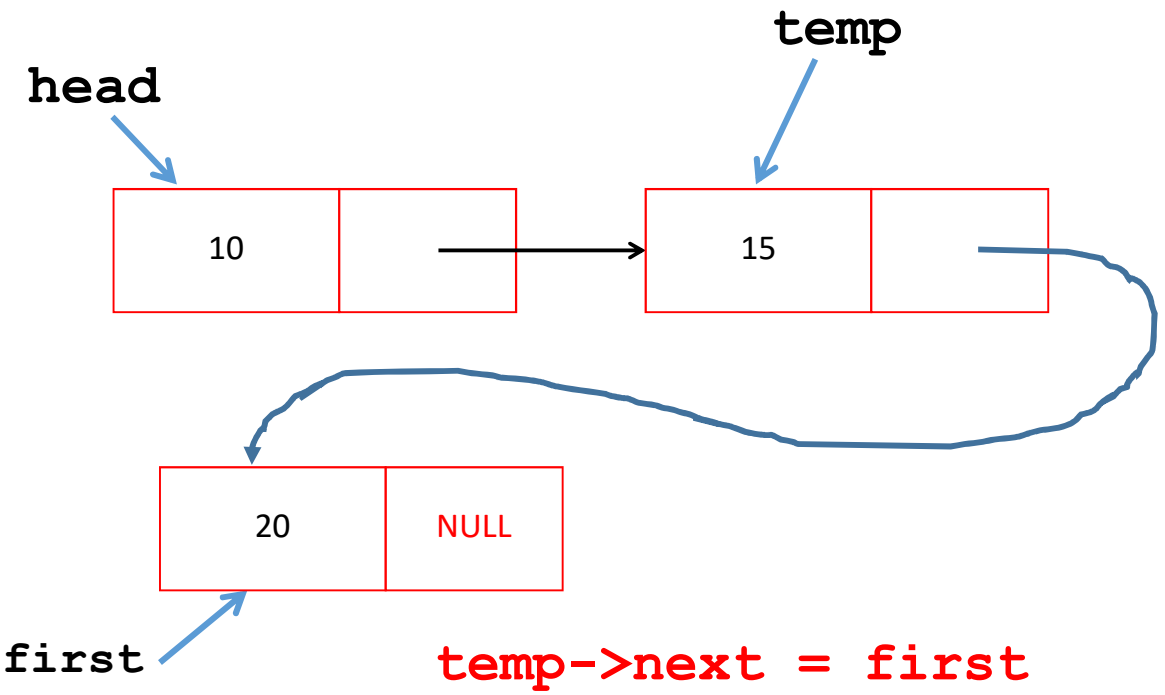
การเพิ่มสมาชิกต่อท้าย



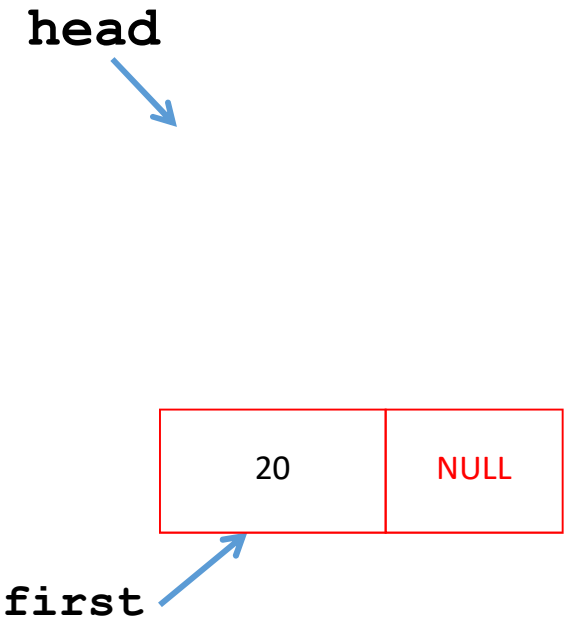
การเพิ่มสมาชิกต่อท้าย



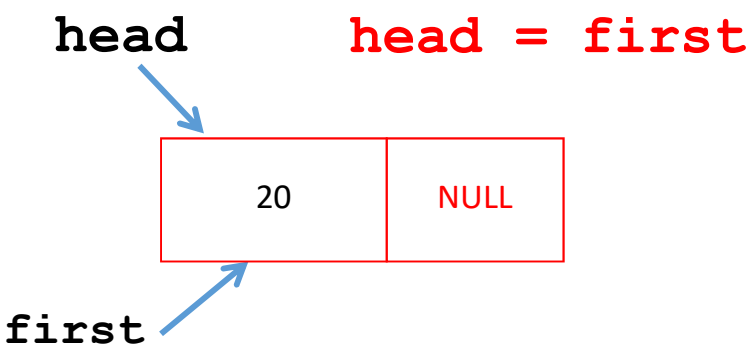
การเพิ่มสมาชิกต่อท้าย



การเพิ่มสมาชิกต่อท้าย



การเพิ่มสมาชิกต่อท้าย

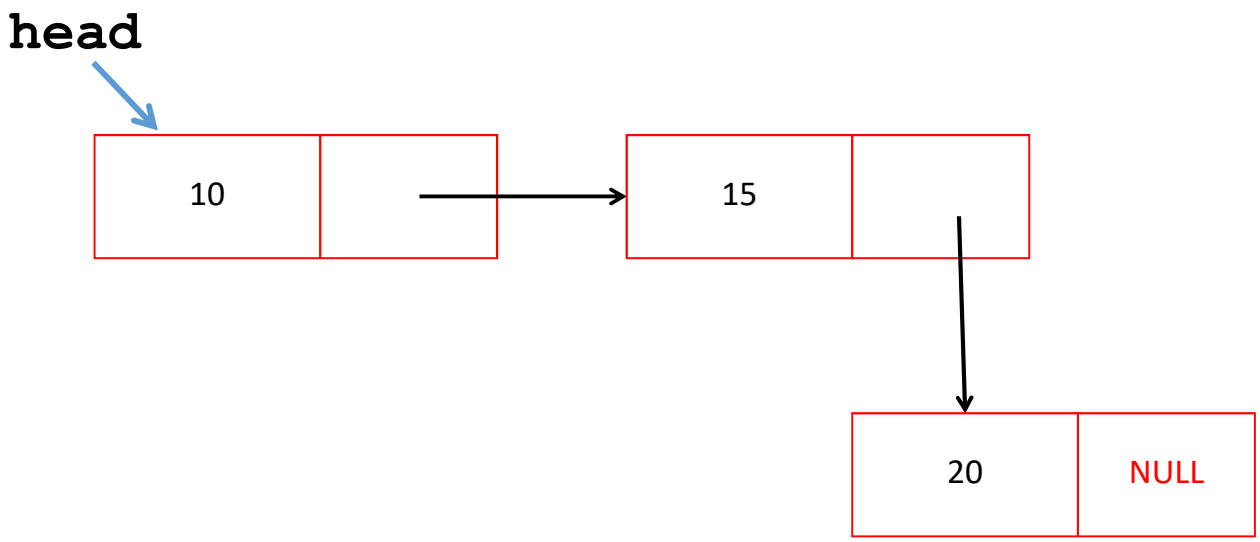


การเพิ่มสมาชิกต่อท้าย

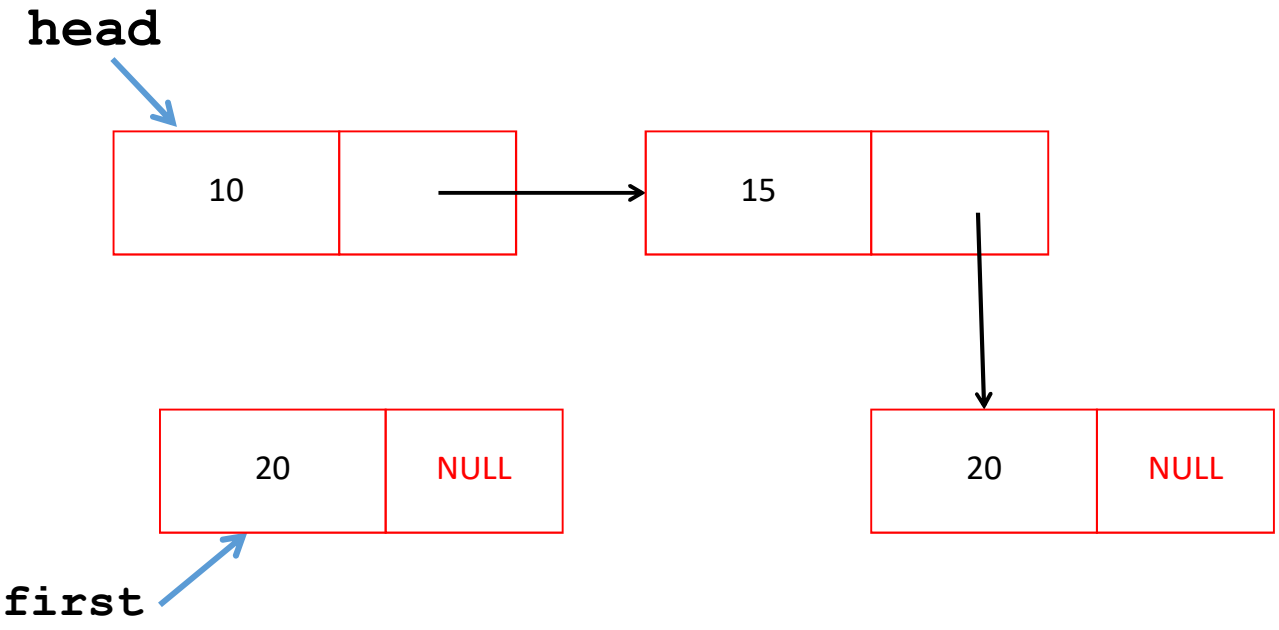
```
void add(int x)
 { struct node *first;
   first=(struct node *)malloc(sizeof(struct node));
   first -> info = x;
   first -> next = NULL;
   if(head != NULL)
    { struct node *temp = head;
      while(temp->next != NULL)
          temp = temp->next;
      temp->next = first;
    }
   else
      head = first;
 }
```

```
void main()
{ add(10);
  add(20);
  add(30);
}
```

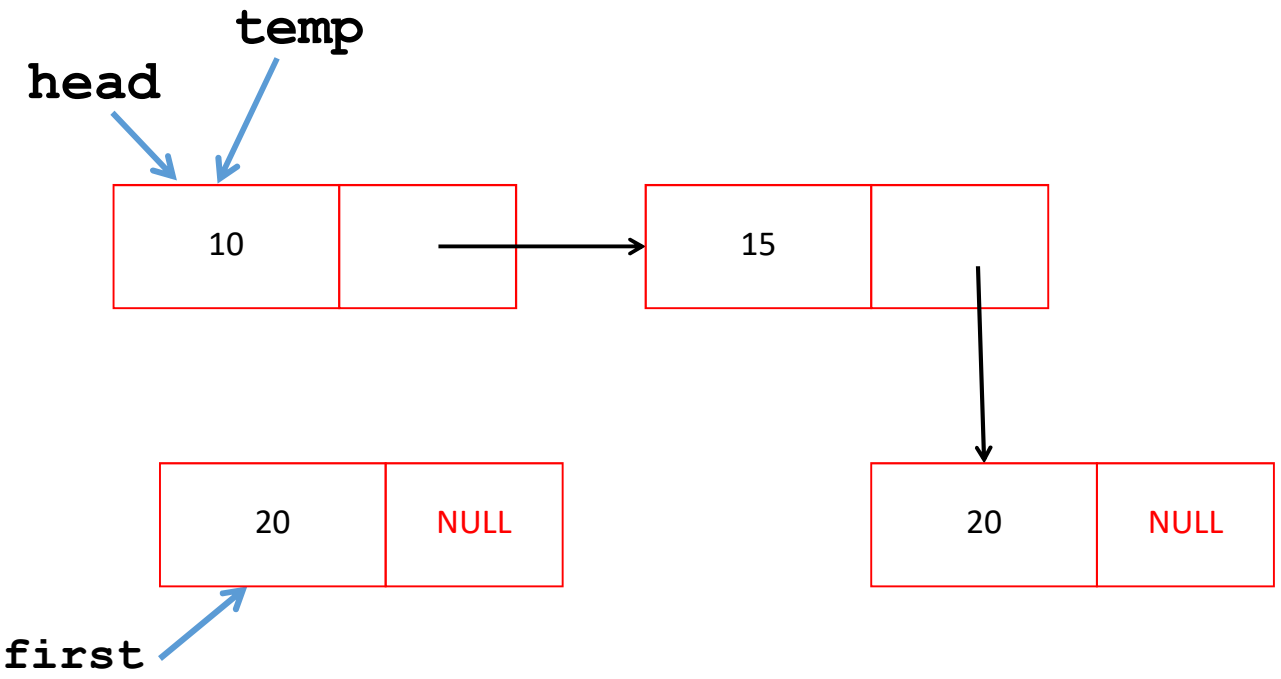
การแทรกข้อมูลระหว่างรายการ



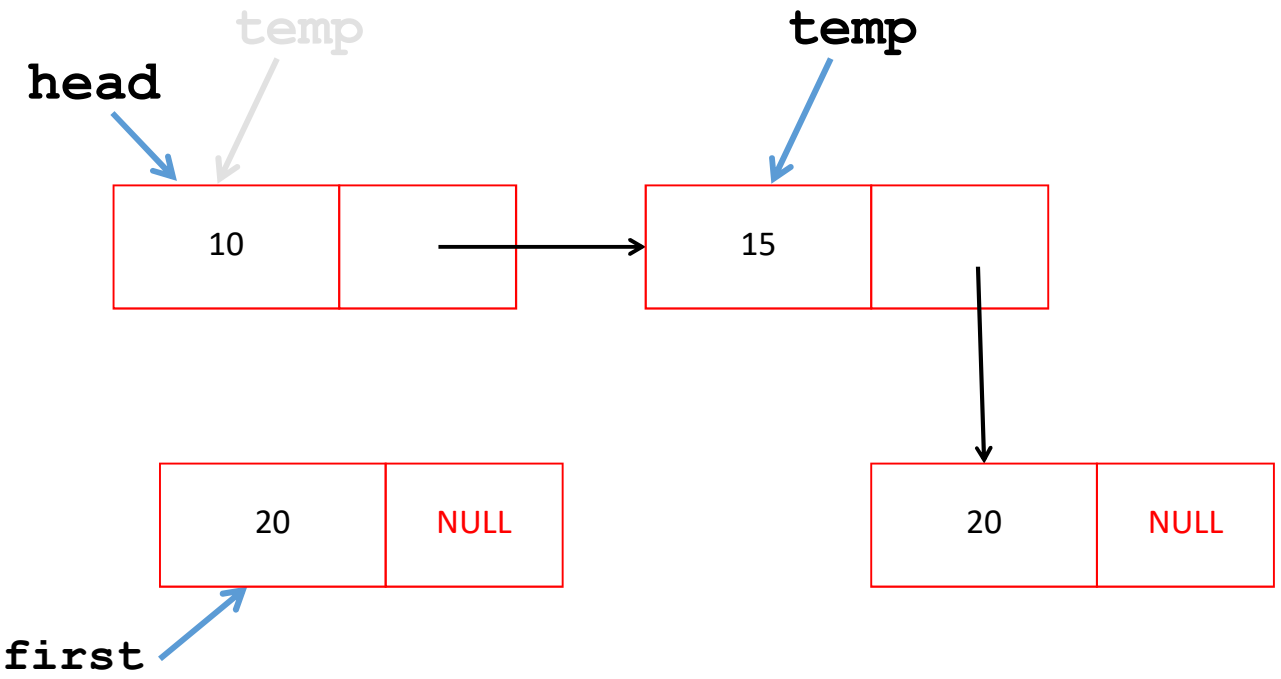
การแทรกข้อมูลระหว่างรายการ



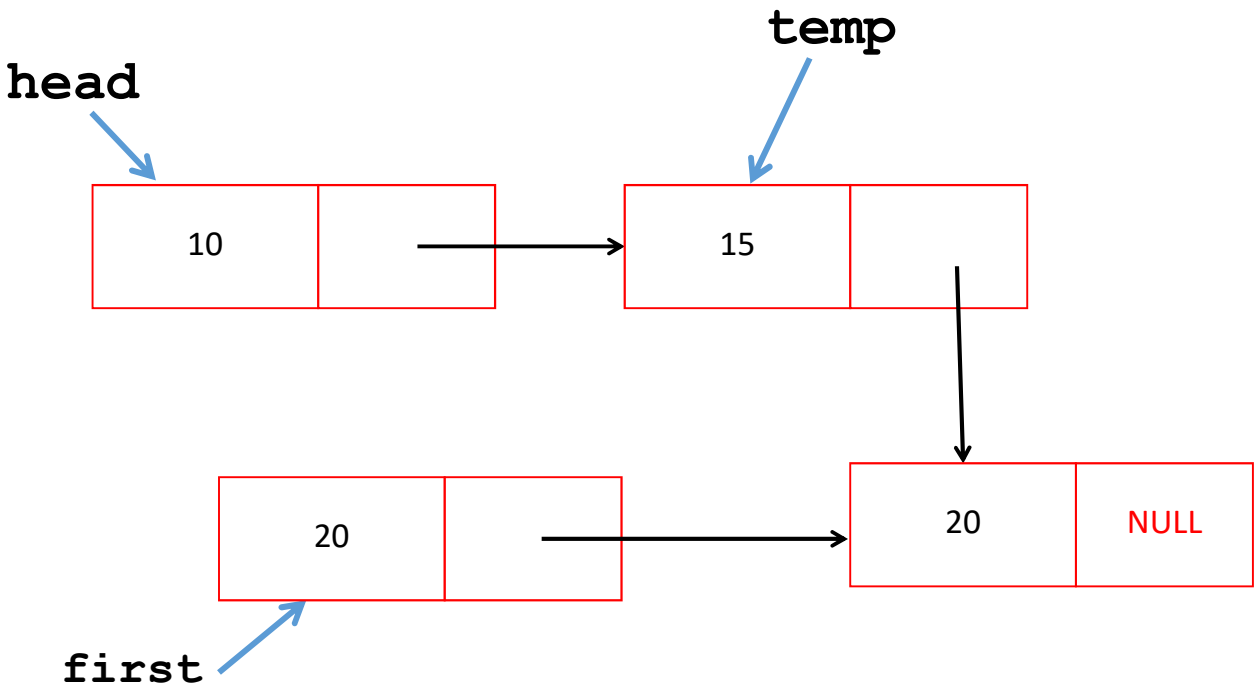
การแทรกข้อมูลระหว่างรายการ c == 0



การแทรกข้อมูลระหว่างรายการ c == 1

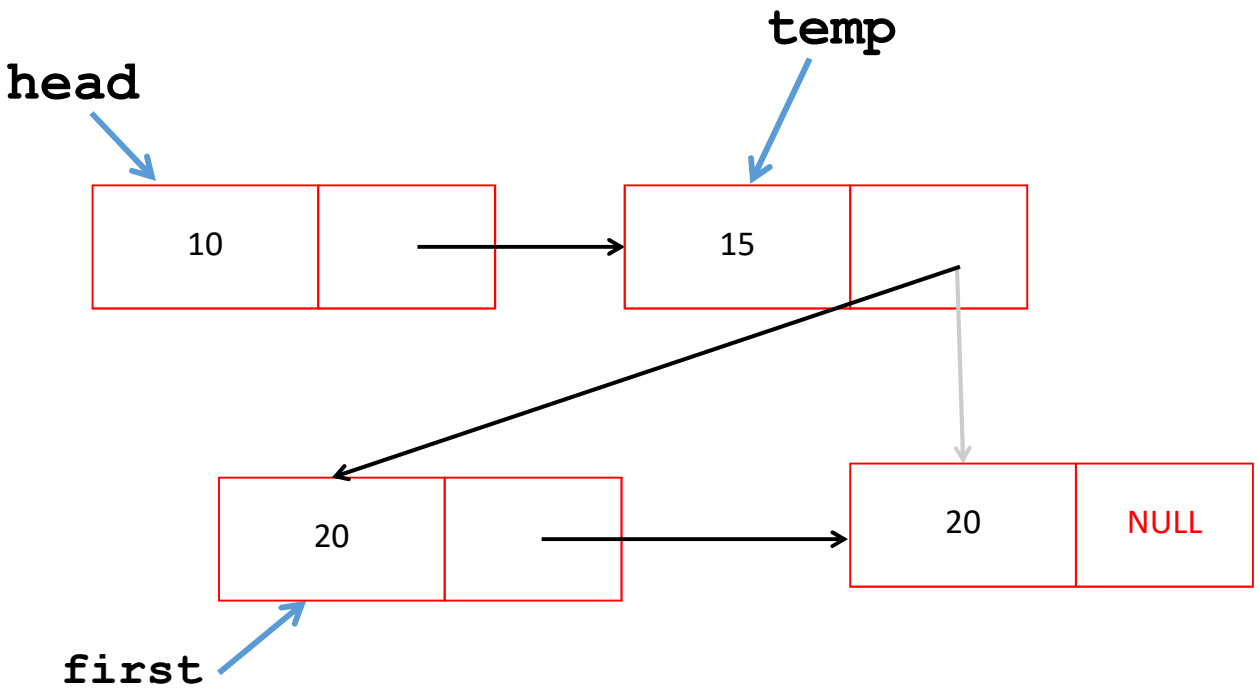


การแทรกข้อมูลระหว่างรายการ c == 1



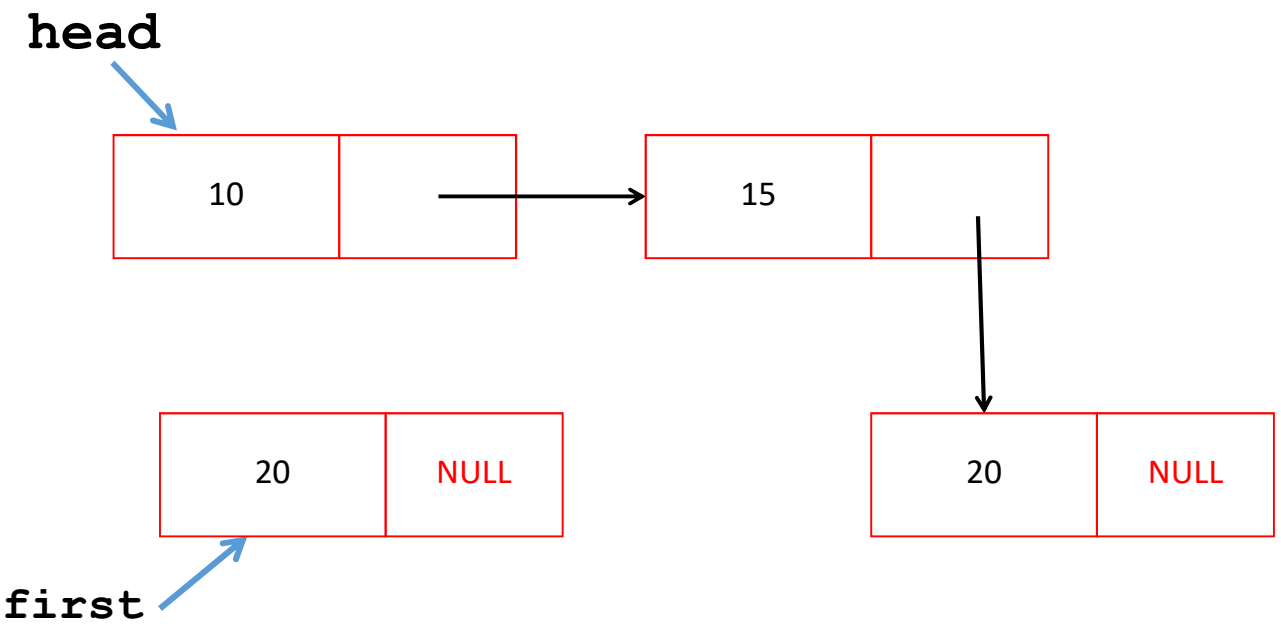
first->next == temp->next

การแทรกข้อมูลระหว่างรายการ c == 1

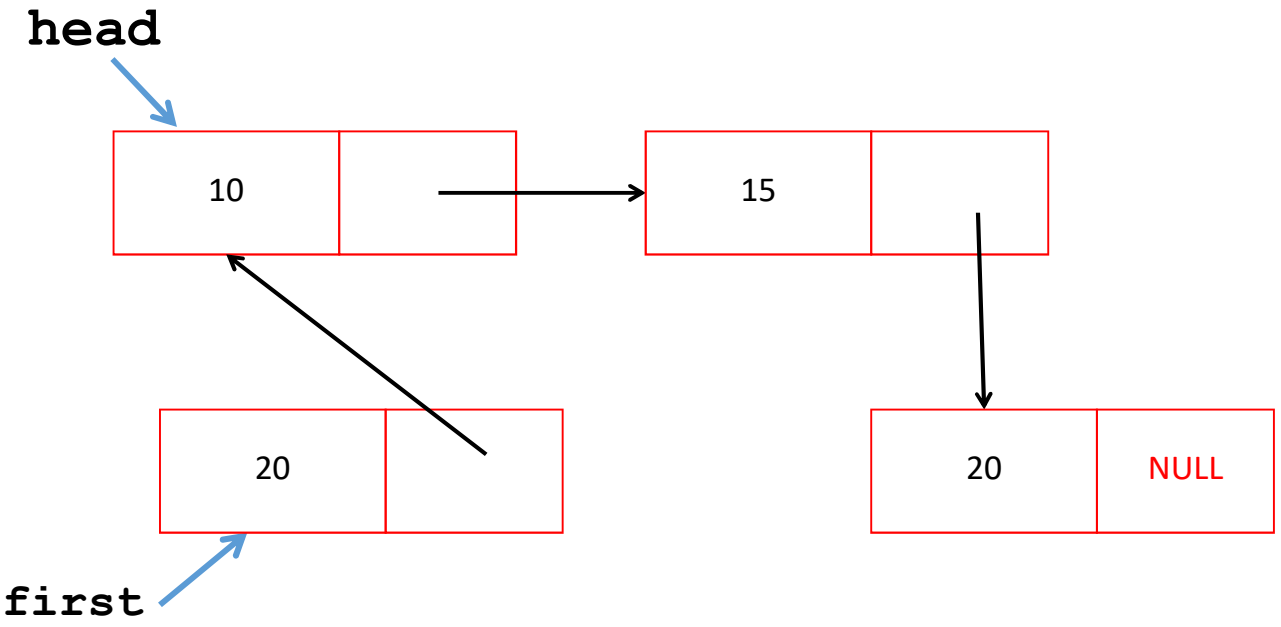


temp->next = first

การแทรกข้อมูลระหว่างรายการ

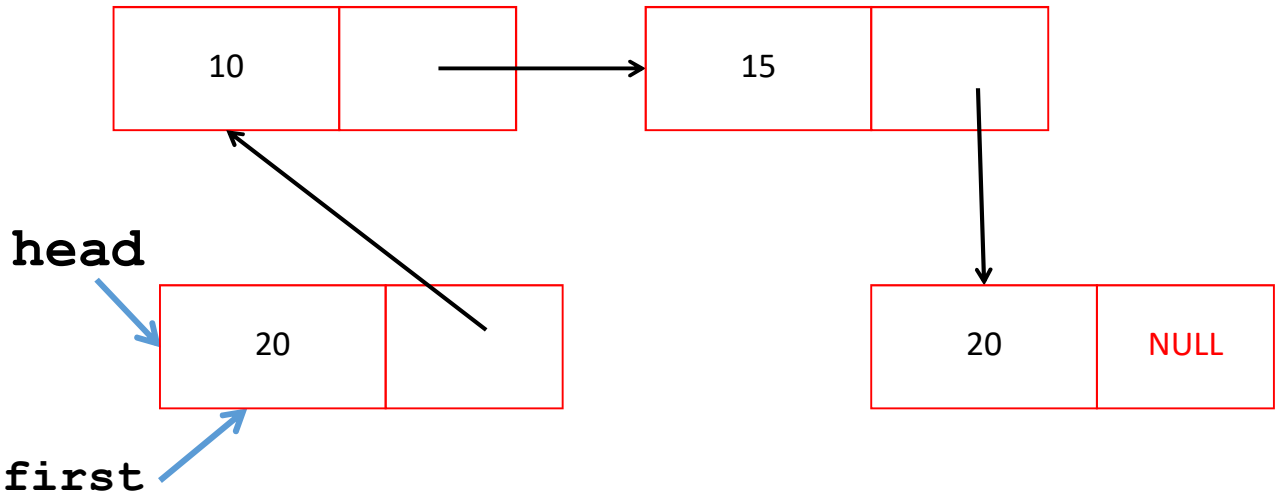


การแทรกข้อมูลระหว่างรายการ



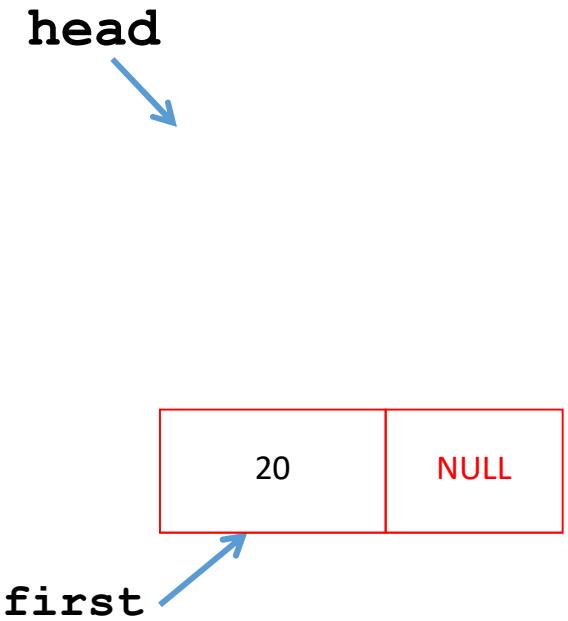

```
first->next = head
```

การแทรกข้อมูลระหว่างรายการ

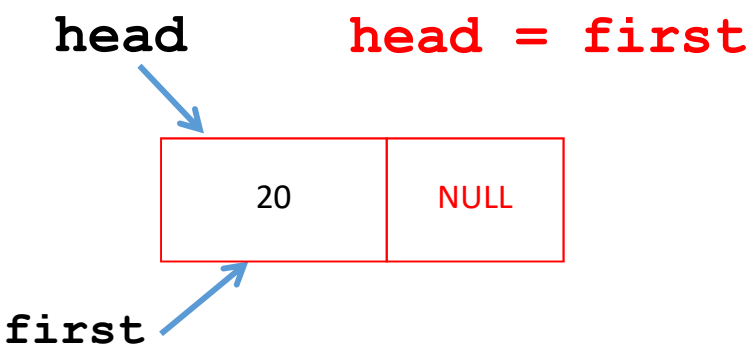


head = first

การเพิ่มสมาชิกต่อท้าย



การเพิ่มสมาชิกต่อท้าย

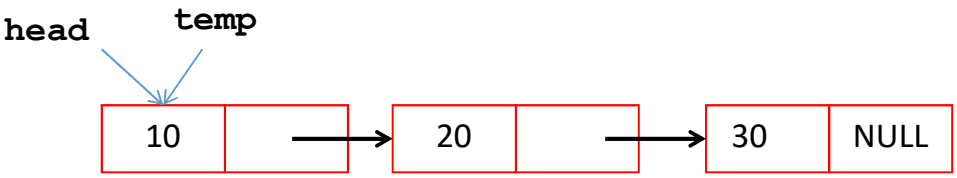



```
void insert(int pos, int x)
 { struct node *first;
   first = (struct node *) malloc( sizeof (struct node) );
   first -> info = x;
   first -> next = NULL;
  if(head != NULL)
    {  struct *temp = head;
       int c;
          if(pos == 0){
               first->next = head;
               head = first;
          } else {
         for(c=0; c<pos-1 ;c++)
            temp = temp->next;
        first->next = temp->next;
        temp->next = first;
      }
    } else  head = first;
 }
```

การท่องรายการเชื่อมโยง

- เริ่มจากพอยน์เตอร์ head จากนั้นก็ย้ายไปชี้ที่โหนดแต่ละโหนดจนกระทั่งพบว่าเป็นโหนดสุดท้าย

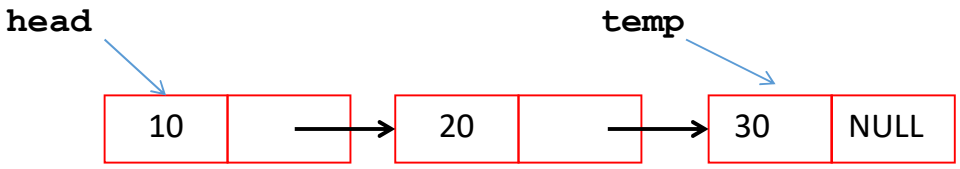
```
void printList()
{ struct node *temp;

  temp = head;        // เริ่มจาก head

  while(temp->next != NULL)
   {  printf("%d ", temp->info);
      temp = temp->next;
   }

}
```



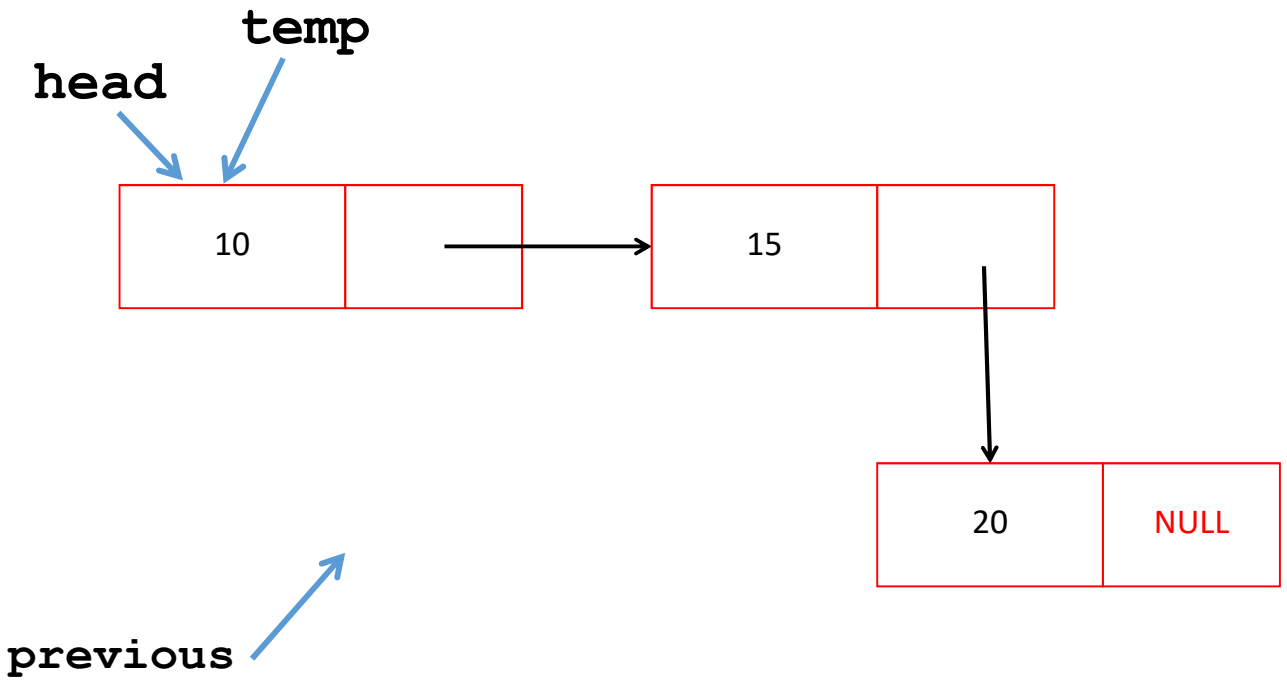
การค้นหาข้อมูลในรายการเชื่อมโยง

```
struct node* find(int key)
{  struct node* temp = head;
  if(head == NULL)
      return NULL;

 while(temp->info != key) {
     if(temp->next == NULL)
         return NULL;
     else
        temp = temp->next;
   }
 return temp;
}
```

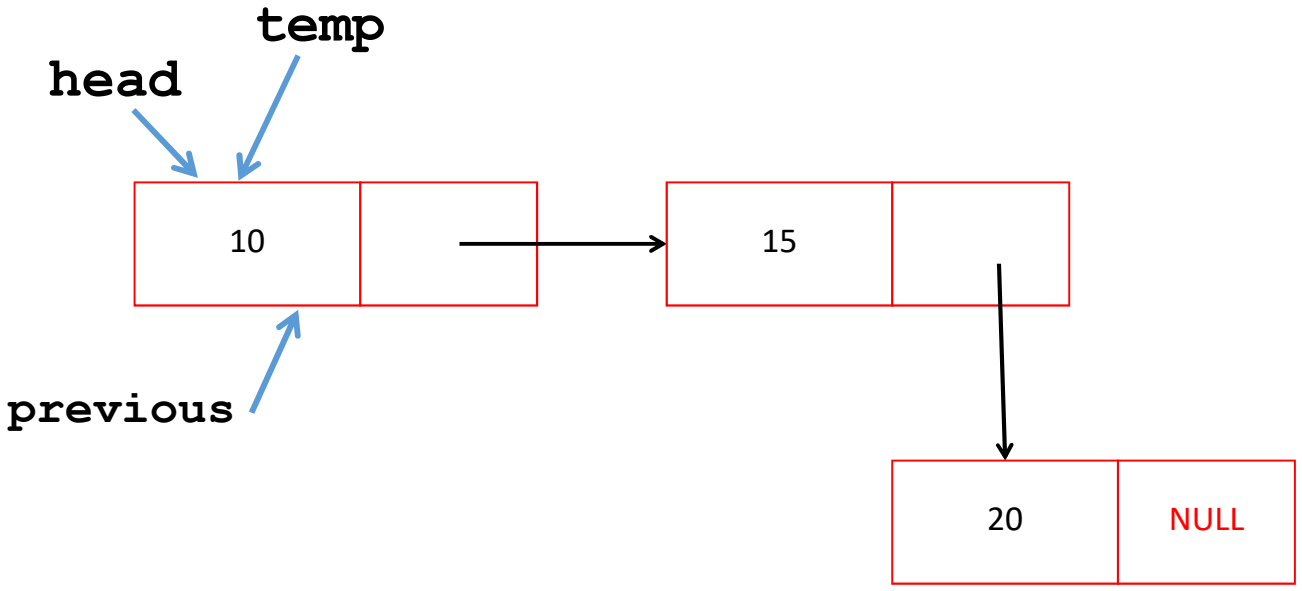
- เริ่มจากพอยน์เตอร์ head จากนั้นก็ย้าย temp ไปชี้ที่โหนดเพื่อเทียบว่า info มีค่าเท่ากับ key หรือไม่
- ทำจนกระทั่งพบข้อมูลที่ค้นหาหรือไม่ก็เป็นโหนดสุดท้ายในรายการ
- เช่น find(30)



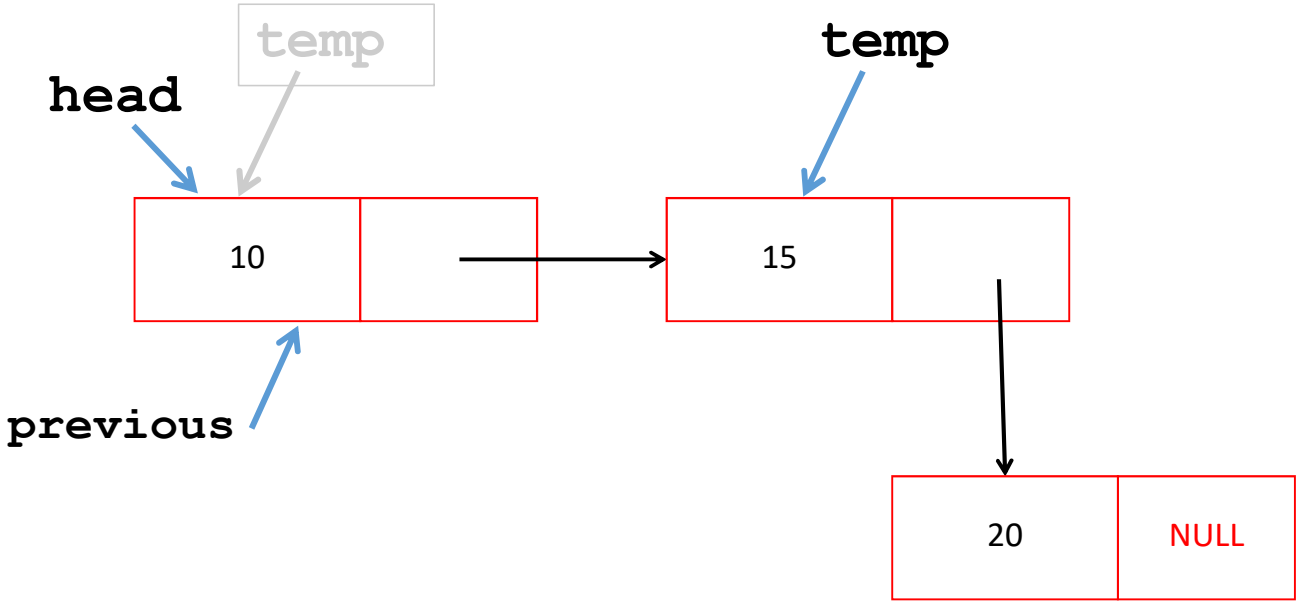
การลบข้อมูลในรายการเชื่อมโยง



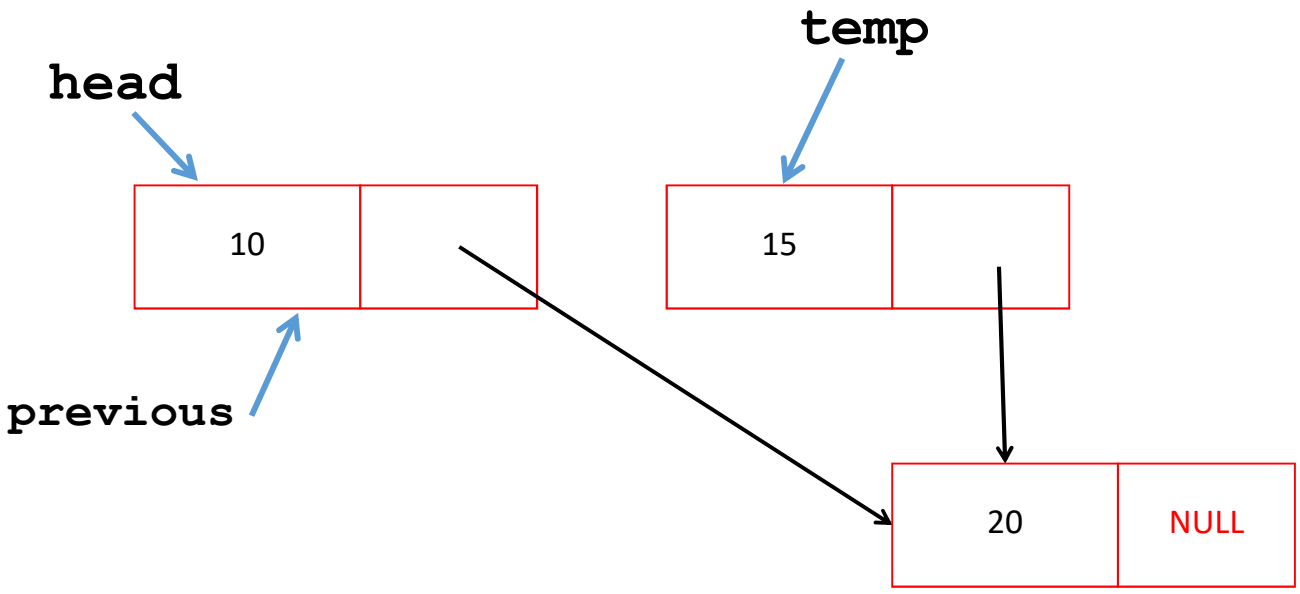
การลบข้อมูลในรายการเชื่อมโยง



การลบข้อมูลในรายการเชื่อมโยง

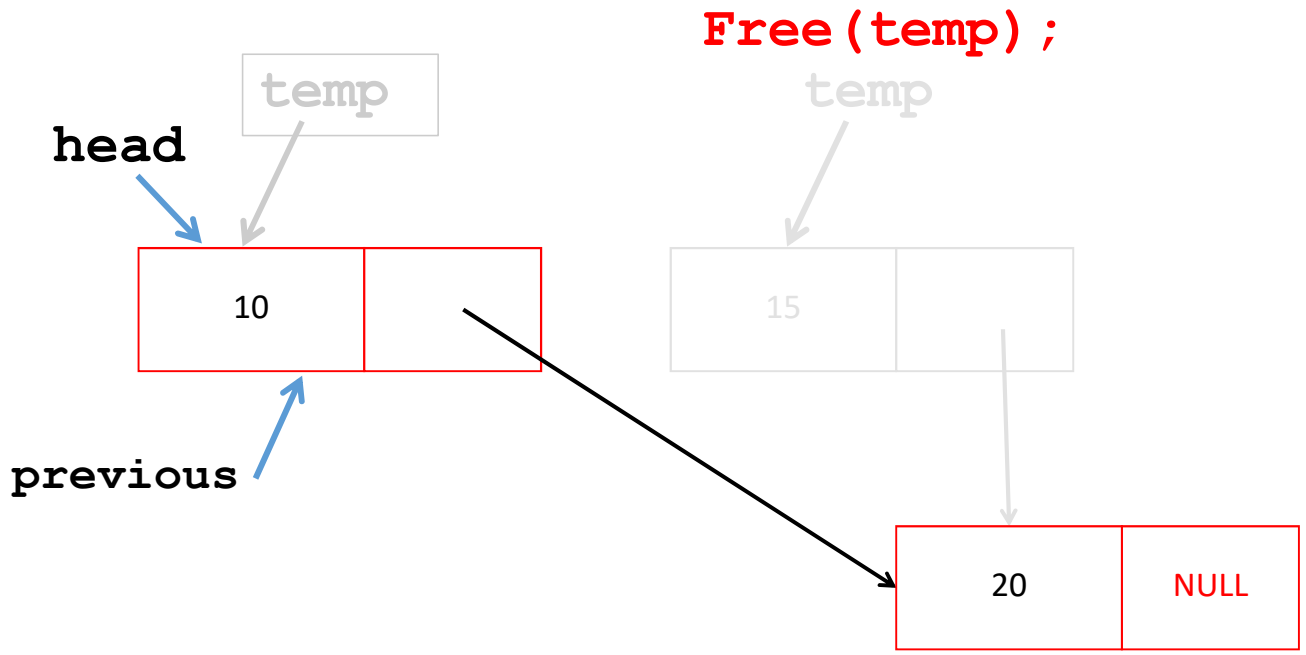


การลบข้อมูลในรายการเชื่อมโยง



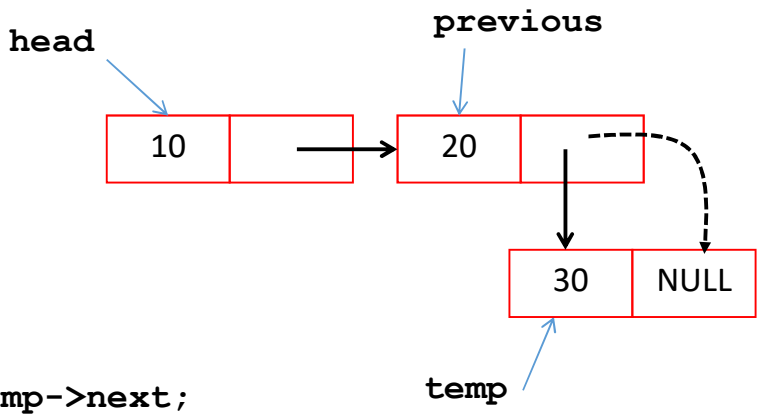

```
previous->next = temp->next;
```

การลบข้อมูลในรายการเชื่อมโยง



การลบข้อมูลใน
รายการเชื่อมโยง

```
struct node* delete(int key)
{ struct node *temp = head;
  struct node *previous = NULL;

  if(head == NULL)
      return NULL;

   while(temp->info != key)   {
     if(temp->next == NULL)
         return NULL;
     else {
      previous = temp;
      temp = temp->next;
      }
   }
   if(temp == head)
      head = head->next;
   else
     { previous->next = temp->next;
       free(temp);
     }
return temp;
}
```



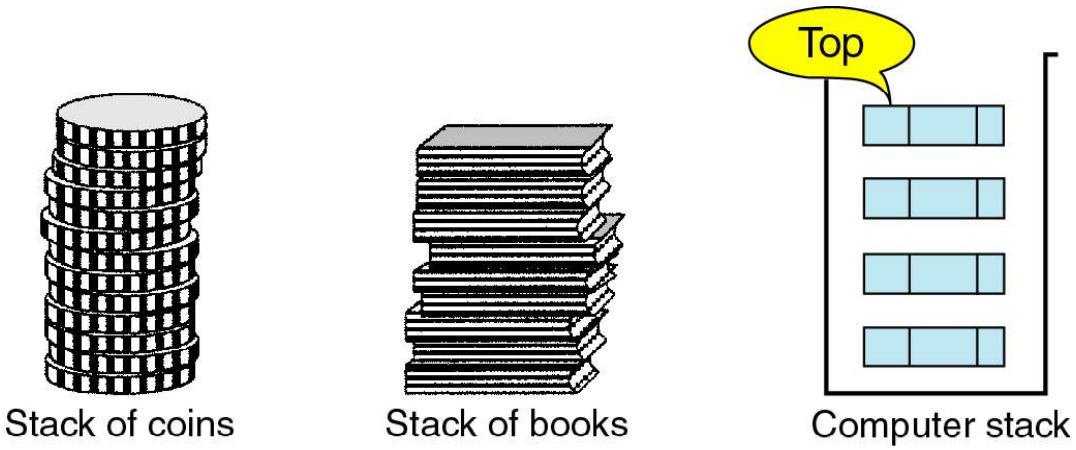
Stack

ที่มา : อ.เลาขวัญ งามประสิทธิ์ โรงเรียนมหิดลวิทยานุสรณ์

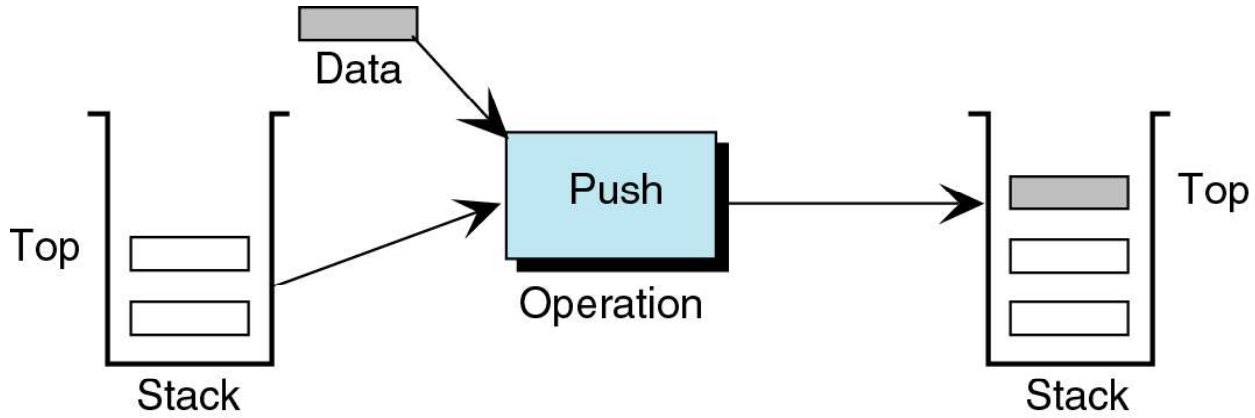
Stack

- **Stack เป็นโครงสร้างข้อมูลแบบ LIFO (Last-In, First-Out)**
- **Operations พื้นฐานของ Stack ได้แก่**
  - **การนำข้อมูลเข้าสู่ Stack เรียกว่า Push**
  - **การนำข้อมูลออกจาก Stack เรียกว่า Pop**
  - **การเรียกใช้ข้อมูลจาก Stack เรียกว่า Top**
- **การสร้าง Stack**
  - **ใช้ Array แทน Stack**
  - **ใช้ Linked list แทน Stack**

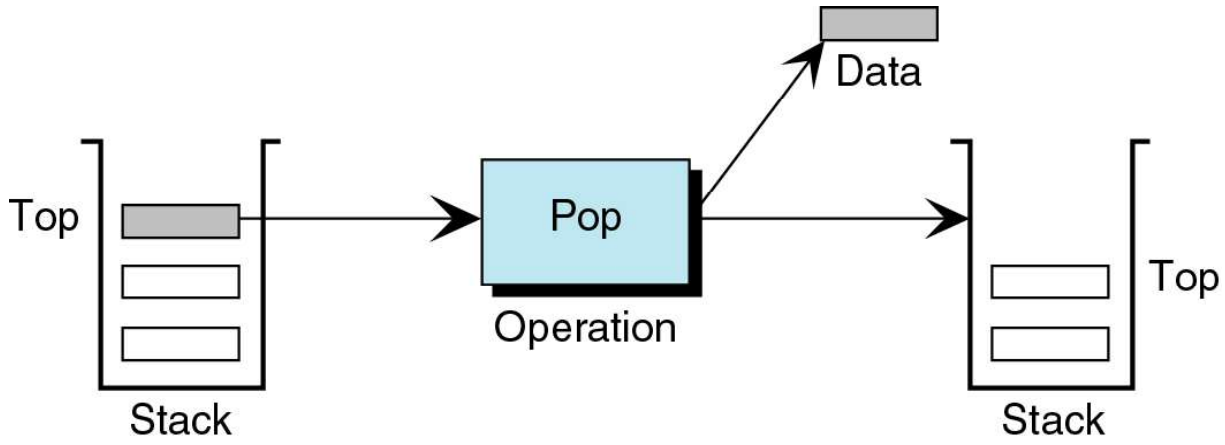
Stack



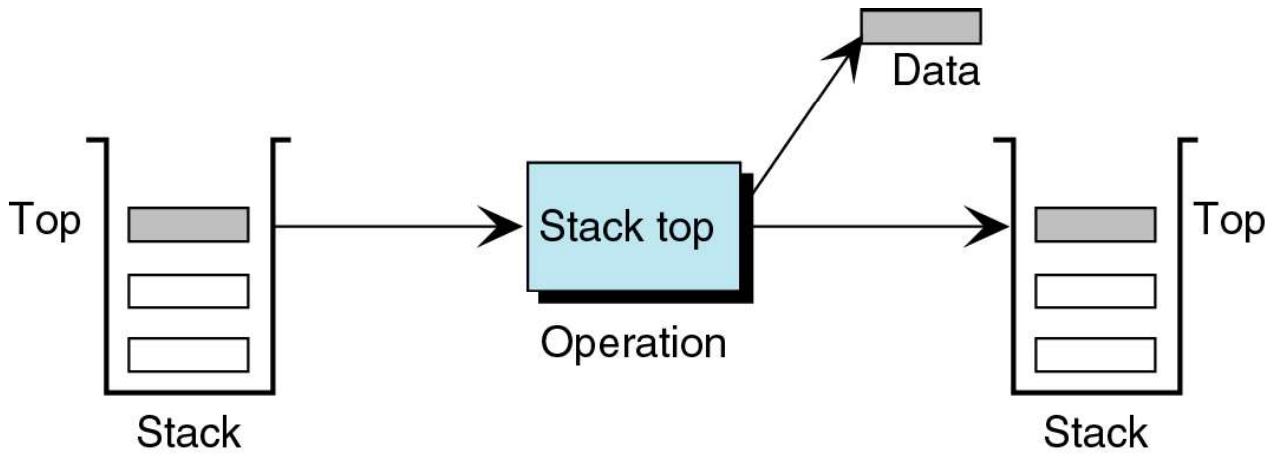
เพิ่มข้อมูลใน Stack: Push

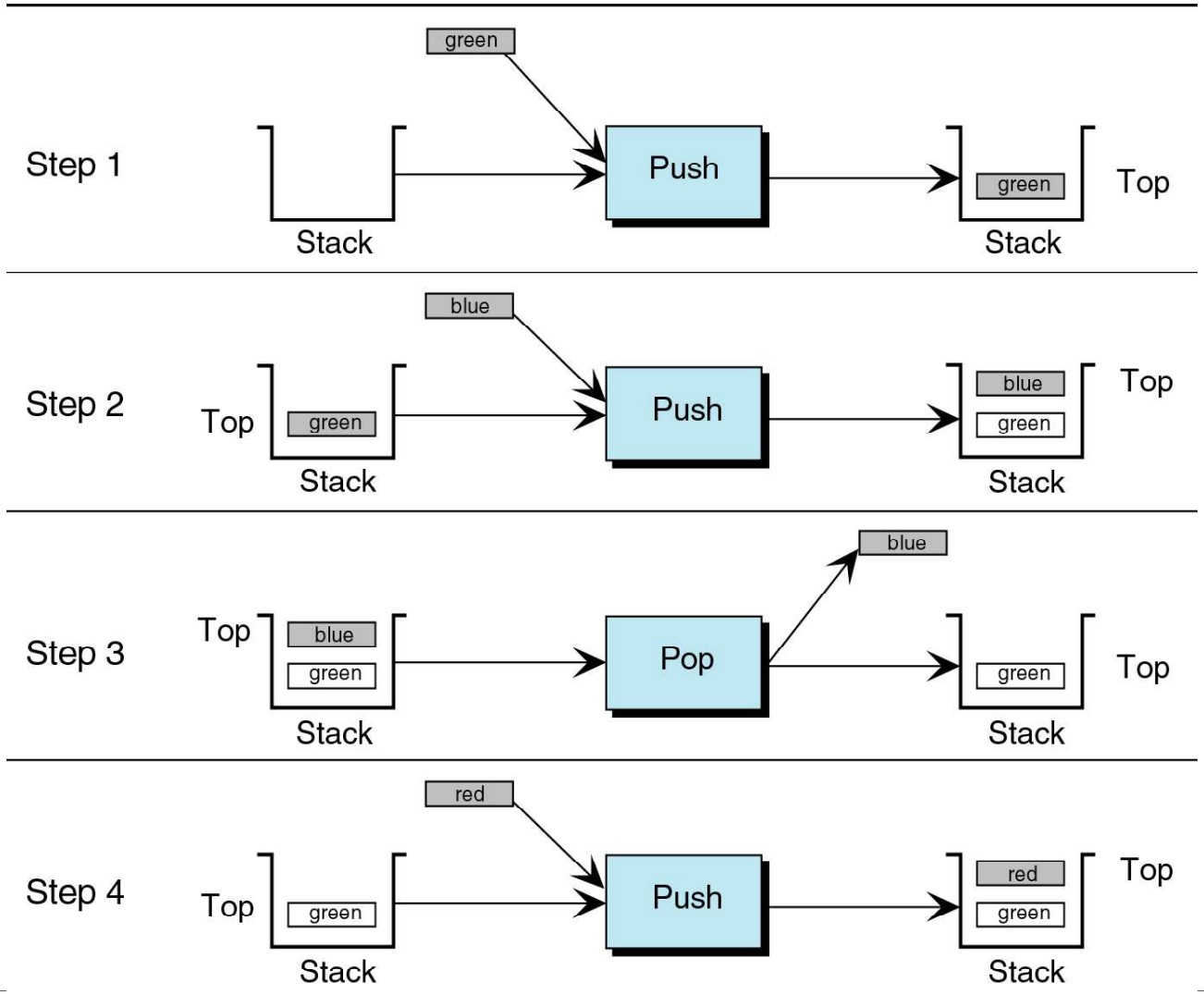


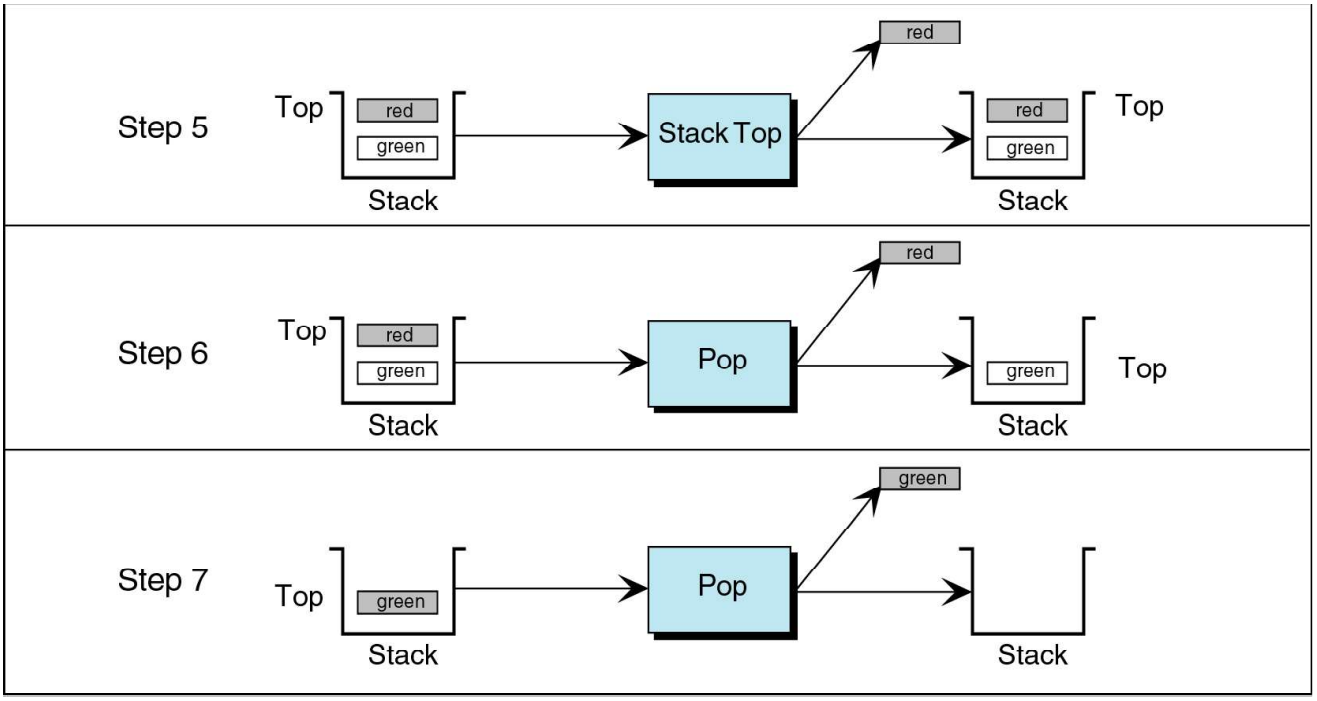
นำข้อมูลออกจาก Stack : Pop



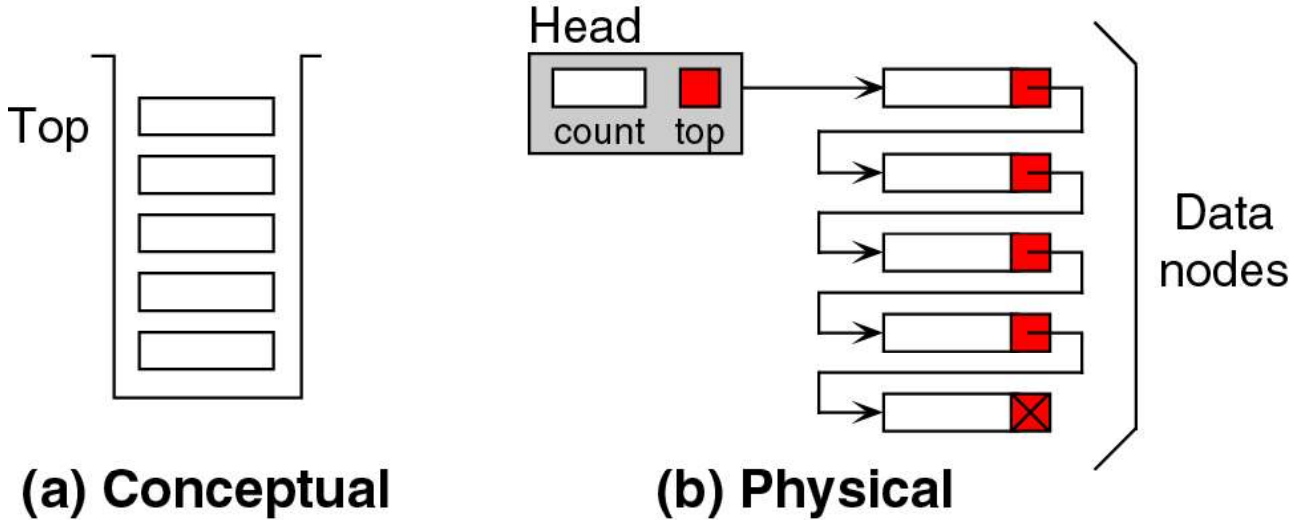
เรียกใช้ข้อมูลใน Stack: Top




Step 1
green
Push
Stack
green
Top
Stack
Step 2
blue
Top
green
Push
Stack
blue
green
Top
Stack
Step 3
Top
blue
green
Pop
blue
Stack
green
Top
Stack
Step 4
red
Top
green
Push
Stack
red
green
Top
Stack

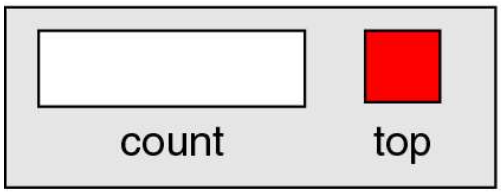

Step 5
Top
red
green
Stack Top
red
Stack
red
green
Top
Stack
Step 6
Top
red
green
Pop
red
Stack
green
Top
Stack
Step 7
Top
green
Pop
green
Stack
Stack

Linked list แทน Stack



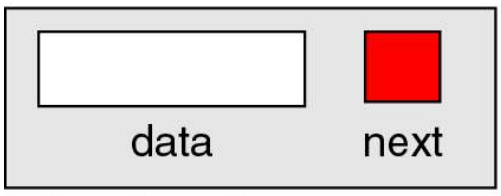
(a) Conceptual (b) Physical

Linked list แทน Stack



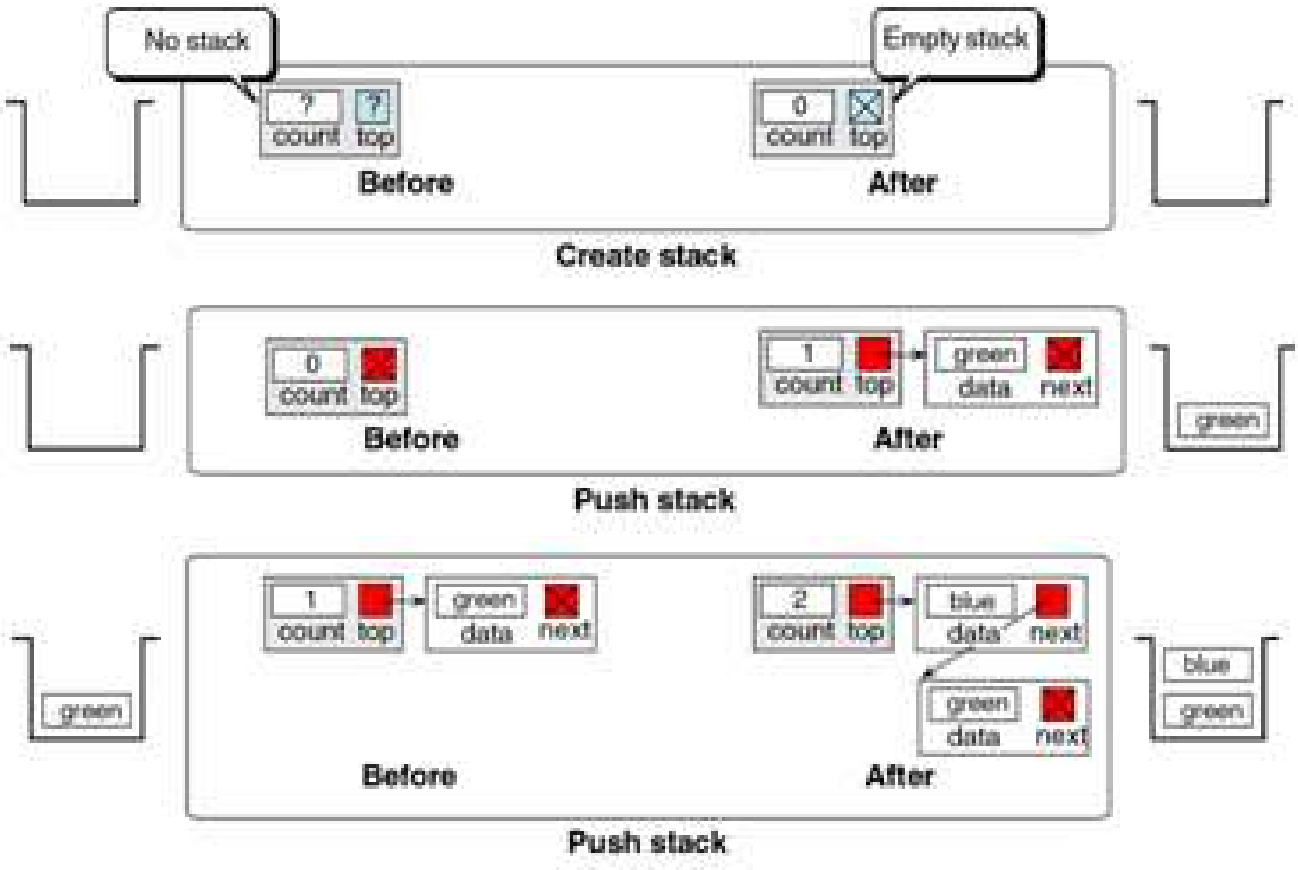
Stack head structure

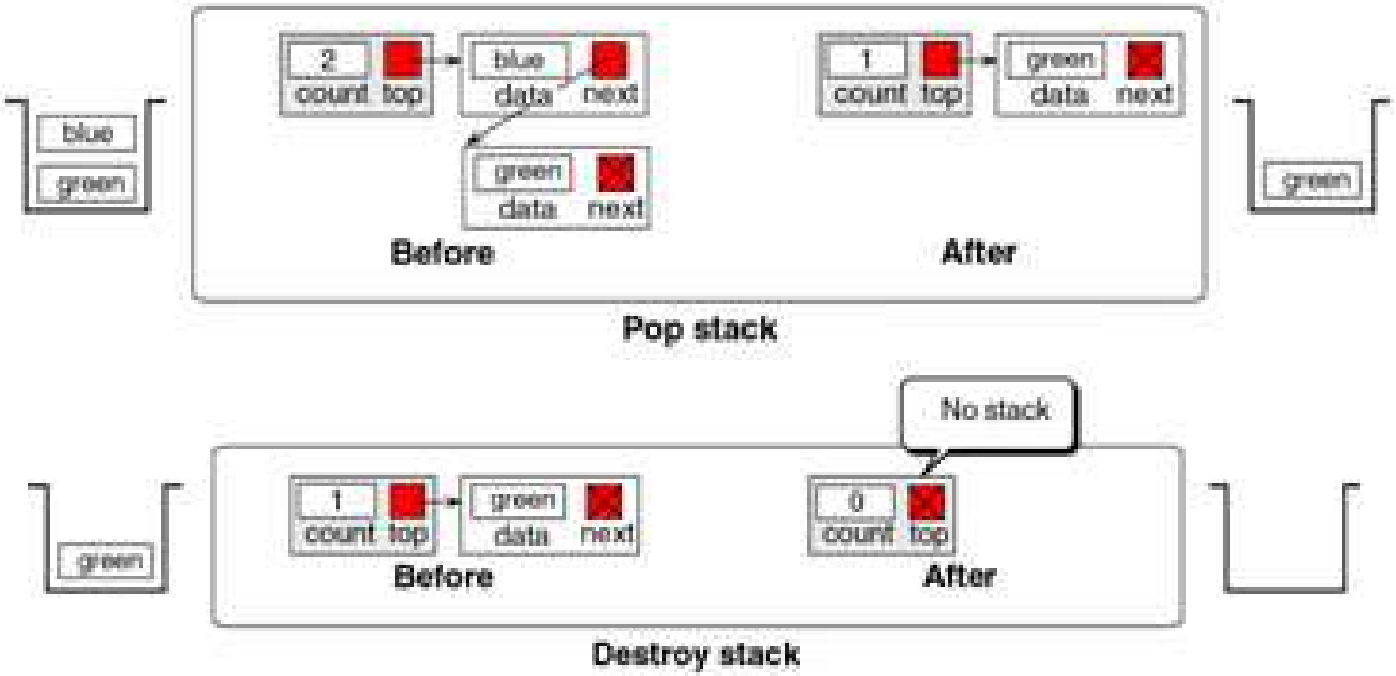
```
struct  stack {
    int count;
    struct node *top;
};
```



Stack node structure

```
struct  node {
    int data;
    struct node *next;
};
```


No stack
Empty stack
? count ? top
0 count top
Before
After
Create stack
0 count top
1 count top
green data next
green
Before
After
Push stack
1 count top
green data next
2 count top
blue data next
green data next
green
blue
green
Before
After
Push stack


2 count top
blue data next
green data next
1 count top
green data next
blue
green
green
Before
After
Pop stack
No stack
1 count top
green data next
0 count top
green
Before
After
Destroy stack

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

struct  stack {
    int count;
    struct node *top;
};
struct stack *stk = NULL;

struct  node {
    int data;
    struct node *next;
};
```

```
void push(int val){
     if(stk == NULL){
       stk = (struct stack*)malloc
             (sizeof(struct stack));
       stk->count=0;
       stk->top = NULL;
     }
   struct node *temp = (struct node*)malloc
                       (sizeof(struct node));
   temp->data = val;
   temp->next = stk->top;
   stk->top=temp;
   stk->count++;
}
```

```
int pop(){
     if(stk == NULL||stk->count==0){
          return -1;
     }
     int val = stk->top->data;
     struct node *temp = stk->top;
     stk->top = temp->next;
     stk->count--;
     free(temp);
     return val;
}
```

Operations พื้นฐานของ Stack ที่สร้างด้วย Linked list

1. Create stack:	สร้าง stack head node
2. Push stack:	เพิ่มรายการใน stack
3. Pop stack:	ลบรายการใน stack
4. Stack top:	เรียกใช้รายการข้อมูลที่อยู่บนสุดของ stack
5. Empty stack:	ตรวจสอบว่า stack ว่างเปล่าหรือไม่
6. Full stack:	ตรวจสอบว่า stack เต็มหรือไม่
7. Stack count:	ส่งค่าจำนวนรายการใน stack
8. Destroy stack:	คืนหน่วยความจำของทุก node ใน stack ให้ระบบ

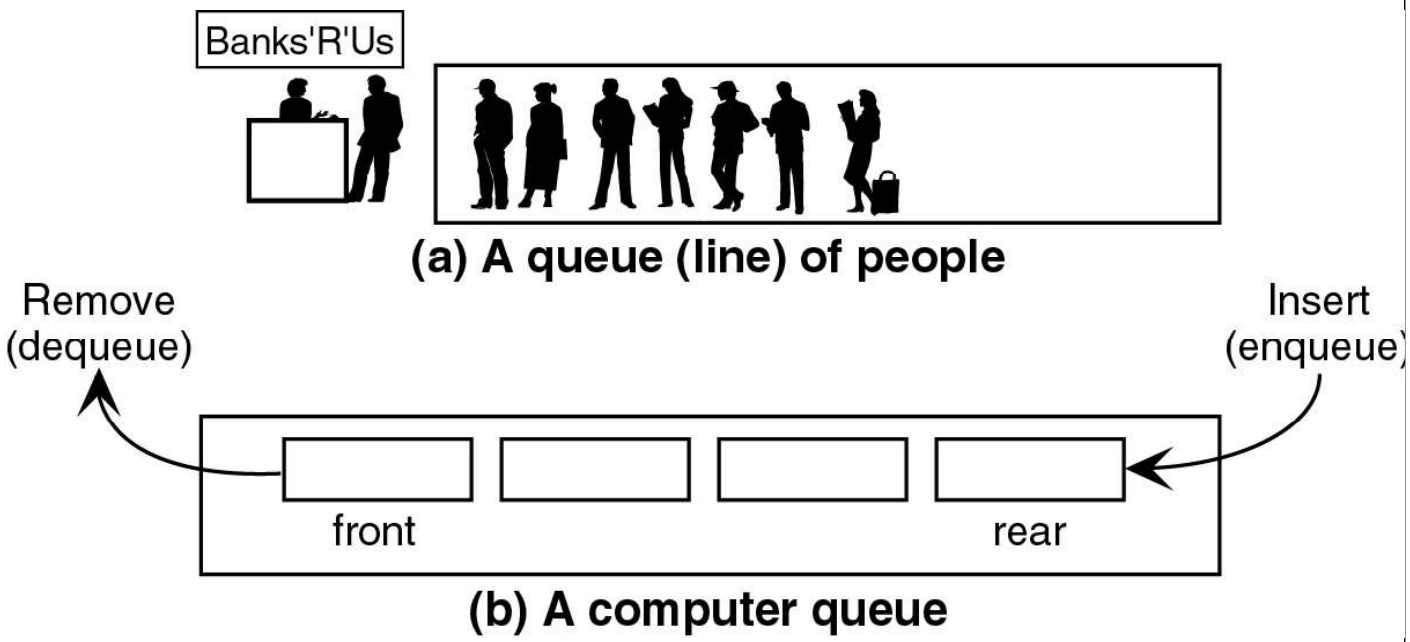
Queue

ที่มา : อ.เลาขวัญ งามประสิทธิ์ โรงเรียนมหิดลวิทยานุสรณ์

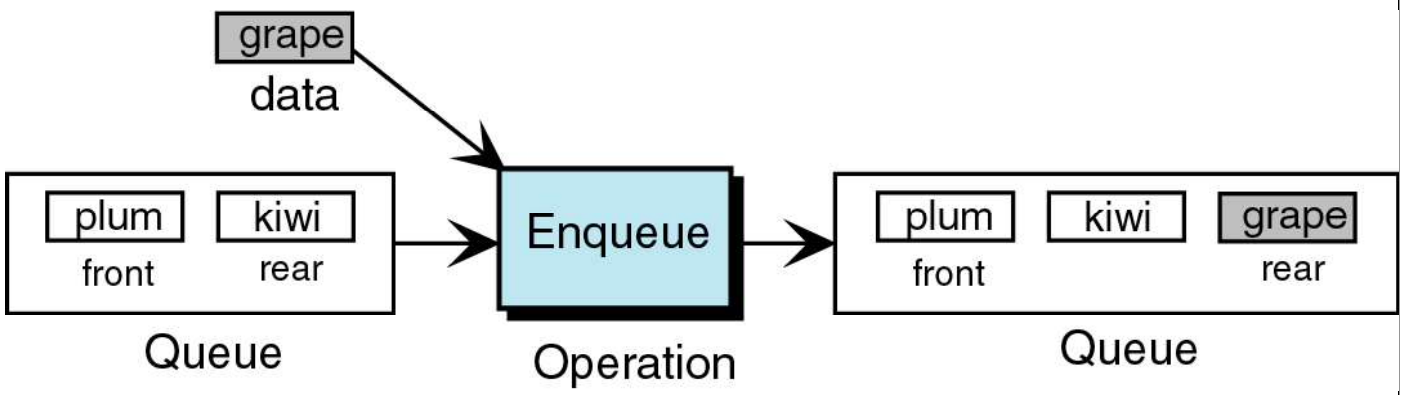
Queue

- Queue เป็นโครงสร้างข้อมูลแบบ FIFO (First-In, First-Out)
- Operations พื้นฐานของ Queue ได้แก่
  - การนำข้อมูลเข้าสู่ Queue เรียกว่า Enqueue
  - การนำข้อมูลออกจาก Queue เรียกว่า Dequeue
  - การเรียกใช้ข้อมูลจากหัวแถวของ Queue เรียกว่า Front
  - การเรียกใช้ข้อมูลจากท้ายแถวของ Queue เรียกว่า Rear
- การสร้าง Queue
  - ใช้ Array แทน queue
  - ใช้ Linked list แทน queue

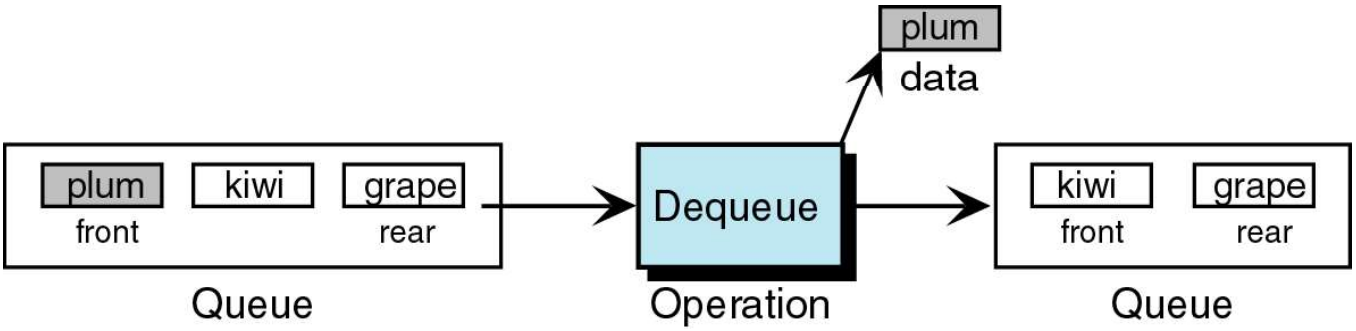
The Queue concept



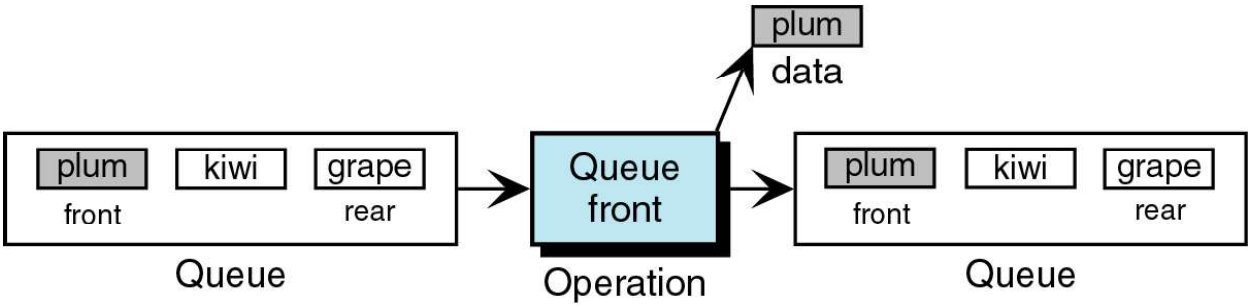
Operation Enqueue



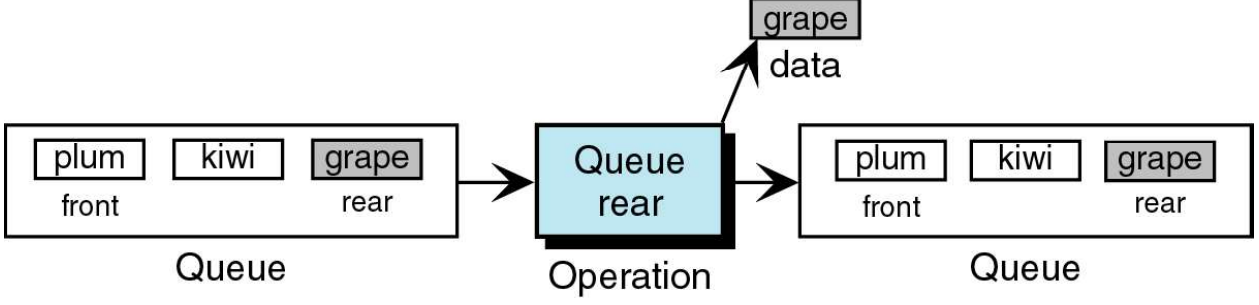
Operation Dequeue



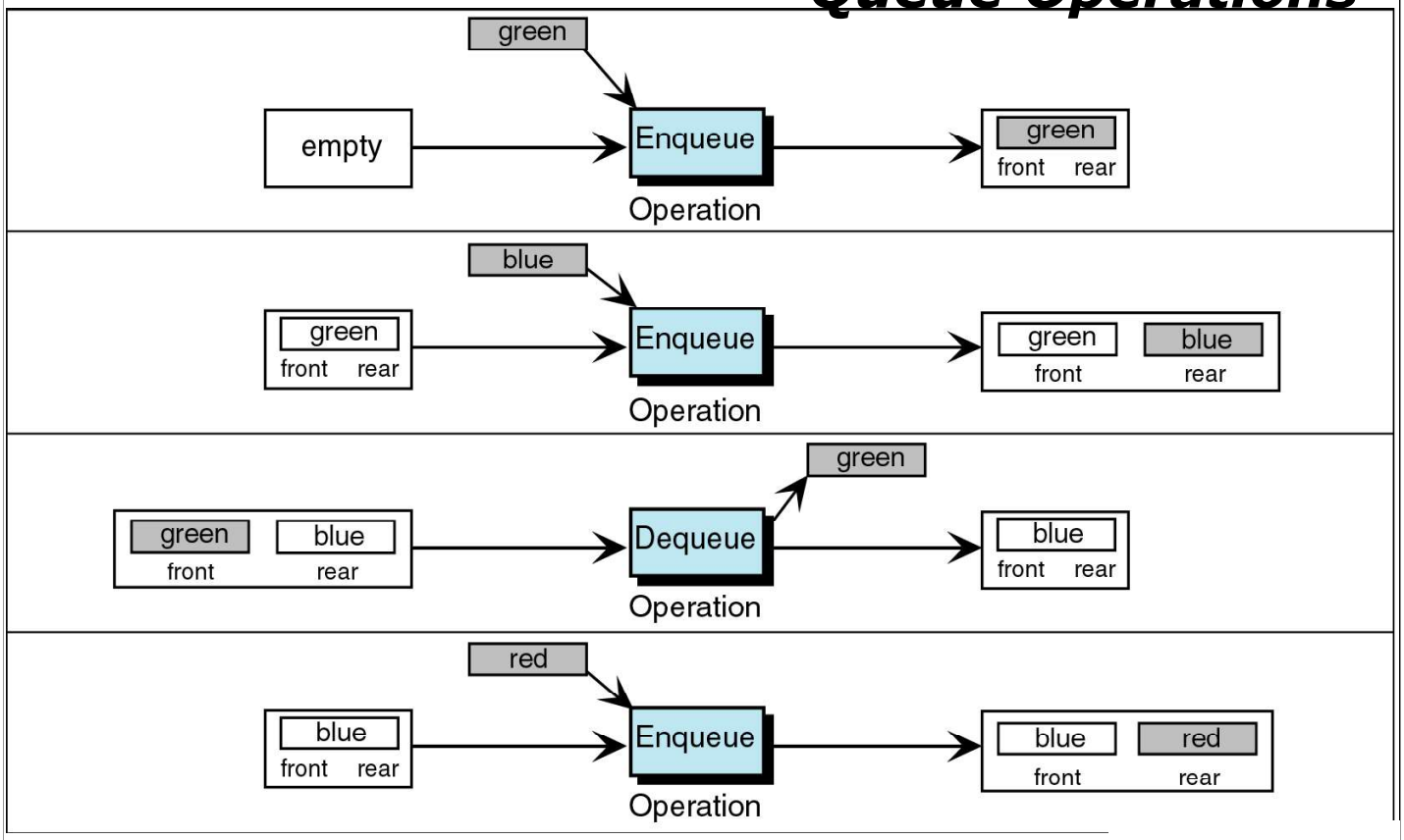
Operation QueueFront



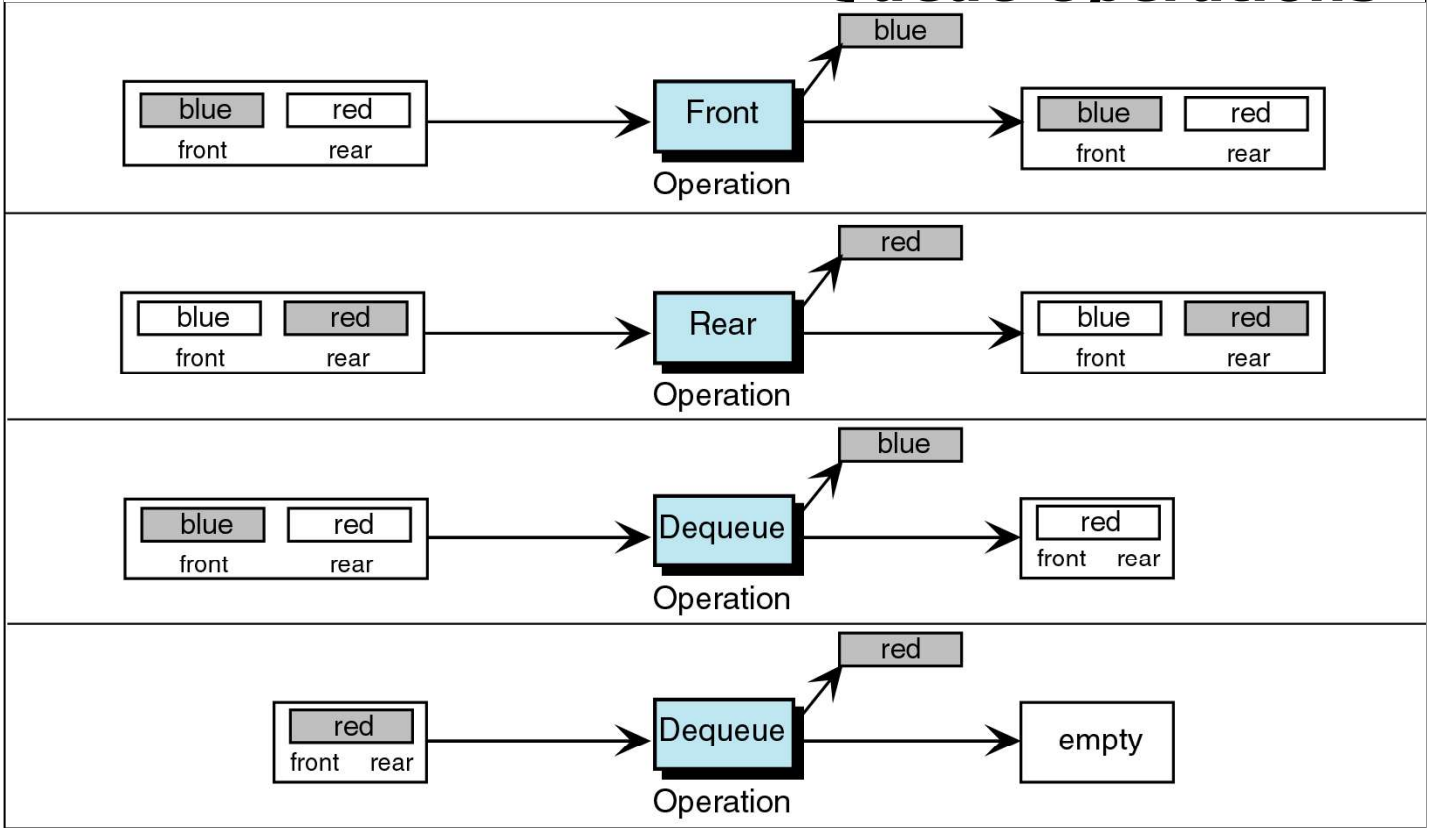
Operation QueueRear



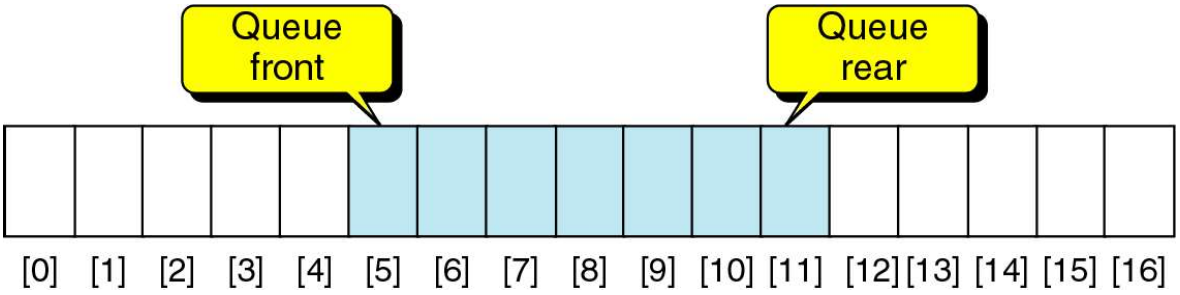
Queue Operations



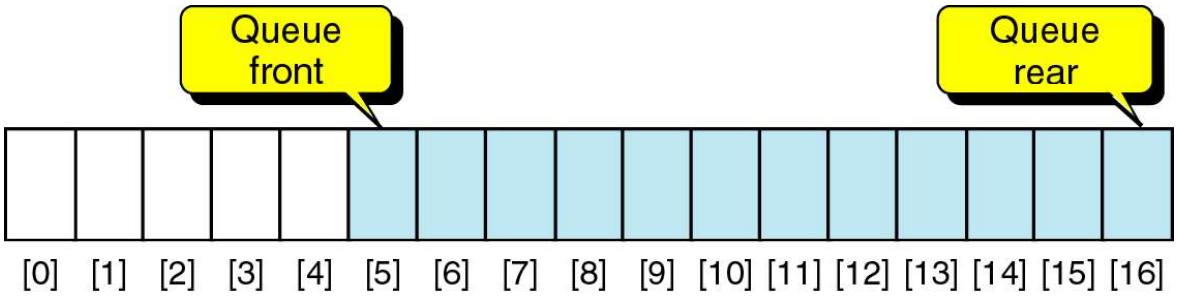
Queue Operations



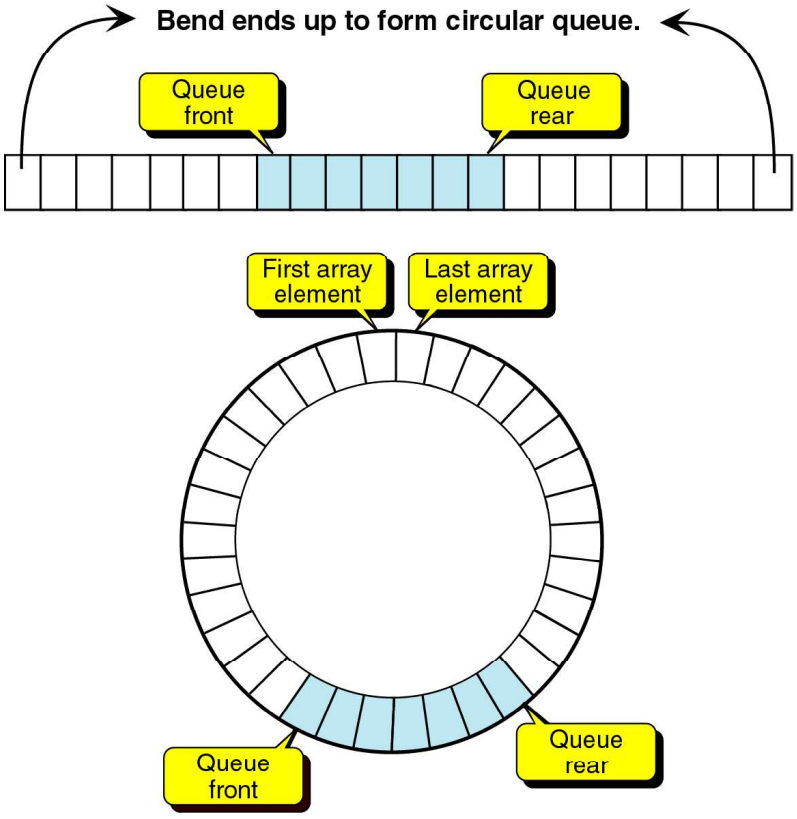
โครงสร้างของ **Queue** แบบ **Array**



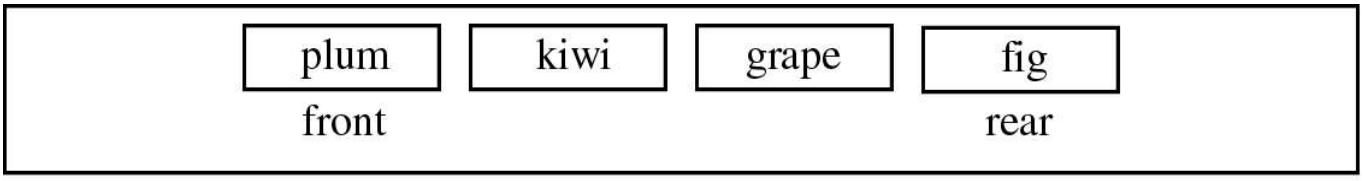
Is queue full?



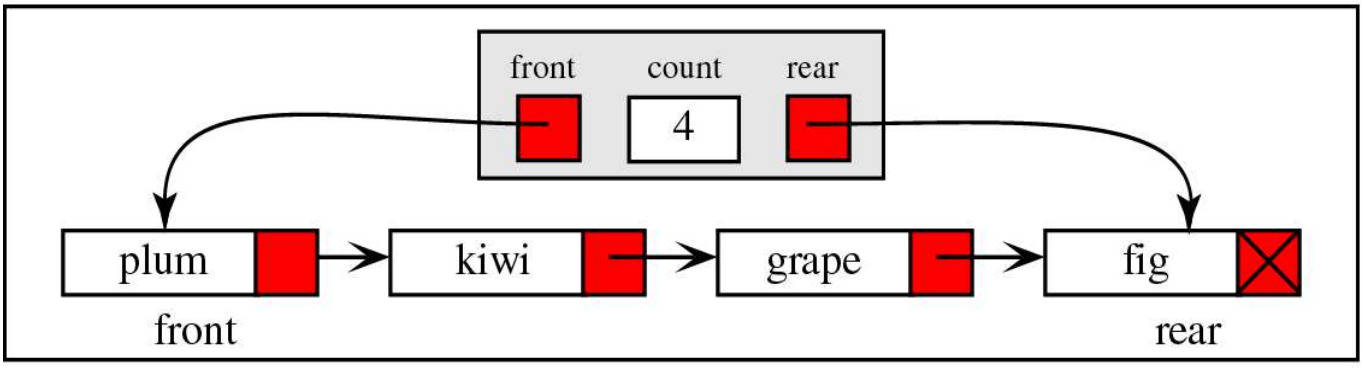
Circular Queue



โครงสร้างของ Queue แบบ Linked list

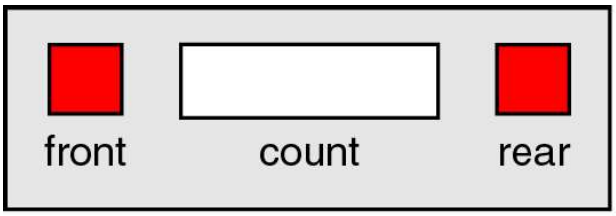


(a) Conceptual queue

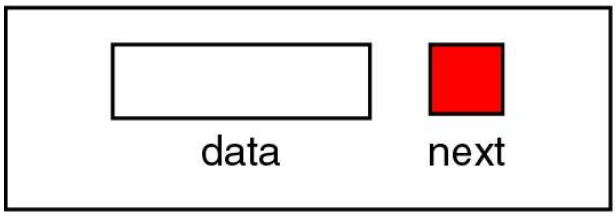


(b) Physical queue

Queue data structure



Head structure



Node structure

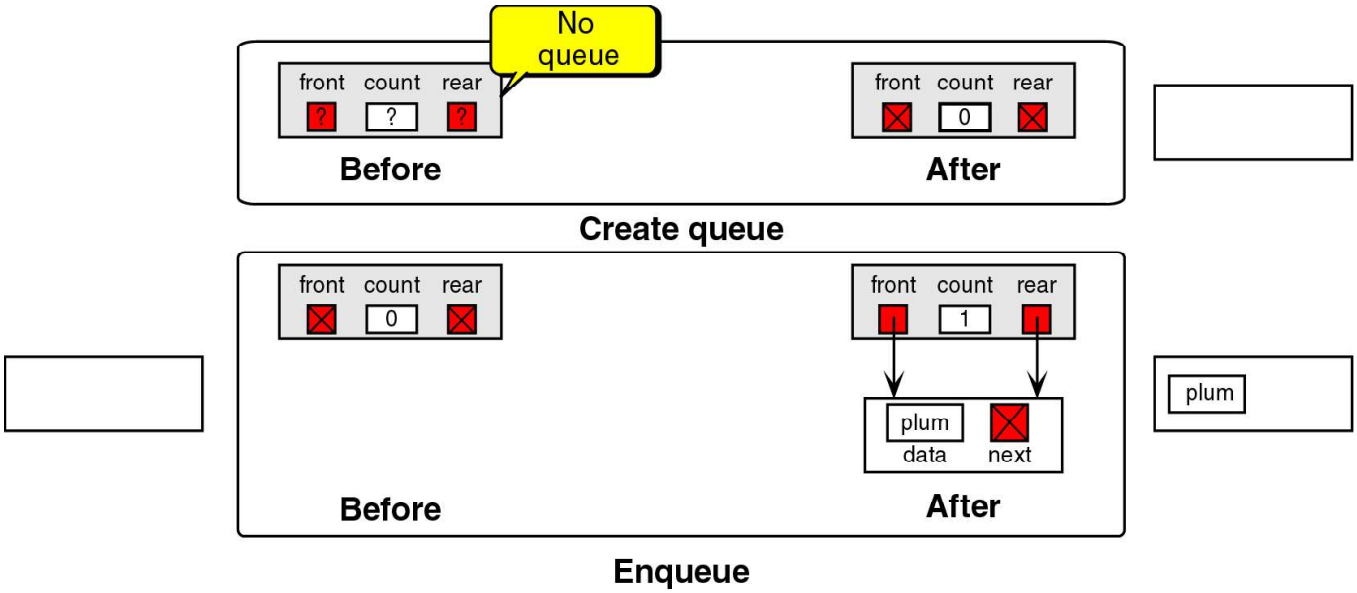
```
struct  queue {
    struct node *front;
    int count;
    struct node *rear;
};
```

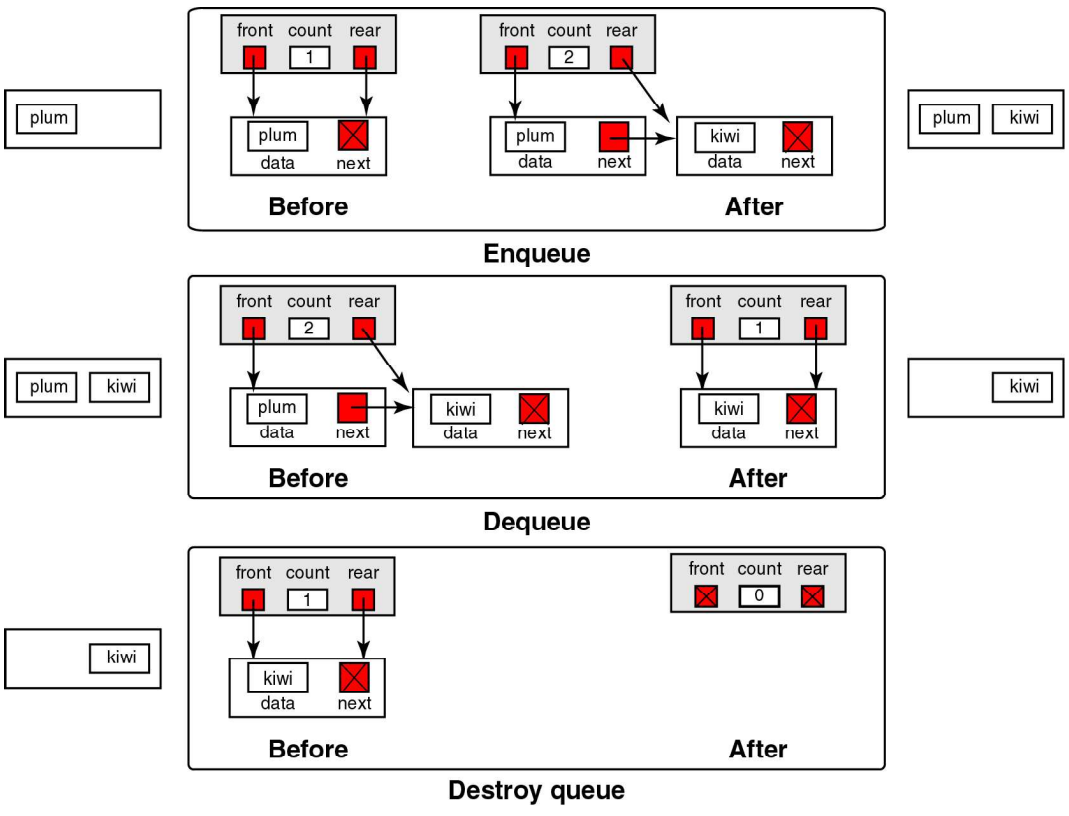
```
struct  node {
    int data;
    struct node *next;
};
```

Algorithm พื้นฐานของ Queue

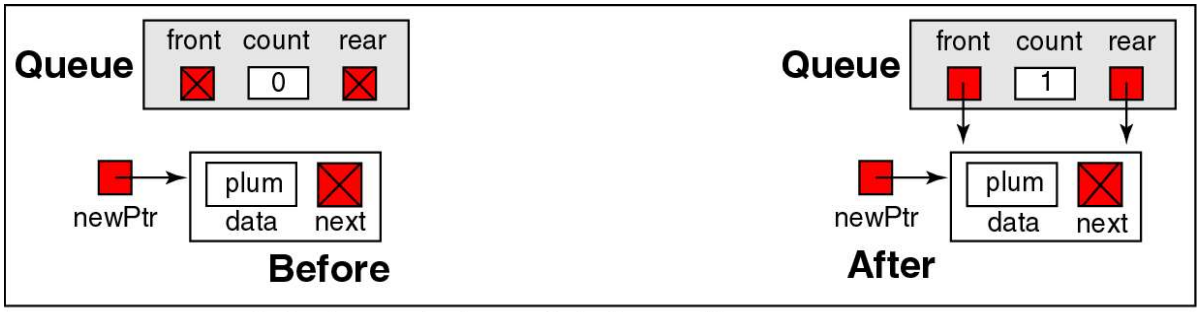
1. **Create queue:** สร้าง queue head จาก dynamic memory
2. **Enqueue:** เพิ่มรายการเข้าไปใน queue
3. **Dequeue:** ลบรายการออกจาก queue
4. **Queue front:** เรียกใช้ข้อมูลที่ด้านหน้าของ queue
5. **Queue rear:** เรียกใช้ข้อมูลที่ด้านหน้าของ queue
6. **Empty queue:** ตรวจสอบว่า queue ว่างหรือไม่
7. **Full queue:** ตรวจสอบว่า queue เต็มหรือไม่
   (มีหน่วยความจำ จัดให้ได้หรือไม่)
8. **Queue count:** บอกจำนวนรายการใน queue
9. **Destroy queue:** ลบข้อมูลทั้งหมดใน queue และคืนหน่วยความจำ
   ให้ระบบแล้วลบและคืนหน่วยความจำของ head node

Create and enqueue

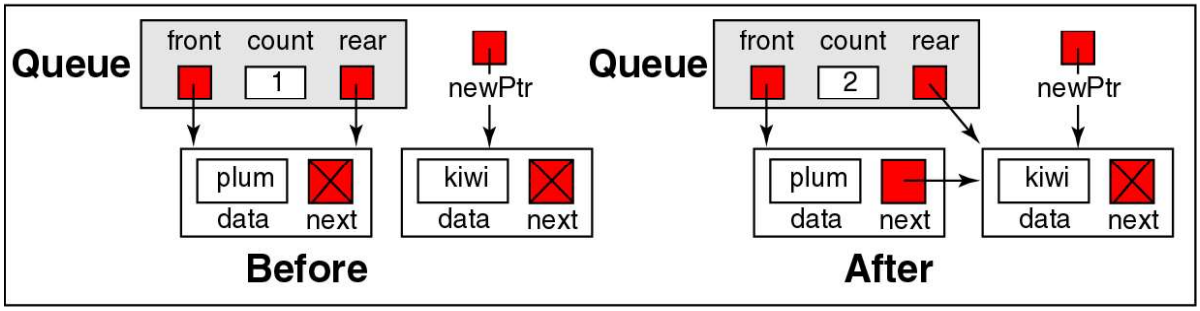




เพิ่มข้อมูลเข้า Queue

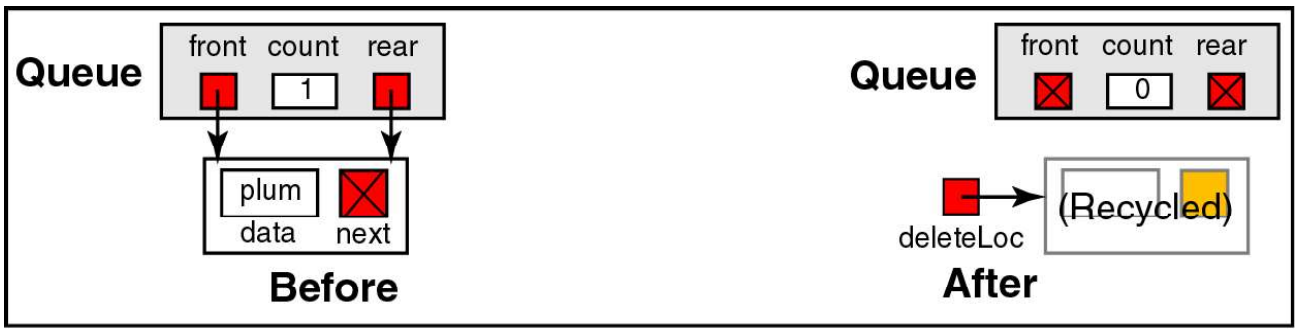


(a) Case 1: insert into null queue

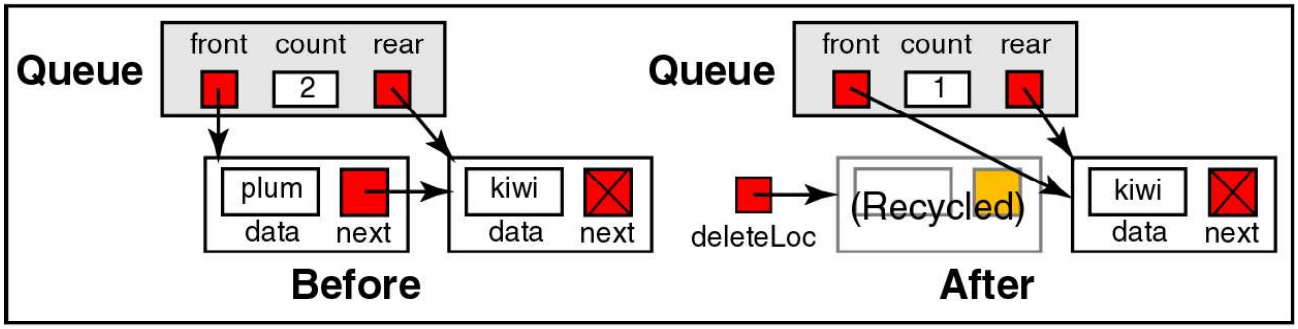


(b) Case 2: insert into queue with data

ลบข้อมูลออกจากเข้า Queue



(a) Case 1: delete only item in queue



(b) Case 2: delete item at front of queue