• 10301222 โครงสร้างข้อมูลและอัลกอริทึม

•Chapter 4

•Linked List

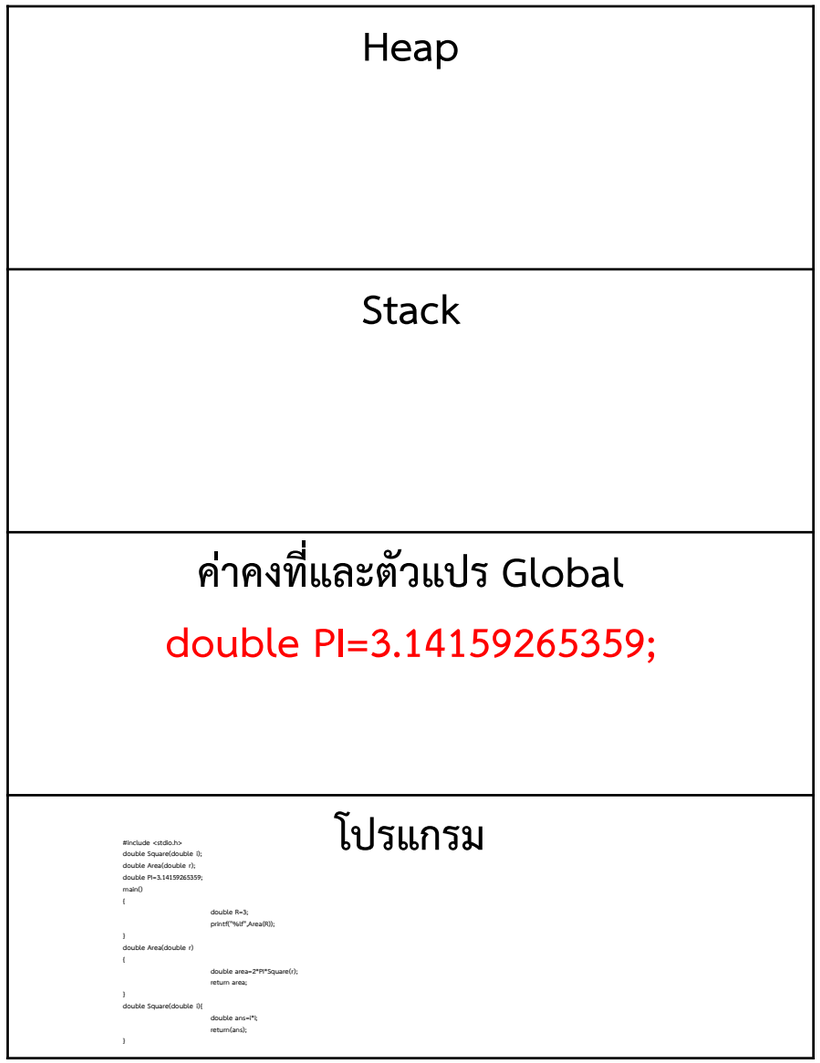
ผศ.ดร. ปวีณ เขื่อนแก้ว

สาขาวิชาวิทยาการคอมพิวเตอร์ มหาวิทยาลัยแม่โจ้

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
	double R=3;
	printf("%lf",Area(R));
}
double Area(double r)
{
	double area=PI*Square(r);
	return area;
}
double Square(double i){
	double ans=i*i;
	return(ans);
}
```

แผนที่หน่วยความจำ




```c
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

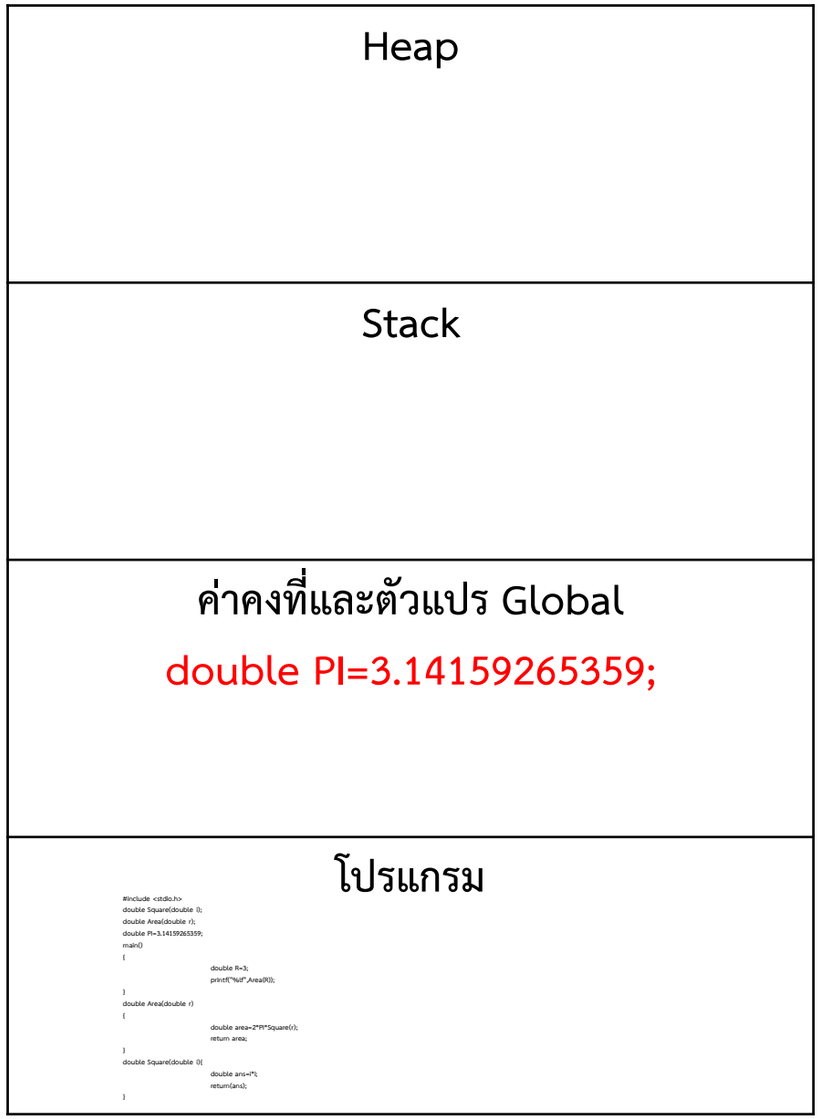
แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ

```
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

แผนที่หน่วยความจำ




```c
#include <stdio.h>
double Square(double i);
double Area(double r);
double PI=3.14159265359;
main()
{
        double R=3;
        printf("%lf",Area(R));
}
double Area(double r)
{
        double area=PI*Square(r);
        return area;
}
double Square(double i){
        double ans=i*i;
        return(ans);
}
```

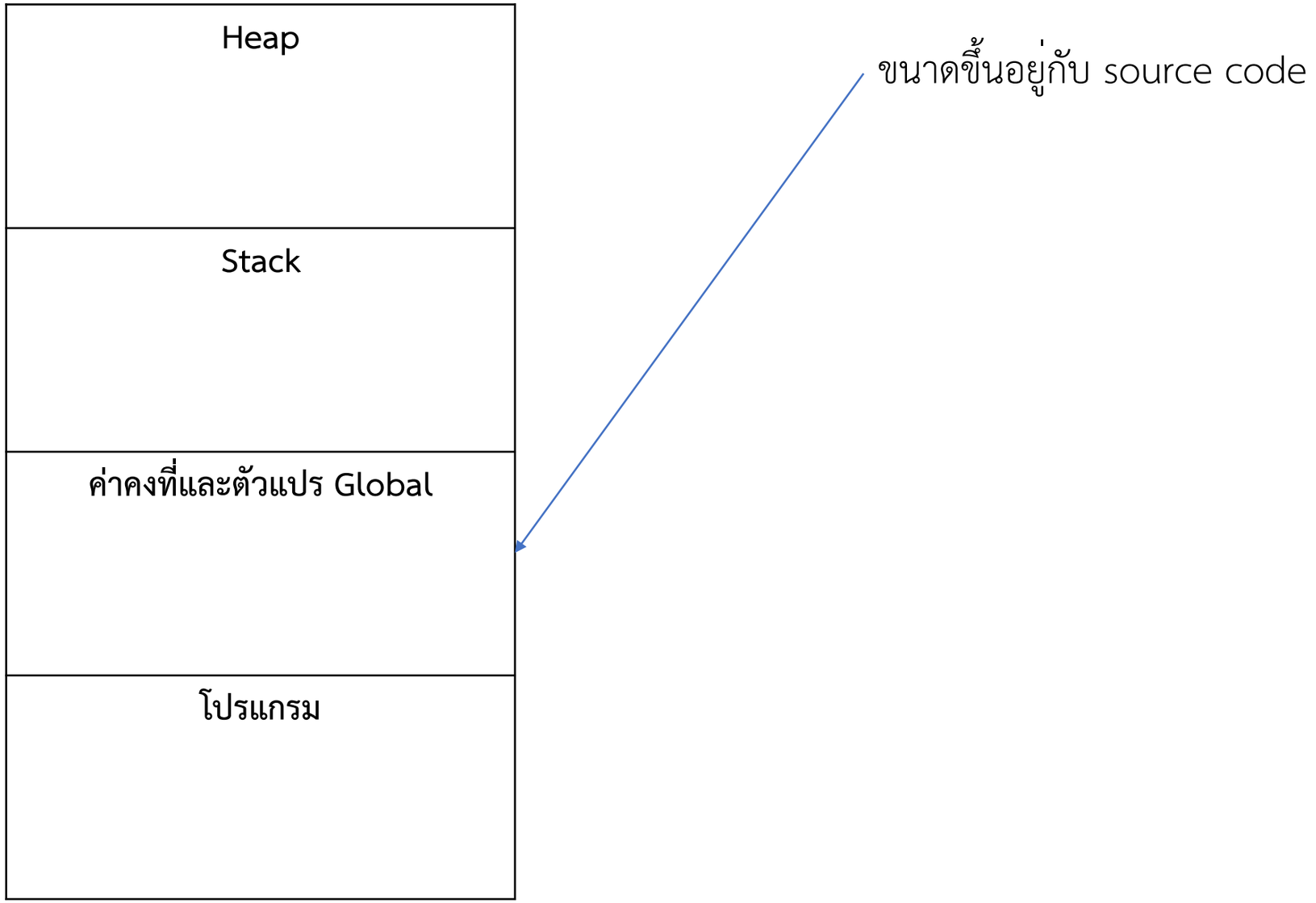
แผนที่หน่วยความจำ

โครงสร้างนี้อาจแตกต่างกันไปบ้างในรายละเอียด ขึ้นอยู่กับภาษา และสถาปัตยกรรม ตัวอย่างนี้เป็นของโปรแกรมที่เขียนด้วยภาษา C

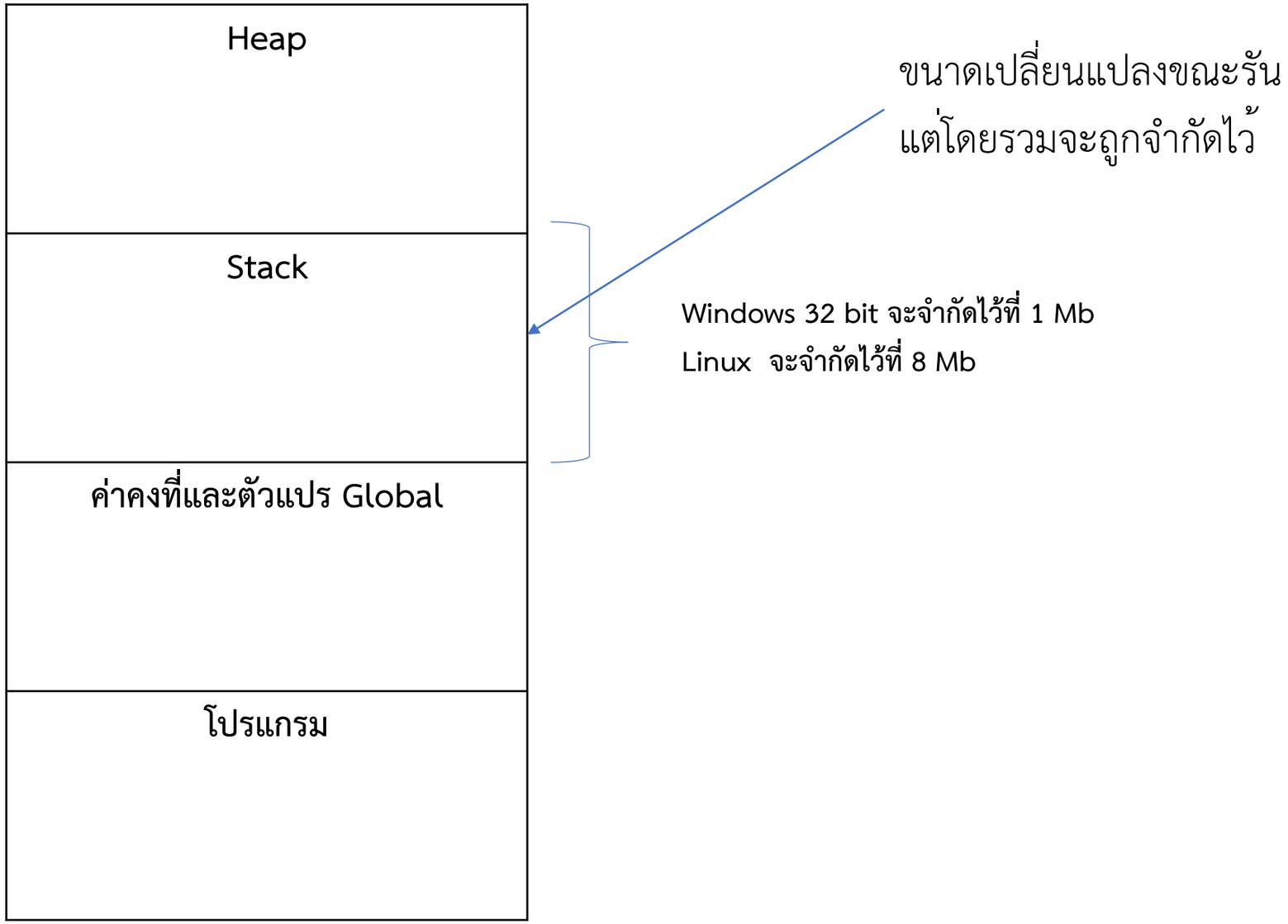
แผนที่หน่วยความจำ

ขนาดเท่าโปรแกรมที่ถูกคอมไพล์แล้ว
โปรแกรม 64 บิต ก็จะได้ใช้พื้นที่
มากกว่าโปรแกรม 32 บิต

แผนที่หน่วยความจำ



แผนที่หน่วยความจำ



แผนที่หน่วยความจำ

ขนาดเปลี่ยนแปลงขณะรัน
แต่โดยรวมจะถูกจำกัดไว้

นี่คือเหตุผลว่าทำไมเราต้องประกาศตัวแปรก่อนใช้ทุกครั้ง
ก็เพราะว่า compiler จำเป็นต้องรู้ว่าแต่ละ function หรือ method จะใช้พื้นที่ใน stack เท่าใด

แม้แต่ Array ถ้าประกาศใน main()
ก็จะเก็บในนี้ ดังนั้น array จึงต้องมีขนาดแน่นอนตั้งแต่ตอนประกาศ และเปลี่ยนแปลงขนาดไม่ได้ไม่ได้

แผนที่หน่วยความจำ

ขนาดเปลี่ยนแปลงขณะรัน
แต่โดยรวมจะถูกจำกัดไว้

Windows 32 bit จะจำกัดไว้ที่ 1 Mb
Linux จะจำกัดไว้ที่ 8 Mb

ขนาด 1 Mb หมายถึง ขณะรัน
โปรแกรมจะเก็บข้อมูลได้ไม่เกิด 1 Mb ใช่หรือไม่ ?

ถ้าเก็บ char จะได้มากถึง 1 ล้านตัวแปร

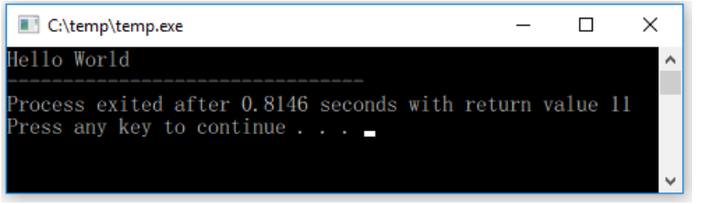
มากพอใช้หรือไม่ ?

แผนที่หน่วยความจำ

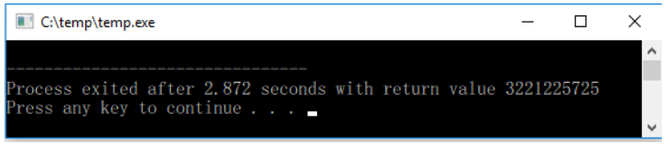
ขนาดเปลี่ยนแปลงขณะรัน
แต่โดยรวมจะถูกจำกัดไว้

Windows 32 bit จะจำกัดไว้ที่ 1 Mb
Linux จะจำกัดไว้ที่ 8 Mb

```
#include <stdio.h>
main()
{  double Giant[200000];
   printf("Hello World");
}
```




```
#include <stdio.h>
main()
{  double Giant[300000];
   printf("Hello World");
}
```

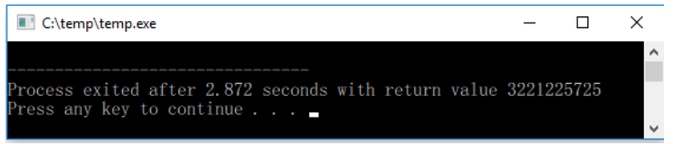


แผนที่หน่วยความจำ

ขนาดเปลี่ยนแปลงขณะรัน
แต่โดยรวมจะถูกจำกัดไว้

Windows 32 bit จะจำกัดไว้ที่ 1 Mb
Linux จะจำกัดไว้ที่ 8 Mb

```
#include <stdio.h>
main()
{  double Giant[300000];
   printf("Hello World");
}
```



Giant มีขนาด =8 x 300000=2.4Mb

ล้นพื้นที่ stack ส่งผลให้โปรแกรมทำงานผิดปรกติ

แบบนี้เรียกว่าเกิด Stack overflow

ภาษา C ไม่มีระบบตรวจสอบ ดังนั้น programmer ต้องรู้เอง

แผนที่หน่วยความจำ

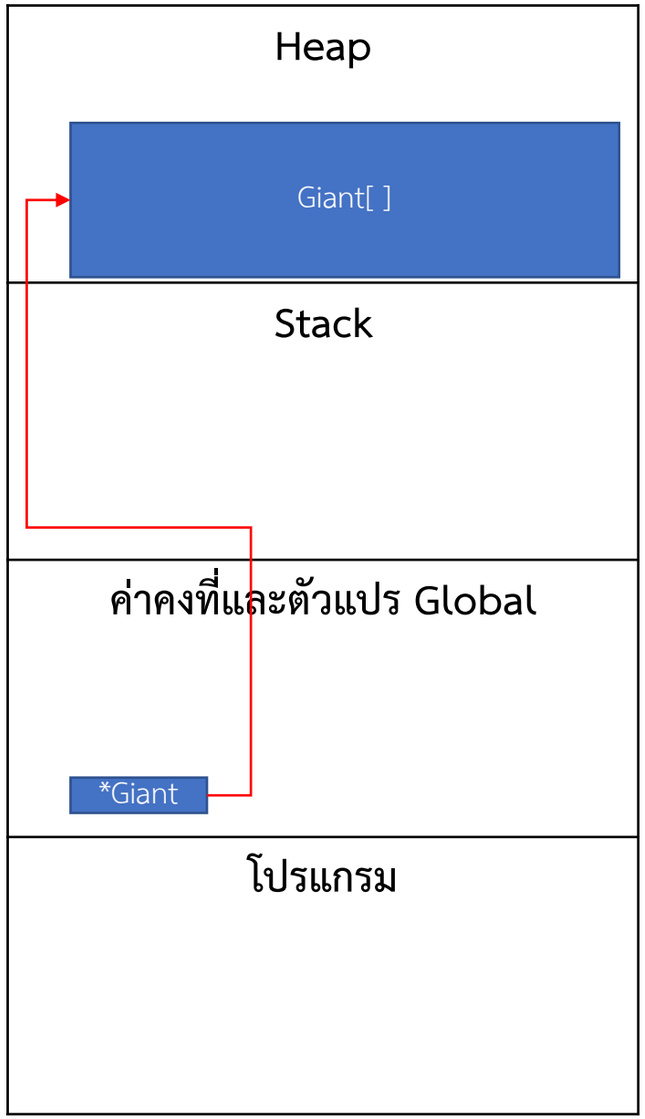
หน่วยความที่สามารถร้องขอใช้จากระบบปฏิบัติการได้

ไม่มีจำกัด

ตัวแปรชนิดเดียวในภาษา C ที่เข้าถึงหน่วยความจำส่วนนี้นั่นคือ

Pointer

แผนที่หน่วยความจำ



Pointer

จริง ๆแล้วตัวแปร pointer อยู่ใน Stack

แต่มันสามารถอ้างตำแหน่งใน heap ได้

การเข้าถึงข้อมูลแบบนี้เรียกว่า

By reference หรือ เข้าถึงโดยการอ้างอิง

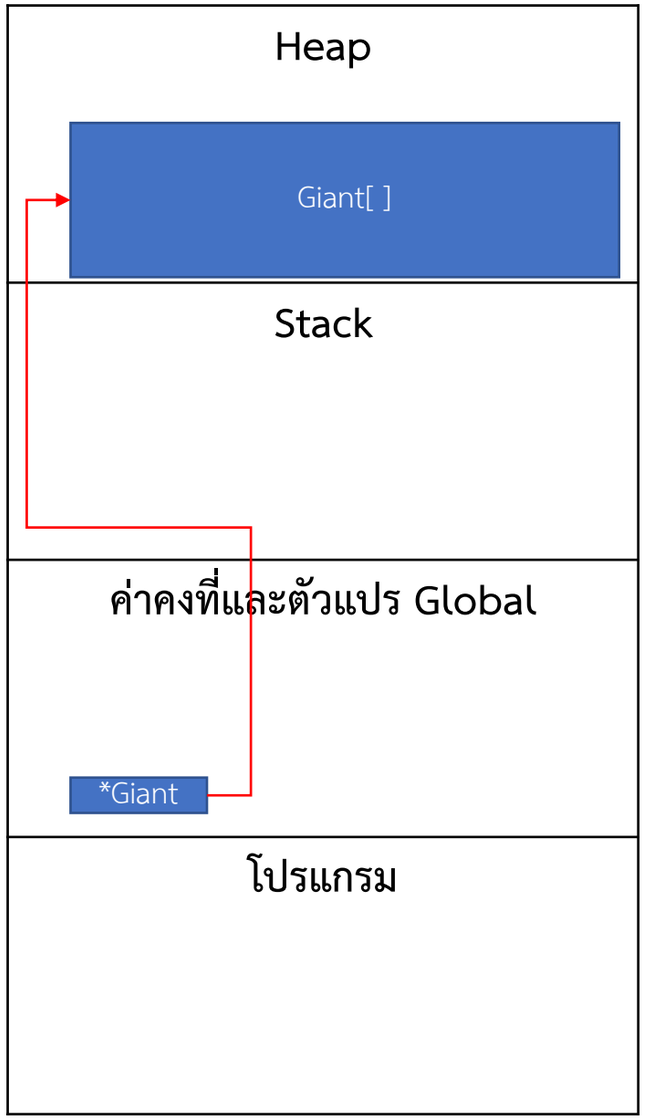
ตัวแปรที่ใช้เก็บข้อมูล เรียกว่า

Value Type

ตัวแปรที่ใช้เก็บตำแหน่ง เรียกว่า

Reference Type

แผนที่หน่วยความจำ



Pointer

การใช้งานหน่วยความจำใน heap มี 2 ขั้นตอนคือ

1 ร้องขอพื้นที่ตามขนาดที่ต้องการ

C ใช้คำสั่ง ตำแหน่ง=malloc(ขนาดที่ต้องการ)

Java ใช้คำสั่ง new

2 คืนพื้นที่เมื่อไม่ใช้แล้ว

C ใช้คำสั่ง free(ระบุตำแหน่ง)

Java ไม่ต้องคืน

เพราะมีระบบหลังบ้าน คอยตรวจสอบและคืนให้

แผนที่หน่วยความจำ

Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
}
```

แผนที่หน่วยความจำ

Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
}
```

แผนที่หน่วยความจำ

Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
}
```

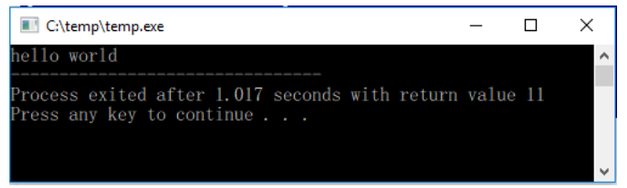
แผนที่หน่วยความจำ

Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
}
```



แผนที่หน่วยความจำ

Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
        free(s);
        printf("After:%s\n",s);
}
```

แผนที่หน่วยความจำ

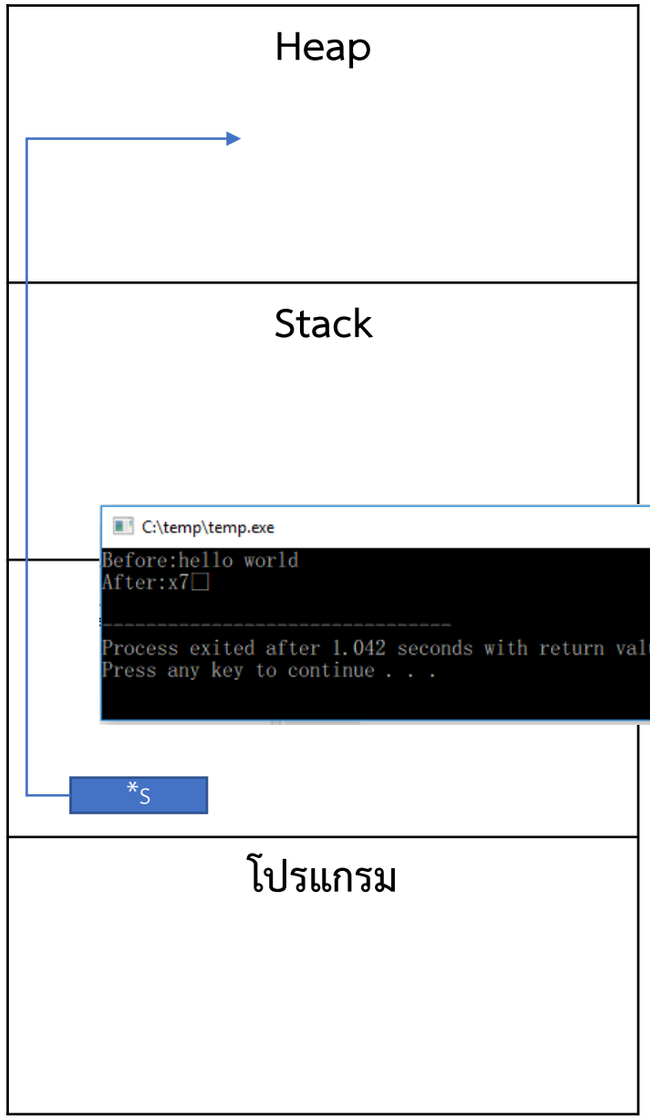
Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf("%s",s);
        free(s);
        printf("After:%s\n",s);
}
```

แผนที่หน่วยความจำ



Pointer

ตัวอย่างการใช้พื้นที่ใน Heap

ใช้ 12 bytes เพื่อเก็บ hello world

```
#include <stdio.h>
main()
{       char *s;
        s=(char *)malloc(12);
        strcpy(s,"hello world");
        printf(“Before:%s",s);
        free(s);
        printf("After:%s\n",s);
}
```

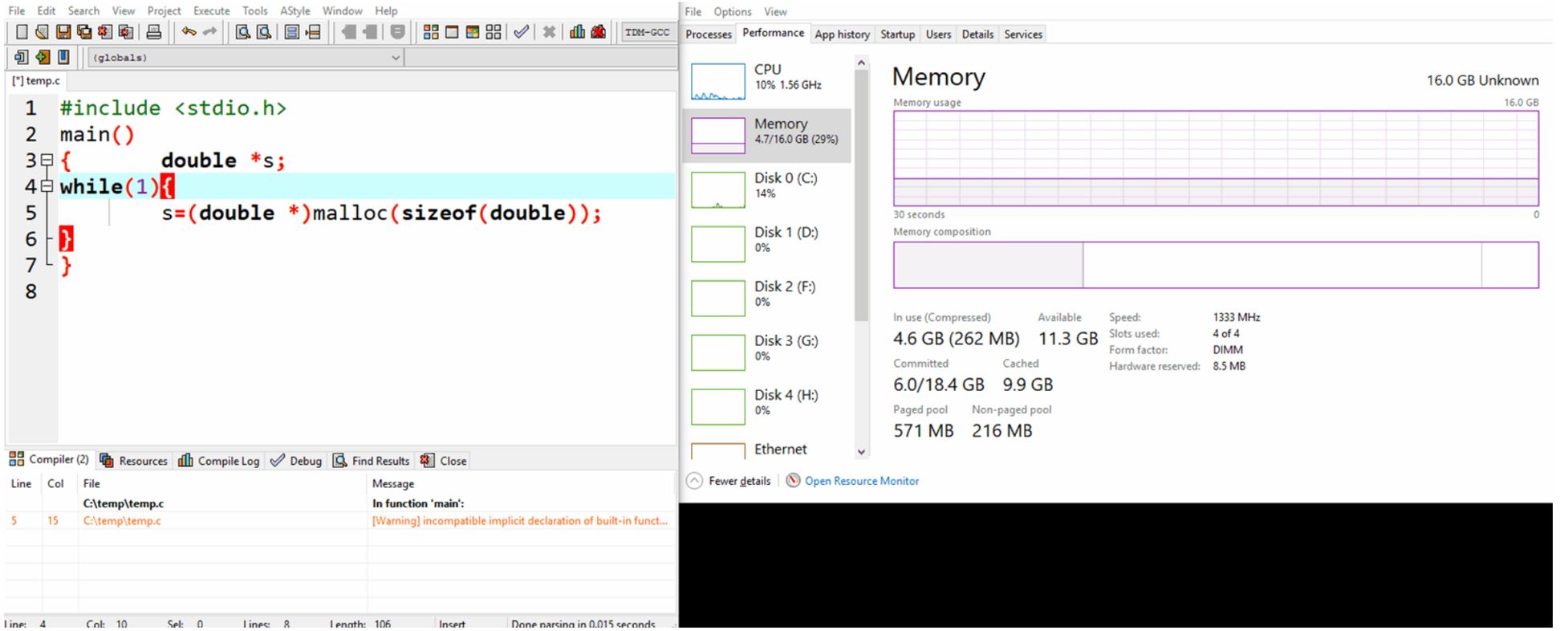
แผนที่หน่วยความจำ

Pointer

ถ้าใช้ภาษา C ห้ามลืมคืนหน่วยความจำเด็ดขาด
ถ้าไม่คืน ถึงแม้จะปิดโปรแกรมไปแล้ว
หน่วยความจำส่วนนั้นก็จะนำกลับมาใช้ใหม่ไม่ได้
ต้อง Reboot เครื่องอย่างเดียว
การแบบนี้เรียกว่า Memory Leaked

ตัวอย่าง code ที่ทำให้เกิด memory leaked แบบที่เลวร้ายที่สุด

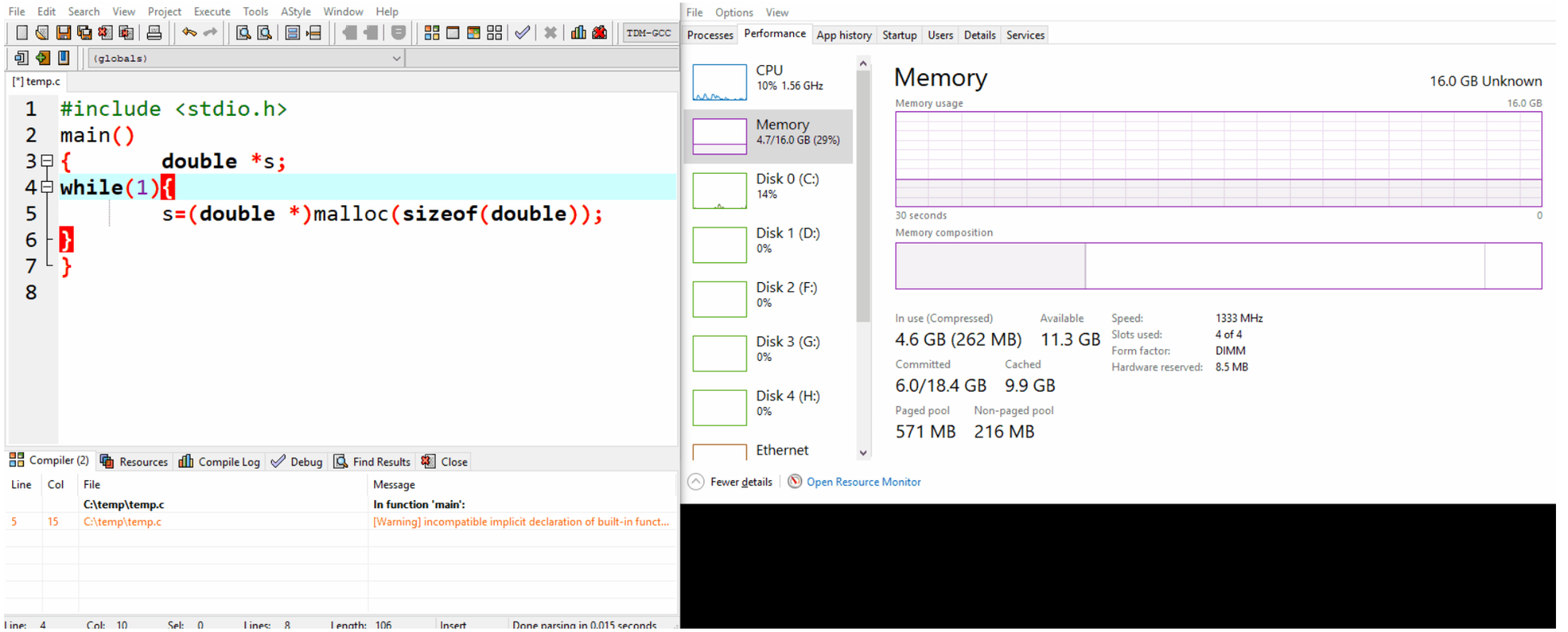
ต้องใช้เวลานานเท่าไหร่จึงจะรั่วหน่วยความจำขนาด 16G จนหมด ?



*ตำเตือน : ห้ามทดลองเขียนโปรแกรมนี้บน notebook

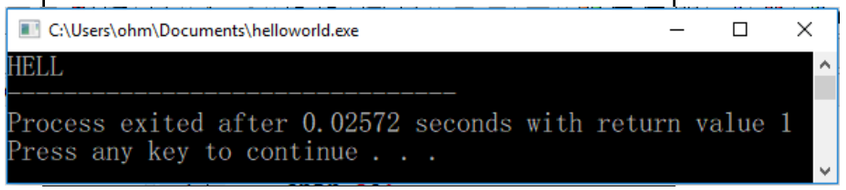
ตัวอย่าง code ที่ทำให้เกิด memory leaked แบบที่เลวร้ายที่สุด

ต้องใช้เวลานานเท่าไหร่จึงจะรั่วหน่วยความจำขนาด 16G จนหมด ?



*ตำเตือน : ห้ามทดลองเขียนโปรแกรมนี้บน notebook

แผนที่หน่วยความจำ



Pointer

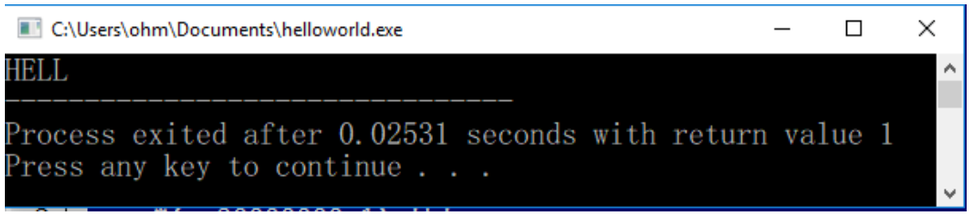
ย้าย Array มาไว้ที่ Heap จะได้ไหม ?

```
#include<stdio.h>
main(){
        char *s;
        s=(char *)malloc(sizeof(char)*2000);
        *(s+0)='H';
        *(s+100)='E';
        *(s+999)='L';
        *(s+1999)='L';
        printf("%c",*(s+0));
        printf("%c",*(s+100));
        printf("%c",*(s+999));
        printf("%c",*(s+1999));
        free(s);
}
```

แผนที่หน่วยความจำ

Pointer

```
#include<stdio.h>
main(){
        char *s;
        s=(char *)malloc(sizeof(char)*20000000);
        *(s+0)='H';
        *(s+100)='E';
        *(s+999)='L';
        *(s+20000000-1)='L';
        printf("%c",*(s+0));
        printf("%c",*(s+100));
        printf("%c",*(s+999));
        printf("%c",*(s+20000000-1));
        free(s);
}
```



Linked List

ใช้ Pointer เพื่อย้าย Array ไปไว้ในหน่วยความจำส่วน heap ช่วยแก้ปัญหา stack overflow แต่ไม่แก้ปัญหา

- ต้องใช้พื้นที่ว่างติดต่อกันเป็นผืนเดียว
- การ แทรก และลบ ยังต้องใช้เวลา O(N)

หากเลือกที่จะเก็บข้อมูลไว้ในหน่วยความจำส่วน heap
มันมีโครงสร้างข้อมูลอีกชนิด ซึ่งจะมีความเหมาะสมกว่า การใช้ Array และ Primitive Type

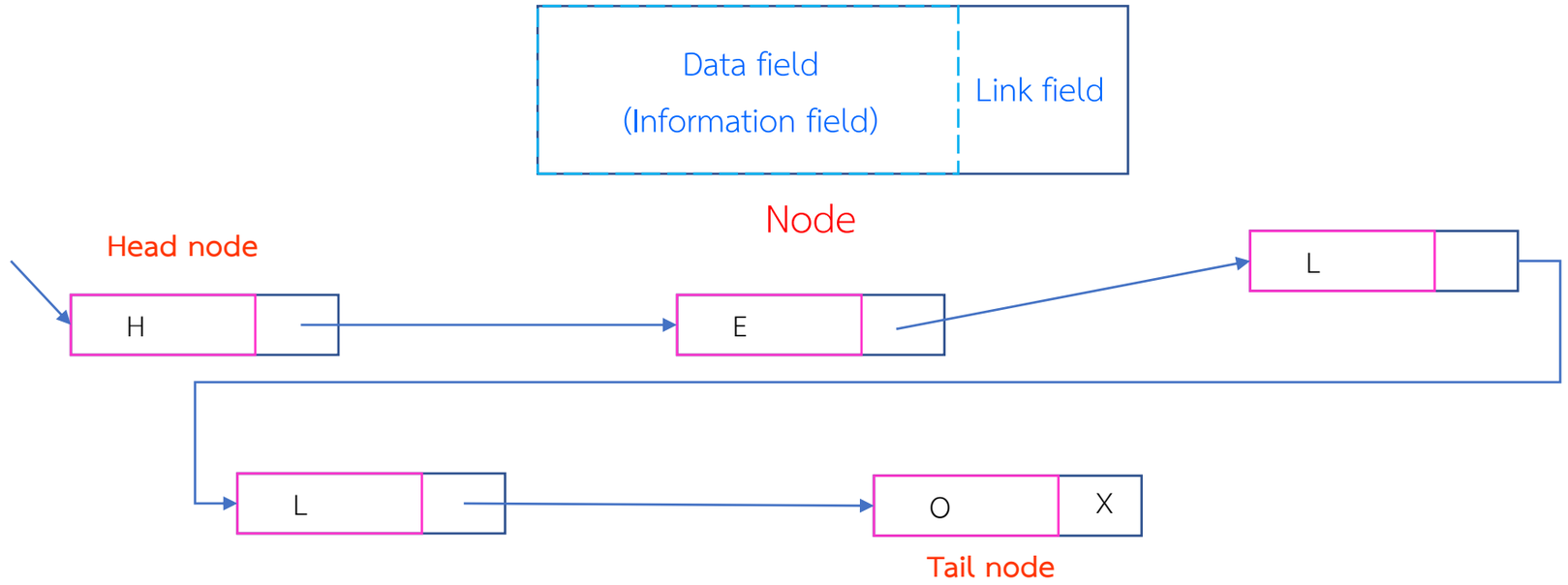
Linked List
รายการโยง

- Singly Linked List
- Doubly Linked List
- Circular Linked List
- Circular Doubly Linked List
- Multi Listed

Linked List: Singly Linked List

Singly Linked List หรือ Regular Linked List ภาษาไทยเรียกว่า การโยงเดี่ยว
เป็นวิธีการเก็บ ข้อมูลอย่างต่อเนื่องของอิลิเม้นต์ต่าง ๆ โดยมีพอยเตอร์เป็นตัวเชื่อมต่อแต่ละอิลิเม้นท์ เรียกว่าโนด (Nodeซึ่งในแต่ละโนดจะประกอบไปด้วย 2 ส่วน คือ Dataจะเก็บข้อมูลของอิลิเม้นท์ และส่วนที่สอง คือ Link Field จะทำหน้าที่เก็บ ตำแหน่งของโนดต่อไปในลิสต

ในส่วนของdataอาจจะเป็นรายการเดียวหรือเป็นเรคคอร์ดก็ได้ในส่วนของ linkจะเป็นส่วนที่เก็บตำแหน่งของโหนดถัดไปใน โหนดสุดท้ายจะเก็บคา Null ซึ่งไม่ได้ชี้ไปยังตำแหน่งใด ๆ เป็นตัวบอกการ สิ้นสุดของลิสตในลิงค์ลิสตจะมีตัวแปรสำหรับชี้ ตำแหน่งลิสต (List pointer variable)ซึ่งเป็นที่เก็บตำแหน่งเริ่มต้นของลิสต ซึ่งก็ คือโหนดแรกของลิสตนั้นเอง ถ้าลิสต์ไม่มีข้อมูล ข้อมูลในโหนดแรกของลิสตจะเป็น Null

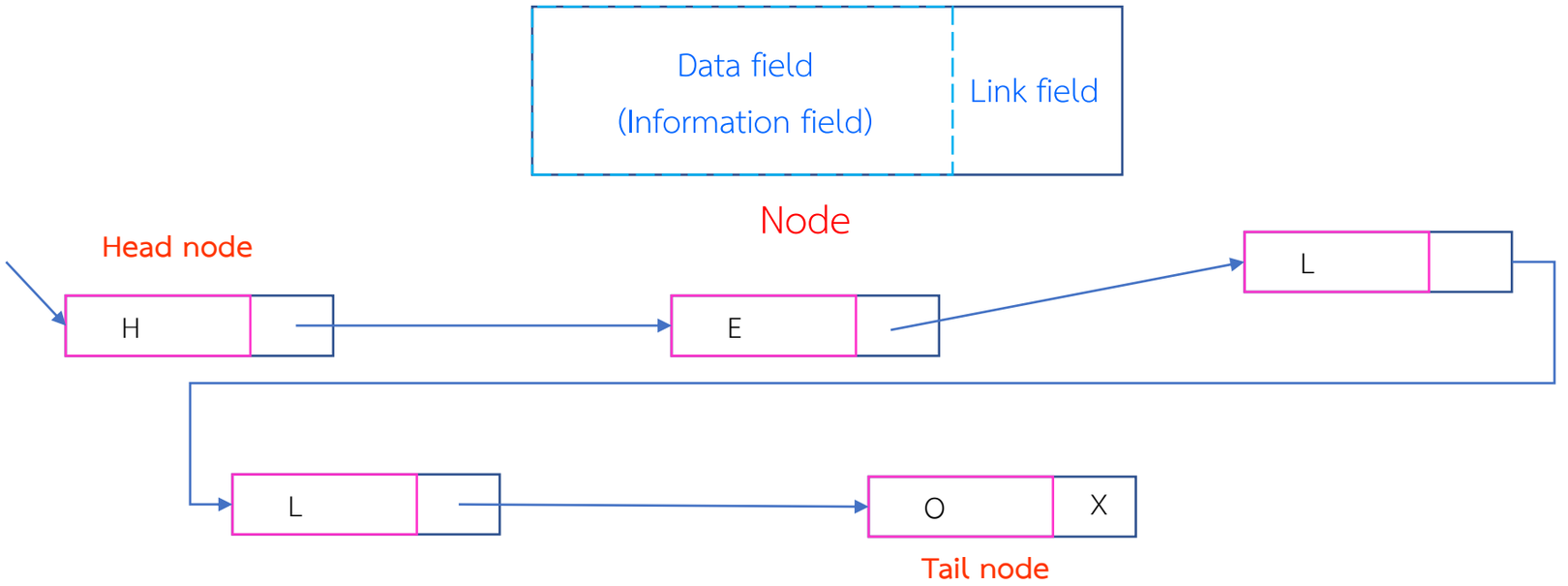


Linked List: Singly Linked List

Data field คือข้อมูลที่ต้องการบันทึก อาจจะเป็น primitive type หรือ composite type ก็ได้ มีกนิยมใส่ข้อมูลเพิ่มเติม เช่น ลำดับของข้อมูล

Link field คือ ตำแหน่งในหน่วยความจำ (address) ในหน่วยความจำของ node ถัดไป

Tail node มักนิยมให้ Link field มีค่าเป็น Null หรือ -1 เพื่อบอกว่า สิ้นสุดข้อมูลแล้ว



Linked List: Singly Linked List

การสร้าง Linked List

ภาษา C นิยมสร้าง Node โดยใช้ตัวแปรแบบ โครงสร้าง (structure)

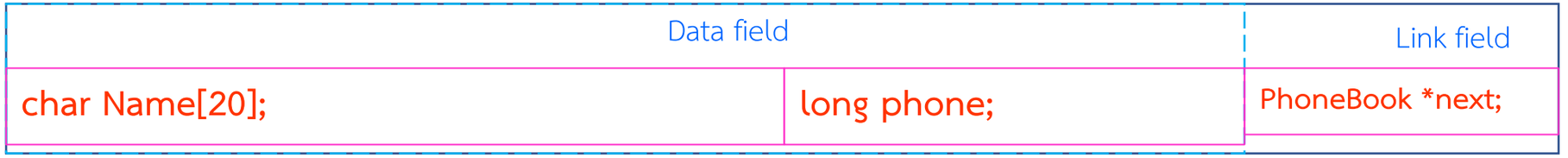
Link field ใช้ตัวแปรแบบ pointer

Node

Linked List: Singly Linked List

การสร้าง Linked List

ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์



PhoneBook

```
struct Node{
        char Name[20];
        long phone;
        struct Node *next;
}PhoneBook;
```

ถ้าคอมไพล์แบบ 32 บิต
PhoneBook มีขนาดเท่าได?

ถ้าคอมไพล์แบบ 64 บิต
PhoneBook มีขนาดเท่าได?

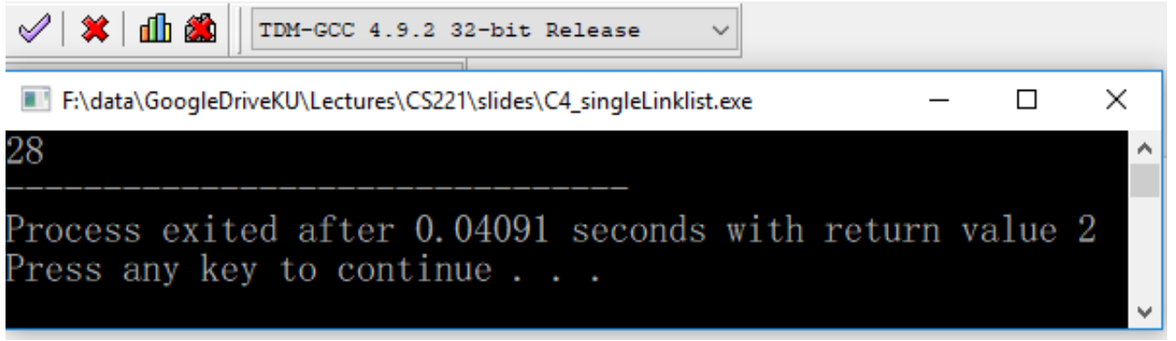
Linked List: Singly Linked List

การสร้าง Linked List

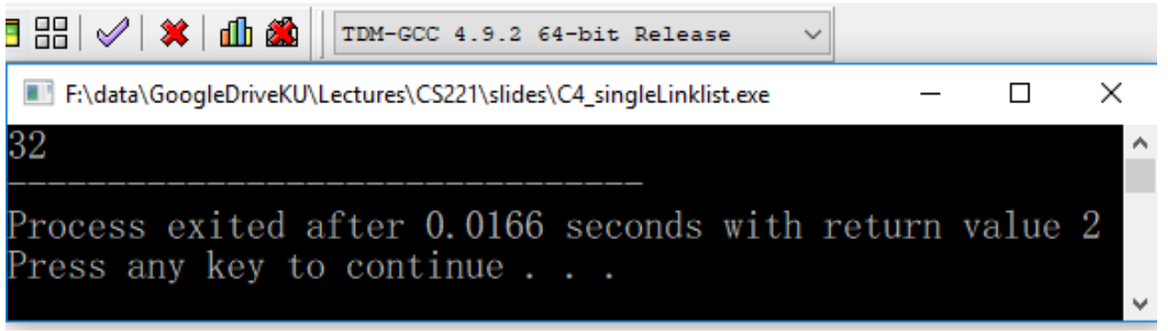
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์

```c
#include <stdio.h>
main(){
    struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;
    printf("%d",sizeof(PhoneBook));
}
```

ถ้าคอมไพล์แบบ 32 บิต PhoneBook มีขนาดเท่าใด?



ถ้าคอมไพล์แบบ 64 บิต PhoneBook มีขนาดเท่าใด?



Linked List: Singly Linked List

การสร้าง Linked List

ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์

การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

*head

*tail

Linked List: Singly Linked List

การสร้าง Linked List
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์

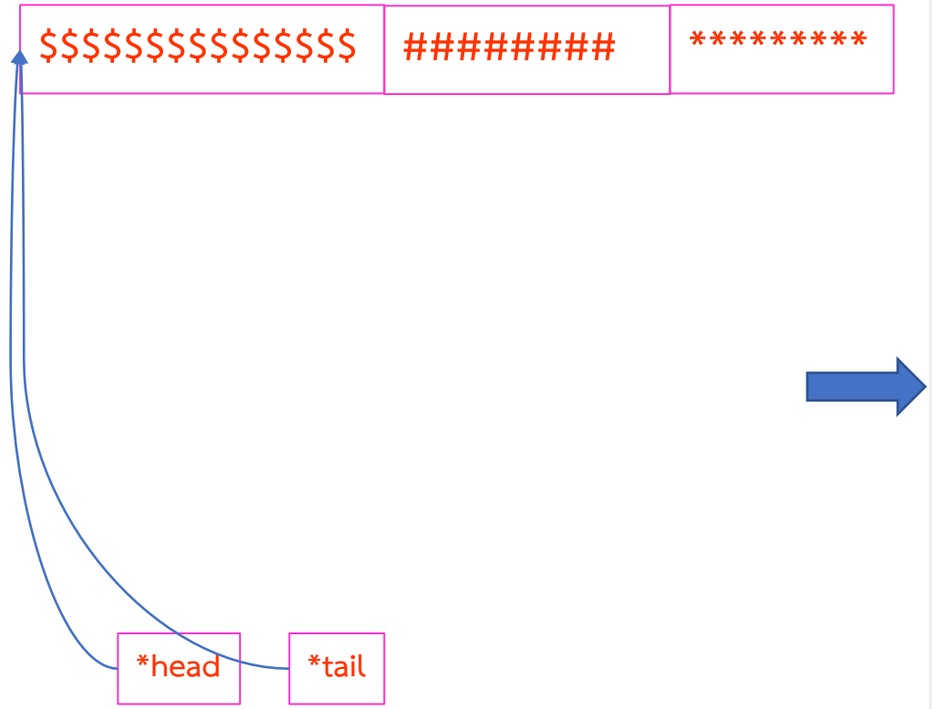
การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

Linked List: Singly Linked List

การสร้าง Linked List
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์



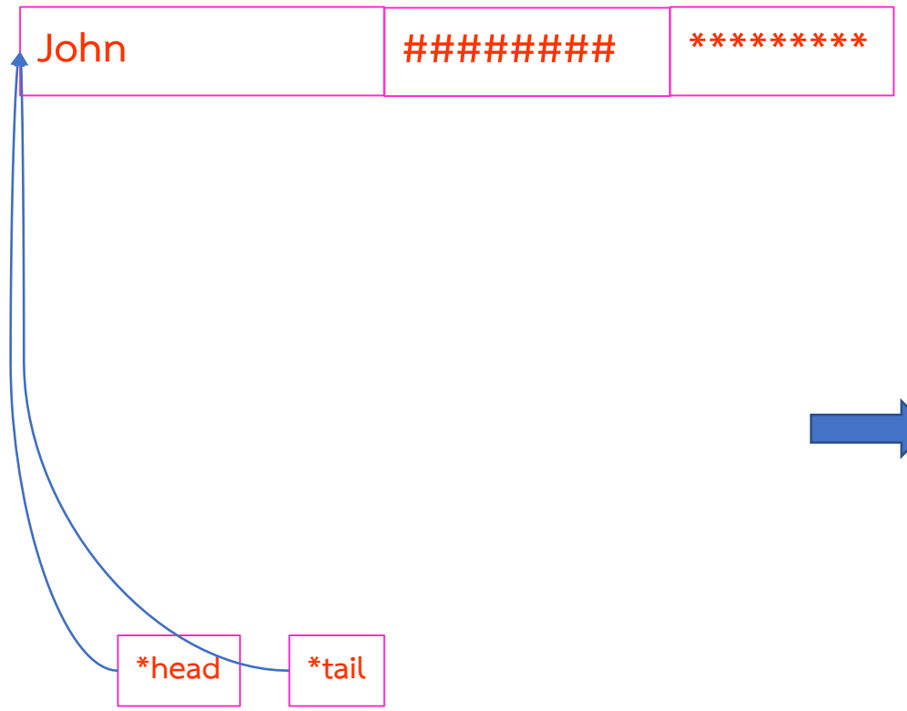
การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

Linked List: Singly Linked List

การสร้าง Linked List
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์



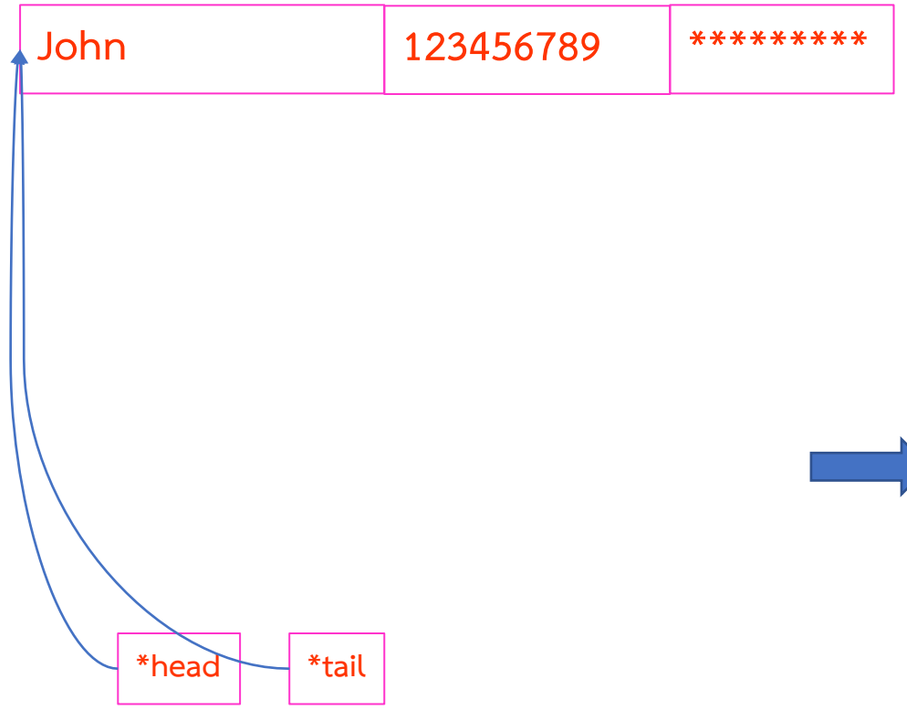
การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

Linked List: Singly Linked List

การสร้าง Linked List
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์



การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

Linked List: Singly Linked List

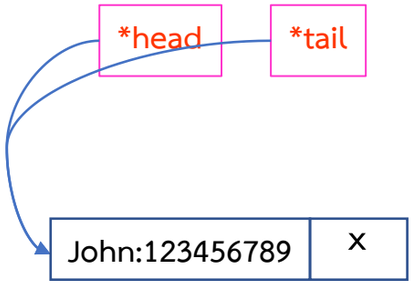
การสร้าง Linked List
ตัวอย่างโปรแกรมเก็บสมุดโทรศัพท์

การเพิ่มข้อมูลลงใน Linked List

Node แรก

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
tail=head;
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;

printf("Name:%s\nPhone:%ld",head->Name,head->phone);

free(head);

}
```

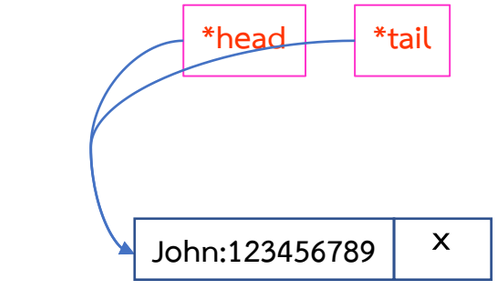
การเพิ่มข้อมูลลงใน Linked List



Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

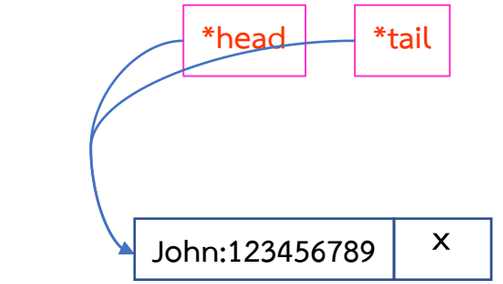
การเพิ่มข้อมูลลงใน Linked List



Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

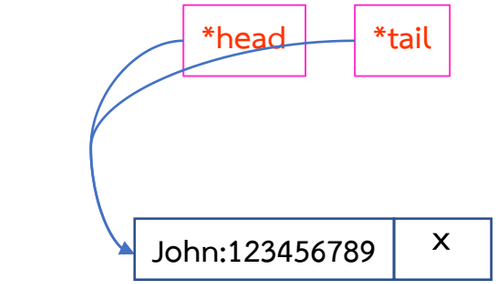
การเพิ่มข้อมูลลงใน Linked List

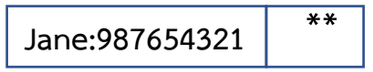


Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List

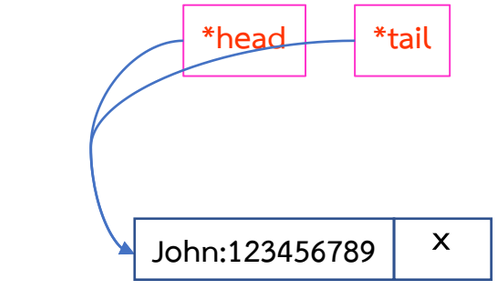


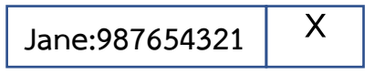


Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List

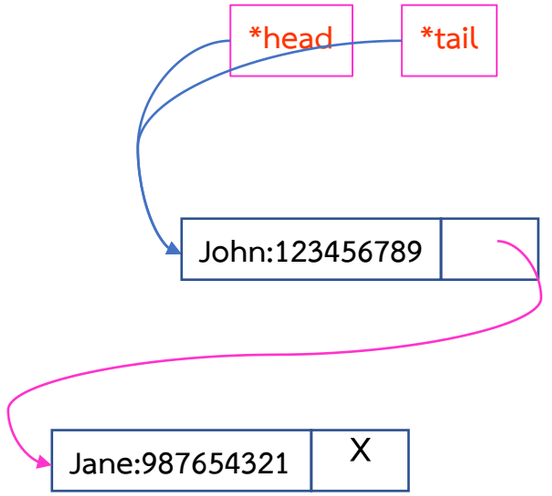




Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

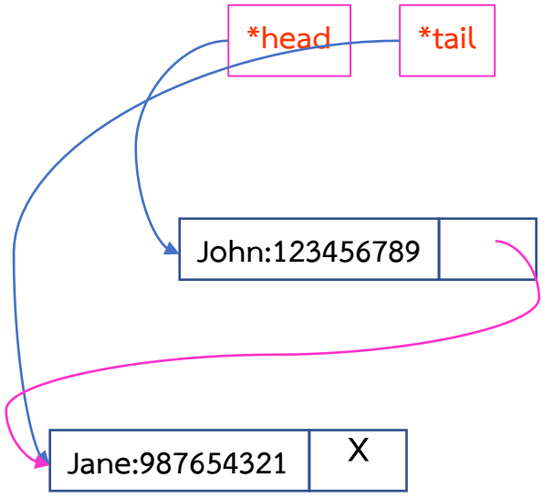
การเพิ่มข้อมูลลงใน Linked List



Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

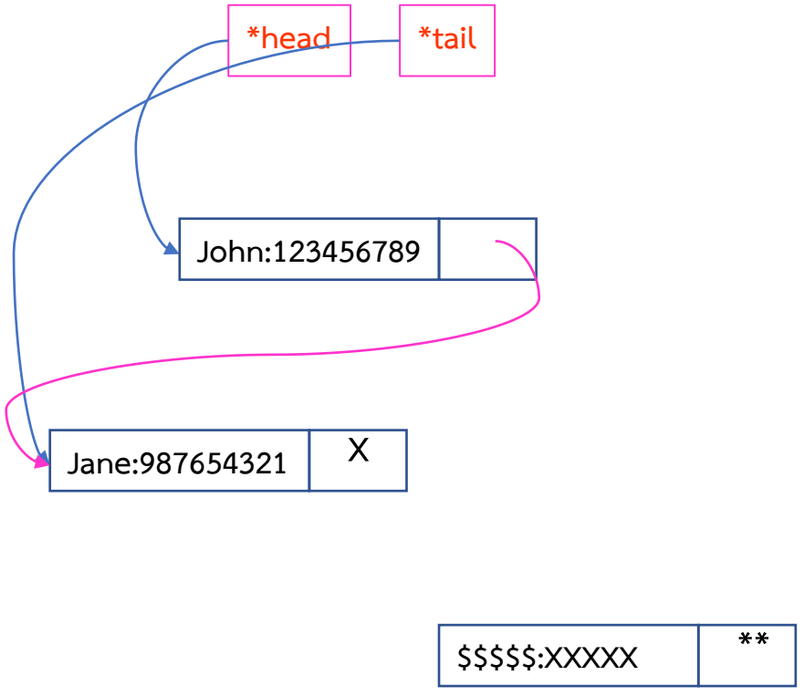
การเพิ่มข้อมูลลงใน Linked List



Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

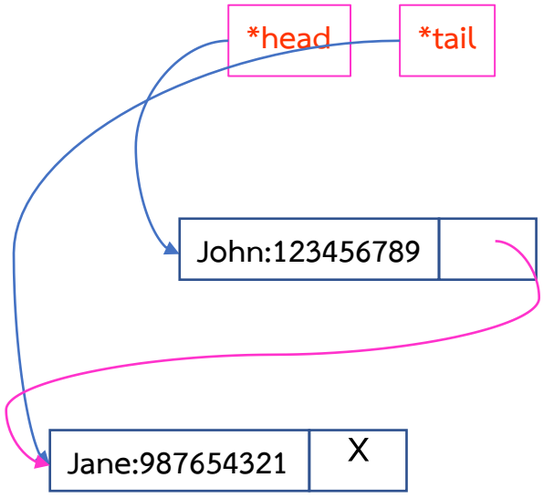
การเพิ่มข้อมูลลงใน Linked List

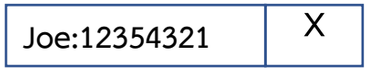


Node ต่อมา

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List

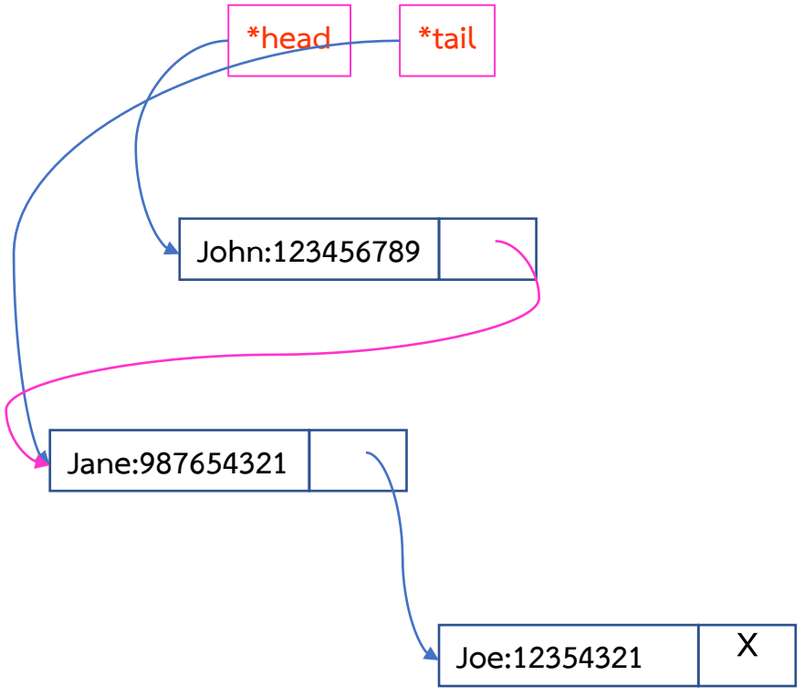




Node ต่อมา

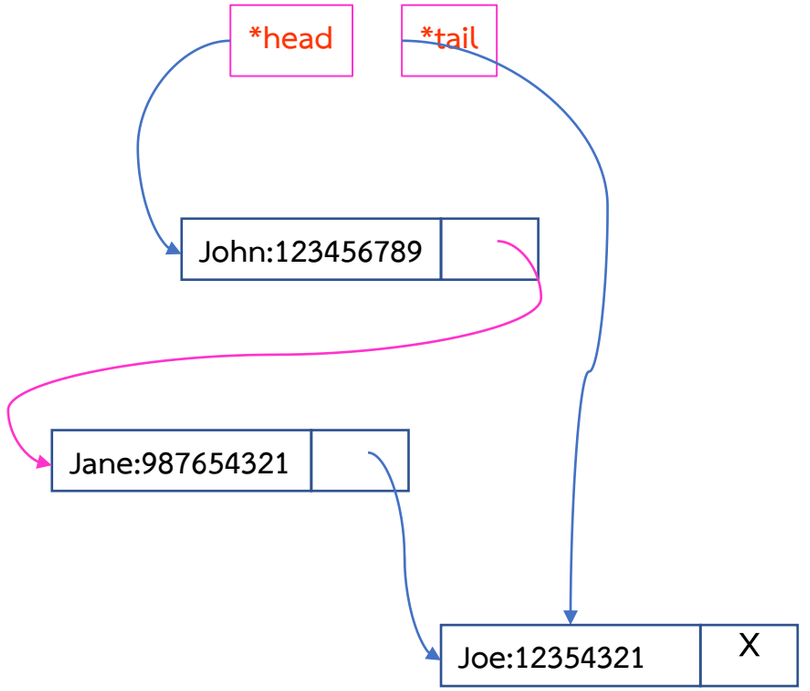
```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List Node ต่อมา



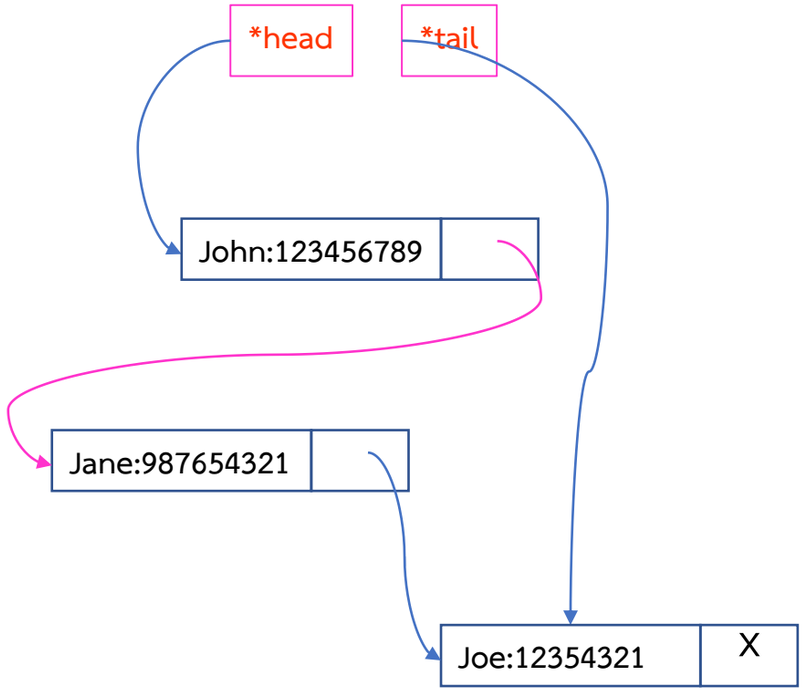

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List Node ต่อมา




```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List Node ต่อมา



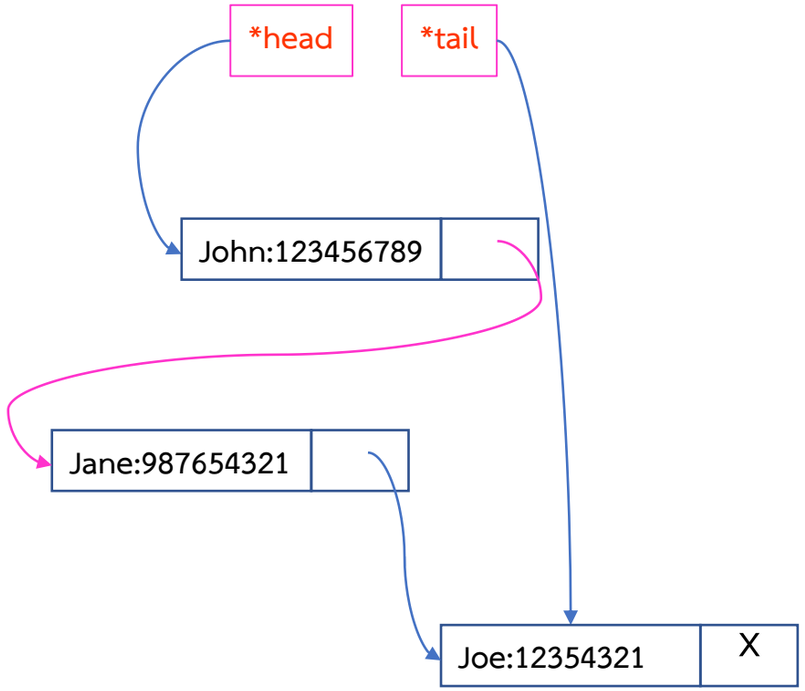
การเพิ่ม Node ต่อท้าย ใน Linked List มีความซับซ้อนทางเวลาเป็น

O(1)

ถ้าไม่มีตัวแปร *tail จะใช้เวลา O(N)

```c
#include <stdio.h>
main(){
typedef struct Node{
        char Name[20];
        long phone;
        struct Node *next;
    }PhoneBook;

PhoneBook *head, *tail,*pt;
head =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(head->Name,"John");
head->phone=123456789;
head->next=0;
tail=head;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Jane");
pt->phone=987654321;
pt->next=0;
tail->next=pt;
tail=pt;

pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Joe");
pt->phone=123454321;
pt->next=0;
tail->next=pt;
tail=pt;
}
```

การเพิ่มข้อมูลลงใน Linked List การแทรก Node แรก

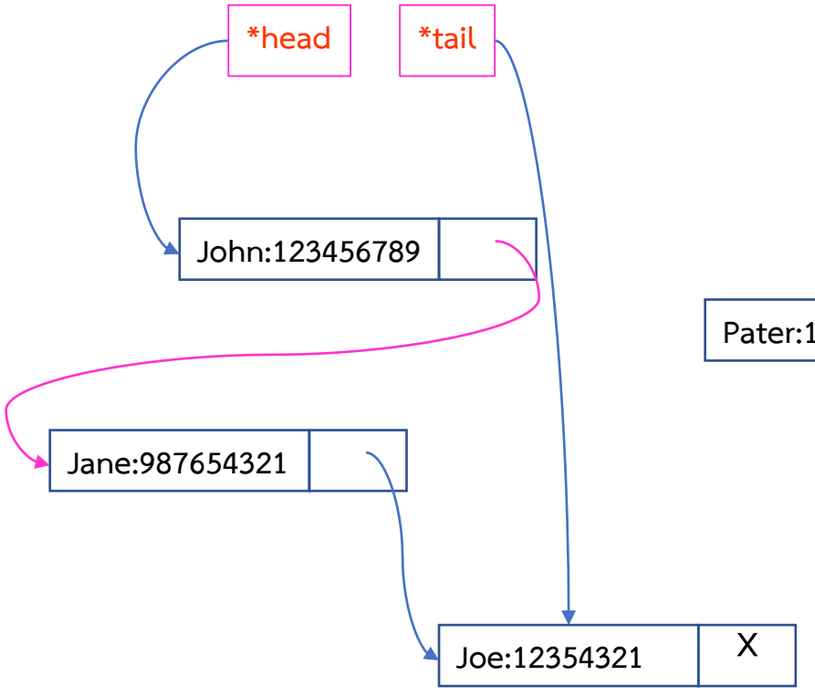


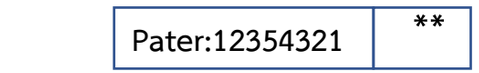
1 สร้าง Node ใหม่

2 ให้ link field ของ node ใหม่ไปชี้ head node

3 เปลี่ยน head node ไปชี้ node ใหม่

การเพิ่มข้อมูลลงใน Linked List การแทรก Node แรก





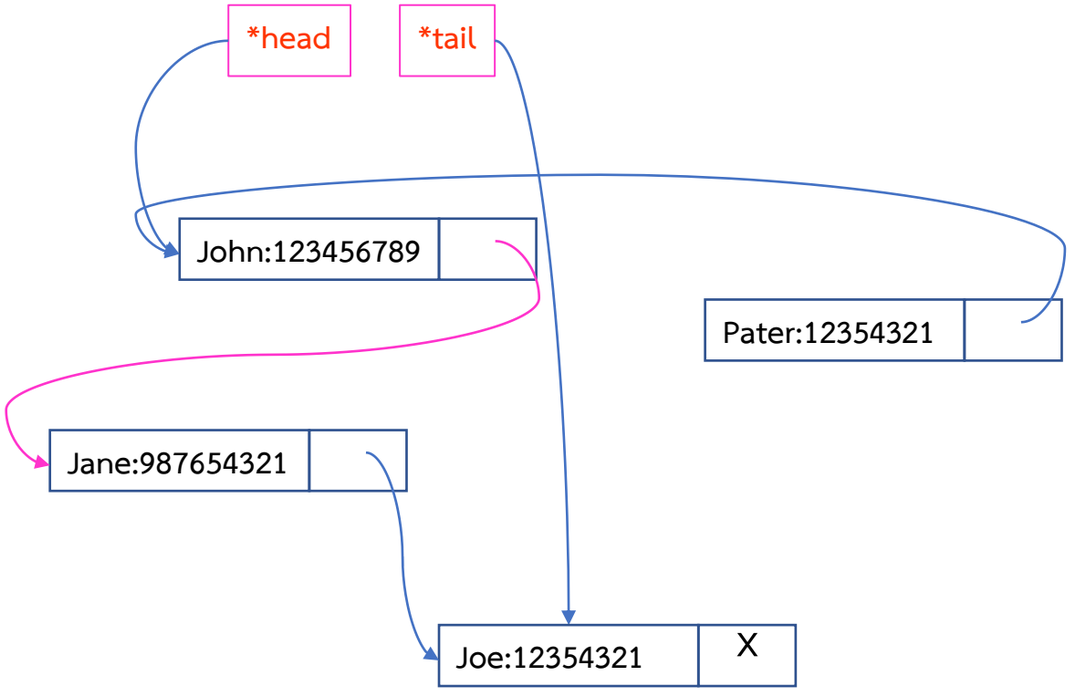

```
pt =(PhoneBook *)
malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Peter");
pt->phone=123454321;
pt->next=head;
head=pt;
```

1 สร้าง Node ใหม่

2 ให้ link field ของ node ใหม่ไปชี้ head node

3 เปลี่ยน head node ไปชี้ node ใหม่

การเพิ่มข้อมูลลงใน Linked List การแทรก Node แรก



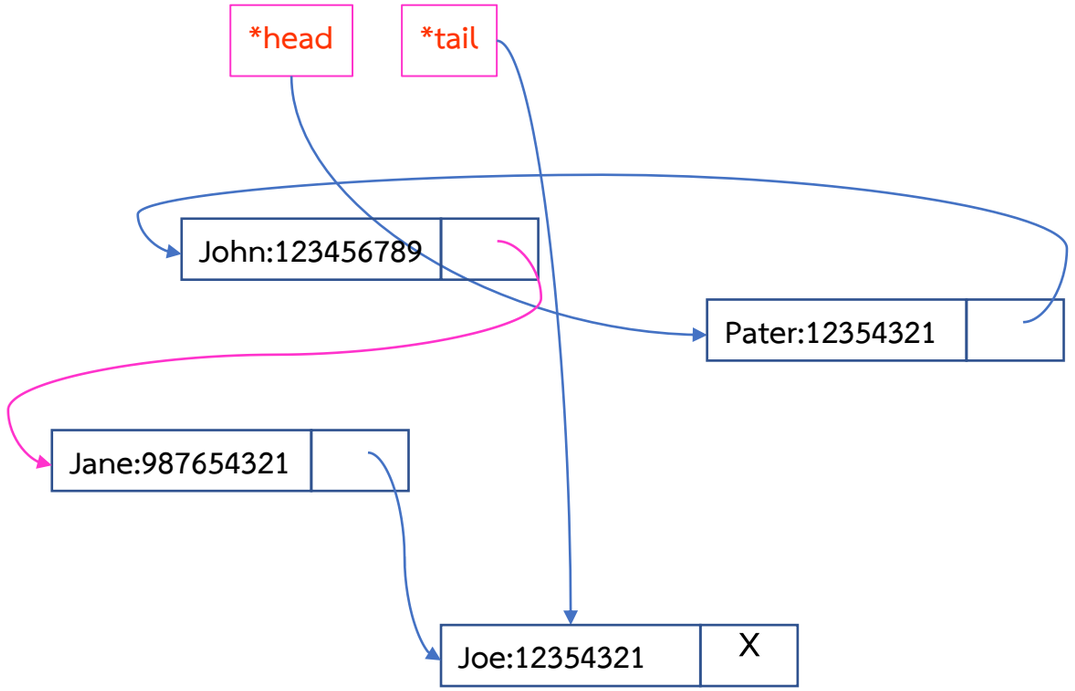

```
pt =(PhoneBook *)
malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Peter");
pt->phone=123454321;
pt->next=head;
head=pt;
```

1 สร้าง Node ใหม่

2 ให้ link field ของ node ใหม่ไปชี้ head node

3 เปลี่ยน head node ไปชี้ node ใหม่

การเพิ่มข้อมูลลงใน Linked List การแทรก Node แรก

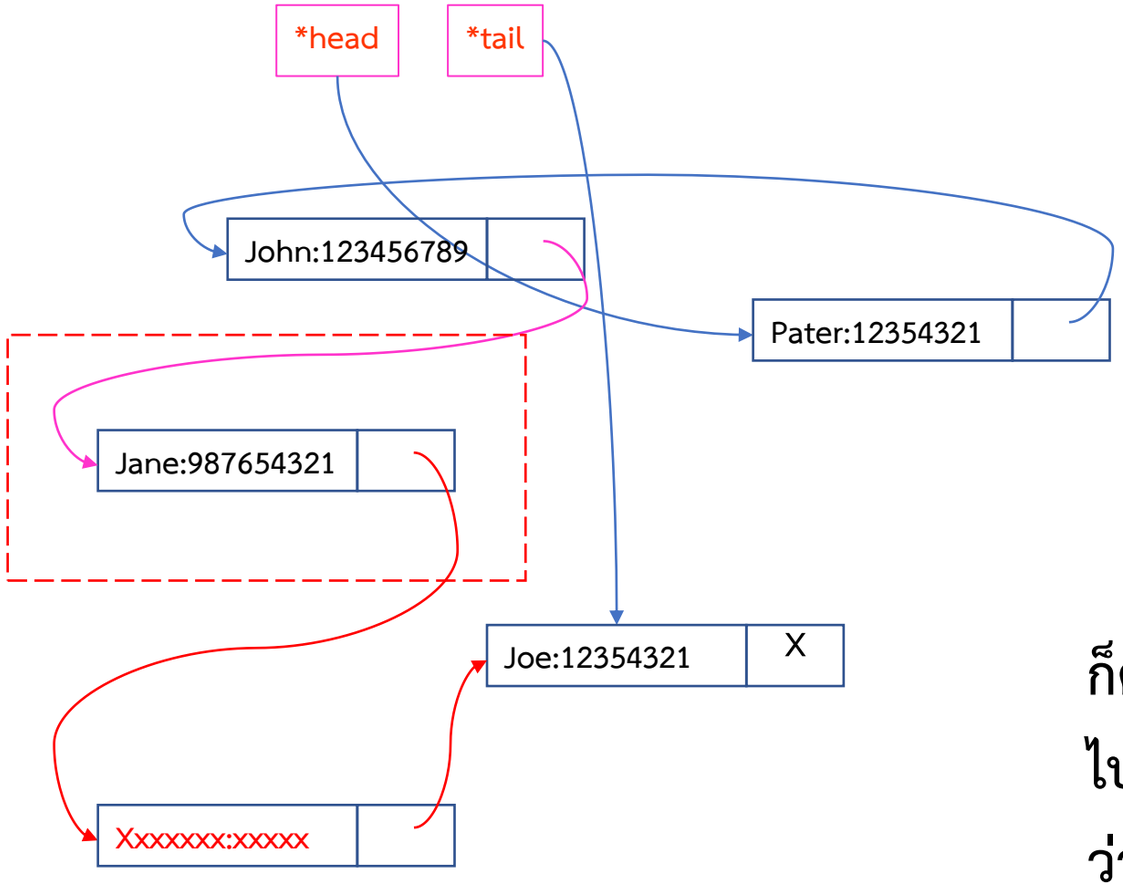



```
pt =(PhoneBook *)
malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Peter");
pt->phone=123454321;
pt->next=head;
head=pt;
```

ความซับซ้อนทางเวลาของการแทรก node ไว้ด้านหน้าสุดคือ

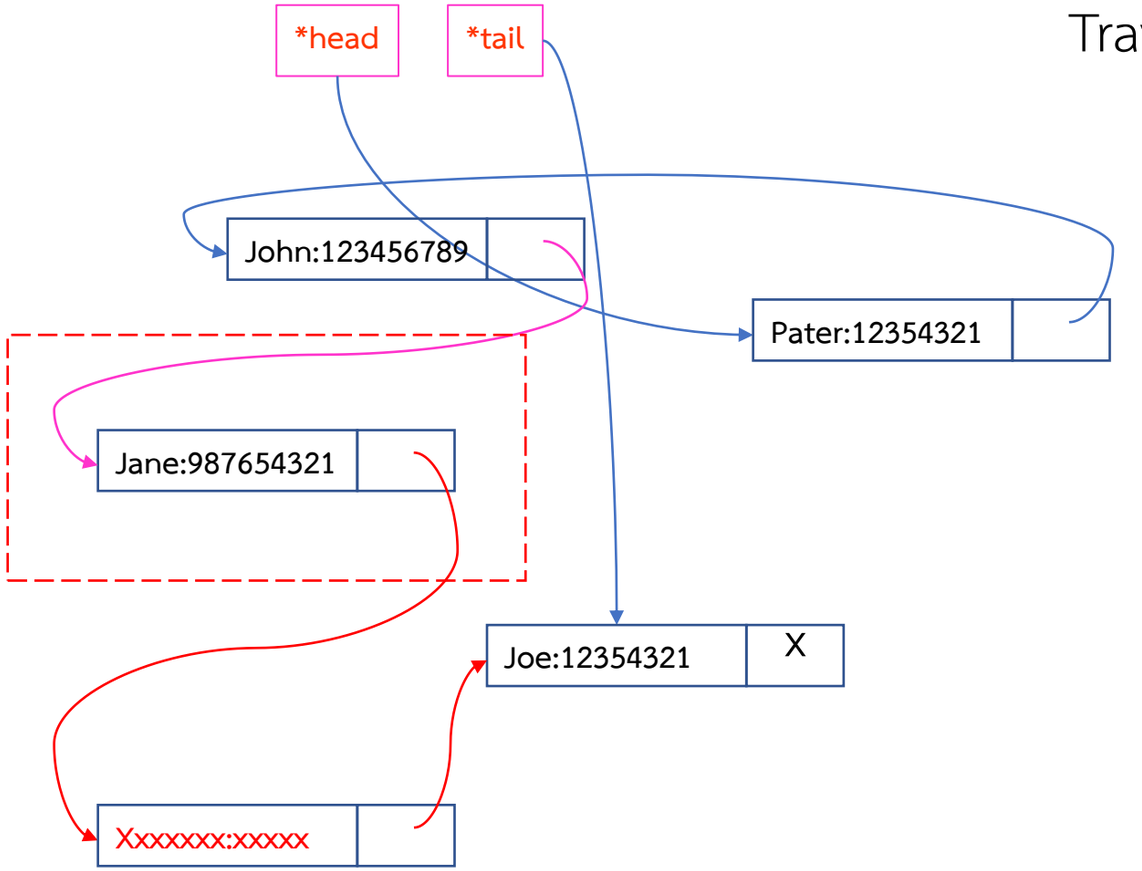
O(1)

การเพิ่มข้อมูลลงใน Linked List



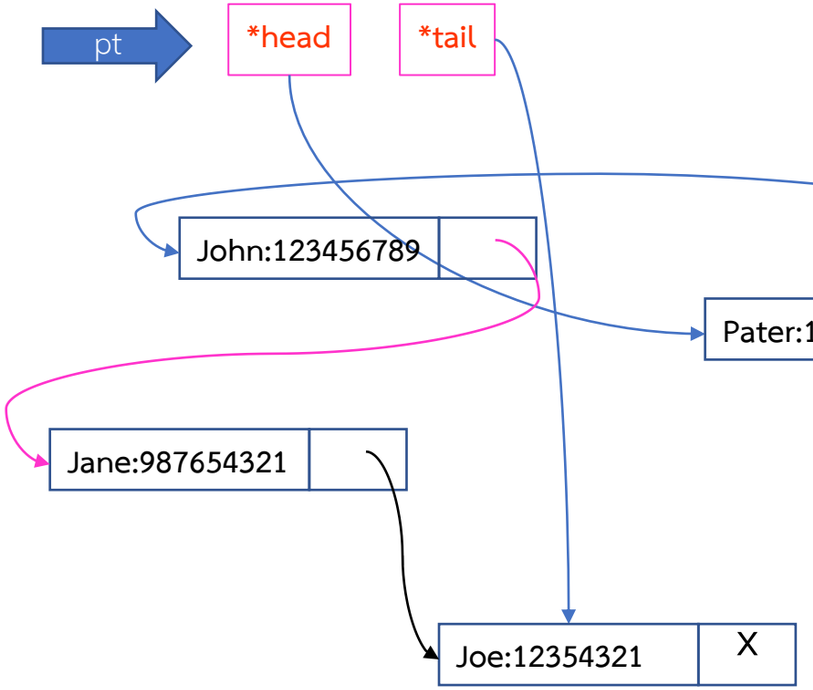
หากต้องการแทรกหน้า Node สุดท้าย จะทำอย่างไร ?

ก็ต้อง Traverse ตั้งแต่ head ไปจนถึง node สุดท้าย เพื่อหาว่ามัน link มาจาก node ไหน



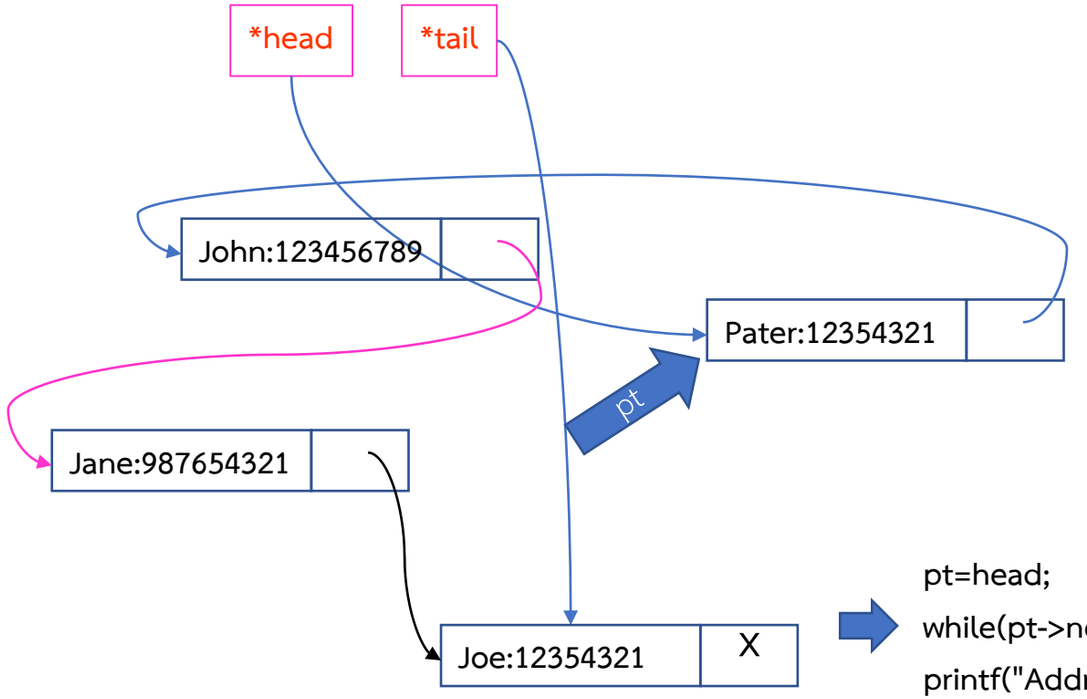
Traverse ทำอย่างไร ?

การ Traverse



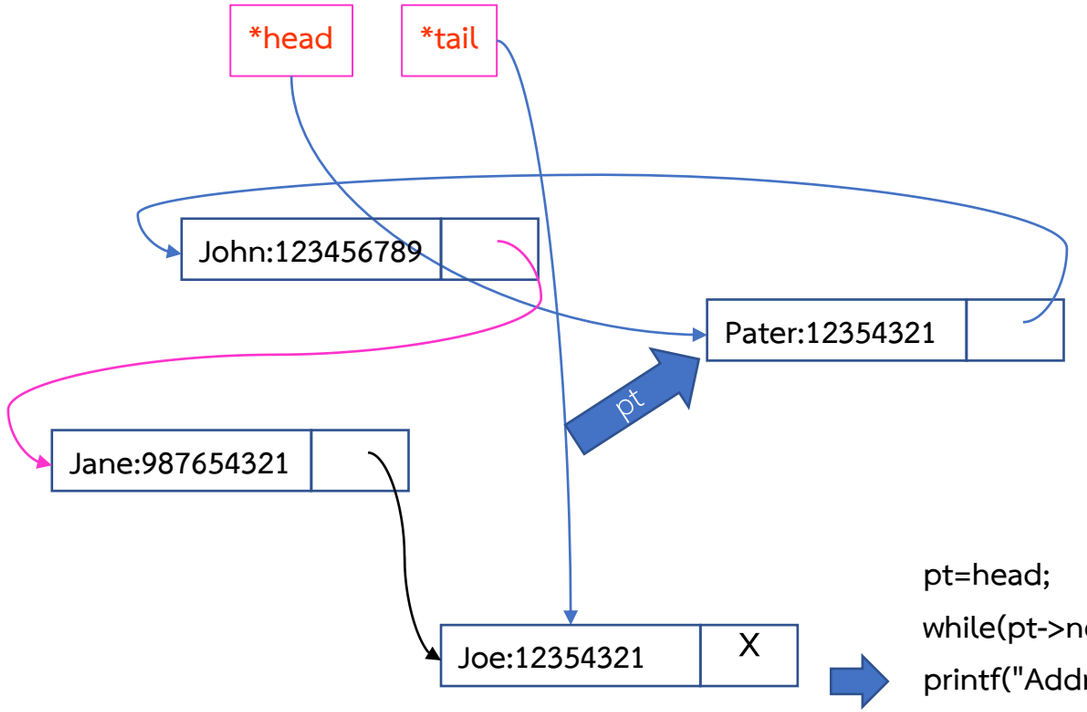

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse




```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

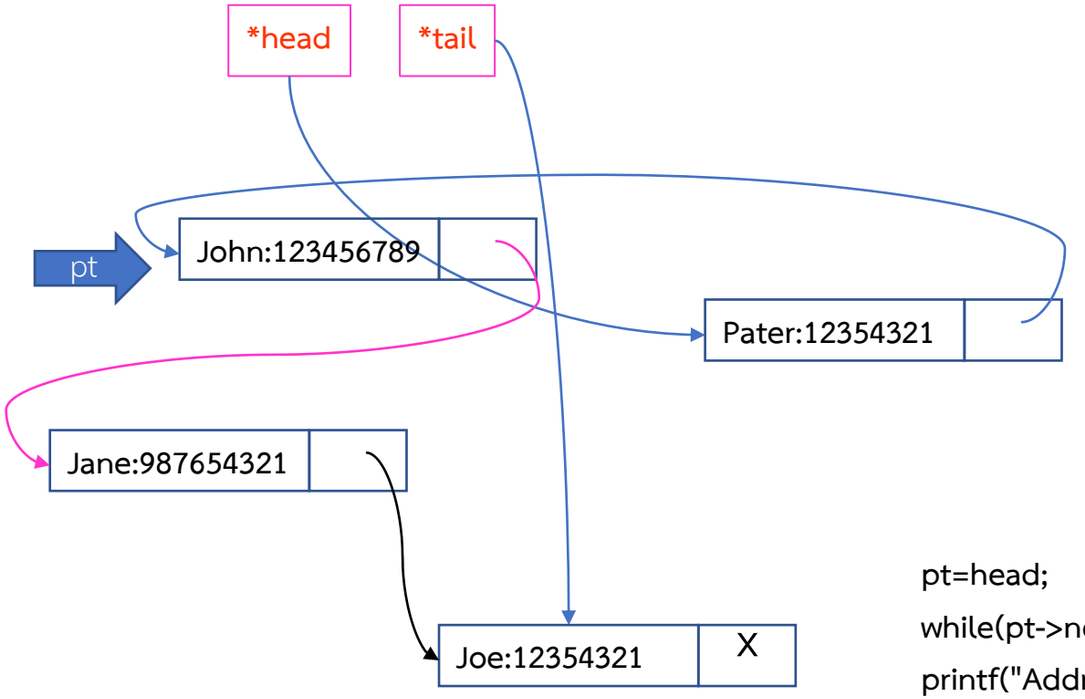
การ Traverse



Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

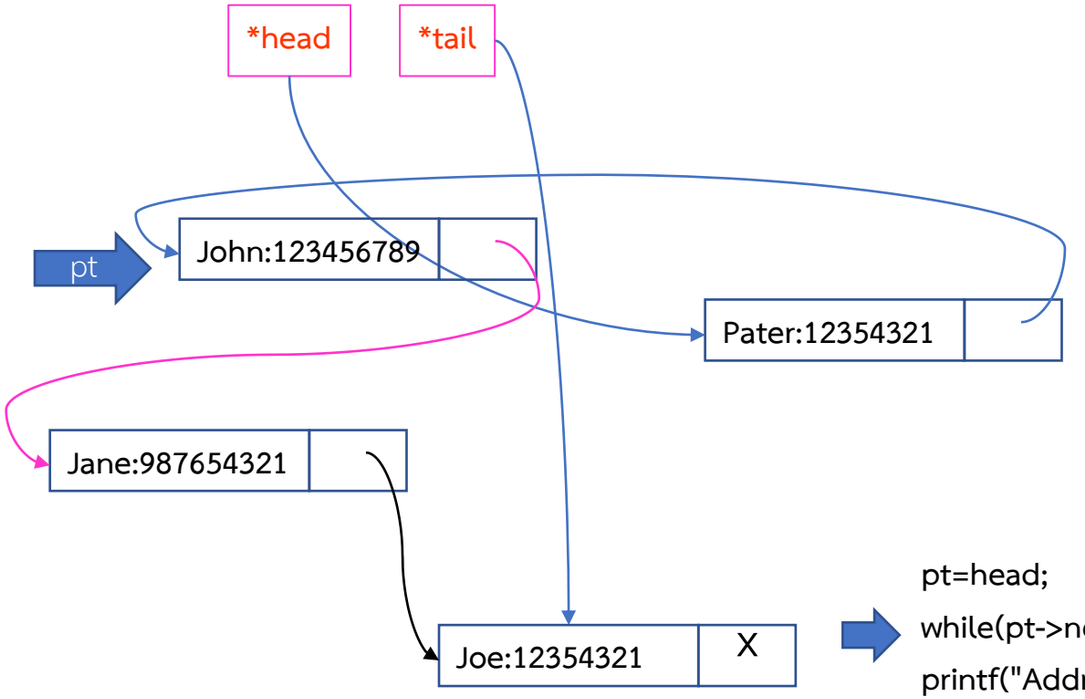
การ Traverse



Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

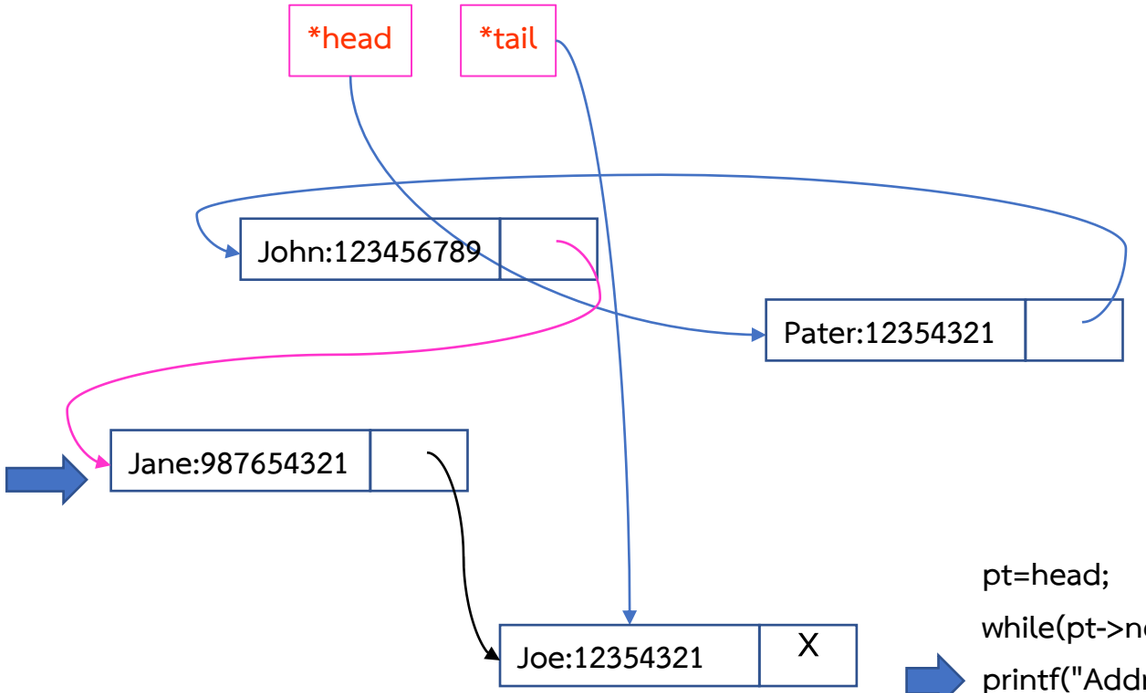
การ Traverse



Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse

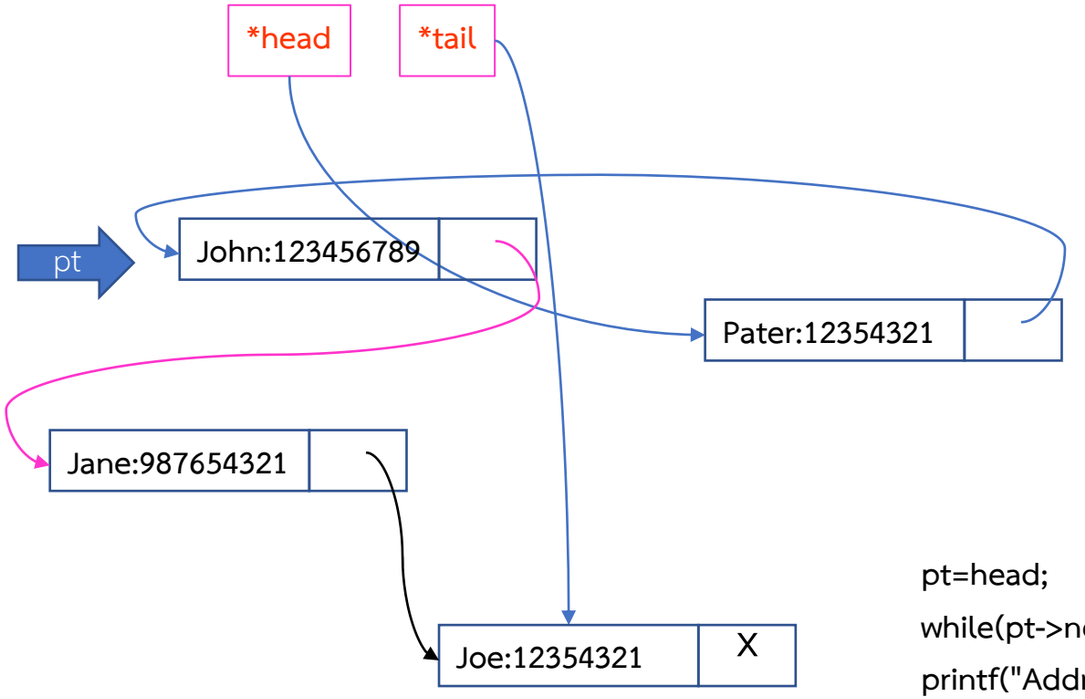


Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse

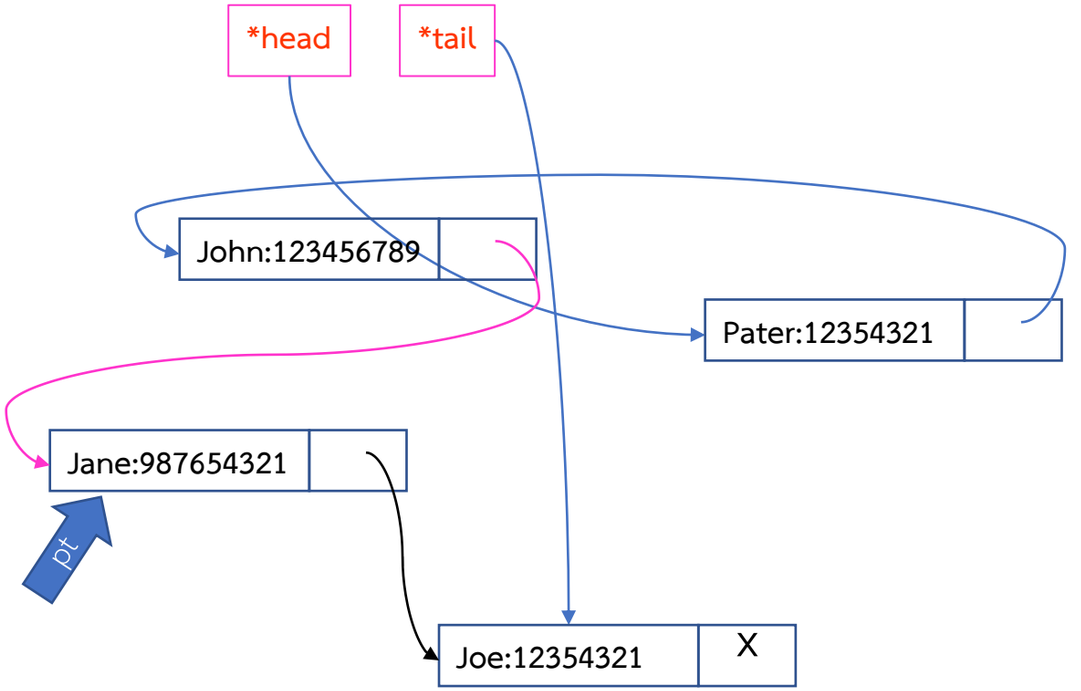


Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse

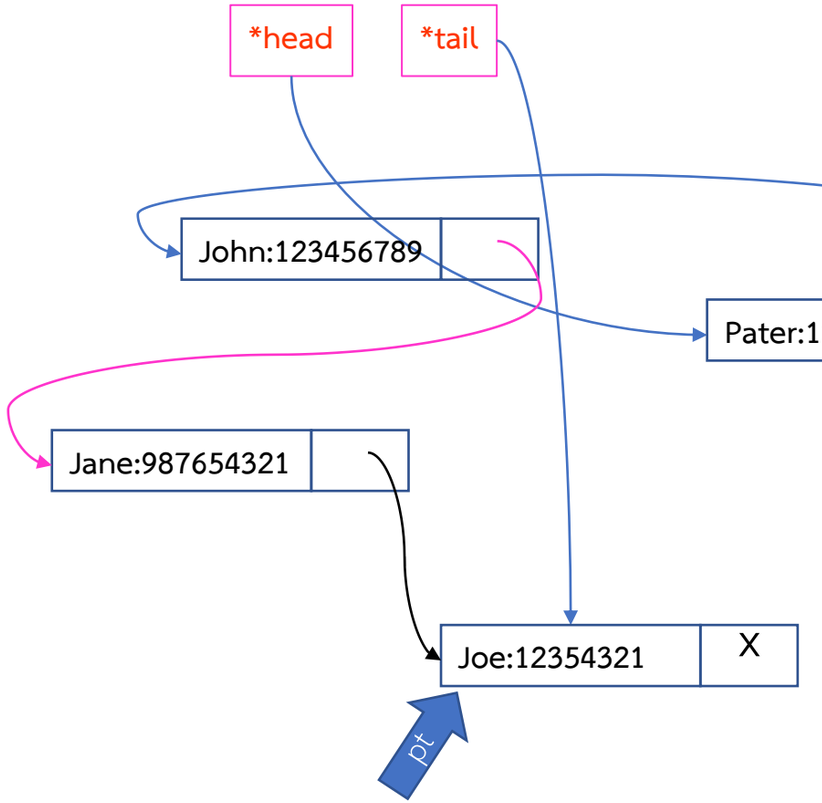


Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430
Address:00000000001C1430
Name:Jane
Phone:987654321
Next:00000000001C1460

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse

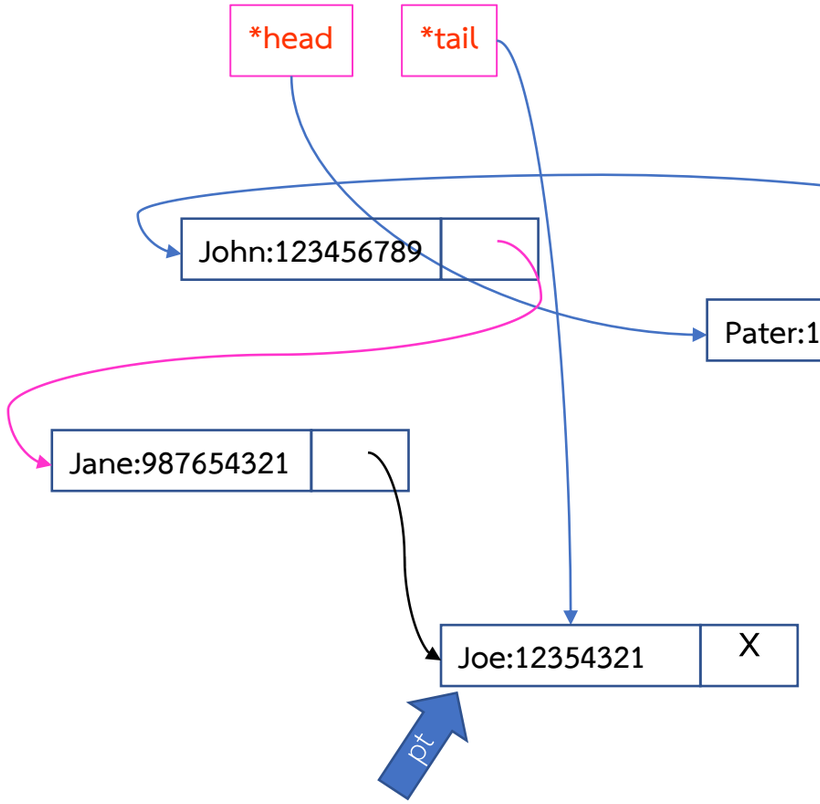


Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430
Address:00000000001C1430
Name:Jane
Phone:987654321
Next:00000000001C1460

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse

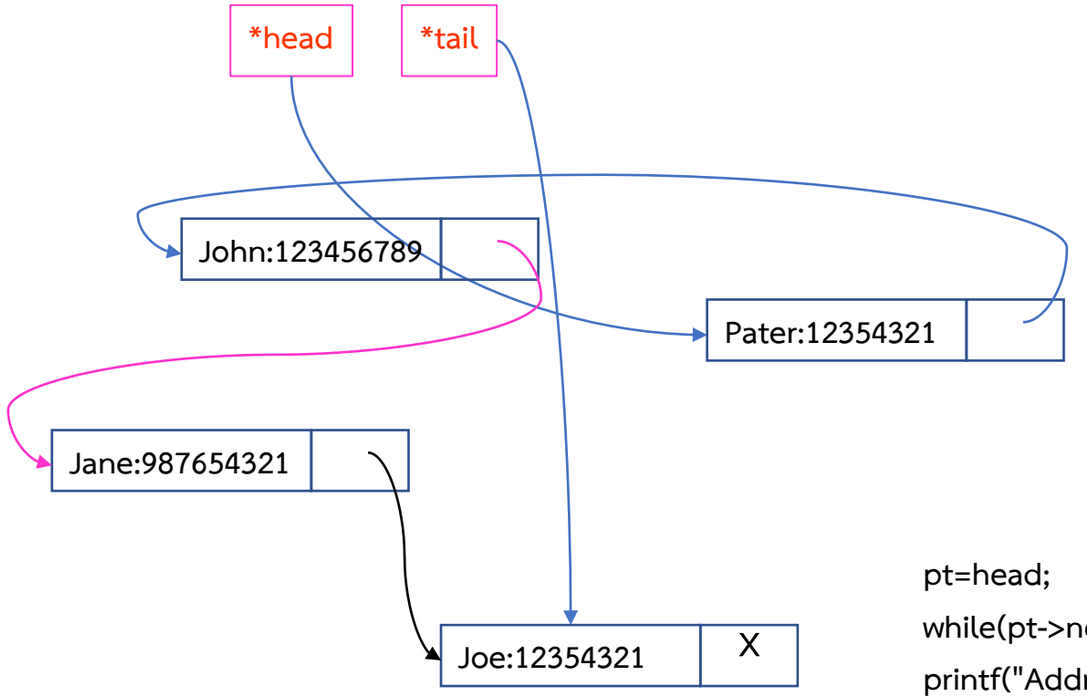


Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430
Address:00000000001C1430
Name:Jane
Phone:987654321
Next:00000000001C1460
Address:00000000001C1460
Name:Joe
Phone:123454321
Next:0000000000000000

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

การ Traverse



Address:00000000001C1490
Name:Peter
Phone:123454321
Next:00000000001C1400

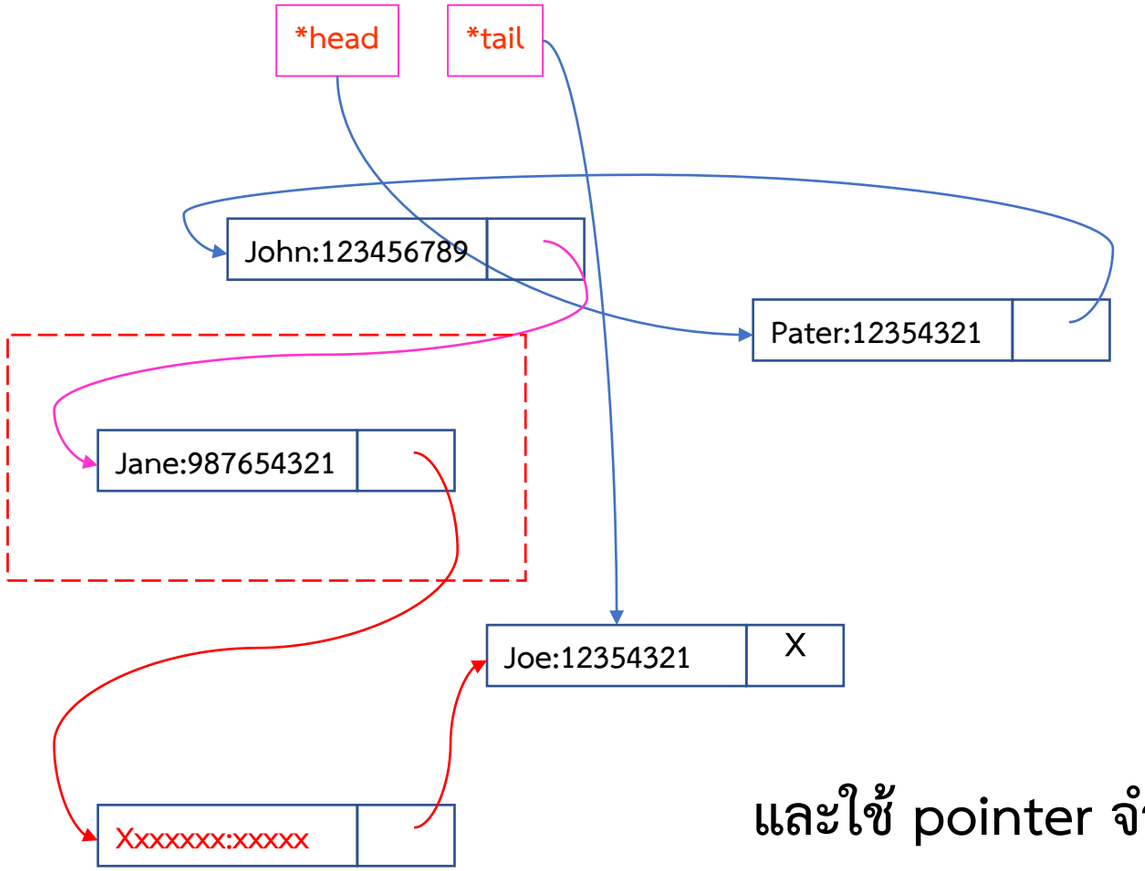
Address:00000000001C1400
Name:John
Phone:123456789
Next:00000000001C1430
Address:00000000001C1430
Name:Jane
Phone:987654321
Next:00000000001C1460
Address:00000000001C1460
Name:Joe
Phone:123454321
Next:0000000000000000

```
pt=head;
while(pt->next!=0){
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n\n",
pt,pt->Name,pt->phone,pt->next);
pt=pt->next;
}
printf("Address:%p\nName:%s\nPhone:%ld\nNext:%p\n",pt,
pt->Name,pt->phone,pt->next);
}
```

ความซับซ้อนทางเวลาของการ Traverse คือ

O(N)

การเพิ่มข้อมูลลงใน Linked List



หากต้องการแทรกหน้า Node สุดท้าย จะทำอย่างไร ?

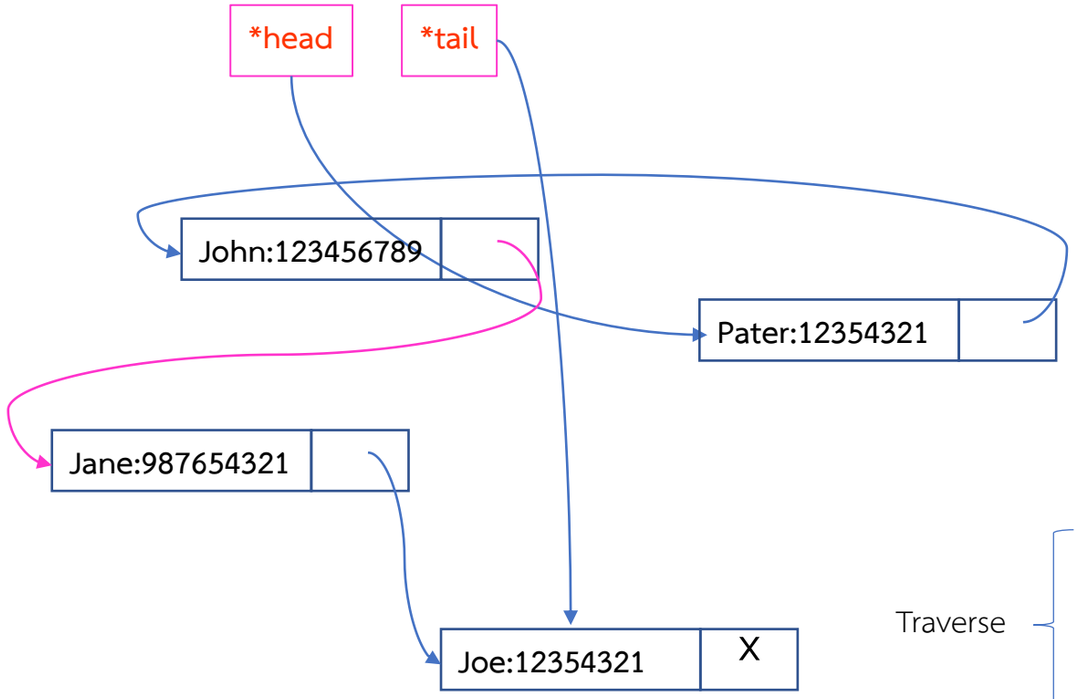
ก็ต้อง Traverse ตั้งแต่ head ไปจนถึง node สุดท้าย เพื่อหาว่ามัน link มาจาก node ไหน

และใช้ pointer จำนวน 2 ตัวในการ Traverse

ตัวแรกเก็บ address ของ node ปัจจุบัน

ตัวที่สอง เก็บ address ของ node ก่อนหน้า

การเพิ่มข้อมูลลงใน Linked List



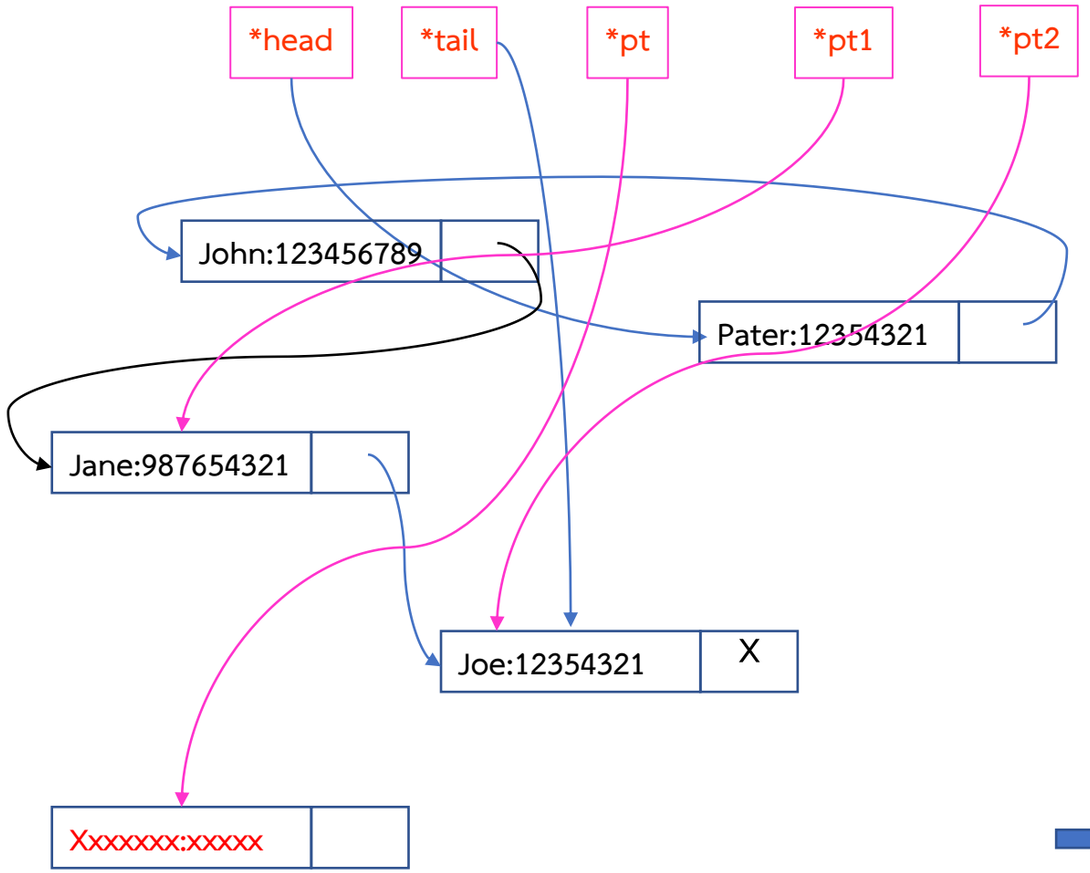
Traverse

```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
```

แทรก

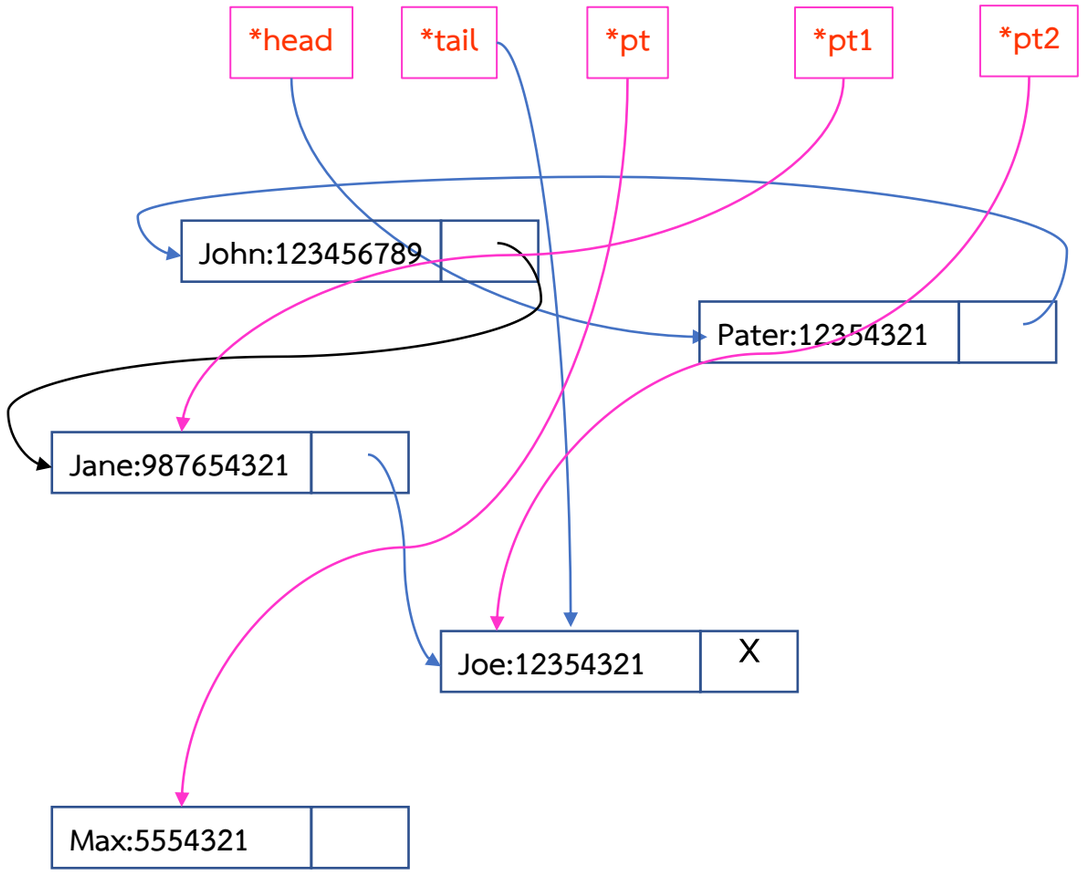
```
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

การเพิ่มข้อมูลลงใน Linked List



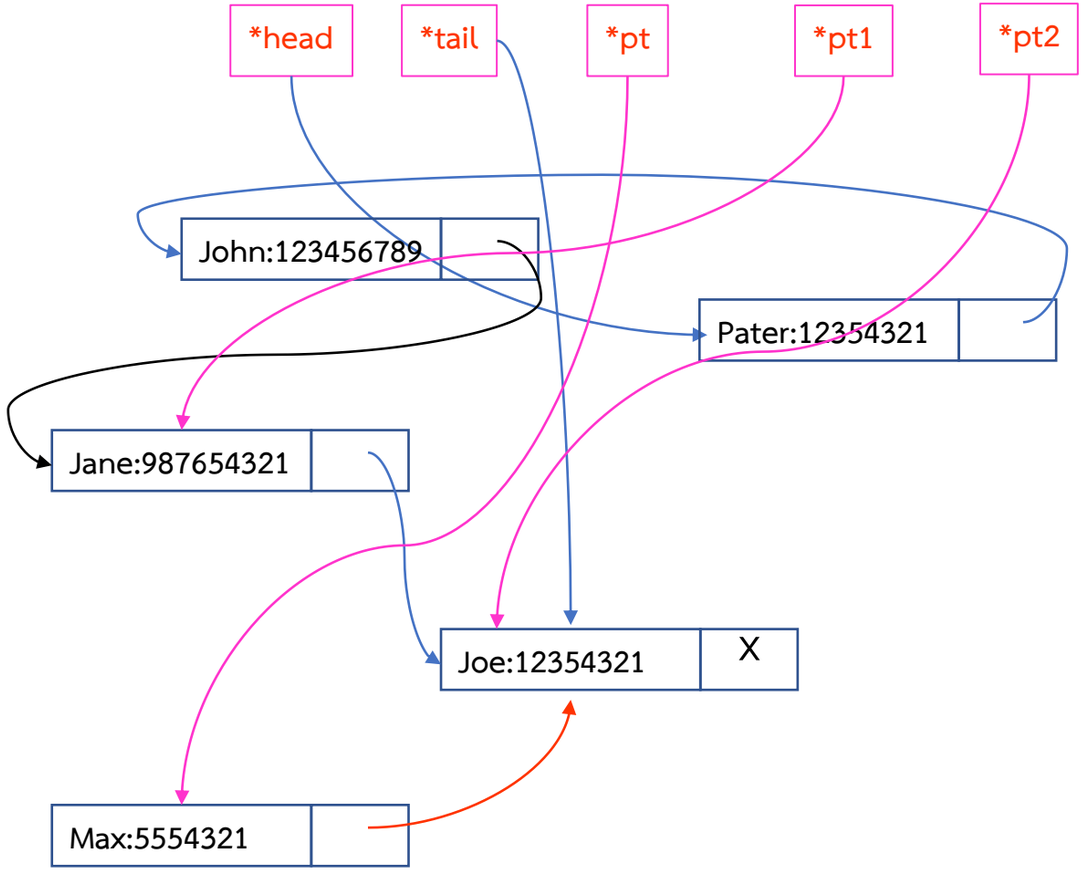

```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

การเพิ่มข้อมูลลงใน Linked List



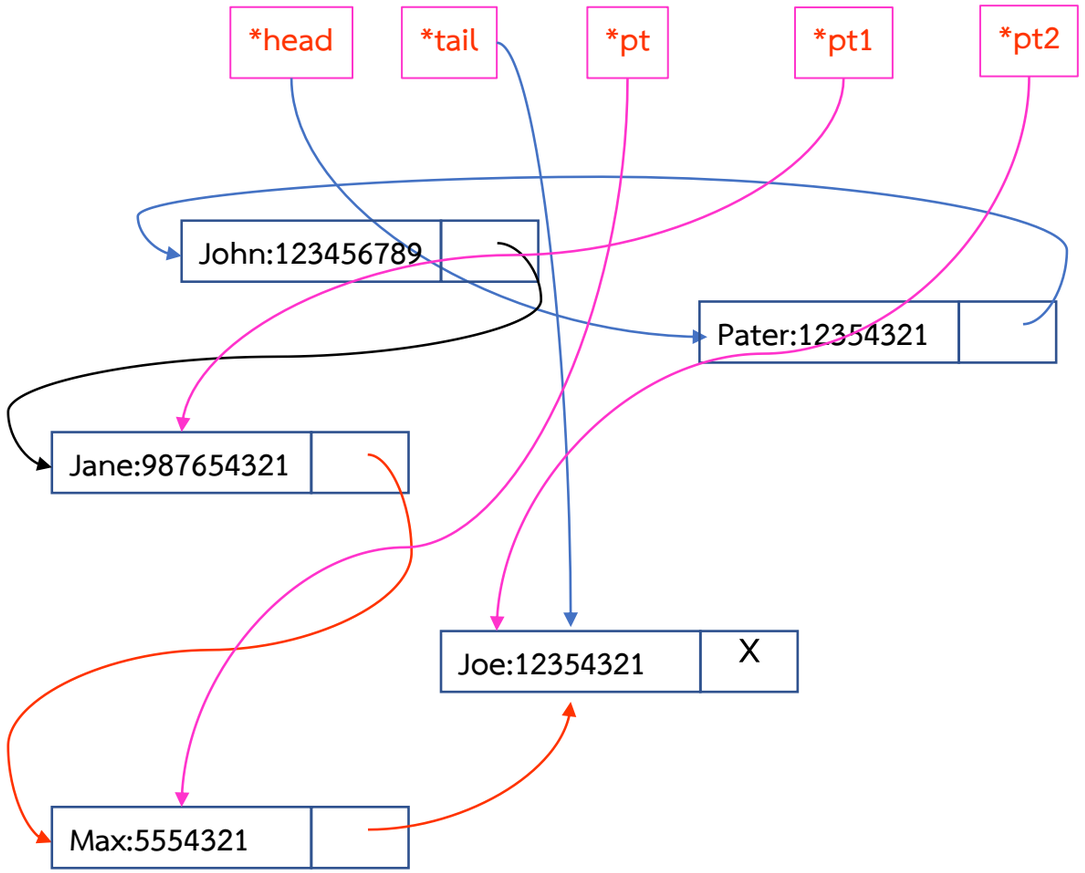

```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

การเพิ่มข้อมูลลงใน Linked List



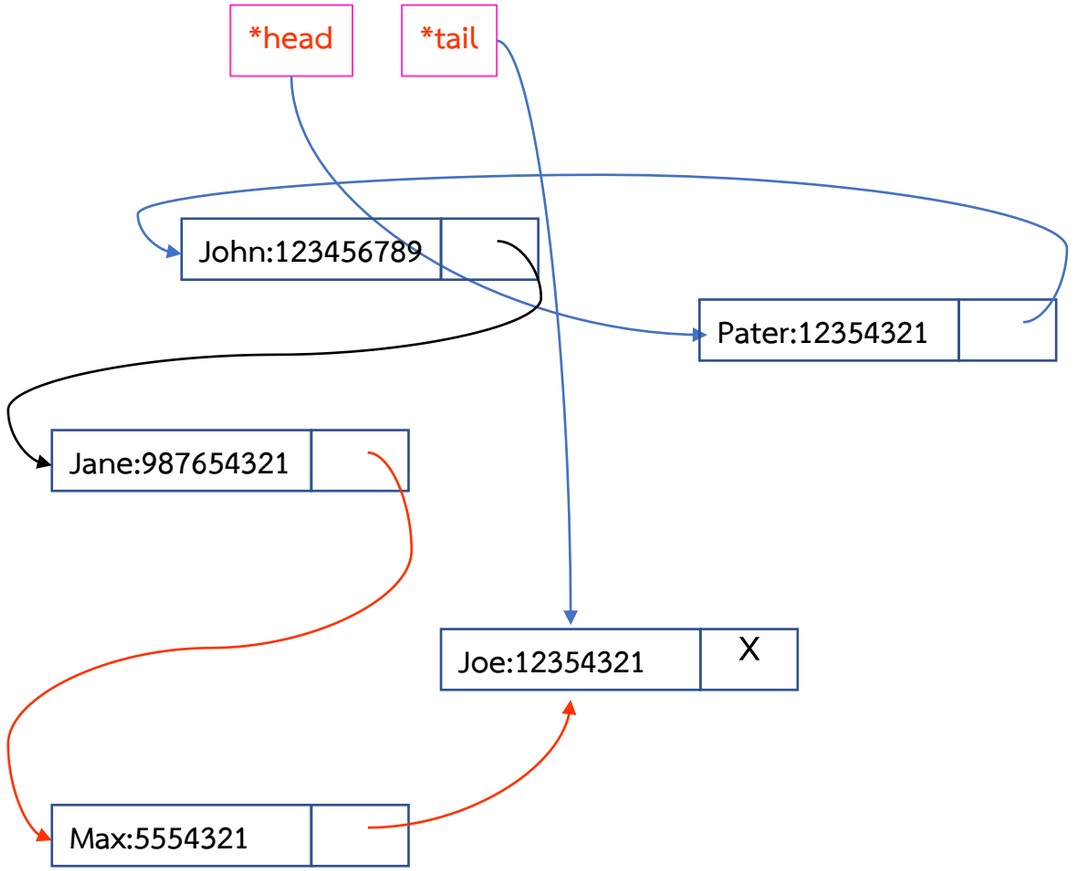

```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

การเพิ่มข้อมูลลงใน Linked List




```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

การเพิ่มข้อมูลลงใน Linked List

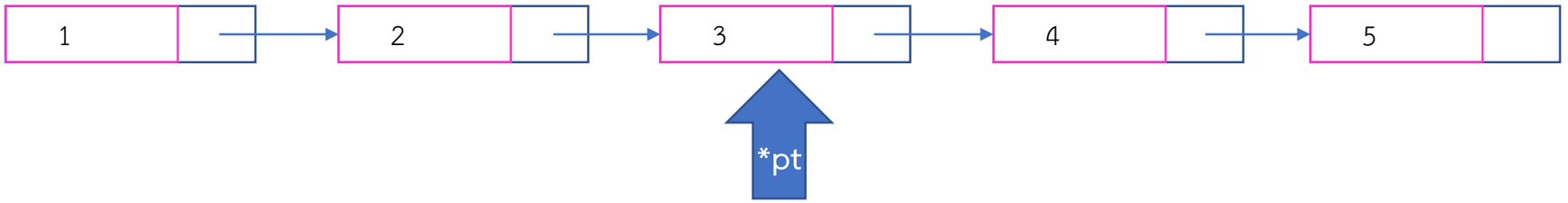



```
PhoneBook *head, *tail,*pt,*pt1,*pt2;
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}
pt =(PhoneBook *) malloc(sizeof(PhoneBook));
strcpy(pt->Name,"Max");
pt->phone=5554321;
pt->next=pt2;
pt1->next=pt;
```

ความซับซ้อนทางเวลาของการแทรก node ไว้ด้านหน้าตัวสุดท้ายคือ

O(N)

Linked List: Singly Linked List

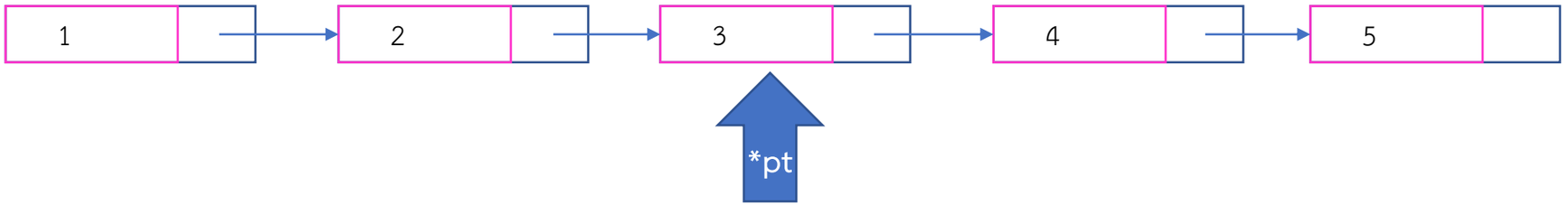


การแทรก Node จากตำแหน่งปัจจุบันจะมี 2 ประเด็นคือ

1 การแทรกทางด้านขวา สามารถทำได้ทันที เนื่องจากมีข้อมูล address ของ node ถัดไป ความซับซ้อนทางเวลาจึงเป็น O(1)

2 การแทรกทางด้านซ้าย ต้อง Traverse ก่อนเพื่อหา Address ของ node ก่อนหน้า ความซับซ้อนทางเวลาจึงเป็น O(N)

Linked List: Singly Linked List



การลบ Node จะมี 4 ประเด็นคือ

1 ลบตัวแรก (Head)

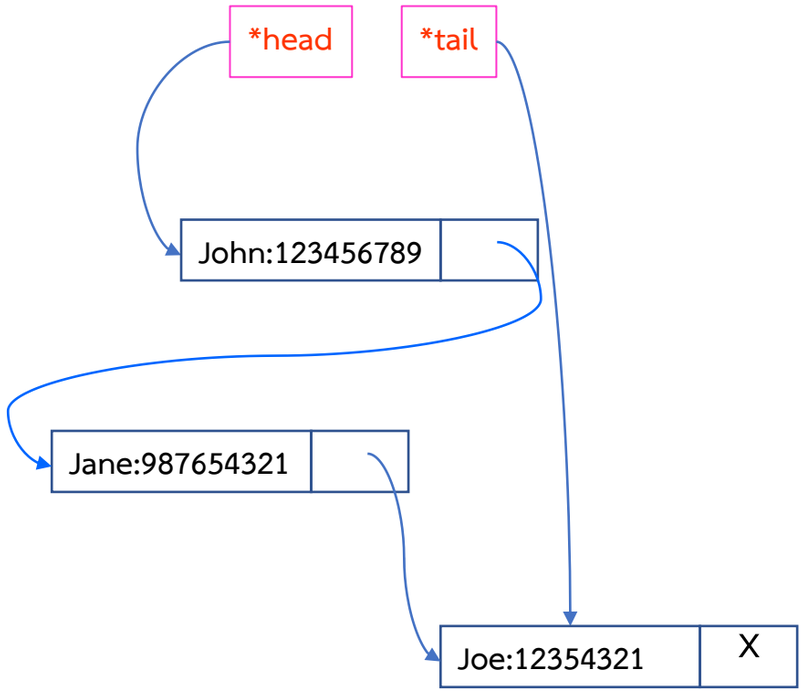
2 ลบตัวสุดท้าย (Tail)

3 ลบตัวที่อยู่ด้านขวา

4 ลบตัวปัจจุบัน

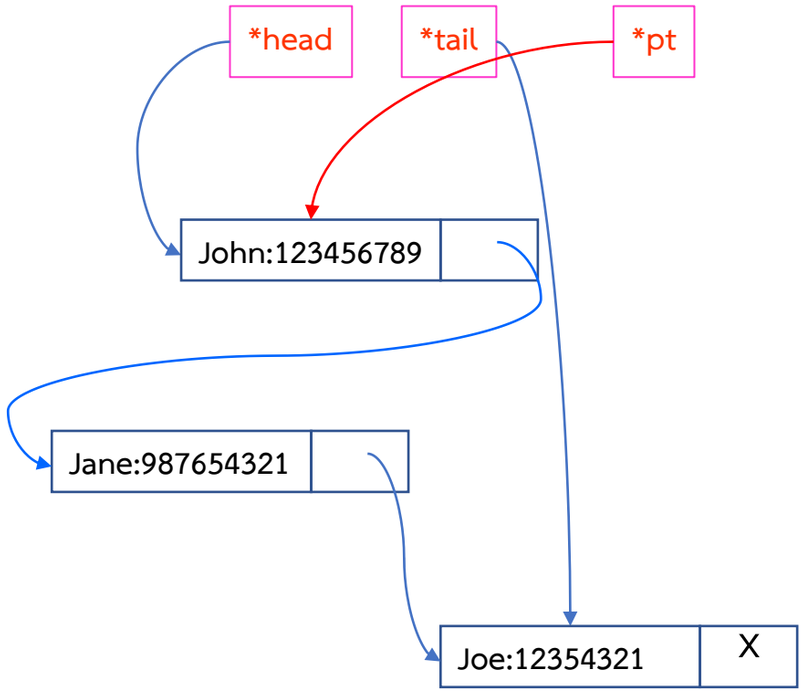
Linked List: Singly Linked List

การลบ Node : ลบตัวแรก



1) ย้าย *head มาชี้ที่ link ของ node แรก
2) คืนพื้นที่ของ node แรกให้กับหน่วยความจำ

การลบ Node : ลบตัวแรก

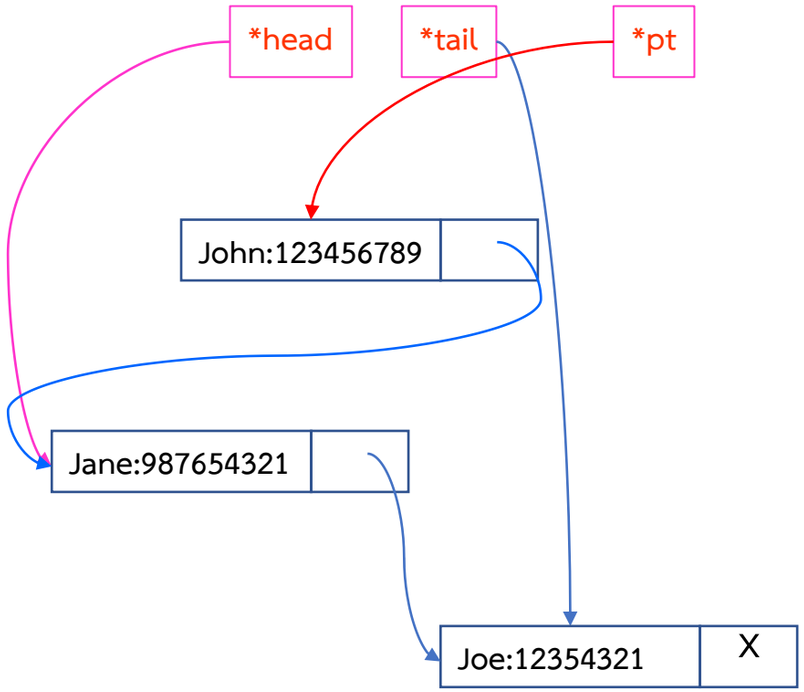


1) ย้าย *head มาชี้ที่ link ของ node แรก
2) คืนพื้นที่ของ node แรกให้กับหน่วยความจำ

```
pt=head;
head=pt->next;
free(pt);
```

Linked List: Singly Linked List

การลบ Node : ลบตัวแรก

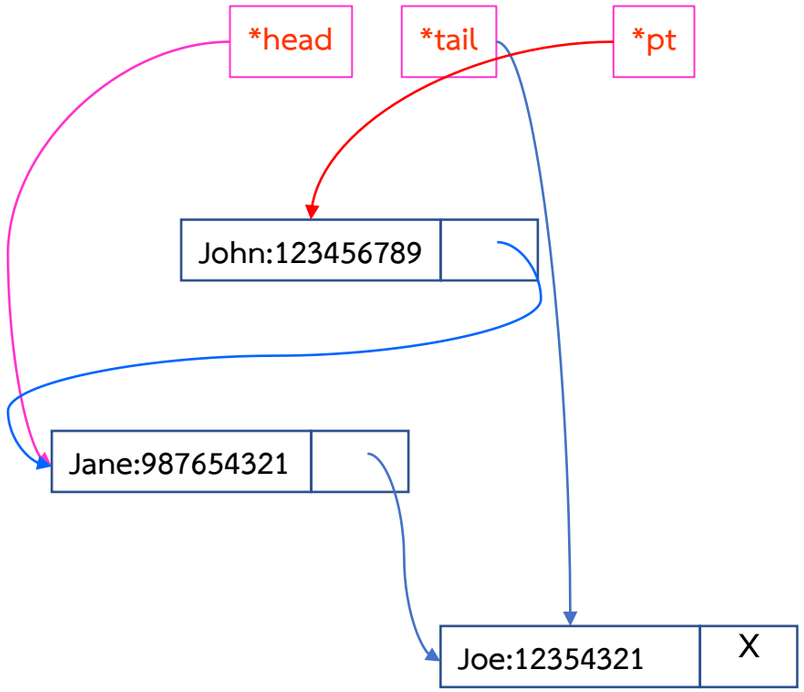


1) ย้าย *head มาชี้ที่ link ของ node แรก
2) คืนพื้นที่ของ node แรกให้กับหน่วยความจำ

```
pt=head;
head=pt->next;
free(pt);
```

Linked List: Singly Linked List

การลบ Node : ลบตัวแรก

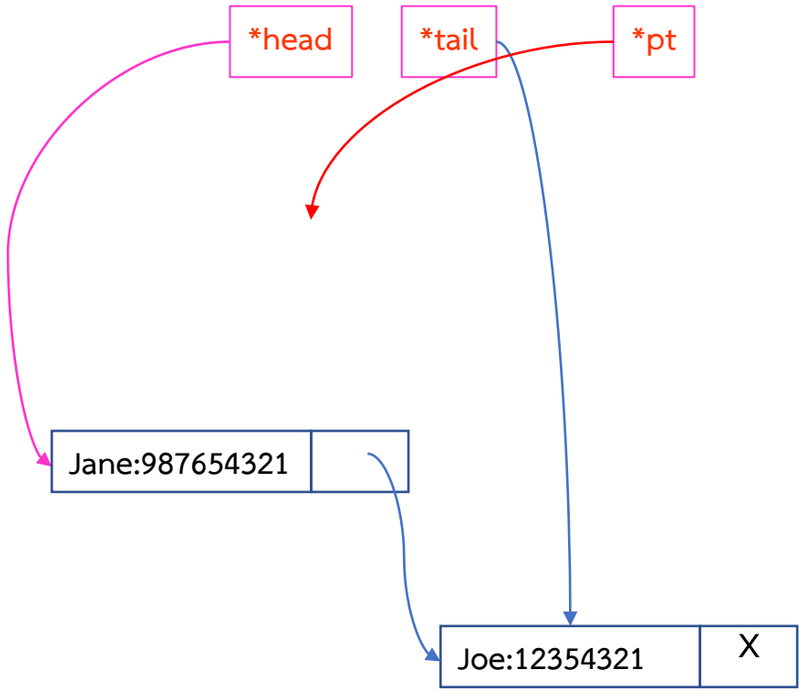


1) ย้าย *head มาชี้ที่ link ของ node แรก
2) คืนพื้นที่ของ node แรกให้กับหน่วยความจำ

```
pt=head;
head=pt->next;
free(pt);
```

Linked List: Singly Linked List

การลบ Node : ลบตัวแรก



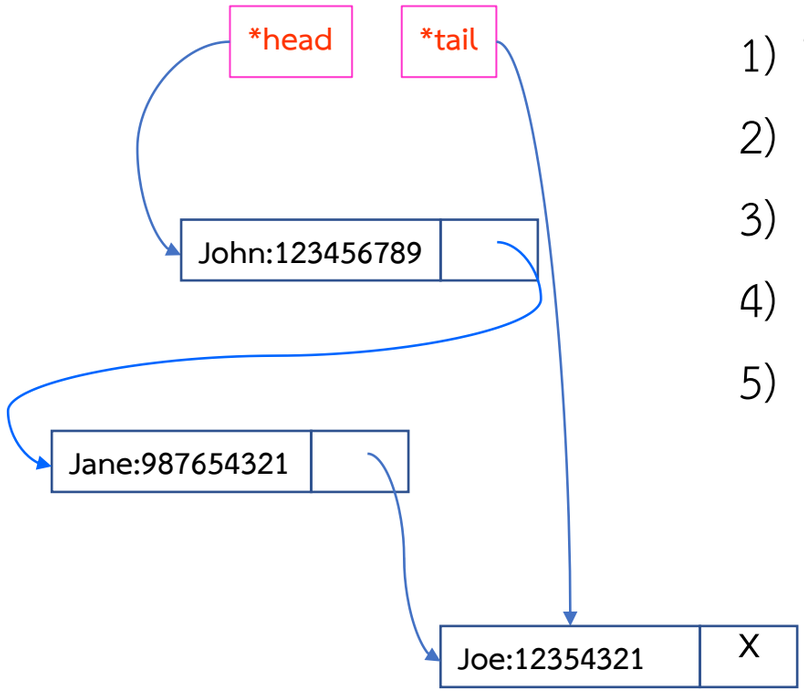
1) ย้าย *head มาชี้ที่ link ของ node แรก
2) คืนพื้นที่ของ node แรกให้กับหน่วยความจำ

```
pt=head;
head=pt->next;
free(pt);
```

O(1)

Linked List: Singly Linked List

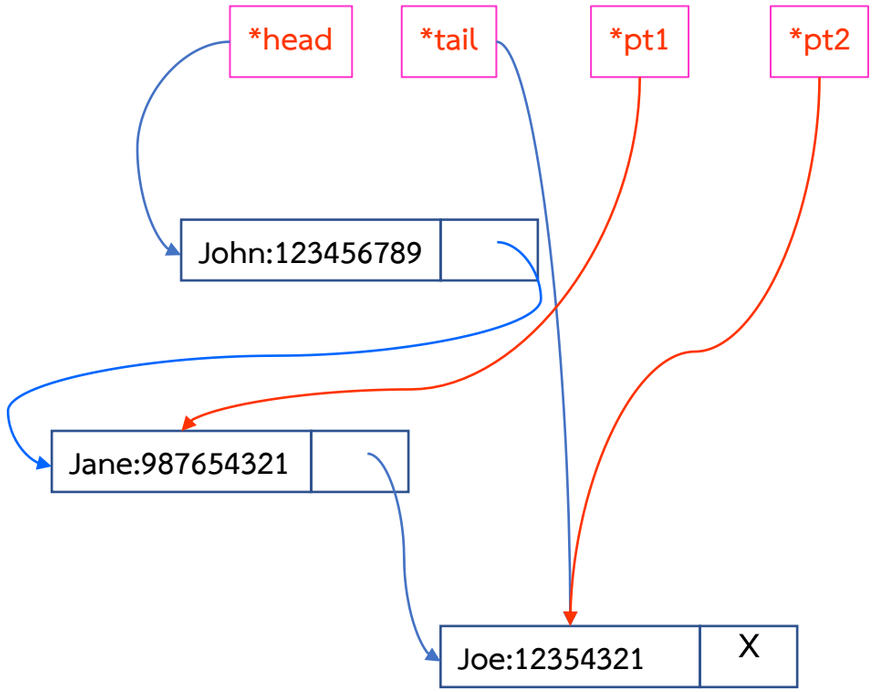
การลบ Node : ลบสุดท้าย



1) Traverse จาก node แรกไปยัง node สุดท้าย
2) เก็บ address ของ note ก่อนสุดท้าย และ node สุดท้าย
3) เปลี่ยน next ของ node ก่อนสุดท้ายเป็น null
4) เปลี่ยน *tail มาชี้ node ก่อนสุดท้าย
5) คืนพื้นที่ของ node สุดท้าย

Linked List: Singly Linked List

การลบ Node : ลบสุดท้าย

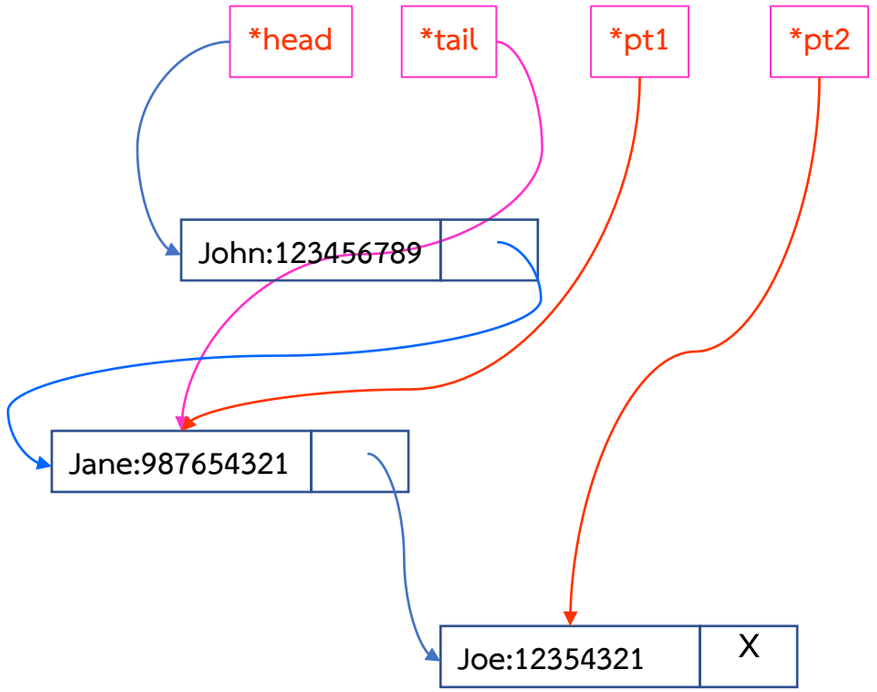



```
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}

tail=pt1;
pt1->next=0;
free(pt2);
```

1) Traverse จาก node แรกไปยัง node สุดท้าย
2) เก็บ address ของ note ก่อนสุดท้าย และ node สุดท้าย
3) เปลี่ยน next ของ node ก่อนสุดท้ายเป็น null
4) เปลี่ยน *tail มาชี้ node ก่อนสุดท้าย
5) คืนพื้นที่ของ node สุดท้าย

Linked List: Singly Linked List

การลบ Node : ลบสุดท้าย

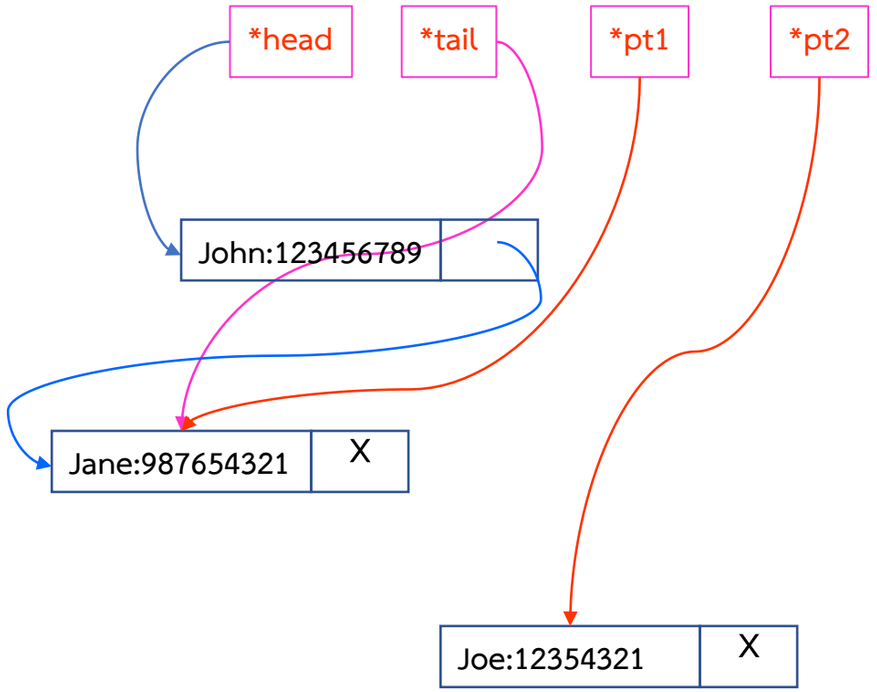



```
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}

tail=pt1;
pt1->next=0;
free(pt2);
```

1) Traverse จาก node แรกไปยัง node สุดท้าย
2) เก็บ address ของ note ก่อนสุดท้าย และ node สุดท้าย
3) เปลี่ยน next ของ node ก่อนสุดท้ายเป็น null
4) เปลี่ยน *tail มาชี้ node ก่อนสุดท้าย
5) คืนพื้นที่ของ node สุดท้าย

Linked List: Singly Linked List

การลบ Node : ลบสุดท้าย

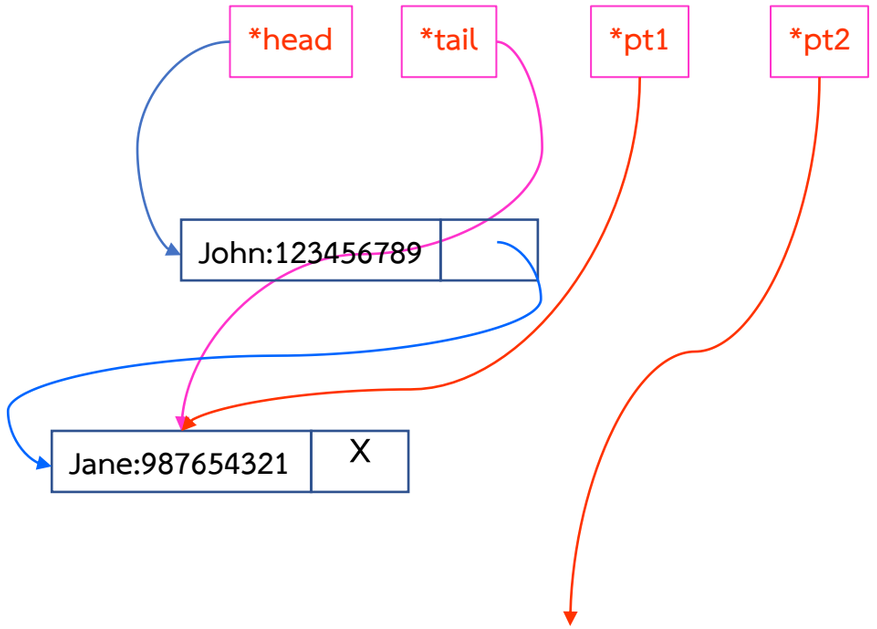



```
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}

tail=pt1;
pt1->next=0;
free(pt2);
```

1) Traverse จาก node แรกไปยัง node สุดท้าย
2) เก็บ address ของ note ก่อนสุดท้าย และ node สุดท้าย
3) เปลี่ยน next ของ node ก่อนสุดท้ายเป็น null
4) เปลี่ยน *tail มาชี้ node ก่อนสุดท้าย
5) คืนพื้นที่ของ node สุดท้าย

Linked List: Singly Linked List

การลบ Node : ลบสุดท้าย



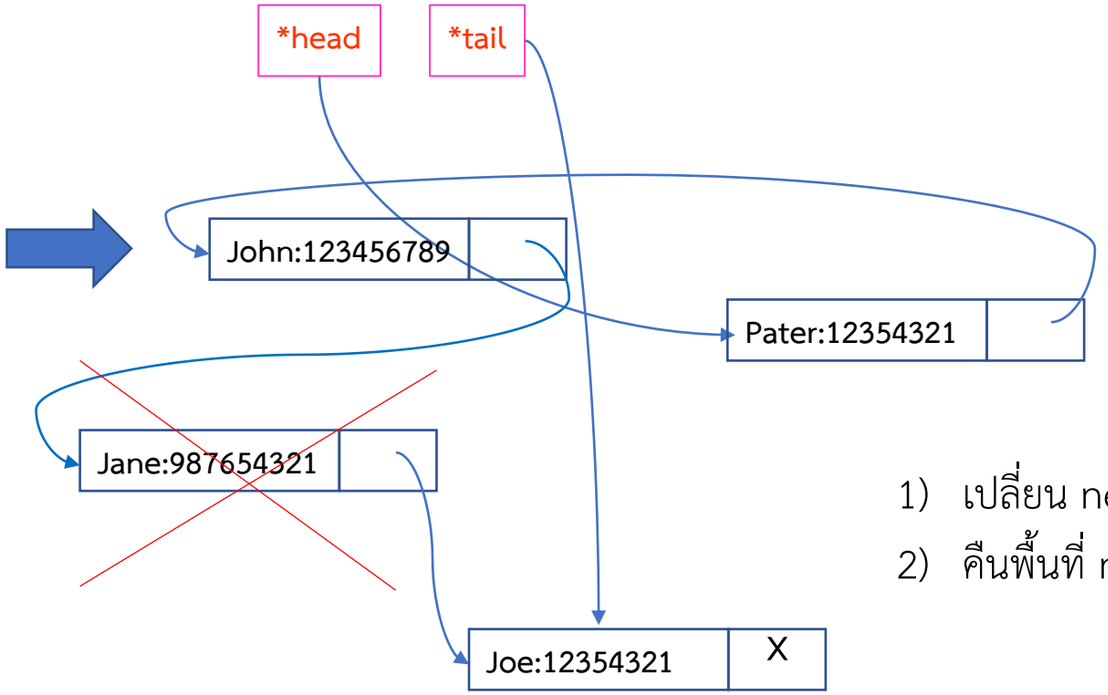

```
pt2=head;
while(pt2->next!=0){
pt1=pt2;
pt2=pt2->next;
}

tail=pt1;
pt1->next=0;
free(pt2);
```

1) Traverse จาก node แรกไปยัง node สุดท้าย
2) เก็บ address ของ note ก่อนสุดท้าย และ node สุดท้าย
3) เปลี่ยน next ของ node ก่อนสุดท้ายเป็น null
4) เปลี่ยน *tail มาชี้ node ก่อนสุดท้าย
5) คืนพื้นที่ของ node สุดท้าย

O(N)

Linked List: Singly Linked List

การลบ Node : ลบด้านขวา

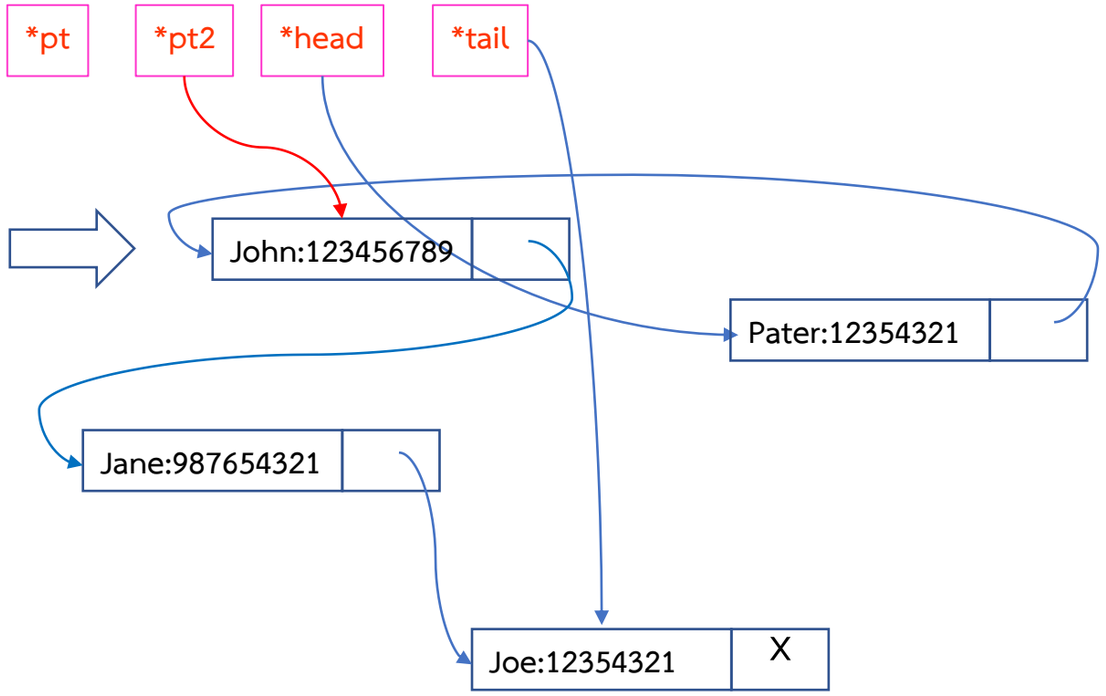


1) เปลี่ยน next ของตัวปัจจุบัน ไปชี้ยัง node ที่ต้องการลบชี้อยู่
2) คืนพื้นที่ node ที่ต้องการลบ

Linked List: Singly Linked List

การลบ Node : ลบด้านขวา

1) เปลี่ยน next ของตัวปัจจุบัน ไปชี้ยัง node ที่ต้องการลบชี้อยู่
2) คืนพื้นที่ node ที่ต้องการลบ



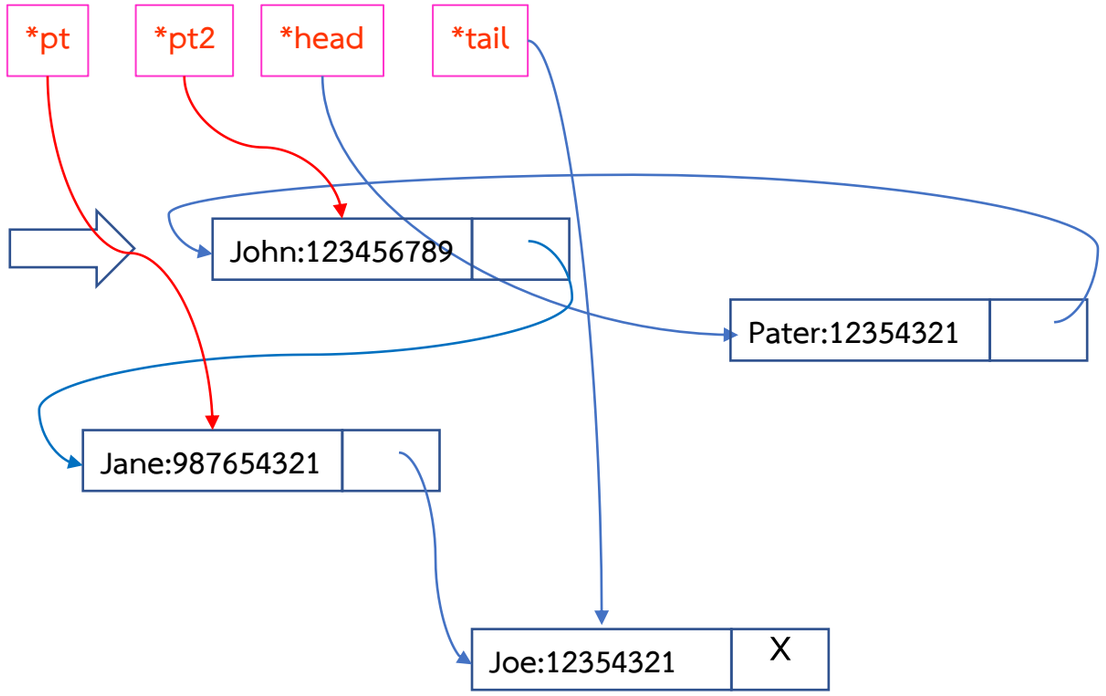

```
pt2=head;
while(pt2->next!=0){
if(!strcmp(pt2->Name,"John"))break;
pt2=pt2->next;
}
```

```
pt=pt2->next;
pt2->next=pt->next;
free(pt);
```

Linked List: Singly Linked List

การลบ Node : ลบด้านขวา

1) เปลี่ยน next ของตัวปัจจุบัน ไปชี้ยัง node ที่ต้องการลบชี้อยู่
2) คืนพื้นที่ node ที่ต้องการลบ



pt=pt2->next;

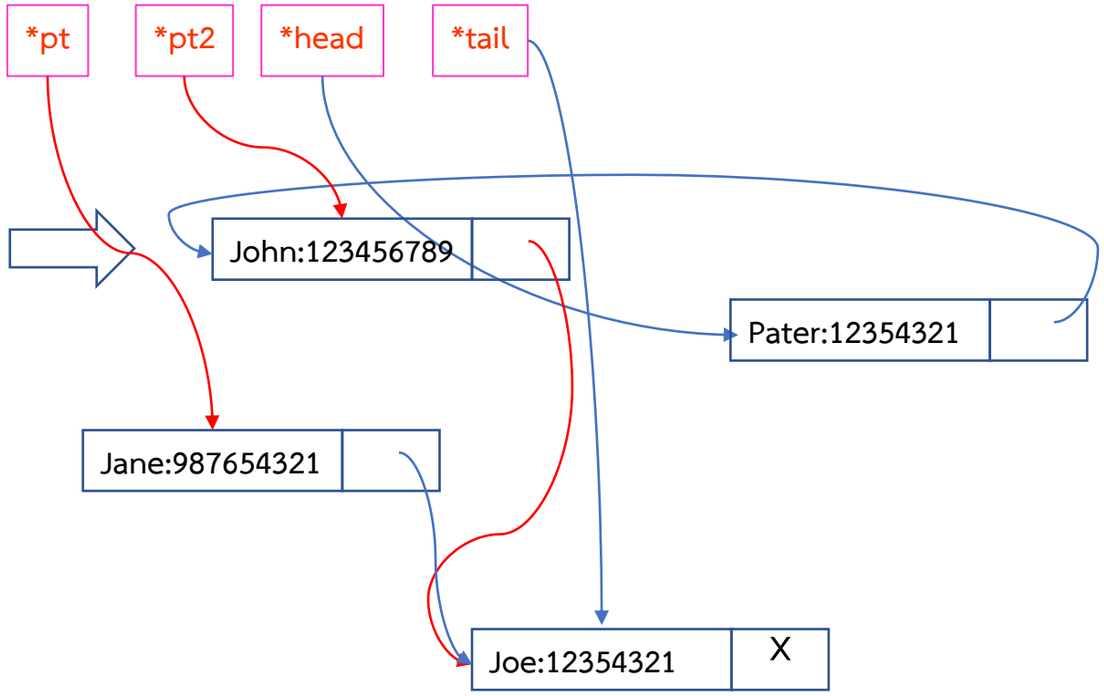
pt2->next=pt->next;

free(pt);

Linked List: Singly Linked List

การลบ Node : ลบด้านขวา

1) เปลี่ยน next ของตัวปัจจุบัน ไปชี้ยัง node ที่ต้องการลบชี้อยู่
2) คืนพื้นที่ node ที่ต้องการลบ

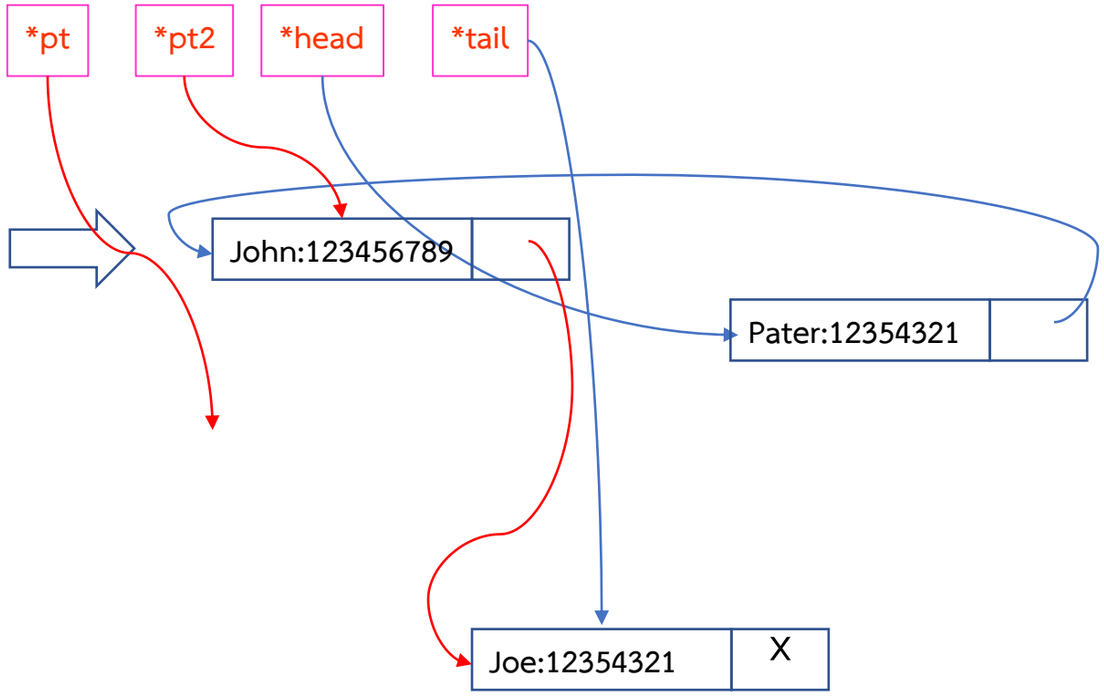



```
pt=pt2->next;
pt2->next=pt->next;
free(pt);
```

Linked List: Singly Linked List

การลบ Node : ลบด้านขวา

1) เปลี่ยน next ของตัวปัจจุบัน ไปชี้ยัง node ที่ต้องการลบชี้อยู่
2) คืนพื้นที่ node ที่ต้องการลบ

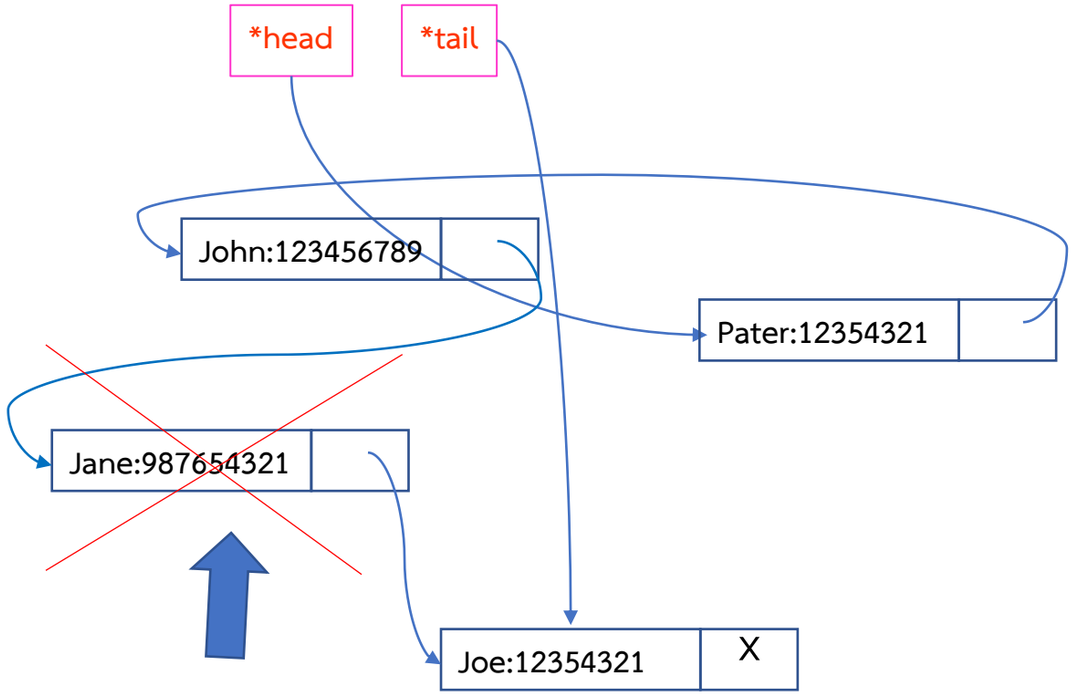



```
pt=pt2->next;
pt2->next=pt->next;
free(pt);
```

O(1)

Linked List: Singly Linked List

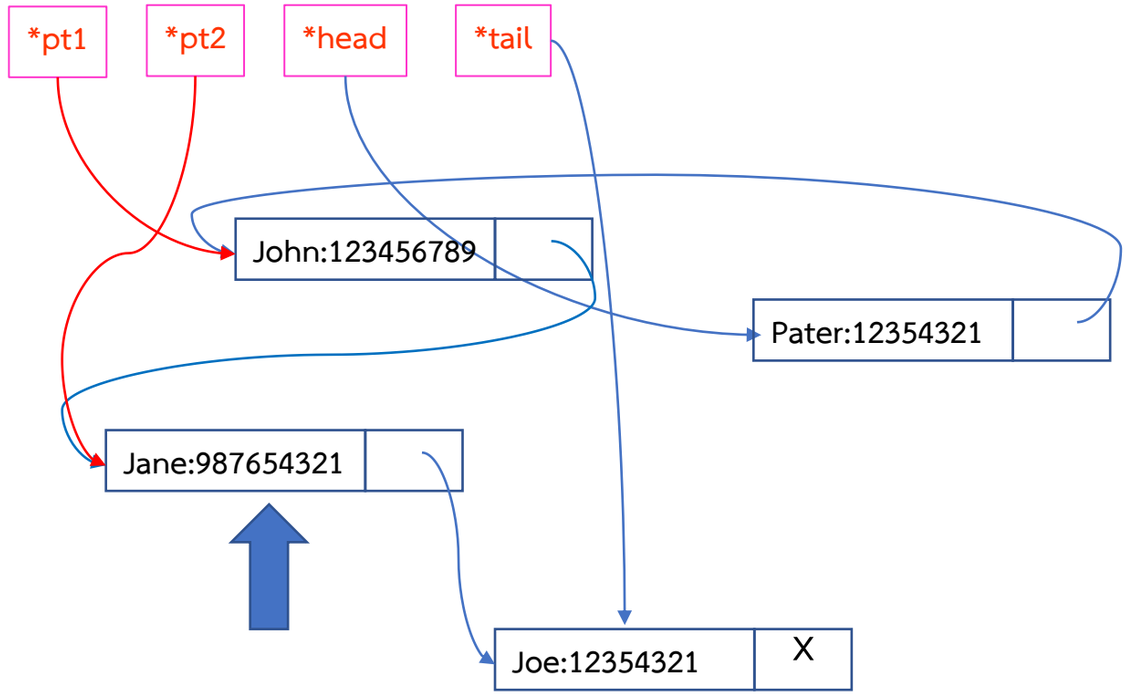
การลบ Node : ลบ node ปัจจุบัน



1) Traverse เพื่อหา Address ของ node ด้านซ้าย
2) เปลี่ยน next ของ node ด้านซ้ายมาชี้ next ของ node ปัจจุบัน
3) คืนพื้นที่หน่วยความจำของ node ปัจจุบัน

Linked List: Singly Linked List

การลบ Node : ลบ node ปัจจุบัน



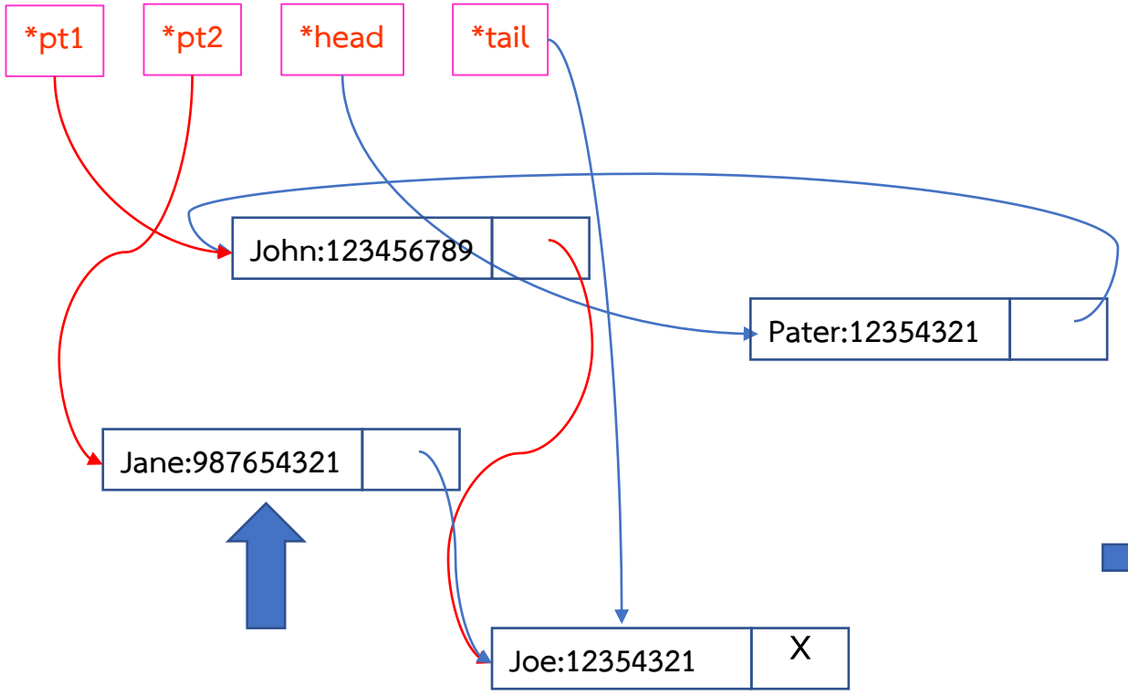

```
pt2=head;
while(pt2->next!=0){
if(!strcmp(pt2->Name,"Jane"))break;
pt1=pt2;
pt2=pt2->next;
}
```

```
pt1->next=pt2->next;
free(pt2);
```

1) Traverse เพื่อหา Address ของ node ด้านซ้าย
2) เปลี่ยน next ของ node ด้านซ้ายมาชี้ next ของ node ปัจจุบัน
3) คืนพื้นที่หน่วยความจำของ node ปัจจุบัน

Linked List: Singly Linked List

การลบ Node : ลบ node ปัจจุบัน



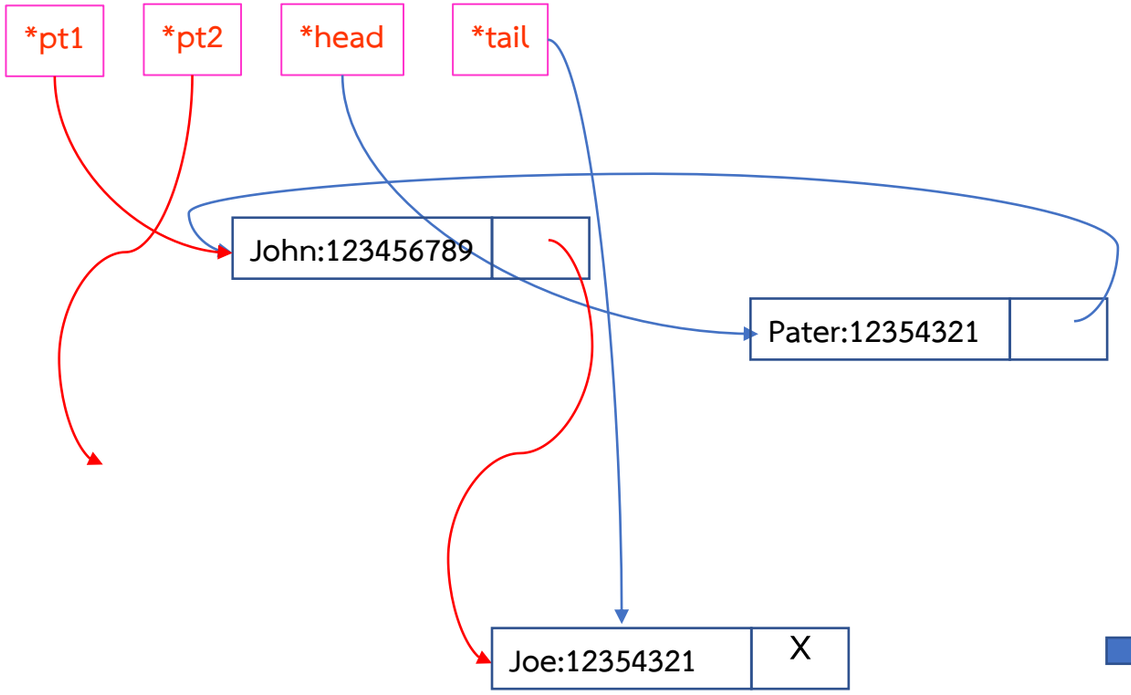

```
pt2=head;
while(pt2->next!=0){
if(!strcmp(pt2->Name,"Jane"))break;
pt1=pt2;
pt2=pt2->next;
}
```

```
pt1->next=pt2->next;
free(pt2);
```

1) Traverse เพื่อหา Address ของ node ด้านซ้าย
2) เปลี่ยน next ของ node ด้านซ้ายมาชี้ next ของ node ปัจจุบัน
3) คืนพื้นที่หน่วยความจำของ node ปัจจุบัน

Linked List: Singly Linked List

การลบ Node : ลบ node ปัจจุบัน



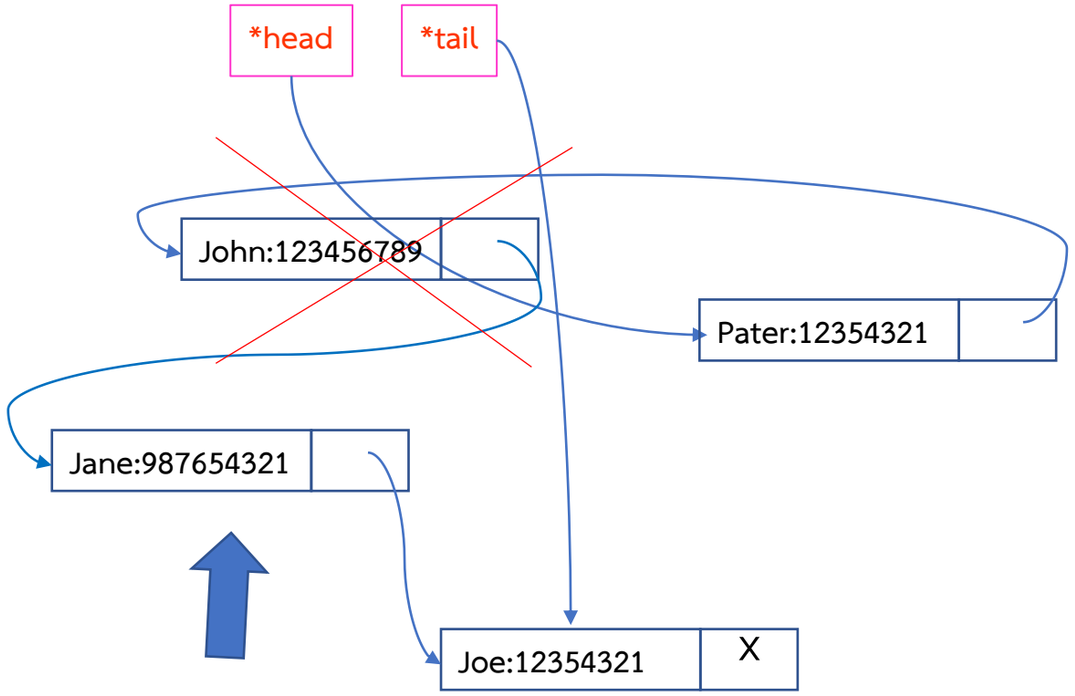

```
pt2=head;
while(pt2->next!=0){
if(!strcmp(pt2->Name,"Jane"))break;
pt1=pt2;
pt2=pt2->next;
}
```

```
pt1->next=pt2->next;
free(pt2);
```

O(N)

แต่ถ้าทำงานโดยใช้ pointer 2 ตัวตลอด จะลดเหลือ O(1)

การลบ Node : ลบ node ปัจจุบัน



ลบตัวที่อยู่ด้านซ้ายทำอย่างไร ?

ความซับซ้อนจะสูง

จึงนิยมใส่ลำดับที่ไม่ซ้ำกัน (index) ไว้ใน data field

ตอน traverse จะเก็บลำดับของ node ที่เคยผ่าน

หากต้องการลบด้านซ้ายก็จะ traverse หา index นี้ เมื่อเจอก็ทำการลบ node ปัจจุบันออก ขั้นตอนนี้ใช้เวลา O(N)

สรุป

ข้อดี

1 เปลี่ยนแปลงจำนวนข้อมูลได้ตลอดเวลา

2 การแทรกและลบทำได้เร็ว

3 การ Traverse ทำได้อย่างรวดเร็ว

ข้อเสีย

1 ต้องเสียพื้นที่หน่วยความจำเพิ่มสำหรับเก็บ Link field

2 ข้อมูลแต่ละ node ถ้าจะรู้ลำดับ ก็ไม่สามารถเข้าถึงโดยตรงได้แบบ Array

3 การเรียงลำดับข้อมูลทำได้ยาก (sorting)

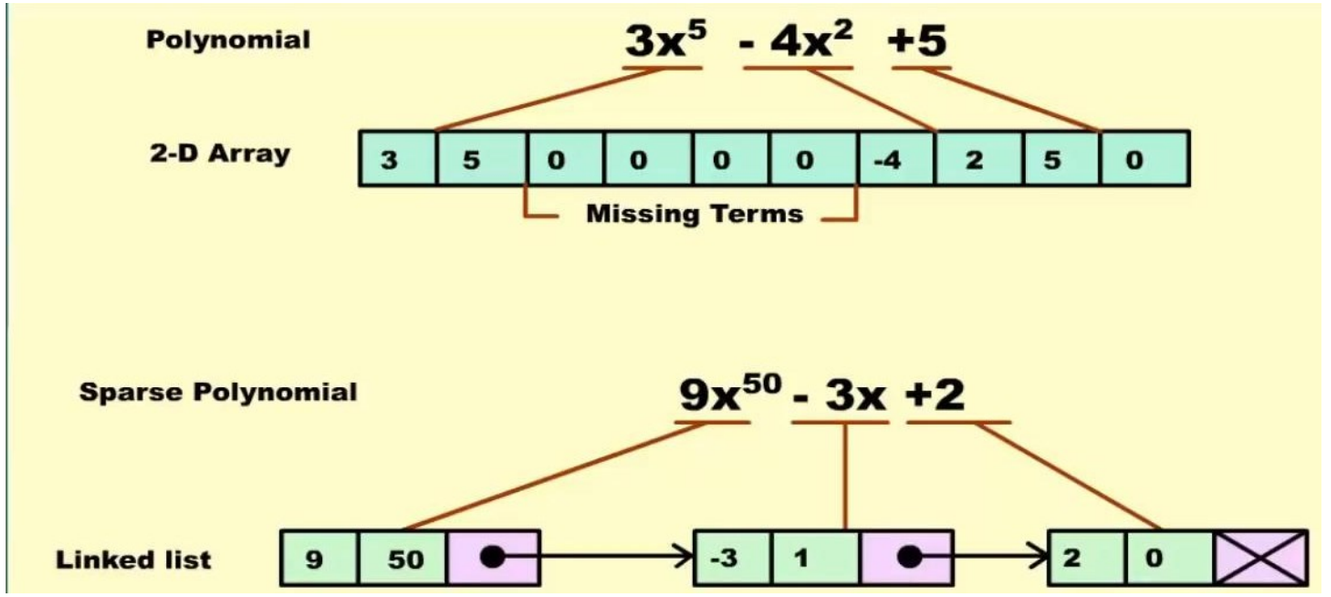
4 การ Traverse ทำได้ทิศทางเดียว

ตัวอย่างการใช้งาน

นิยมใช้ในการสร้างโครงสร้างข้อมูลชิดอื่น ๆ เช่น ต้นไม้ และ กราฟ

โดยเฉพาะโครงสร้างข้อมูลที่มีการเปลี่ยนแปลงขนาดข้อมูลตลอดเวลา เช่น stack

นิยมใช้ในการเก็บ polynomial เพื่อประหยัดพื้นที่



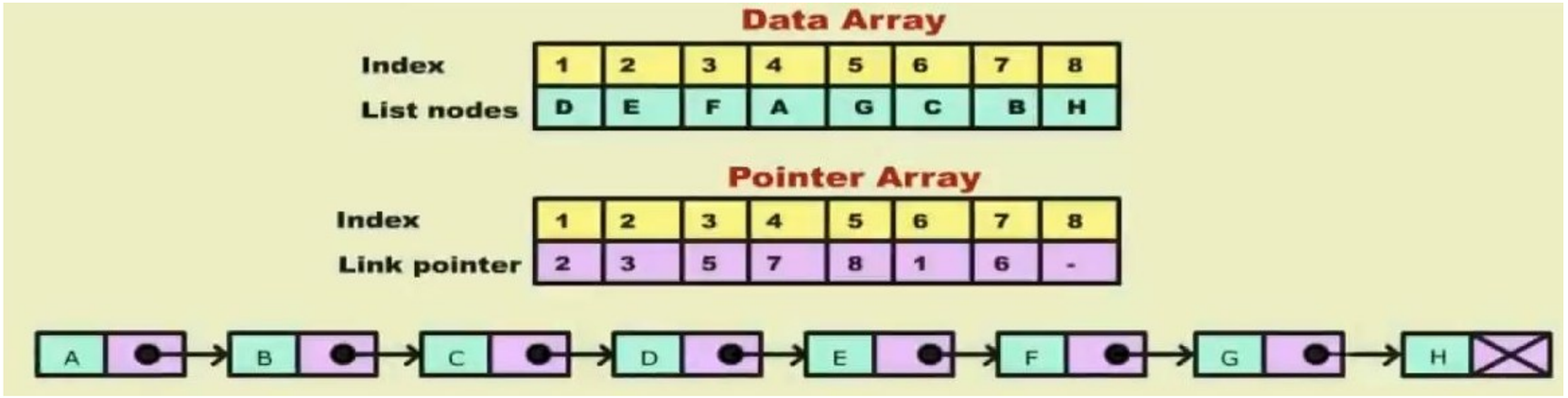
Linked List: Singly Linked List

ภาษา Java ไม่มี Pointer จะสร้าง Linked List ได้อย่างไร

สร้างคลาสขึ้นมาเพื่อเก็บข้อมูลเหมือนภาษา C และใช้ Array แทนการใช้ malloc()

การสร้าง Linked list แบบนี้เรียกว่าการสร้างโดยใช้ Cursors

จะใช้ Array 2 ตัววิ่งไปพร้อมกัน ตัวแรกเก็บ Data filed และ ตัวที่สองเก็บ Link field



Linked List: Singly Linked List

แต่อย่างไรก็ตาม Java มีโครงสร้างข้อมูลแบบ List ในรูปแบบของ ADT ให้ใช้อยู่แล้ว

```
class Node {
public Node(String string, int i) {
Name=string;
Phone=i;
}
String Name;
int Phone;
}
```

สร้าง Node

```
ArrayList<Node> PhoneBook = new ArrayList<Node>();
```

สร้าง List

```
PhoneBook.add(new Node("John", 123456789));
PhoneBook.add(new Node("Jane", 987654321));
```

เพิ่มข้อมูลต่อท้าย

```
PhoneBook.add(0, new Node("Peter",5554321));
```

แทรกข้อมูลไว้หน้าสุด

```
for(Node x:PhoneBook) {
System.out.println(String.format("Name:%s\nPhone:%d\n\n", x.Name,x.Phone));
}
```

Traverse

```
<terminated> HelloList [Java Application]
Name:Peter
Phone:5554321


Name:John
Phone:123456789


Name:Jane
Phone:987654321
```

Linked List: Singly Linked List

แต่อย่างไรก็ตาม Java มีโครงสร้างข้อมูลแบบ List ในรูปแบบของ ADT ให้ใช้อยู่แล้ว

```
class Node {
public Node(String string, int i) {
Name=string;
Phone=i;
}
String Name;
int Phone;
}
```

สร้าง Node

```
ArrayList<Node> PhoneBook = new ArrayList<Node>();
```

สร้าง List

```
PhoneBook.add(new Node("John", 123456789));
PhoneBook.add(new Node("Jane", 987654321));
```

เพิ่มข้อมูลต่อท้าย

```
PhoneBook.add(0, new Node("Peter",5554321));
```

แทรกข้อมูลไว้หน้าสุด

```
PhoneBook.remove(1);
```

ลบข้อมูลตำแหน่งที่ 1

```
for(Node x:PhoneBook) {
System.out.println(String.format("Name:%s\nPhone:%d\n\n", x.Name,x.Phone));
}
```

```
<terminated> HelloList [Java Application]
Name:Peter
Phone:5554321


Name:John
Phone:123456789


Name:Jane
Phone:987654321
```

```
<terminated> HelloList [Java Application]
Name:Peter
Phone:5554321


Name:Jane
Phone:987654321
```

Linked List: Singly Linked List

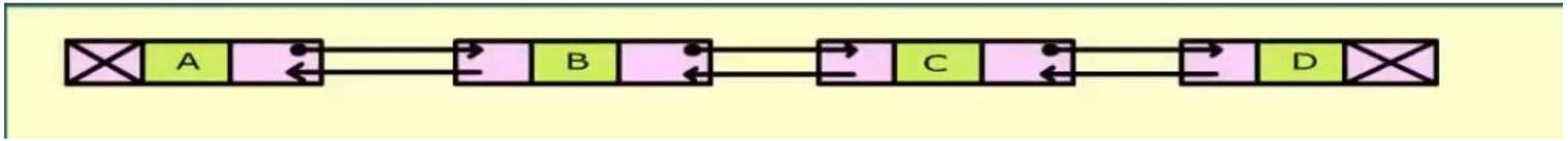
แต่อย่างไรก็ตาม Java มีโครงสร้างข้อมูลแบบ List ในรูปแบบของ ADT ให้ใช้อยู่แล้ว

Java เก็บทุกอย่างที่สร้างด้วยคำสั่ง New เอาไว้ที่ Heap อยู่แล้ว
ดังนั้น stack จะไม่เต็มง่ายๆ เหมือนเขียนด้วย C

Linked List
รายการโยง

- Singly Linked List ✔
- Doubly Linked List
- Circular Linked List
- Circular Doubly Linked List
- Multi Listed

Doubly Linked List คือ Linked List ที่มีการเชื่อมโยงสองทางทั้งไปและกลับ

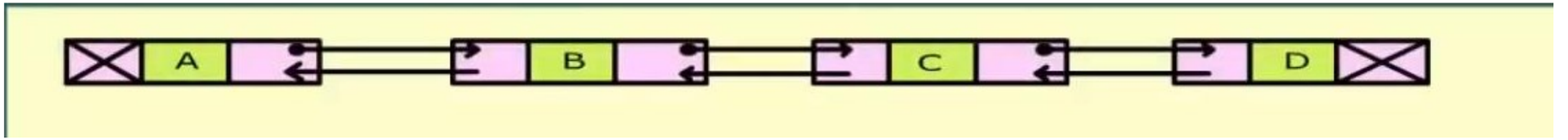


ทำให้สามารถ Traverse ได้ทั้ง 2 ทิศทาง (Bidirectional)

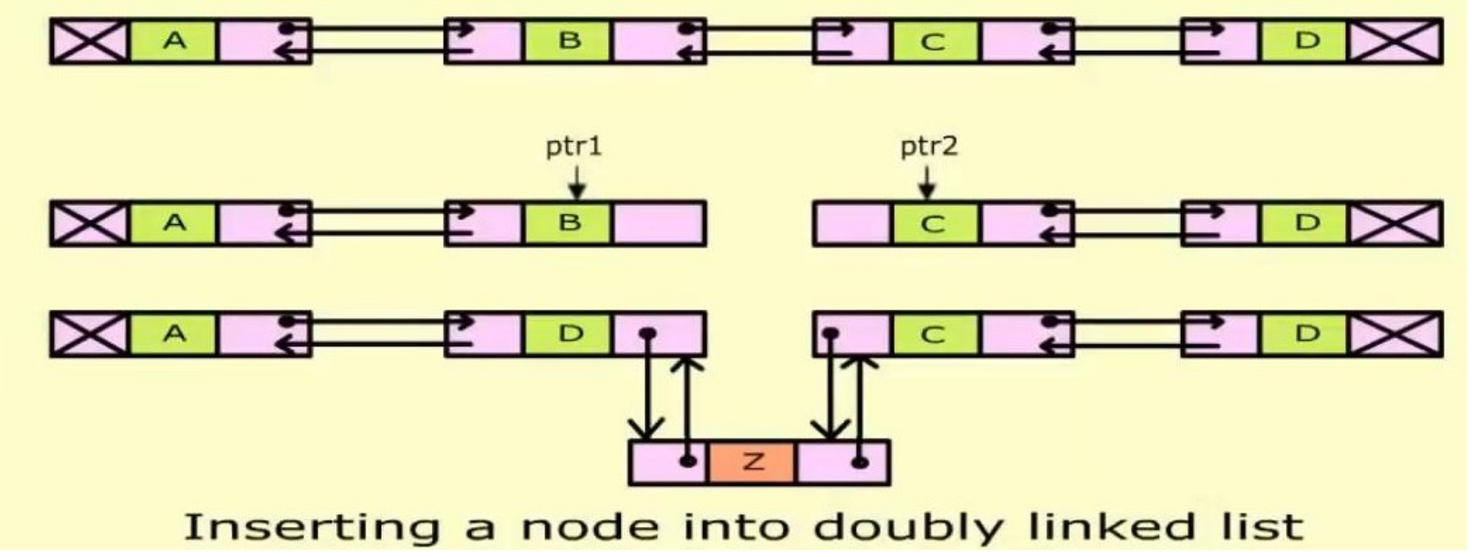
การแทรกและลบ สามารถกระทำได้โดยการปรับ link pointer ทั้ง 2 ด้าน
จากนั้นก็ทำการปรับ pointer ของ node ซ้ายและขวา

เนื่องจากสามารถ traverse ได้ทั้ง 2 ทิศทาง จึงสามารถเดินทางไปยัง node ด้านซ้ายได้เลย
โดยไม่ต้องเริ่ม Traverse มาตั้งแต่ head

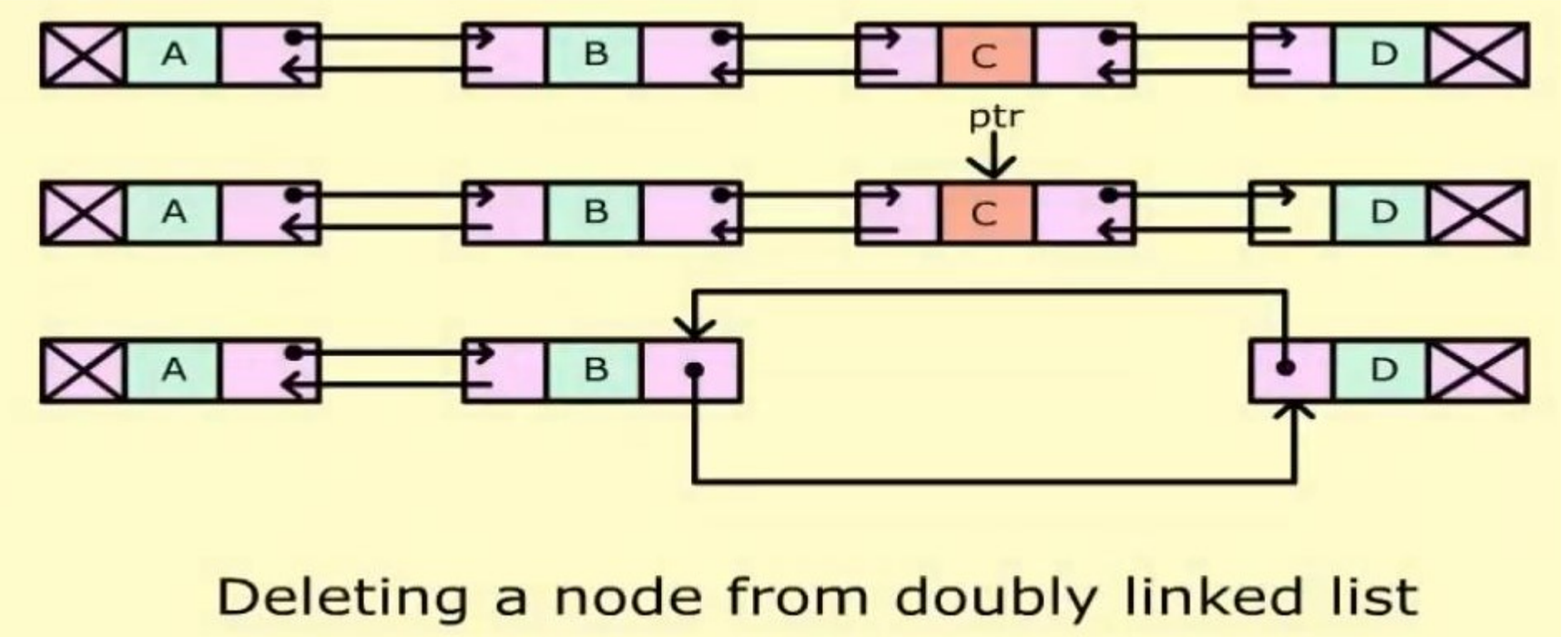
Doubly Linked List คือ Linked List ที่มีการเชื่อมโยงสองทางทั้งไปและกลับ



การแทรก node ตรงกลาง จะใช้ pointer ชั่วคราว 2 ตัว ช่วยในการตัดต่อ node



การลบ node ตรงกลาง จะใช้ pointer ชั่วคราว 2 ตัว ช่วยในการตัดต่อ node



Doubly Linked List

ข้อดี

1 สามารถ Traverse ได้สองทิศทาง

2 การลบและแทรกทำได้เร็ว

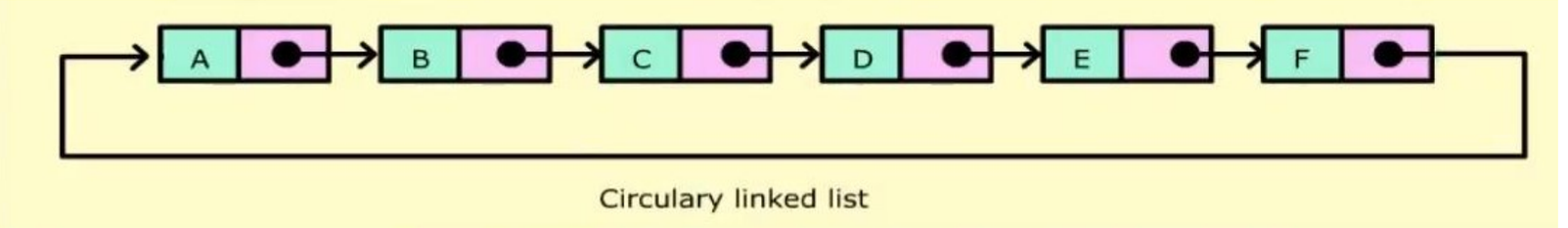
ข้อเสีย

1 ต้องเสียพื้นที่หน่วยความจำเพิ่มสำหรับเก็บ Link field วิ่งกลับ

2 ต้องคอยติดตาม Address ของ node ทั้งซ้ายและขวา

Linked List

Circular Linked List คือ Linked List ที่มีการเชื่อมโยง Node สุดท้ายกลับไปยัง Node แรก



Circulary linked list

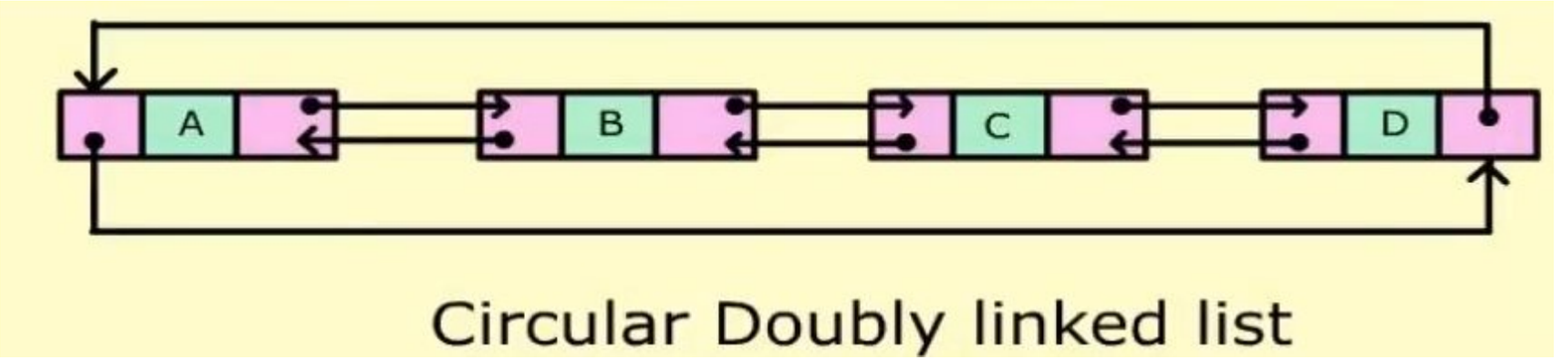
ช่วยให้ Node สุดท้ายวนกลับไป Node แรกได้เร็ว

ไม่มี head และ tail ดังนั้นการ traverse จะเริ่มจาก node ใดก็ได้ node นั้นจะถูกเรียกว่า head node

มักนิยมใช้ในการวนเพื่อประมวลผลซ้ำ ๆ เช่นการจัดลำกับเก็บข้อมูล หรือประมวลผลข้อมูล ที่ต้องวนทำซ้ำ ๆ และอาจมีงานที่เพิ่มหรือลดได้ หรือการประมวลผลสายอักขระ

ข้อเสีย: ระวังการ traverse อาจติด infinite loop

Circular Doubly Linked List คือ Linked List ที่มีการเชื่อมโยงสองทางและ เชื่อมโยง Node สุดท้ายกลับไปยัง Node แรก



เริ่ม traverse จาก node ไหน และไปทางทิศไหนก็ได้ มีข้อดีเหมือน Double และ Circular Linked list มารวมกัน

ข้อเสีย: ระวังการ traverse อาจติด infinite loop

Linked List

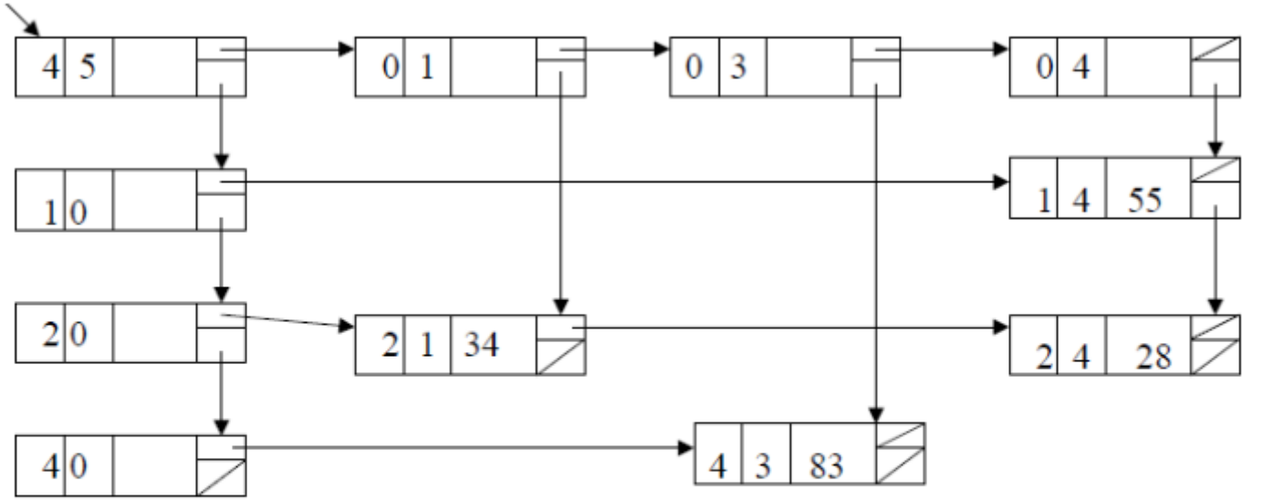
Multi Linked List คือ Linked List ที่มีการเชื่อมโยงมากกว่า 1

แล้วต่างจาก Doubly Linked List ตรงไหน ?

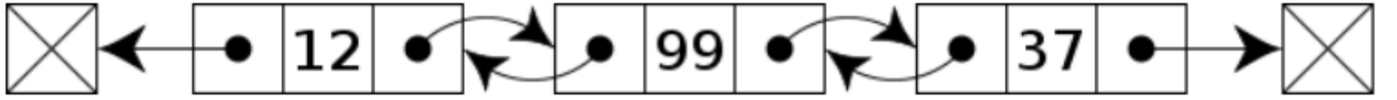
ต่างกันตรงที่ Doubly Linked List ส่วนของ link ตัวแรกจะชี้กลับไปยังตัวซ้ายเท่านั้น

แต่ Multi Linked List ไม่ได้บังคับว่าต้องทำแบบนั้น

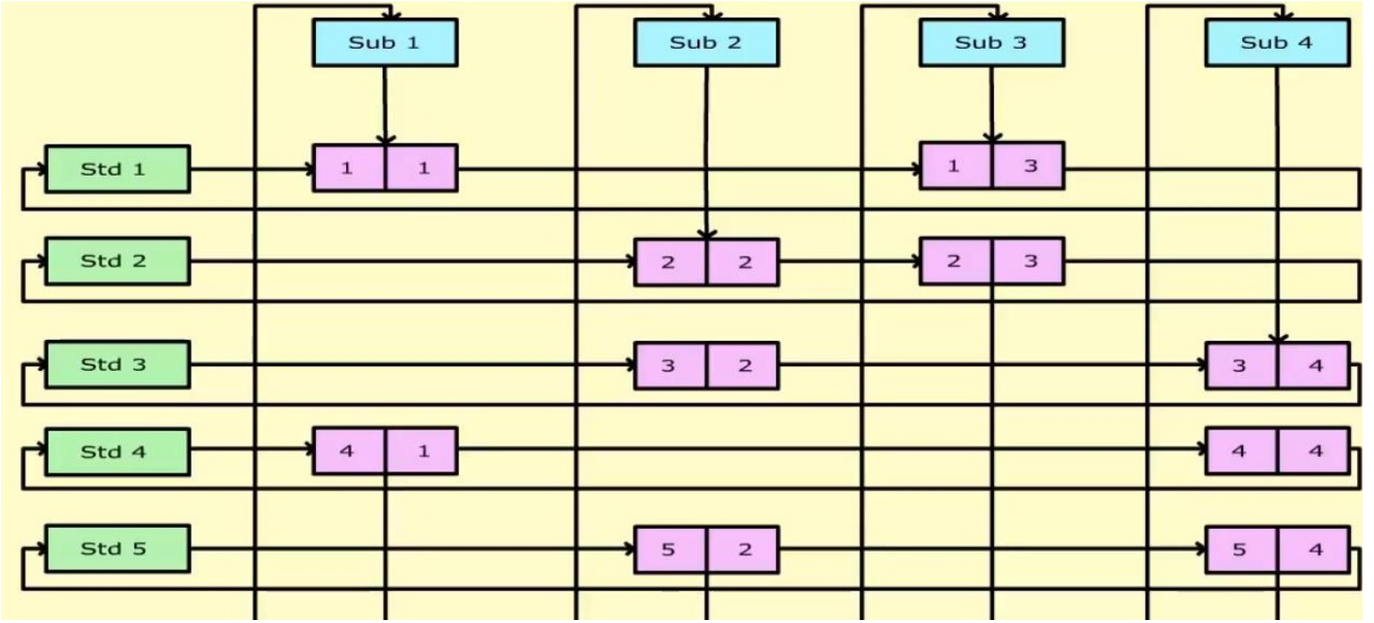
A multi linked list:



A doubly linked list:



Multi Linked List นิยมใช้ในการสร้างฐานข้อมูลเชิงสัมพันธ์ ดังตัวอย่าง



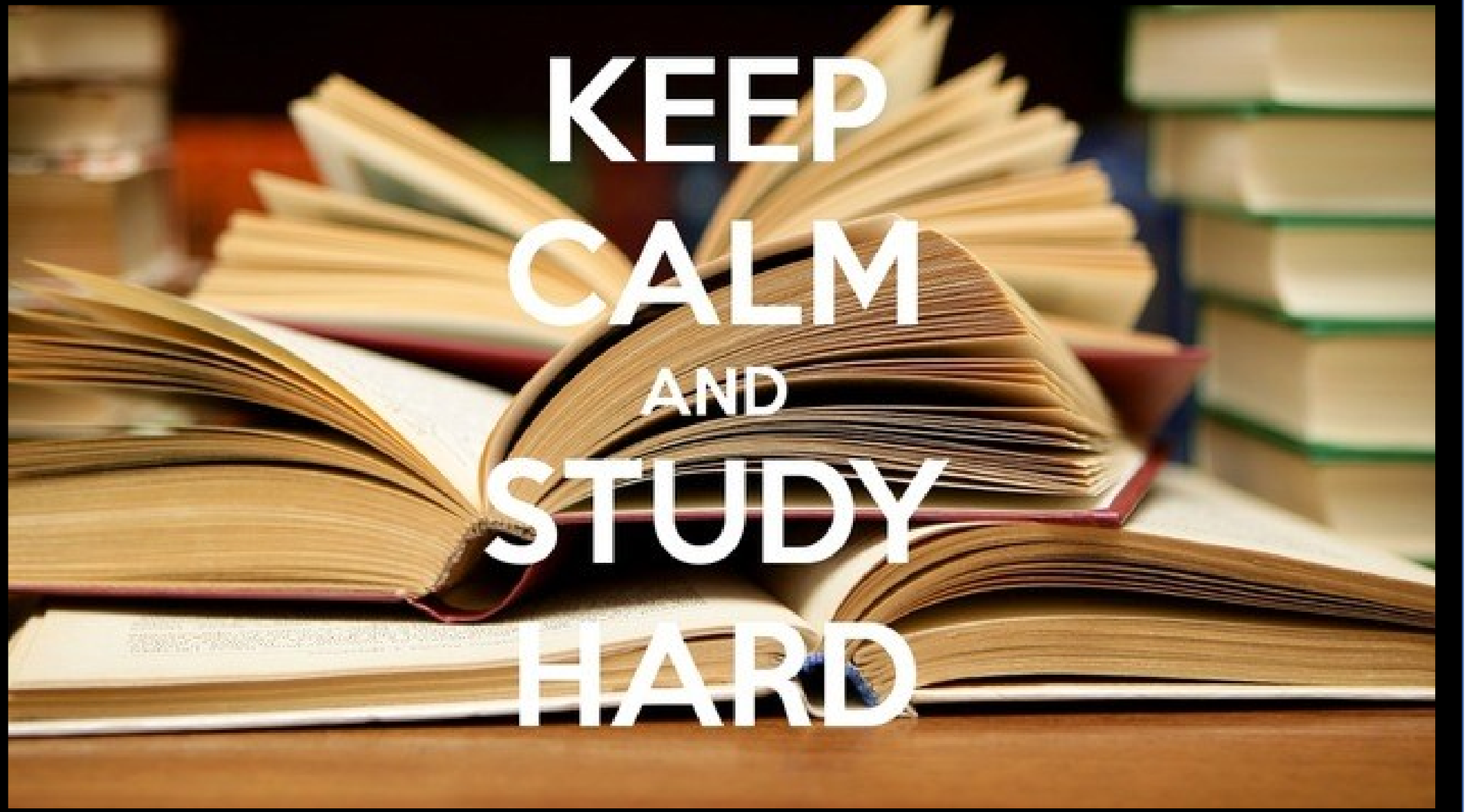
โดยการ traverse สามารถจะเริ่มจาก นักศึกษา หรือ วิชาก็ได้

อยากรู้คะแนนรวมทุกวิชาของ นักศึกษา

หรือ เริ่มจากวิชา แล้วหาว่ามีนักศึกษาคนไหนลง บ้าง และได้คะแนนเท่าไหร่ ก็สามารถทำได้

Multi Linked List นักศึกษาจะได้เรียนอย่างละเอียด

ทั้ง พีชคณิตเชิงสัมพันธ์ และ แคลคูลัสเชิงสัมพันธ์ ในวิชา ฐานข้อมูล


KEEP
CALM
AND
STUDY
HARD