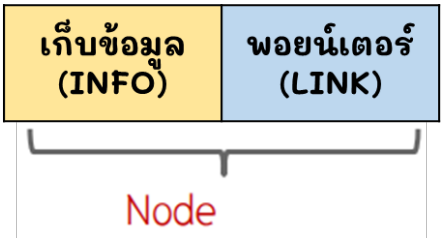
บทที่ 3 โครงสร้างลิงค์ลิสต์ ต้นไม้ และกราฟ

(Linked List Tree and Graph Structure)

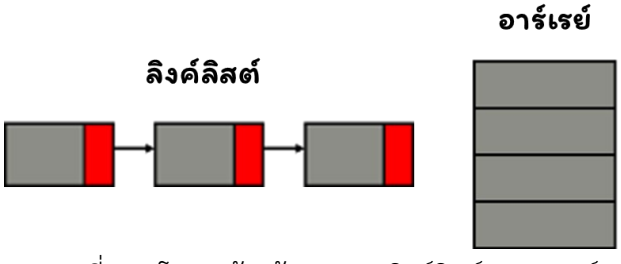
3.1 โครงสร้างลิงค์ลิสต์ (Linked List)

ลิงค์ลิสต์เป็นการจัดเก็บชุดข้อมูลเชื่อมโยงต่อเนื่องกันไปตามลำดับ โครงสร้างข้อมูลแบบลิงค์ลิสต์นี้ต้องมีส่วนที่เป็นพอยน์เตอร์ ซึ่งการแทนแบบใช้ พอยน์เตอร์นี้ต้องใช้พื้นที่เพิ่มเติมเป็นส่วนของพอยน์เตอร์ (Pointer) หรือลิงค์ (Link) เพื่อแสดงให้เห็นชัดว่าโหนดที่ต่อจากโหนดนั้นคือโหนดใด ลักษณะของแต่ละโหนดจึง ประกอบด้วย 2 ส่วน ดังรูป 3.1



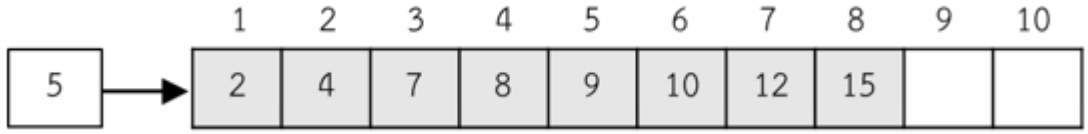
รูปที่ 3.1 ลักษณะโหนดของลิงค์ลิสต์

โครงสร้างข้อมูลแบบลิงค์ลิสต์นี้จะให้ความสะดวกแก่การจัดการข้อมูลมาก อย่างไรก็ดีในบางครั้งก็ให้ความยุ่งยากแก่ผู้ใช้เนื่องจากจะซับซ้อนกว่าแบบใช้อาเรย์ ข้อจำกัดของการใช้อาเรย์คืออาเรย์มีขนาดคงที่ ดังรูป 3.2 ถ้าใช้อาเรย์ขนาด 100 ช่อง เมื่อใช้หมด 100 ช่องก็ใช้อาเรย์นั้นไม่ได้ ถ้าไม่อนุญาตให้เคลียร์พื้นที่อาเรย์เพื่อใช้ซ้ำถึงแม้อนุญาตให้เคลียร์พื้นที่อาเรย์ เพื่อใช้ซ้ำก็ไม่เป็นการสะดวก เนื่องจากอาจต้องตรวจทุกช่องในอาเรย์เพื่อหาว่าช่องไหนสามารถใช้ซ้ำได้ นอกจากนี้การใช้พื้นที่ซ้ำยังต้องเลื่อนข่าวสารทั้งหมดไปอยู่ในส่วนปลาย หรือส่วนต้นของอาเรย์ให้หมด การทำเช่นนี้จะเสียเวลามากอีกด้วย ข้อจำกัดของการใช้อาเรย์อีกประการคือ ในภาวะการทำงานแบบเวลาจริง (Real-time) ที่ต้องมีการนำข้อมูลเข้า (Insertion) หรือดึงข้อมูลออก (Deletion) อาเรย์จะไม่สะดวกแก่การใช้มาก ในภาวะการทำงานแบบนี้โครงสร้างแบบลิงค์ลิสต์จะสะดวกกว่า

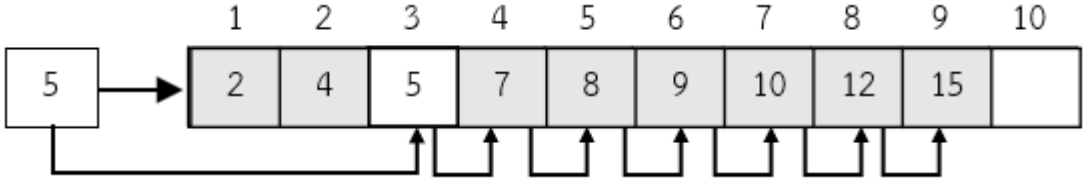


รูปที่ 3.2 โครงสร้างข้อมูลแบบลิงค์ลิสต์และอาเรย์

ความไม่สะดวกในแทรกข้อมูลสู่อาเรย์ ขอให้พิจารณาดูการนำข่าวสาร 5 เข้าสู่อาเรย์ A (ซึ่งมี 10 ช่อง) เพื่อให้ค่าต่าง ๆ เรียงอยู่ในลำดับที่ถูกต้อง จะเห็นได้ว่าในลักษณะนี้จะต้อง มีการเลื่อนข่าวสารในอาเรย์จำนวนมาก เพื่อเปิดทางให้กับตัวที่เข้ามาใหม่ กระบวนการเช่นนี้ จะเสียเวลามากถ้าแต่ละข่าวสารเป็นเรคอร์ด (Record) ขนาดใหญ่และมีรายการจำนวนมาก ดังรูป 3.3
ในระบบที่มีการใช้ลิงค์ลิสต์ การแทรกข้อมูลเข้าหรือลบข้อมูลออกจะสะดวกมาก เนื่องจากสามารถทำสำเร็จได้โดยคำสั่งไม่กี่คำสั่ง เพื่อเปลี่ยนค่าพอยน์เตอร์



(ก) ก่อนเข้า



(ข) ต้องเลื่อนข้อมูล 6 ชิ้นเพื่อเปิดทางให้แก่ข้อมูลใหม่

รูปที่ 3.3 แสดงความไม่สะดวกในการแทรกข้อมูลเข้าสู่อาเรย์

นิยามการเข้าถึงข้อมูลในลิงค์ลิสต์

ในลิงค์ลิสต์จะมี

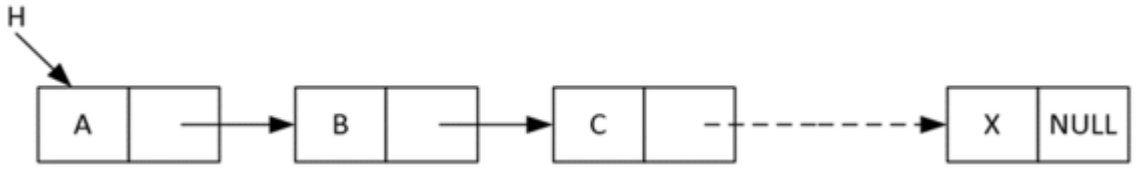
H (HeadNode) เป็นตัวชี้ตำแหน่งแรกของลิสต์

NULL เป็นการแสดงการสิ้นสุด (ในที่นี้จะแทนด้วยเครื่องหมาย ^)

H1 เป็นพอยน์เตอร์ผู้ช่วยในการท่องเข้าไปในลิสต์

P เป็นพอยน์เตอร์ชี้โหนดที่ต้องการแทรกหรือลบ

สมมติว่าจะแทนชุดข่าวสารที่ประกอบด้วย A,B,C ลักษณะการแทนจะเป็นไปตามรูป 3.4 ซึ่งแสดงแนวความคิดการเก็บแบบหนึ่ง ส่วนการเก็บข่าวสาร A,B,C แท้จริงแล้วจะเป็นไป ในรูปแบบใดก็ได้



รูปที่ 3.4 การแทนโครงสร้างแบบลิงค์ลิสต์

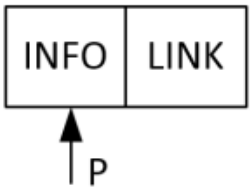
3.1.1 ลิงค์ลิสต์เดี่ยว (Singly Linked List)

ลิงค์ลิสต์เดี่ยว หมายถึง ลิงค์ลิสต์ที่แต่ละข้อมูลมีการแสดงออกถึงลำดับก่อนหลัง อย่างชัดแจ้งโดยใช้พอยน์เตอร์ นั่นคือในแต่ละข้อมูลจะมีส่วนที่บอกว่าข้อมูลตัวต่อไปอยู่ที่ใด แต่ละข้อมูลจะเรียกว่าโหนด (Node) แต่ละโหนดจะมี 2 ส่วน ซึ่งจะใช้สัญลักษณ์ ต่อไปนี้ในการเรียกชื่อแต่ละโหนด ดังรูป 3.5

NODE(P) หมายถึงโหนดที่ระบุ (ถูกชี้) โดยพอยน์เตอร์ P

INFO(P) หมายถึงส่วนข้อมูลของโหนดที่ถูกชี้โดย P

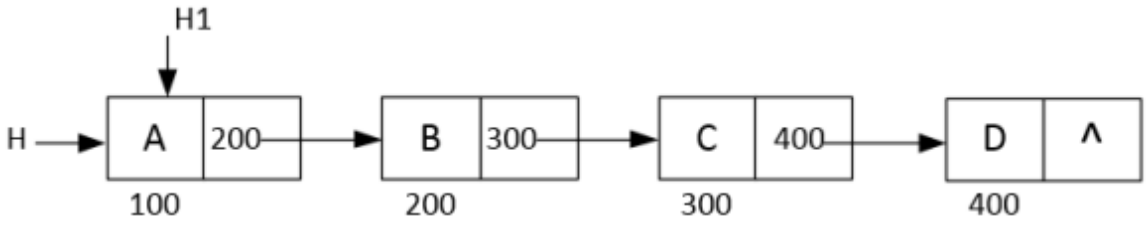
LINK(P) หมายถึงส่วนชี้ตำแหน่งของโหนดที่ถูกชี้โดย P



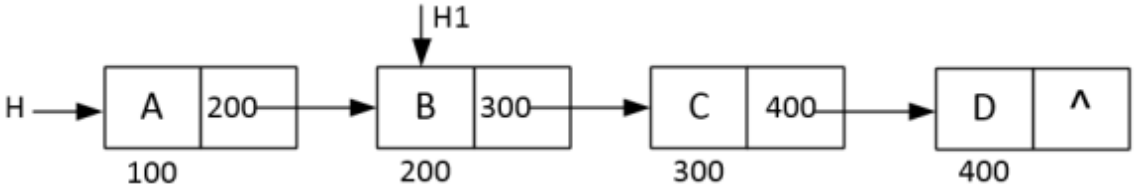
รูปที่ 3.5 ลักษณะลิงค์ลิสต์เดี่ยว

3.1.1.1 การเข้าถึงข้อมูลทั้งหมดในลิงค์ลิสต์เดี่ยว

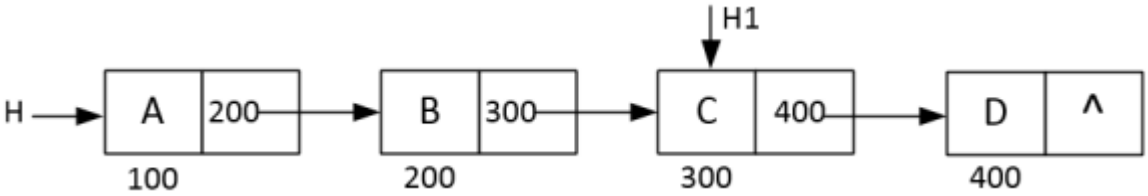
ถ้าต้องการเข้าถึงข้อมูลทุกตัวในลิสต์เพื่อกระทำกับข้อมูลในแต่ละโหนด สามารถเข้าถึงข้อมูลโดยวิธีการแบบลำดับ (Sequential) โดยเริ่มต้นที่ H การท่องไปในลิงค์ลิสต์นั้นจะไม่เลื่อน H เนื่องจากถ้าเลื่อนที่ H แล้วจะกลับไปตั้งต้นที่หัวลิสต์ใหม่ไม่ได้ ดังนั้นจะใช้พอยน์เตอร์อีกตัวมาเป็นผู้ช่วย (Auxiliary Pointer) เพื่อท่องเข้าไปในลิสต์ แทนทุกครั้งที่เลื่อนพอยน์เตอร์ (สมมติว่าเป็น H1) โดยใช้คำสั่ง **H1 = LINK(H1)** เพื่อเลื่อน H1 ชี้ไปยังโหนดถัดไป (ตัวเลข 100 200 300 400 นั้นสมมติว่าเป็นที่อยู่ของแต่ละโหนดในหน่วยความจำ) ดังรูป 3.6



(ก) หลังคำสั่ง H1 = H



(ข) หลังคำสั่ง H1 = LINK(H1) ครั้งแรก



(ค) หลังคำสั่ง H1 = LINK(H1) ครั้งที่สอง

รูปที่ 3.6 แสดงการเข้าถึงข้อมูลในลิงค์ลิสต์เดี่ยว

จะเห็นได้ว่าการเข้าถึงโหนดของลิงค์ลิสต์นั้นจะใช้พอยน์เตอร์ของแต่ละโหนดเป็นตัวชี้เชื่อมโยงกันเป็นทอดๆ ไปเรื่อยๆ โดยเริ่มต้นที่ H ซึ่งสามารถใช้คำสั่งวนรอบ เช่น คำสั่ง WHILE, FOR เพื่อเข้าและดำเนินการกับชุดของข้อมูลในลิงค์ลิสต์นี้ได้

3.1.1.2 การวางโครงสร้างโปรแกรมลิงค์ลิสต์เดี่ยว

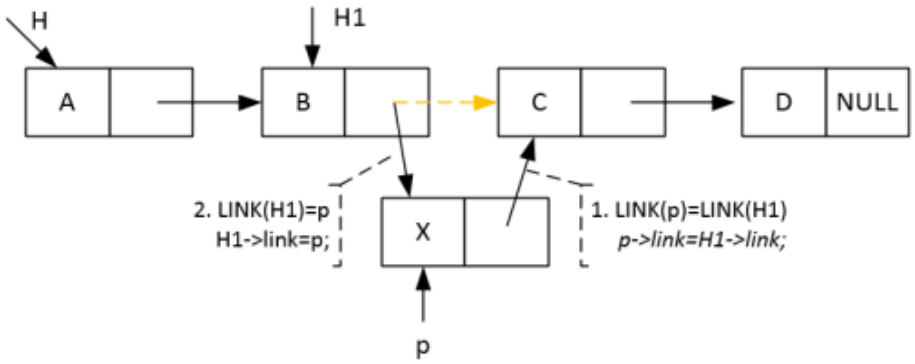
เริ่มแรกจะสร้างลิงค์ลิสต์เดี่ยวก่อนจำนวน N โหนดโดยการเชื่อมต่อเป็นลำดับในฟังก์ชัน CreateNNode จากนั้นก็ไปรับคำสั่งเพื่อทำการแทรก หรือลบ แล้วก็ดำเนินการดังนี้

การแทรกโหนดในลิงค์ลิสต์เดี่ยว (Insertion)

กรณีที่ลิสต์ว่างเปล่าจะไม่อนุญาตให้แทรก หากไม่ว่างเปล่าก็จะค้นหา ว่าข้อมูลที่อ้างอิงอยู่ที่โหนดไหน หากไม่พบก็ไม่สามารถแทรกได้ หากพบก็ใช้ H1 ชี้ตำแหน่งของโหนดนั้นแล้วทำการอ่านค่าข้อมูลใหม่เก็บไว้ใน info แล้วแทรกโหนดใหม่ลงไปหลังโหนด ที่ระบุตามขั้นตอนดังรูป 3.7 หรือ 3.8 แล้วแต่กรณี แล้วทำการค้นหาข้อมูลที่อ้างอิงตัวนั้นต่อ หากพบก็จะแทรกอีก จนกว่าจะหมดข้อมูลในลิสต์

สมมติว่ามีลิงค์ลิสต์ H อยู่แล้ว (โหนดที่ถูกชี้โดย H คือโหนดแรกของลิงค์ลิสต์นี้) และต้องการนำโหนดข่าวสารที่ถูกชี้โดยพอยน์เตอร์ P (หรือโหนด P) ไปอยู่ต่อจากโหนด H1 จะมีลักษณะการนำข้อมูลเข้าอยู่ 2 กรณี คือ

1. การแทรกโหนดหลัง H1 (กรณี H1 ไม่ใช่โหนดสุดท้าย)

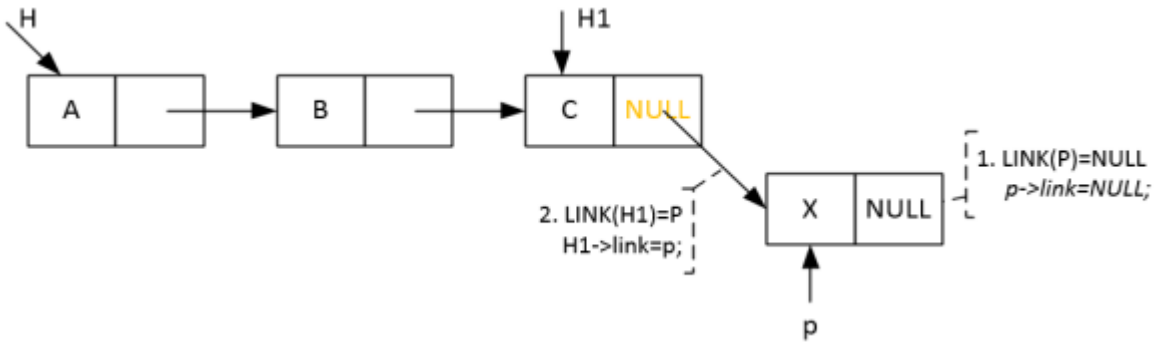


รูปที่ 3.7 การแทรกโหนดหลัง H1 (กรณี H1 ไม่ใช่โหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

1. ALLOCATE P
2. LINK(P)= LINK(H1)
3. LINK(H1)=(P)

2. การแทรกโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)



รูปที่ 3.8 การแทรกโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

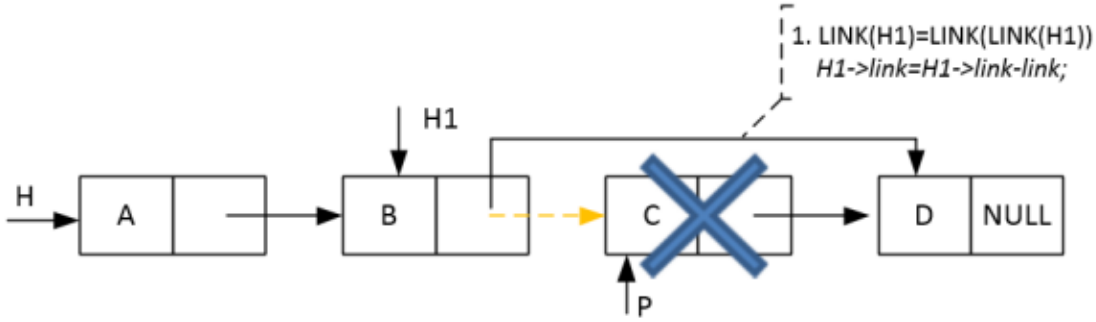
1. ALLOCATE P
2. LINK(P)=NULL
3. LINK(H1)=(P)

การลบโหนดในลิงค์ลิสต์เดี่ยว (Deletion)

กรณีที่ลิสต์ว่างเปล่าจะไม่อนุญาตให้ลบ หากไม่ว่างเปล่าก็จะค้นหาว่า ข้อมูลที่อ้างอิงอยู่ที่โหนดไหน หากเป็นโหนดสุดท้ายจะไม่ลบ(เพราะโหนดที่อยู่ถัดจากนี้ไม่มี) หากไม่ใช่จะใช้ H1 ชี้ตำแหน่งของโหนดนั้นแล้วก็ทำการลบตามขั้นตอนดังรูป 3.9 หรือ 3.10 แล้วแต่กรณี

หากต้องการลบโหนดที่ต่อถัดจากโหนด H1 ออกจากลิงค์ลิสต์ และให้คืนโหนดนั้นไปยังหน่วยความจำ (Storage Pool) จะมี 2 กรณีเช่นกัน คือ

1. การลบโหนดหลัง H1 (กรณี H1 ไม่ใช่โหนดสุดท้าย)

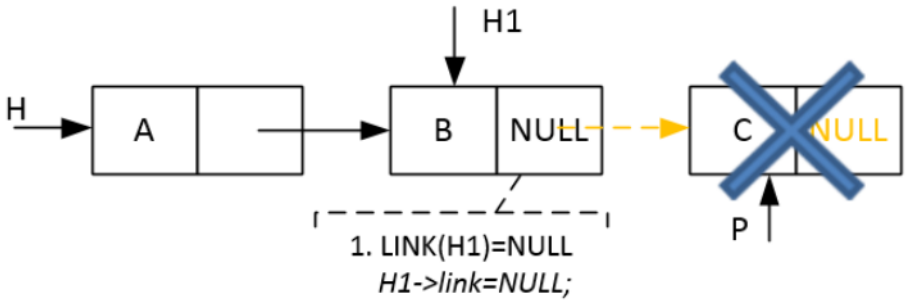


รูปที่ 3.9 การลบโหนดหลัง H1 (กรณีไม่ใช่โหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

1. P=LINK(H1)
2. LINK(H1)=LINK(P)
3. FREE(P) //คืนโหนดให้กับ Storage Pool

2. การลบโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)



รูปที่ 3.10 การลบโหนดหลัง H1 (กรณีเป็นโหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

1. P=LINK(H1)
2. LINK(H1)=NULL
3. FREE(P) //คืนโหนดให้กับ Storage Pool

3.1.1.3 โปรแกรมลิงค์ลิสต์เดี่ยว

```
/* Program create SINGLY LINKED LIST by...
 1. Create Node N Nodes
 2. Can Insert node after data as defined
 3. Can Delete node after data as defined
 4. Show address of Node and LINK
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use molloc
struct Node //Declare structure of node
{
    int info;
    struct Node *link;
};
struct Node *H, *H1, *p, *q; // Declare pointer node
int i,j,k,n,data;
char ch;
Node *Allocate() //Allocate 1 node from storage pool
{
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
    return(temp);
}
void CreateNNode(int n) //Create N Node put data and link it
{
    int i,temp;
    H=NULL;H1=NULL;
    for (i=1;i<=n;i++) //Count N Node
    {
        p=Allocate(); //Allocate Node
        temp=1+rand() % 99; //random difference number 1..99
        p->info=temp; //Put random data in to node
        if (i==1)
            H=p; //Set H point to first node
        else
            H1->link=p; //Link first node to second node
```

```
        H1=p; //Let H1 point to last node
        H1->link=NULL; //Set link of H1 to NULL
    }
}
void ShowAllNode()
{
    printf("H = %x\n",H); //Display address of pointer H
    p=H; //Set start point of p pointer
    i=1; //set start value of counter
    while (p != NULL) //While if NOT NULL
    {
        printf("%d) %x\t",i,p); //Show COUNTER and POINTER
        printf("INFO : %d\t",p->info); //Show INFO
        printf("LINK : %x\n",p->link); //Show LINK
        p=p->link; //Skip to next node
        i++; //Skip Counter
    } //End While
} //Enf Fn.
void InsertAfter(int data1)
{
    int temp; //Temporary variable
    if (H==NULL)
        printf("Linked List have no node!!..\n");
    else
    {
        H1=H; //Let H1 point at start node
        while (H1 != NULL) // Search for the data while H1<>NULL
        {
            if (H1->info == data1) //if Found
            {
                p=Allocate(); //Allocate one node from storage pool
                printf("\nInsert data : " ); //Input data for insert
                scanf("%d",&temp); //Read from KBD
                p->info=temp; // Entry temporary data into INFO of node
                p->link=H1->link; //Change pointer 1st for insert node
(FAR)
                H1->link=p; //Change pointer 2nd for insert node (NEAR)
            } //End if
            H1=H1->link; //Skip H1 to next node
        } //End while
    } //End IF
} //End Fn.
void DeleteAfter(int data1)
{
    int temp; //Temporary variable
    if (H==NULL)
```

```
        printf("Linked List have no node!!..\n");
    else
    {
        H1=H; //Let H1 point at start node
        while (H1 != NULL) //Search for the data while H1<>NULL
        {
            if (H1->info == data1) //if Found
            {
                if (H1->link==NULL) //If no more node
                printf ("No more node from here,Can't delete it!!!\n");
                else
                {
                    p=H1->link; //Mark at node for Delete
                    if(p->link==NULL) //If p is last node
                        H1->link=NULL; //Set link of H1 to NULL
                    else
                        H1->link=p->link; //If not set link of H1 point
same address of p
                    free(p); //Free node to storage pool
                } //End if2
            } //End if1
            H1=H1->link; //Skip H1 to next node
        } //End while
    } //End IF
} //End Fn.
int main() //MAIN Fn.
{   n=10; //Set amount of node
    CreateNNode(n); //Call Fn. Create N nodes
    printf("PROGRAM SINGLY LINKED LIST \n");
    printf("================================= \n");
    printf("All Data in Linked List \n");
    ShowAllNode(); //Call Fn. Show all node
    ch=' ';
    while (ch != 'E')
    {
        printf("MENU : [I:Insert D:Delete E:Exit]");
        ch=getch();
        switch (ch)
        {
        case 'I' : printf("\nInsert After data : " ); //Input data for
insert after
            scanf("%d",&data);
            InsertAfter(data); //Call Fn. Insert after data
            printf("\nAll Data in Linked List AFTER INSERTED\n");
            ShowAllNode(); //Call Fn. Show all node
            break;
```

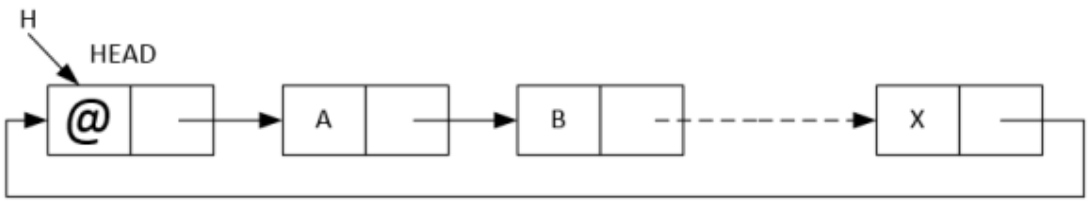
```
        case 'D' : printf("\nDelete After data : " ); //Input data for
Delete after
            scanf("%d",&data);
            DeleteAfter(data); //Call Fn. Delete after data
            printf("\nAll Data in Linked List AFTER DELETED\n");
            ShowAllNode(); //Call Fn. Show all node
            break;
        } // End Switch...case
    } //End While
    return(0);
} //End MAIN
```

```
C:\Users\iammai\Downloads\project\code-data_structure\8\Lab_8_SLL.exe
PROGRAM SINGLY LINKED LIST
==================================
All Data in Linked List
H = c81420
1) c81420       INFO : 42       LINK : c81440
2) c81440       INFO : 54       LINK : c81460
3) c81460       INFO : 98       LINK : c81480
4) c81480       INFO : 68       LINK : c814a0
5) c814a0       INFO : 63       LINK : c814c0
6) c814c0       INFO : 83       LINK : c814e0
7) c814e0       INFO : 94       LINK : c81500
8) c81500       INFO : 55       LINK : c81520
9) c81520       INFO : 35       LINK : c81540
10) c81540      INFO : 12       LINK : 0
MENU : [I:Insert D:Delete E:Exit]
```

รูปที่ 3.11 แสดงผลการทำงานของโปรแกรมลิงค์ลิสต์เดี่ยว

3.1.1.4 ลิงค์ลิสต์เดี่ยววงกลม (Singly Circular Linked List)

เมื่อให้พอยน์เตอร์ของโหนดสุดท้ายในลิงค์ลิสต์เดี่ยว (ที่มีโหนดแรกคือโหนด H) ชี้ไปยังโหนดแรกซึ่งเป็นโหนดพิเศษ เรียกว่า เฮดโหนด (Head Node) จะได้โครงสร้างลิงค์ลิสต์ เดี่ยววงกลมดังรูป 3.12 โดยโครงสร้างแบบนี้จะสะดวกกว่าแบบลิงค์ลิสต์เดี่ยวธรรมดา ในแง่ที่ว่าถัดจากโหนดสุดท้ายก็เป็นโหนดแรกเลย ดังนั้นการไล่พอยน์เตอร์จึงมีความสะดวกในบางครั้ง เพราะพอยน์เตอร์จะไม่มีโอกาสตกหายไประหว่างการท่องไปในลิงค์ลิสต์



รูปที่ 3.12 แสดงลิงค์ลิสต์เดี่ยววงกลม (Singly Circular Linked List)

คุณสมบัติของเฮดโหนด (Head Node)

1. มีข้อมูลแปลก ๆ ที่ไม่ตรงกับข้อมูลปกติ (ดังรูป 3.12 H จะมีข้อมูลเป็นประเภท char เช่นเดียวกับ A,B,C แต่ไม่ใช่ A-Z)

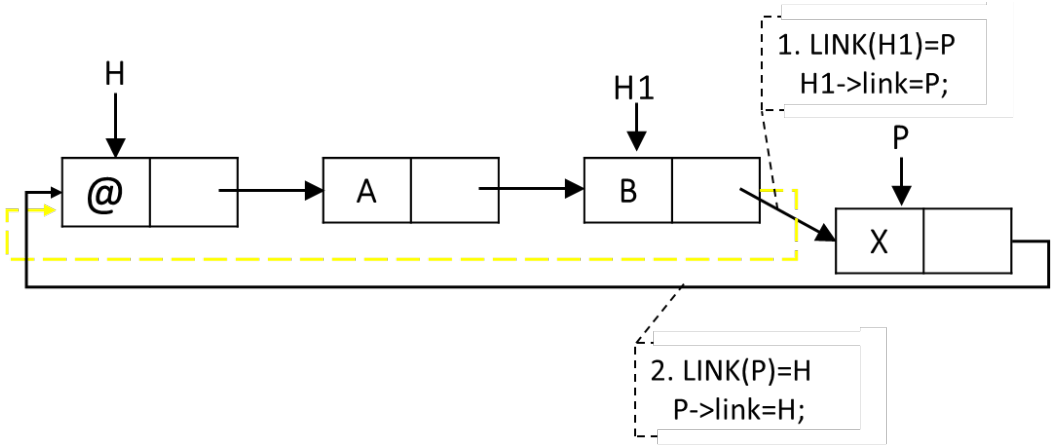
2. ห้ามลบ

3. หากไม่มีโหนดปกติอื่น พอยน์เตอร์ LINK จะชี้เข้าหาตนเอง

การวางโครงสร้างโปรแกรมลิงค์ลิสต์เดี่ยววงกลม

จะคล้ายกับลิงค์ลิสต์เดี่ยวเพียงแต่ต้องคำนึงถึงเฮดโหนด (Head Node) ซึ่งจะอยู่ใน โมดูล **“การแทรก”** และ **“การลบ”** ดังนี้

การแทรก : กรณีที่โหนดที่แทรกเป็นโหนดสุดท้ายของลิสต์ ต้องให้ LINK ของโหนด สุดท้าย ต้องกลับมาชี้ที่เฮดโหนด



รูปที่ 3.13 การแทรกโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

1. ALLOCATE P

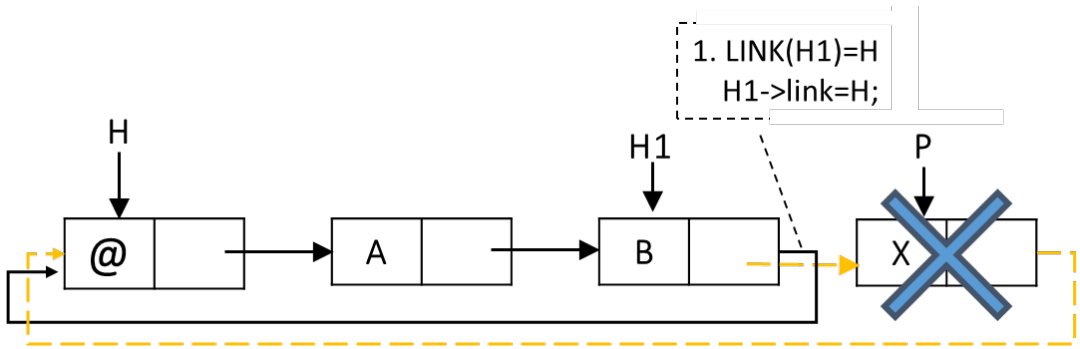
2. LINK(H1)=P

3. LINK(P)=H

การลบ :

1. หากโหนดที่อยู่หลัง H1 เป็นเฮดโหนดจะไม่อนุญาตให้ลบเพราะคุณสมบัติของ ลิงค์ลิสต์วงกลมที่มีเฮดโหนดนั้นระบุว่า **“ห้ามลบเฮดโหนด”**

2. หากโหนดที่ลบเป็นโหนดสุดท้ายต้องให้ LINK ของโหนด H1 วกกลับไปชี้เฮดโหนด



รูปที่ 3.14 การลบโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)

ขั้นตอนการย้ายพอยน์เตอร์และสั่งงานจะเป็นดังนี้

1. P=LINK(H1)
2. LINK(H1)=H
3. FREE(P)

3.1.1.5 โปรแกรมลิงค์ลิสต์เดี่ยววงกลม

```
/* Program create SINGLY CIRCULAR LINKED LIST by...
 1. Create Node N Nodes
 2. Can Insert node after data as defined
 3. Can Delete node after data as defined
 4. Show address of Node
Note.- Not allow to delete the HEAD Node
*/
#include <stdio.h> //use printf
#include <conio.h> //use getch
#include <stdlib.h> //use molloc
#define HeadData -999 //Special Data of Head Node
struct Node //Declare structure of node
{
    int info;
    struct Node *link;
};
struct Node *H, *H1, *p, *q; // Declare pointer node
int i,j,k,n,data;
char ch;
Node *Allocate() //Allocate 1 node from storage pool
{
    struct Node *temp;
    temp=(Node*)malloc(sizeof(Node)); //Allocate node by size declare
```

```
    return(temp);
}
void CreateNNode(int n) //Create N Node put data and link it
{
    int i,temp;
    H=p;H1=p;
    for (i=1;i<=n;i++) //Count N Node
    {
        p=Allocate(); //Allocate Node
        temp=1+rand() % 99; //random difference number 1..99
        p->info=temp; //Put random data in to node
        H1->link=p; //Let Last node point to new node
        H1=p; //Let H1 point to new node
        H1->link=H; //Set link of H1 to HEAD for Circular Linked List
    }
}
void ShowAllNode()
{
    printf("H = %x\n",H); //Display address of pointer H
    p=H->link; //Set start point of p pointer at first node
    i=1; //set start value of counter
    while (p->info != HeadData) //While if INFO is NOT HeadData
    {
        printf("%d) : %x\t",i,p); //Show COUNTER and POINTER
        printf("INFO : %d\t",p->info); //Show INFO
        printf("LINK : %x\n",p->link); //Show LINK
        p=p->link; //Skip to next node
        i++; //Skip Counter
    } //End While
} //Enf Fn.
void InsertAfter(int data1)
{
    int temp; //Temporary variable
    if (H->link == H) //If Link point back to HeadNode
        printf("Circular Linked List have no node!!..\n");
    else
    {
        H1=H->link; //Let H1 point at first node
        while (H1->info != HeadData) // Search for the data while
INFO<>HeadData
        {
            if (H1->info == data1) //if Found
            {
                p=Allocate(); //Allocat one node from storage pool
                printf("\nInsert data : " ); //Input data for insert
                scanf("%d",&temp); //Read from KBD
```

```
                p->info=temp; // Entry temporary data into INFO of node
                p->link=H1->link; //Change pointer 1st for insert node
(FAR)
                H1->link=p; //Change pointer 2nd for insert node (NEAR)
            } //End if
            H1=H1->link; //Skip H1 to next node
        } //End while
    } //End IF
} //End Fn.
void DeleteAfter(int data1)
{
    int temp; //Temporary variable
    if (H->link == H) //If Link point back to HeadNode
        printf("Circular Linked List have no node!!..\n");
    else
    {
        H1=H->link; //Let H1 point at start node
        while (H1->info != HeadData) // Search for the data while
INFO<>HeadData
        {
            if (H1->info == data1) //if Found
            {
                if (H1->link==H) //If no more node
                    printf ("This is the HEAD Node,Can't delete
it!!!\n");
                else
                {
                    p=H1->link; //Mark at node for Delete
                    if(p->link==H) //If p is last node
                        H1->link=H; //Set link of H1 to Head node
                    else
                        H1->link=p->link; //If not set link of H1 point
same address of p
                    free(p); //Free node to storage pool
                } //End if2
            } //End if1
            H1=H1->link; //Skip H1 to next node
        } //End while
    } //End IF
} //End Fn.

int main() //MAIN Fn.
{
    p=Allocate(); //Create HEAD Node
    p->info=HeadData; //Assign Special data
    p->link=p; //Link back to Node
```

```
    n=10; //Set amount of node
    CreateNNode(n); //Call Fn. Create N nodes
    printf("PROGRAM SINGLY CIRCULAR LINKED LIST \n");
    printf("======================================== \n");
    printf("All Data in Linked List \n");
    ShowAllNode(); //Call Fn. Show all node
    ch=' ';
    while (ch != 'E')
    {
        printf("MENU : [I:Insert D:Delete E:Exit]");
        ch=getch();
        switch (ch)
        {
            case 'I' : printf("\nInsert After data : " ); //Input data
for insert after
                scanf("%d",&data);
                InsertAfter(data); //Call Fn. Insert after data
                printf("\nAll Data in Linked List AFTER INSERTED\n");
                ShowAllNode(); //Call Fn. Show all node
                break;
            case 'D' : printf("\nDelete After data : " ); //Input data
for Delete after
                scanf("%d",&data);
                DeleteAfter(data); //Call Fn. Delete after data
                printf("\nAll Data in Linked List AFTER DELETED\n");
                ShowAllNode(); //Call Fn. Show all node
                break;
        } // End Switch...case
    } //End While
    return(0);
} //End MAIN
```

```
PROGRAM SINGLY CIRCULAR LINKED LIST
========================================
All Data in Linked List
H = b21440
1) : b21460     INFO : 42       LINK : b21480
2) : b21480     INFO : 54       LINK : b214a0
3) : b214a0     INFO : 98       LINK : b214c0
4) : b214c0     INFO : 68       LINK : b214e0
5) : b214e0     INFO : 63       LINK : b21500
6) : b21500     INFO : 83       LINK : b21520
7) : b21520     INFO : 94       LINK : b21540
8) : b21540     INFO : 55       LINK : b21560
9) : b21560     INFO : 35       LINK : b21580
10) : b21580    INFO : 12       LINK : b21440
MENU : [I:Insert D:Delete E:Exit]
```

รูปที่ 3.15 แสดงผลการทำงานของโปรแกรมลิงค์ลิสต์เดี่ยววงกลม