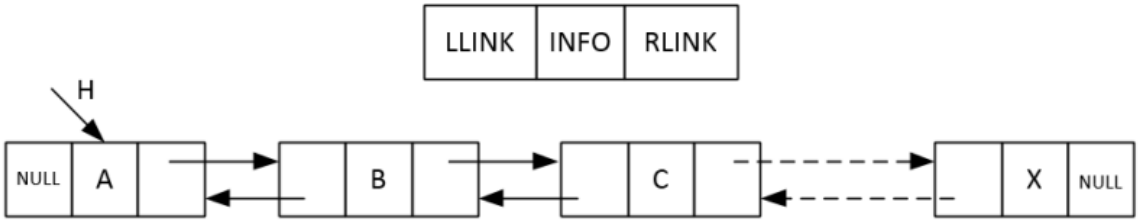
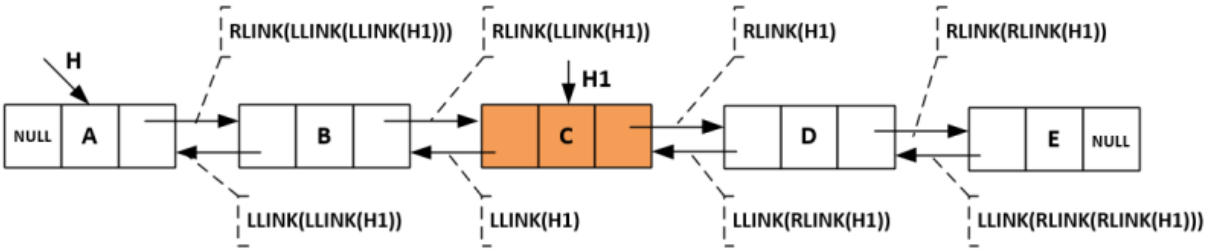
3.1.2 ลิงค์ลิสต์คู่ (Doubly Linked List)

ลิงค์ลิสต์แบบนี้เป็นลิงค์ลิสต์ที่แต่ละโหนดมีสองพอยน์เตอร์คือ LLLNK และ RLINK ซึ่งจะชี้ไปยังโหนดทางซ้ายและทางขวาของโหนดนั้นตามลำดับ ดังตัวอย่างเช่น ถ้าต้องการจะ เก็บค่า A,B,C X ในลิงค์ลิสต์ชนิดนี้จะได้โครงสร้างข้อมูลดังรูป 3.16 ซึ่งเมื่อเปรียบเทียบ ลิงค์ลิสต์คู่กับลิงค์ลิสต์เดี่ยว จะเห็นได้ว่าลิงค์ลิสต์คู่ต้องใช้เนื้อที่มากกว่า เนื่องจากต้องใช้ 2 พอยน์เตอร์สำหรับแต่ละโหนดแน่นอน แต่ที่เสียไปนั้นก็เพื่อที่จะได้สิ่งบางอย่างเพิ่มเติม นั่น คือ ความสะดวกในการนำข่าวสารเข้าหรือออก จากลิงค์ลิสต์ ในกรณีลิงค์ลิสต์คู่ ซึ่งสามารถ นำโหนดใหม่แทรกเข้าทาง ด้านหน้า หรือ ด้านหลังโหนด H1 ในลิงค์ลิสต์นั้นได้



รูปที่ 3.16 แสดงลิงค์ลิสต์คู่

ด้วยเหตุที่ลิงค์ลิสต์คู่มีพอยน์เตอร์ถึง 2 ตัวต่อโหนด และสามารถชี้ไปทางซ้าย หรือ ไปทางขวาก็ได้ อีกทั้งยังเชื่อมโยงถึงกันในแต่ละโหนดอีกด้วย ดังนั้นเพื่อให้การระบุตำแหน่งของพอยน์เตอร์แต่ละจุดเป็นไปอย่างถูกต้อง และสัมพันธ์กันในเชิงตำแหน่ง จึงอธิบายชื่อขอ พอยน์เตอร์ในแต่ละจุดดังรูป 3.17 โดยใช้ H1 เป็นตำแหน่งอ้างอิง

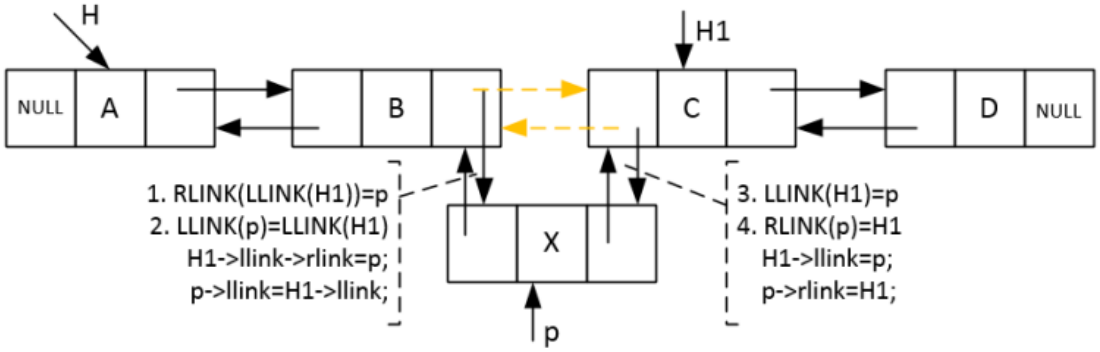


รูปที่ 3.17 แสดงชื่อเรียกของพอยน์เตอร์ต่าง ๆ

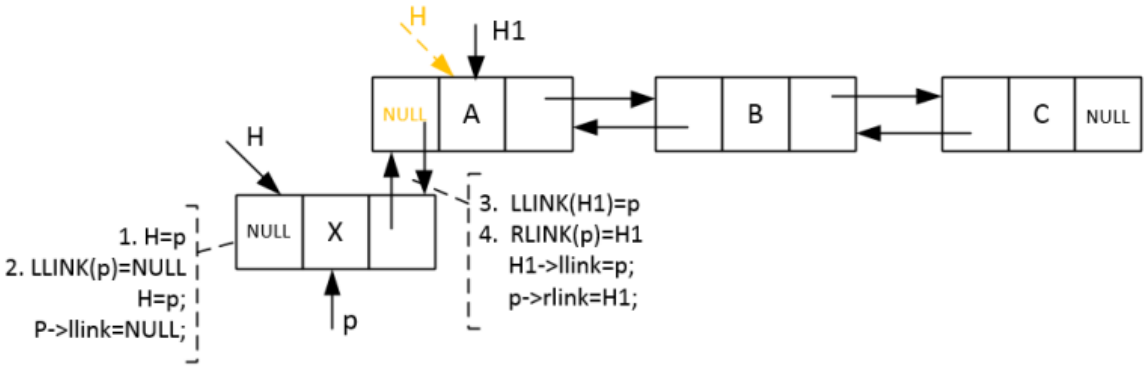
3.1.2.1 การแทรกและลบโหนดในลิงค์ลิสต์คู่

การแทรกโหนดในลิงค์ลิสต์คู่

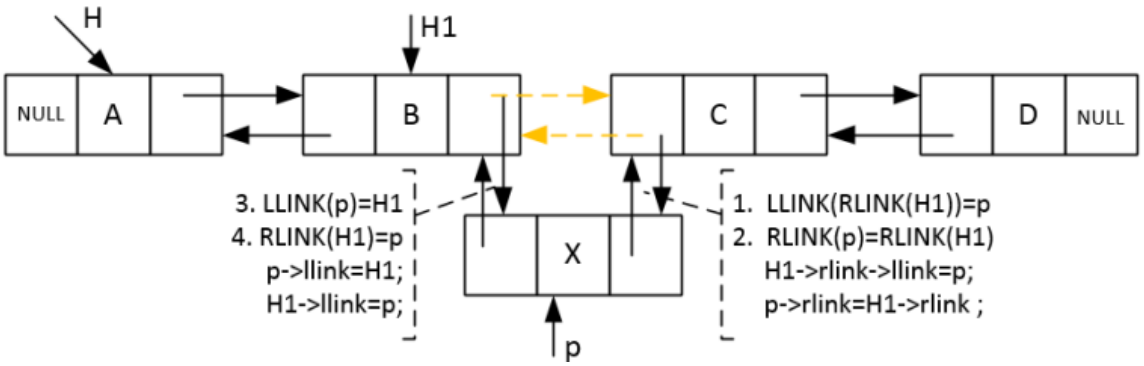
สมมติว่ามีลิงค์ลิสต์คู่อยู่ดังรูป 3.18-3.21 และจะนำโหนด P ซึ่งเก็บข่าวสาร X แทรกเข้าสู่ลิงค์ลิสต์นี้ ณ ตำแหน่งที่ H1 ชี้อยู่ จะเห็นได้ว่ากระบวนการนี้สำเร็จได้โดยการ เปลี่ยนค่าพอยเตอร์ 4 ค่า อย่างไรก็ดีลำดับการเปลี่ยนค่าพอยเตอร์มีความสำคัญมาก หาก เปลี่ยนพอยน์เตอร์ไม่ถูกขั้นตอน จะทำให้การแทรกโหนดล้มเหลว และด้วยเหตุที่การแทรก โหนด สามารถกระทำได้ทั้งแทรกก่อนแทรกหลังโหนดที่ระบุ (H1) ทั้งยังต้องพิจารณากรณีที่ H1 เป็นโหนดแรก หรือ เป็นโหนดสุดท้ายหรือไม่ด้วย ดังนั้นลักษณะการแทรกจึงมีทั้งหมด 4 แบบ ดังนี้



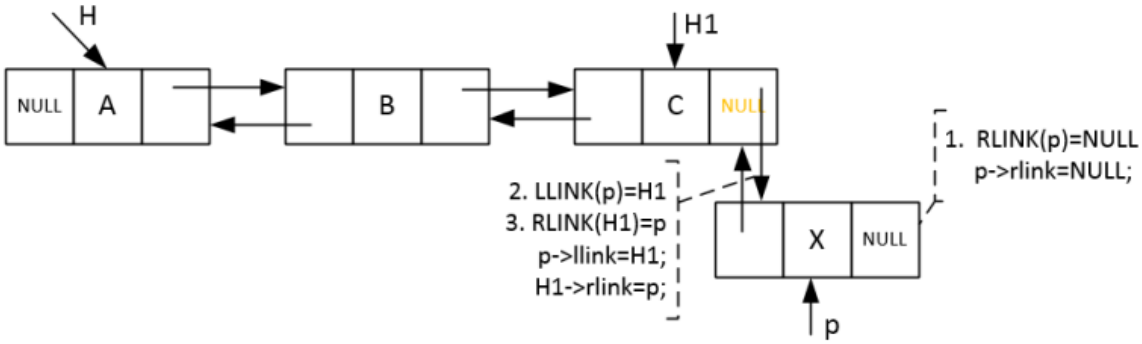
รูปที่ 3.18 การแทรกโหนดก่อน H1 (กรณี H1 ไม่ใช่โหนดแรก)



3.19 การแทรกโหนดก่อน H1 (กรณี H1 เป็นโหนดแรก)



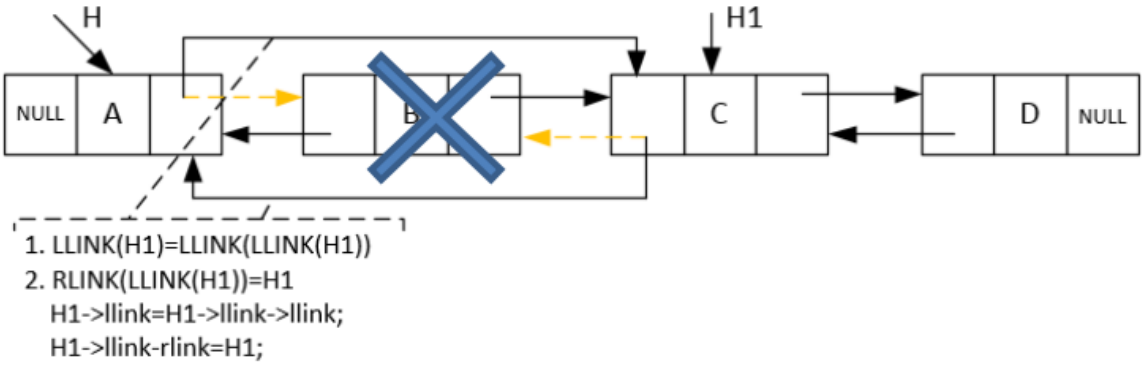
รูปที่ 3.20 การแทรกโหนดหลัง H1 (กรณี H1 ไม่ใช่โหนดสุดท้าย)



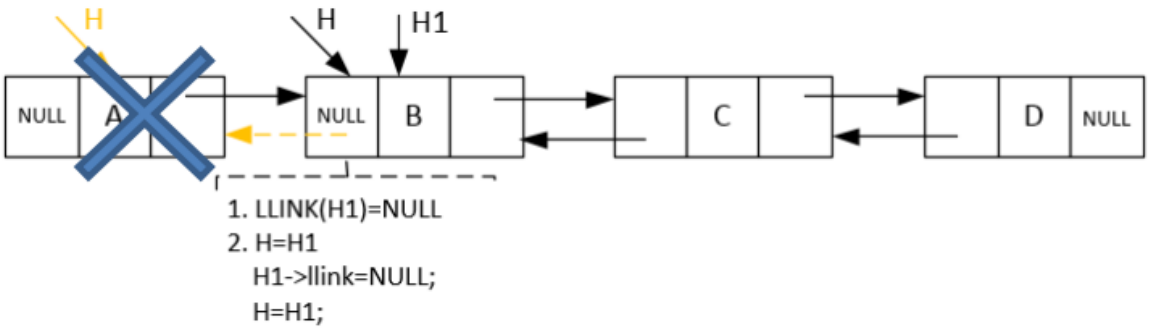
รูปที่ 3.21 การแทรกโหนดหลัง H1 (กรณี H1 เป็นโหนดสุดท้าย)

การลบโหนดในลิงค์ลิสต์คู่

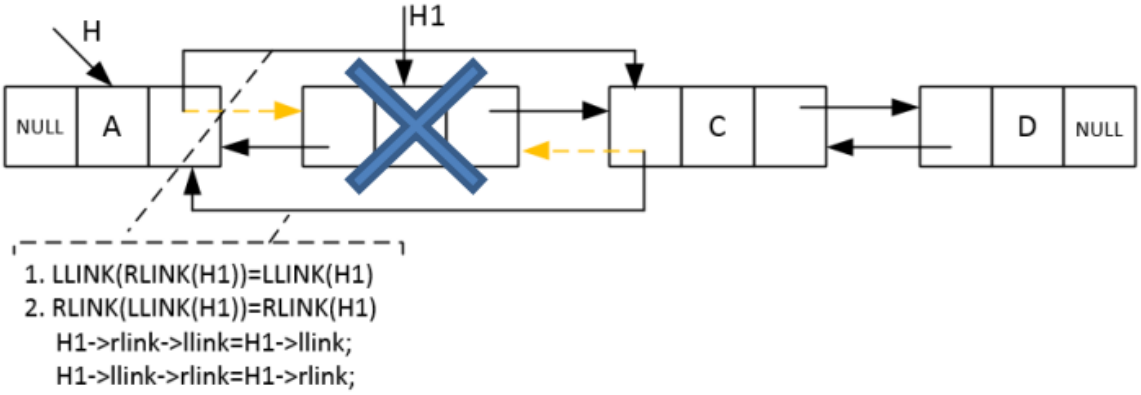
การลบโหนดที่อ้างอิงโดย H1 ออกจากลิงค์ลิสต์คู่สามารถทำได้ด้วยการเปลี่ยนค่า พอยน์เตอร์ 2 ค่า และการลบสามารถลบได้ทั้งโหนดก่อนหน้า ลบโหนดตนเอง และลบโหนด หลัง ทั้งยังต้องพิจารณากรณีที่เป็นโหนดแรก หรือ เป็นโหนดสุดท้ายด้วย ดังนั้นลักษณะการ ลบจึงมีทั้งหมด 7 รูปแบบ ดังรูป 3.22



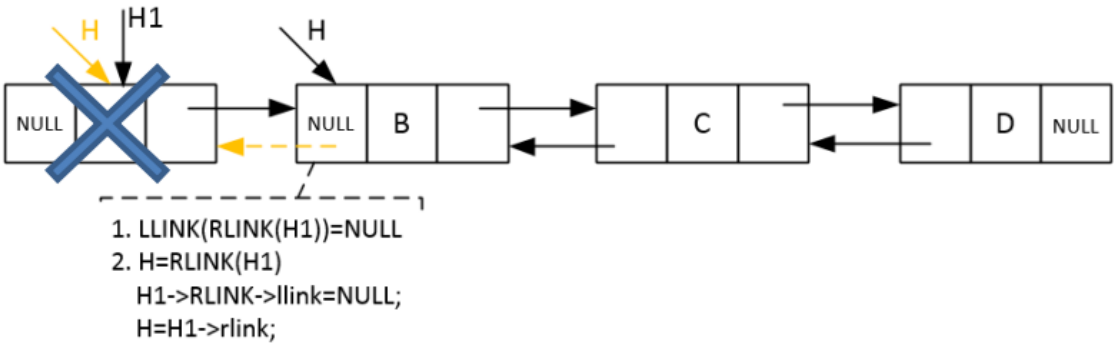
รูปที่ 3.22 การลบโหนดก่อน H1 (กรณีโหนดก่อน H1 ไม่ใช่โหนดแรก)



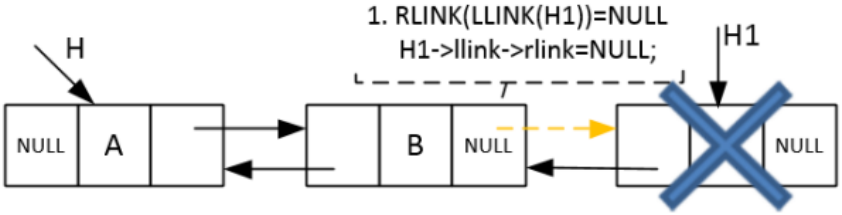
รูปที่ 3.23 การลบโหนดก่อน H1 (กรณีโหนดก่อน H1 เป็นโหนดแรก)



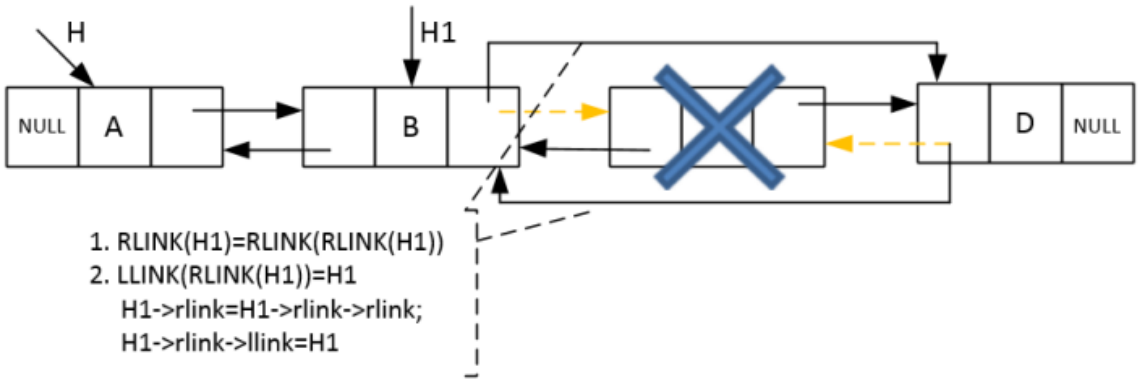
รูปที่ 3.24 การลบโหนด H1 (กรณีโหนดก่อน H1 ไม่ใช่โหนดแรก)



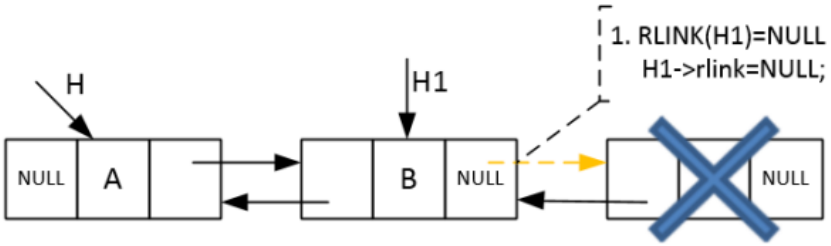
รูปที่ 3.25 การลบโหนด H1 (กรณีโหนด H1 เป็นโหนดแรก)



รูปที่ 3.26 การลบโหนด H1 (กรณีโหนด H1 เป็นโหนดสุดท้าย)



รูปที่ 3.27 การลบโหนดหลังโหนด H1 (กรณีไม่ใช้โหนดสุดท้าย)



รูปที่ 3.28 การลบโหนดหลังโหนด H1 (กรณีเป็นโหนดสุดท้าย)

การวางโครงสร้างโปรแกรมลิงค์ลิสต์คู่

ด้วยเหตุที่รูปแบบการแทรก และการลบของลิงค์ลิสต์คู่มีหลากหลายรูปแบบ ดังนั้น - จึงแบ่งโมดูลการทำงานออกตามลักษณะรูปแบบการทำงานซึ่งมีทั้งสิ้น 5 โมดูลดังนี้

1. แทรกก่อน (InsertBefore) : หลังจากที่ค้นหาข้อมูลของโหนดที่จะทำการแทรก พบแล้ว ก็จะใช้ H1 ชี้ไว้ และทำการสร้างโหนดใหม่ขึ้นมา 1 โหนด รับค่าข้อมูลเข้าไปเก็บใน info จากนั้นก็ตรวจสอบว่า H1 เป็นโหนดแรกหรือไม่ ถ้าใช่ หรือถ้าไม่ใช่ ก็ดำเนินการแทรก ตามขั้นตอนในรูปที่ 3.18 หรือ 3.19 แล้วแต่กรณี

2. แทรกหลัง (InsertAfter) : หลังจากที่ค้นหาข้อมูลของโหนดที่จะทำการแทรก พบแล้ว ก็จะใช้ H1 ชี้ไว้ และทำการสร้างโหนดใหม่ขึ้นมา 1 โหนด รับค่าข้อมูลเข้าไปเก็บในinfo จากนั้นก็ตรวจสอบว่า H1 เป็นโหนดสุดท้ายหรือไม่ ถ้าใช่ หรือถ้าไม่ใช่ ก็ด แทรกตามขั้นตอนในรูปที่ 3.20 หรือ 3.21 แล้วแต่กรณี

3. ลบก่อน (DeleteBefore) : หลังจากที่ค้นหาข้อมูลของโหนดที่จะทำการ แล้ว ก็จะใช้ H1 ชี้ไว้และตรวจสอบว่า H1 เป็นโหนดแรกหรือไม่ ถ้าเป็นก็จะไม่อนุญาตให้ โหนดก่อนหน้า (เพราะโหนดก่อนหน้า H1 ไม่มี) แต่หากไม่ใช่โหนดแรกก็ตรวจสอบต่อไป โหนดก่อน H1 ที่จะทำการลบเป็นโหนดแรกหรือไม่ ถ้าใช่ หรือไม่ใช่ ก็ดำเนินการลบต ขั้นตอนดังรูป 3.22 หรือ 3.23 แล้วแต่กรณี

4. ลบโหนดตนเอง (DeleteSelf) : หลังจากที่ค้นหาข้อมูลของโหนดที่จะทำการ ลบพบแล้ว ก็จะใช้ H1 ชี้ไว้และตรวจสอบว่า H1 เป็นโหนดแรกหรือไม่ หรือไม่ใช่โหนดแรก หรือเป็นโหนดสุดท้าย ก็ดำเนินการลบตามขั้นตอนดังรูป 3.24 หรือ 3.25 หรือ 3.26 แล้วแต่ กรณี

5. ลบโหนดหลัง (DeleteAfter) : หลังจากที่ค้นหาข้อมูลของโหนดที่จะทำการ ลบพบแล้ว ก็จะใช้ H1 ชี้ไว้และตรวจสอบว่า H1 เป็นโหนดสุดท้ายหรือไม่ ถ้าเป็นก็จะไม่ อนุญาตให้ลบโหนดหลัง (เพราะโหนดหลัง H1 ไม่มี) แต่หากไม่ใช่โหนดสุดท้ายก็ตรวจสอบ ต่อไปว่า โหนดหลัง H1 ที่จะทำการลบเป็นโหนดสุดท้ายหรือไม่ ถ้าใช่ หรือไม่ใช่ ก็ดำเนินการ ลบตามขั้นตอนดังรูป 3.27 หรือ 3.28 แล้วแต่กรณี

3.1.2.1 โปรแกรมลิงค์ลิสต์คู่

```
/* Program create DOUBLY LINKED LIST by...
 1. Create Node N Nodes
 2. Can Insert node before/after data as defined
 3. Can Delete node before/itself/after data as defined
 4. Show address of Node
*/
#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 *llink;
  struct Node *rlink;
 };
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->rlink=p; //Link first node to second node
  p->llink=H1;
  H1=p; //Let H1 point to last node
  H1->rlink=NULL; //Set rlink 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("LLINK : %x\t",p->llink); //Show lLINK
    printf("INFO : %d\t",p->info); //Show INFO
    printf("RLINK : %x\n",p->rlink); //Show RLINK
    p=p->rlink; //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
        if (H1->rlink == NULL)
        {
          p->rlink=NULL;
```

```
        }
        else
        {
          p->rlink=H1->rlink; //Change pointer 1st for insert node (FAR
to NEAR)
          H1->rlink->llink=p; //LLINK(RLINK(H1))=p
        }
        p->llink=H1; //LLINK(P)=H1
        H1->rlink=p; //RLINK(H1)=p
      } //End if
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn.
void InsertBefore(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
        if (H1->llink == NULL)
        {
          p->llink=NULL;
          H=p;
        }
        else
        {
          H1->llink->rlink=p; //RLINK(LLINK(H1))=p
          p->llink=H1->llink; //LLINK(p)=LLINK(H1)
        }
        H1->llink=p;//LLINK(H1)=p
        p->rlink=H1;//RLINK(P)=H1
      } //End if
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
```

```
 } //End Fn.
void DeleteBefore(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->llink==NULL) //If no more node
          printf ("No more node from here,Can't delete it!!!\n");
        else
        {
          p=H1->llink; //Mark at node for Delete
          if(p->llink==NULL) //If p is first node
          {
            H1->llink=NULL; //Set link of H1 to NULL
            H=H1; //Set H to first NODE
          }
          else
          {
            H1->llink=p->llink; //If not set link of H1 point same
address of p
            p->llink->rlink=H1;
          }
          free(p); //Free node to storage pool
        } //End if2
      } //End if1
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn.

void DeleteSelf(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=H1; //Mark at node for Delete
        if(p->llink==NULL && p->rlink==NULL) //If only one node
          H=NULL;
        else
        {
          if(p->llink==NULL) //Check if first NODE
          {
            H=p->rlink; //Let H point to RLINK(P)
            H->llink=NULL; //Assign LLINK(H)=NULL
          }
          else
          {
            if(p->rlink==NULL) //Check if Last NODE
              p->llink->rlink=NULL;
            else //NORMAL
            {
              p->llink->rlink=p->rlink;
              p->rlink->llink=p->llink;
            }
          }
        }
        free(p); //Free node to storage pool
      } //End if1
      H1=H1->rlink; //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->rlink==NULL) //If no more node
          printf ("No more node from here,Can't delete it!!!\n");
        else
```

```
        {
          p=H1->rlink; //Mark at node for Delete
          if(p->rlink==NULL) //If p is last node
            H1->rlink=NULL; //Set link of H1 to NULL
          else
          {
            H1->rlink=p->rlink; //If not set link of H1 point same
address of p
            p->rlink->llink=H1;
          }
          free(p); //Free node to storage pool
        } //End if2
      } //End if1
      H1=H1->rlink; //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 DOUBLY LINKED LIST \n");
  printf("================================= \n");
  printf("All Data in Linked List \n");
  ShowAllNode(); //Call Fn. Show all node
  ch=' ';
  while (ch != 'E')
  {
    printf("MENU>> [B:InsertBefore    A:InsertAfter\n");
    printf(" O:DeleteBefore X:Delete itself\n");
    printf(" D:DeleteAfter E:Exit]");
    ch=getch();
    switch (ch)
    {
    case 'B' :  printf("\nInsert After data : " ); //Input data for
insert before
          scanf("%d",&data);
          InsertBefore(data); //Call Fn. Insert after data
          printf("\nAll Data in Linked List AFTER INSERTED\n");
          ShowAllNode(); //Call Fn. Show all node
          break;

    case 'A' :  printf("\nInsert After data : " ); //Input data for
insert after
          scanf("%d",&data);
```

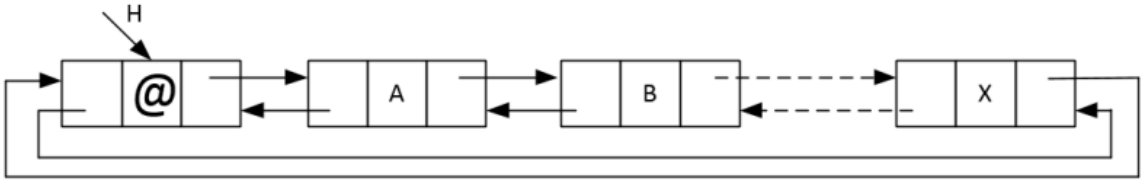
```
            InsertAfter(data); //Call Fn. Insert after data
            printf("\nAll Data in Linked List AFTER INSERTED\n");
            ShowAllNode(); //Call Fn. Show all node
            break;
        case 'O' :  printf("\nDelete Before data : " ); //Input data for
Delete after
            scanf("%d",&data);
            DeleteBefore(data); //Call Fn. Delete after data
            printf("\nAll Data in Linked List AFTER DELETED\n");
            ShowAllNode(); //Call Fn. Show all node
            break;
        case 'X' :  printf("\nDelete ItSelf data : " ); //Input data for
Delete after
            scanf("%d",&data);
            DeleteSelf(data); //Call Fn. Delete after data
            printf("\nAll Data in Linked List ITSELF DELETED\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 */
      getch(); //Wait for KBD hit
} //End MAIN
```

```
C:\Users\LENOVO\Downloads\DLL.exe
PROGRAM DOUBLY LINKED LIST
==================================
All Data in Linked List
H = 1c13e0
1) : 1c13e0     LLINK : 0       INFO : 42       RLINK : 1c1400
2) : 1c1400     LLINK : 1c13e0  INFO : 54       RLINK : 1c1420
3) : 1c1420     LLINK : 1c1400  INFO : 98       RLINK : 1c1440
4) : 1c1440     LLINK : 1c1420  INFO : 68       RLINK : 1c1460
5) : 1c1460     LLINK : 1c1440  INFO : 63       RLINK : 1c1480
6) : 1c1480     LLINK : 1c1460  INFO : 83       RLINK : 1c14a0
7) : 1c14a0     LLINK : 1c1480  INFO : 94       RLINK : 1c14c0
8) : 1c14c0     LLINK : 1c14a0  INFO : 55       RLINK : 1c14e0
9) : 1c14e0     LLINK : 1c14c0  INFO : 35       RLINK : 1c1500
10) : 1c1500    LLINK : 1c14e0  INFO : 12       RLINK : 0
MENU>> [B:InsertBefore          A:InsertAfter
 O:DeleteBefore X:Delete itself
 D:DeleteAfter E:Exit]
```

รูปที่ 3.29 แสดงผลการทำงานของโปรแกรมลิงค์ลิสต์คู่

3.1.2.3 ลิงค์ลิสต์คู่วงกลม (Doubly Circular Linked List)

เมื่อให้พอยน์เตอร์ RLINK ของโหนดสุดท้ายในลิงค์ลิสต์คู่ (ที่มีโหนดแรกคือโหนด) ชี้ไปยังโหนดแรกซึ่งเป็นโหนดพิเศษ เรียกว่า เฮดโหนด (Head Node) และให้พอยน์เตอร์ LLINK ของเฮดโหนดชี้ไปยังโหนดสุดท้าย จะได้โครงสร้างลิงค์ลิสต์คู่แบบวงกลมดังรูป 3.29



รูปที่ 3.30 แสดงลิงค์ลิสต์คู่วงกลม

คุณสมบัติของเฮดโหนด (Head Node)

1. มีข้อมูลแปลก ๆ ที่ไม่ตรงกับข้อมูลปกติ
2. ห้ามลบ
3. หากไม่มีโหนดปกติอื่น พอยน์เตอร์ LLINK และ RLINK จะชี้เข้าหาตนเอง

การวางโครงสร้างโปรแกรมลิงค์ลิสต์คู่วงกลม

จะคล้ายกับลิงค์ลิสต์คู่เพียงแต่ต้องคำนึงถึงเฮดโหนด (Head Node) ซึ่งจะอยู่ใน โมดูลดังนี้

การแทรก : กรณีที่โหนดที่แทรกเป็นโหนดสุดท้ายของลิสต์ ต้องให้ RLINK ของ โหนดสุดท้ายต้องกลับมาชี้ที่เฮดโหนด

การลบ :

1. หากโหนดที่จะลบเป็นเฮดโหนดจะไม่อนุญาตให้ลบเพราะคุณสมบัติของลิงค์ ลิสต์วงกลมที่มีเฮดโหนดนั้นระบุว่า “ห้ามลบเฮดโหนด”
2. หากโหนดที่ลบเป็นโหนดสุดท้ายต้องให้ RLINK ของโหนด H1 วกกลับไปชี้เฮดโหนด หรือหากลิงค์ลิสต์นั้นมีสมาชิกเพียงแค่โหนดเดียวหากลบโหนดนั้นแล้ว ต้องให้ทั้ง LINK และ RLINK วกชี้ที่เฮดโหนด

3.1.2.4 โปรแกรมลิงค์ลิสต์คู่วงกลม

```
/* Program create DOUBLY CIRCULAR LINKED LIST by...
 1. Create Node N Nodes
 2. Insert node before/after data as defined
 3. Delete node before/itself/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> //usemolloc
#define HeadInfo -999 // Define data of Head Node

struct Node { //Declare structure of node
  int info;
  struct Node *llink;
  struct Node *rlink;
 };
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 its
  int i,temp;
  H1=H; //Start H1 at here
  for (i=1;i<=n;i++) {//Count N Node
    p=Allocate(); //Allocate New Node
    temp=1+rand() % 99; //random difference number 1..99
    p->info=temp; //Put random data in to node
    H1->rlink=p; //Link first node to second node
    p->llink=H1; //LLink point back to predecessor node
    H1=p; //Let H1 point to last node
    H1->rlink=H; //Set rlink of H1 point to HEAD NODE
    H->llink=H1; //Set LLink of H point to H1
  }
 }
 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
  if (p->rlink != H ) //if have more node
  {
    p=p->rlink; //Skip pointer to first node
    while (p != H) //While if P <> H
    {
      printf("%d) : %x\t",i,p); //Show COUNTER and POINTER
      printf("LLINK : %x\t",p->llink); //Show lLINK
      printf("INFO : %d\t",p->info); //Show INFO
      printf("RLINK : %x\n",p->rlink); //Show RLINK
      p=p->rlink; //Skip to next node
      i++; //Skip Counter
    } //End While
  } //End if
 } //Enf Fn.
void InsertAfter(int data1)
 {
  int temp; //Temporary variable
  if (H->rlink == H)
    printf("Linked List have no node!!..\n");
  else
  {
    H1=H->rlink; //Let H1 point at 1st node
    while (H1->info != HeadInfo) // Search for the data while H1 loop
back to HAED Node
    {
      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
        if (H1->rlink == H) //IF H1 is Last Node
        {
          p->rlink=H; //Let p Point to HEAD Node
          H->llink=p; //Let H Point to Last Node
        }
        else
        {
          p->rlink=H1->rlink; //Change pointer 1st for insert node (FAR
to NEAR)
          H1->rlink->llink=p; //LLINK(RLINK(H1))=p
        }
        p->llink=H1; //LLINK(P)=H1
        H1->rlink=p; //RLINK(H1)=p
```

```
      } //End if
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn.

void InsertBefore(int data1)
 {
  int temp; //Temporary variable
  if (H->rlink==H)
    printf("Linked List have no node!!..\n");
  else
  {
    H1=H->rlink; //Let H1 point at 1st node
    while (H1->info != HeadInfo) // Search for the data while H1 loop
back to HAED Node
    {
      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
        if (H1->llink == H) //First Node
        {
          p->llink=H; //LLINK(p) Point to HEAD Node
          H->rlink=p; //RLINK(H) Point to p
        }
        else
        {
          H1->llink->rlink=p; //RLINK(LLINK(H1))=p
          p->llink=H1->llink; //LLINK(p)=LLINK(H1)
        }
        H1->llink=p;//LLINK(H1)=p
        p->rlink=H1;//RLINK(P)=H1
      } //End if
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn

 void DeleteBefore(int data1)
 {
  int temp; //Temporary variable
  if (H->rlink==H)
    printf("Linked List have NO NODE!!..\n");
```

```
  else
  {
    H1=H->rlink; //Let H1 point at 1st node
    while (H1->info != HeadInfo) // Search for the data while H1 loop
back to HAED Node
    {
      if (H1->info == data1) //if Found
      {
        if (H1->llink==H) //If no more node
          printf ("No more node before here,Can't delete it!!!\n");
        else
        {
          p=H1->llink; //Mark at node for Delete
          H1->llink=p->llink; //If not set link of H1 point same
address of p
          p->llink->rlink=H1;
          free(p); //Free node to storage pool
        } //End if2
      } //End if1
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn.

void DeleteSelf(int data1)
 {
  int temp; //Temporary variable
  if (H->rlink==H)
    printf("Linked List have NO NODE!!..\n");
  else
  {
    H1=H->rlink; //Let H1 point at 1st node
    while (H1->info != HeadInfo) // Search for the data while H1 loop
back to HAED Node
    {
      if (H1->info == data1) //if Found
      {
        p=H1; //Mark at node for Delete
        if(p->llink==H && p->rlink==H) //If only one node
        {
          H->llink=H; //Set HEAD Pointer point it self
          H->rlink=H;
        }
        else
        {
          p->llink->rlink=p->rlink;
```

```
          p->rlink->llink=p->llink;
        }
        free(p); //Free node to storage pool
      } //End if1
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn.

void DeleteAfter(int data1)
 {
  int temp; //Temporary variable
  if (H->rlink==H)
    printf("Linked List have NO NODE!!..\n");
  else
  {
    H1=H->rlink; //Let H1 point at 1st node
    while (H1->info != HeadInfo) // Search for the data while H1 loop
back to HAED Node
    {
      if (H1->info == data1) //if Found
      {
        if (H1->rlink==H) //If no more node
          printf ("No more node from here,Can't delete it!!!\n");
        else
        {
          p=H1->rlink; //Mark at node for Delete
          H1->rlink=p->rlink; //If not set link of H1 point same
address of p
          p->rlink->llink=H1;
          free(p); //Free node to storage pool
        } //End if2
      } //End if1
      H1=H1->rlink; //Skip H1 to next node
    } //End while
  } //End IF
 } //End Fn

 int main() //MAIN Fn.
{
  p=Allocate();// Create HEAD NODE
  p->info=HeadInfo; //Special data for Head node
  p->llink=p; p->rlink=p; //Let both Link point to itself
  H=p; //Let H Point to Head node
  n=10; //Set amount of node
  CreateNNode(n); //Call Fn. Create N nodes
```

```
  printf("PROGRAM DOUBLY 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>> [B:InsertBefore  A:InsertAfter\n");
    printf(" O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]\n");
    //printf(" D:DeleteAfter E:Exit]");
    ch=getch();
    switch (ch)
    {
    case 'B' :  printf("\nInsert Before data : " ); //Input data for
insert after
          scanf("%d",&data);
          InsertBefore(data); //Call Fn. Insert after data
          printf("\nAll Data in Linked List AFTER INSERTED\n");
          ShowAllNode(); //Call Fn. Show all node
          break;
    case 'A' :  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 'O' :  printf("\nDelete Before data : " ); //Input data for
Delete after
          scanf("%d",&data);
          DeleteBefore(data); //Call Fn. Delete after data
          printf("\nAll Data in Linked List AFTER DELETED\n");
          ShowAllNode(); //Call Fn. Show all node
          break;
    case 'X' :  printf("\nDelete ItSelf data : " ); //Input data for
Delete after
          scanf("%d",&data);
          DeleteSelf(data); //Call Fn. Delete after data
          printf("\nAll Data in Linked List ITSELF DELETED\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 DOUBLY CIRCULAR LINKED LIST
==================================
All Data in Linked List
H = ad13e0
1) : ad1400     LLINK : ad13e0  INFO : 42       RLINK : ad1420
2) : ad1420     LLINK : ad1400  INFO : 54       RLINK : ad1440
3) : ad1440     LLINK : ad1420  INFO : 98       RLINK : ad1460
4) : ad1460     LLINK : ad1440  INFO : 68       RLINK : ad1480
5) : ad1480     LLINK : ad1460  INFO : 63       RLINK : ad14a0
6) : ad14a0     LLINK : ad1480  INFO : 83       RLINK : ad14c0
7) : ad14c0     LLINK : ad14a0  INFO : 94       RLINK : ad14e0
8) : ad14e0     LLINK : ad14c0  INFO : 55       RLINK : ad1500
9) : ad1500     LLINK : ad14e0  INFO : 35       RLINK : ad1520
10) : ad1520    LLINK : ad1500  INFO : 12       RLINK : ad13e0
MENU>> [B:InsertBefore  A:InsertAfter
 O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]
```

รูปที่ 3.31 แสดงผลการทำงานของโปรแกรมลิงค์ลิสต์คู่วงกลม