



รายงาน

เรื่อง โปรแกรม DOUBLY CIRCULAR LINKED LIST

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

รายงาน

เรื่อง โปรแกรม DOUBLY CIRCULAR LINKED LIST

จัดทำโดย

นายพงษ์พันธุ์ เลาวพงศ์

รหัสนักศึกษา 66543206019-2

เสนอ

อาจารย์ปิยพล ยืนยงสถาวร

รายงานนี้เป็นส่วนหนึ่งของวิชา

ENGCE124

โครงสร้างข้อมูลและขั้นตอนวิธี

(Data Structures and Algorithms)

หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์

สาขาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์

มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่

ภาคเรียนที่ 1 ปีการศึกษา 2567

คำนำ

รายงานฉบับนี้เป็นส่วนหนึ่งของรายวิชา ENGCE124 โครงสร้างข้อมูลและขั้นตอนวิธี (Data Structures and Algorithms) หลักสูตร วศ.บ.วิศวกรรมคอมพิวเตอร์ สาขาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์มหาวิทยาลัยเทคโนโลยีราชมงคลล้านนา ภาคพายัพ เชียงใหม่ ในระดับปริญญาตรีปีที่ 2 โดยมีจุดประสงค์ในการอธิบายโค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST รวมถึงอธิบายหลักการการทำงานของโปรแกรม DOUBLY CIRCULAR LINKED LIST และอธิบายผลลัพธ์การใช้งานโปรแกรม DOUBLY CIRCULAR LINKED LIST

ผู้จัดทำรายงานหวังว่า รายงานฉบับนี้จะเป็นประโยชน์กับผู้สนใจ หรือนักศึกษาทุกท่านที่กำลังหาศึกษาในหัวข้อของโปรแกรม DOUBLY CIRCULAR LINKED LIST หากมีข้อเสนอแนะหรือข้อผิดพลาดประการใด ผู้จัดทำขอน้อมรับไว้ และขออภัยมา ณ ที่นี้

ผู้จัดทำ

นายพงษ์พันธุ์ เลาวพงศ์

วันที่ 28/08/2567

สารบัญ

| | หน้า |
|--|------|
| คำนำ | ก |
| สารบัญ | ๗ |
| โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย | 1 |
| หลักการทำงานของโปรแกรม DOUBLY CIRCULAR LINKED LIST | 13 |
| ผลลัพธ์การใช้งานโปรแกรม DOUBLY CIRCULAR LINKED LIST | 29 |
| บรรณานุกรม | 34 |

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย

```
#include <stdio.h> // ใช้ printf

#include <conio.h> // ใช้ getch

#include <stdlib.h> // ใช้ malloc

#define HeadInfo -999 // กำหนดข้อมูลของโหนดหัว (Head Node)

struct Node { // ประกาศโครงสร้างของโหนด

    int info;

    struct Node *llink;

    struct Node *rlink;

};

struct Node *H, *H1, *p, *q; // ประกาศตัวชี้ (Pointer) สำหรับโหนด

int i,j,k,n,data;

char ch;

Node *Allocate() { // จัดสรร 1 โหนดจากพื้นที่เก็บข้อมูล

    struct Node *temp;

    temp=(Node*)malloc(sizeof(Node)); // จัดสรรโหนดตามขนาดที่กำหนด

    return(temp);

}

void CreateNNode(int n) { // สร้าง N โหนด ใส่ข้อมูลและลิงค์ไปยังโหนดอื่นๆ

    int i,temp;

    H1=H; // เริ่มต้น H1 ที่นี่

    for (i=1;i<=n;i++) { // นับ N โหนด
```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

p=Allocate(); // จัดสรรโหนดใหม่

temp=1+rand() % 99; // สุ่มตัวเลขระหว่าง 1..99

p->info=temp; // ใส่ข้อมูลสุ่มในโหนด

H1->rlink=p; // ลิงค์จากโหนดแรกไปยังโหนดที่สอง

p->llink=H1; // LLink ชี้กลับไปที่โหนดก่อนหน้า

H1=p; // ให้ H1 ชี้ไปยังโหนดสุดท้าย

H1->rlink=H; // ตั้งค่า rlink ของ H1 ให้ชี้ไปยังโหนดหัว

H->llink=H1; // ตั้งค่า LLink ของ H ให้ชี้ไปยัง H1

}

}

void ShowAllNode()

{

printf("H = %x\n",H); // แสดงที่อยู่ของ Pointer H

p=H; // กำหนดจุดเริ่มต้นของ Pointer p

i=1; // กำหนดค่าของเคาน์เตอร์เริ่มต้น

if (p->rlink != H ) // ถ้ามีโหนดมากกว่า 1 โหนด

{

p=p->rlink; // ข้าม Pointer ไปยังโหนดแรก

while (p != H) // ขณะที่ P ไม่เท่ากับ H

{

printf("%d) : %x\t",i,p); // แสดงเคาน์เตอร์และ Pointer

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

printf("LLINK : %x\t",p->llink); // แสดง LLink

printf("INFO : %d\t",p->info); // แสดงข้อมูลในโหนด

printf("RLINK : %x\n",p->rlink); // แสดง RLink

p=p->rlink; // ข้ามไปยังโหนดถัดไป

i++; // ข้ามเคาน์เตอร์

} // จบ While

} // จบ if

} // จบฟังก์ชัน

void InsertAfter(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink == H)

        printf("Linked List have no node!!...\n");

    else

    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว

        {

            if (H1->info == data1) // ถ้าพบข้อมูล

            {

                p=Allocate(); // จัดสรรโหนดจากพื้นที่เก็บข้อมูล
            }
        }
    }
}

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

printf("\nใส่ข้อมูล : "); // ป้อนข้อมูลสำหรับการแทรก

scanf("%d",&temp); // อ่านข้อมูลจากคีย์บอร์ด

p->info=temp; // ใส่ข้อมูลชั่วคราวในโหนด

if (H1->rlink == H) // ถ้า H1 เป็นโหนดสุดท้าย

{

    p->rlink=H; // ให้ p ชี้ไปที่โหนดหัว

    H->llink=p; // ให้ H ชี้ไปที่โหนดสุดท้าย

}

else

{

    p->rlink=H1->rlink; // เปลี่ยน Pointer สำหรับแทรกโหนด (ไกลไปไกล)

    H1->rlink->llink=p; // LLink(RLink(H1))=p

}

p->llink=H1; // LLink(p)=H1

H1->rlink=p; // RLink(H1)=p

} // จบ if

H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน

void InsertBefore(int data1)

```


โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have no node!!...\n");

    else

    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว

        {

            if (H1->info == data1) // ถ้าพบข้อมูล

            {

                p=Allocate(); // จัดสรรโหนดจากพื้นที่เก็บข้อมูล

                printf("\nใส่ข้อมูล : "); // ป้อนข้อมูลสำหรับการแทรก

                scanf("%d",&temp); // อ่านข้อมูลจากคีย์บอร์ด

                p->info=temp; // ใส่ข้อมูลชั่วคราวในโหนด

                if (H1->llink == H) // โหนดแรก

                {

                    p->llink=H; // LLink(p) ชี้ไปที่โหนดหัว

                    H->rlink=p; // RLink(H) ชี้ไปที่ p

                }

            }

            else
```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

    {

        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

} // จบ if

H1=H1->rlink; // ซ้ำม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน

void DeleteBefore(int data1)

{

    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!..\n");

    else

    {

        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ยังวนกลับมาที่โหนดหัว

        {

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

    if (H1->info == data1) // ถ้าพบข้อมูล
    {
        if (H1->llink==H) // ถ้าไม่มีโหนดก่อนหน้านี้
            printf ("No more node before here,Can't delete it!!!\n");
        else
        {
            p=H1->llink; // ทำเครื่องหมายที่โหนดสำหรับการลบ
            H1->llink=p->llink; // ตั้งค่า Link ของ H1 ให้ชี้ไปที่ที่อยู่เดียวกับ p
            p->llink->rlink=H1;
            free(p); // ปลดปล่อยโหนดกลับไปยังพื้นที่เก็บข้อมูล
        } // จบ if2
    } // จบ if1

    H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป
} // จบ while

} // จบ if

} // จบฟังก์ชัน

void DeleteSelf(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!!\n");

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

else

{

    H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

    while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว

    {

        if (H1->info == data1) // ถ้าพบข้อมูล

        {

            p=H1; // ทำเครื่องหมายที่โหนดสำหรับการลบ

            if(p->llink==H && p->rlink==H) // ถ้ามีเพียงหนึ่งโหนด

            {

                H->llink=H; // ตั้งค่า Pointer ของ HEAD ให้ชี้ไปที่ตัวเอง

                H->rlink=H;

            }

            else

            {

                p->llink->rlink=p->rlink;

                p->rlink->llink=p->llink;

            }

            free(p); // ปลดปล่อยโหนดกลับไปยังพื้นที่เก็บข้อมูล

        } // จบ if1

        H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป
    
```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

    } // จบ while

    } // จบ if
} // จบฟังก์ชัน

void DeleteAfter(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!..\n");

    else

    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว

        {

            if (H1->info == data1) // ถ้าพบข้อมูล

            {

                if (H1->rlink==H) // ถ้าไม่มีโหนดถัดไป

                    printf ("No more node from here,Can't delete it!!!\n");

                else

                {

                    p=H1->rlink; // ทำเครื่องหมายที่โหนดสำหรับการลบ

                    H1->rlink=p->rlink; // ตั้งค่า Link ของ H1 ให้ชี้ไปที่ที่อยู่เดียวกับ p

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

        p->rlink->llink=H1;

        free(p); // ปลอยโหนดกลับไปยังพื้นที่เก็บข้อมูล

    } // จบ if2

    } // จบ if1

    H1=H1->rlink; // ซ้ำ H1 ไปยังโหนดถัดไป

    } // จบ while

} // จบ if

} // จบฟังก์ชัน

int main() // ฟังก์ชันหลัก (MAIN)
{
    p=Allocate(); // สร้างโหนดหัว (HEAD NODE)

    p->info=HeadInfo; // ข้อมูลพิเศษสำหรับโหนดหัว

    p->llink=p; p->rlink=p; // ให้ทั้งสอง Link ชี้กลับมาที่ตัวเอง

    H=p; // ให้ H ชี้ไปที่โหนดหัว

    n=10; // กำหนดจำนวนโหนด

    CreateNNode(n); // เรียกฟังก์ชันสร้าง N โหนด

    printf("PROGRAM DOUBLY CIRCULAR LINKED LIST \n");

    printf("===== \n");

    printf("All Data in Linked List \n");

    ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

    ch=' ';

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

while (ch != 'E')

{

    printf("MENU>> [B:InsertBefore A:InsertAfter\n");

    printf(" O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]\n");

    ch=getch();

    switch (ch)

    {

    case 'B' : printf("\nInsert Before data : " ); // ป้อนข้อมูลสำหรับแทรกก่อน

                scanf("%d",&data);

                InsertBefore(data); // เรียกฟังก์ชันแทรกก่อนข้อมูล

                printf("\nAll Data in Linked List AFTER INSERTED\n");

                ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

                break;

    case 'A' : printf("\nInsert After data : " ); // ป้อนข้อมูลสำหรับแทรกหลัง

                scanf("%d",&data);

                InsertAfter(data); // เรียกฟังก์ชันแทรกหลังข้อมูล

                printf("\nAll Data in Linked List AFTER INSERTED\n");

                ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

                break;

    case 'O' : printf("\nDelete Before data : " ); // ป้อนข้อมูลสำหรับลบก่อน

                scanf("%d",&data);

```

โค้ดของโปรแกรม DOUBLY CIRCULAR LINKED LIST พร้อมคำอธิบาย (ต่อ)

```

DeleteBefore(data); // เรียกฟังก์ชันลบก่อนข้อมูล

printf("\nAll Data in Linked List AFTER DELETED\n");

ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

break;

case 'X' : printf("\nDelete ItSelf data : " ); // ป้อนข้อมูลสำหรับลบตัวเอง

scanf("%d",&data);

DeleteSelf(data); // เรียกฟังก์ชันลบตัวเอง

printf("\nAll Data in Linked List ITSELF DELETED\n");

ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

break;

case 'D' : printf("\nDelete After data : " ); // ป้อนข้อมูลสำหรับลบหลัง

scanf("%d",&data);

DeleteAfter(data); // เรียกฟังก์ชันลบหลังข้อมูล

printf("\nAll Data in Linked List AFTER DELETED\n");

ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

break;

} // จบ Switch...case

} // จบ While

return(0);

} // จบฟังก์ชันหลัก (MAIN)

```


หลักการทำงานของโปรแกรม DOUBLY CIRCULAR LINKED LIST

โปรแกรม Doubly Circular Linked List เป็นโครงสร้างข้อมูลแบบลิงค์ลิสต์ที่แต่ละโหนด (Node) มีตัวชี้สองตัวคือไปข้างหน้า (RLink) และย้อนกลับ (LLink) และที่สำคัญคือ Linked List นี้จะเป็นแบบวงกลม (Circular) ซึ่งหมายความว่าโหนดสุดท้ายจะลิงค์กลับไปยังโหนดแรก ทำให้สามารถวนซ้ำได้ไม่มีที่สิ้นสุด และโปรแกรมนี้ประกอบด้วยฟังก์ชันต่าง ๆ ที่มีหน้าที่ในการจัดการกับโหนดใน Linked List เช่น การเพิ่มโหนด การลบโหนด และการแสดงผลข้อมูลของโหนดทั้งหมด โดยประกอบไปด้วย

1. การประกาศไลบรารีของโปรแกรม

```
#include <stdio.h> // ใช้ printf

#include <conio.h> // ใช้ getch

#include <stdlib.h> // ใช้ malloc

#define HeadInfo -999 // กำหนดข้อมูลของโหนดหัว (Head Node)
```

ในส่วนของการประกาศไลบรารีของโปรแกรม มีรายละเอียดดังต่อไปนี้

- #include <stdio.h> : โปรแกรมนี้ใช้ไลบรารีมาตรฐาน stdio.h ซึ่งมีฟังก์ชันสำหรับการรับและแสดงผลข้อมูลในโปรแกรม ตัวอย่างเช่น การใช้ฟังก์ชัน printf เพื่อแสดงข้อมูลต่าง ๆ ออกทางหน้าจอ
- #include <conio.h> : ฟังก์ชัน conio.h เป็นไลบรารีที่มีฟังก์ชันที่เกี่ยวข้องกับการจัดการอินพุตและเอาต์พุตของคอนโซล โดยโปรแกรมนี้ใช้ฟังก์ชัน getch จากไลบรารีนี้ ฟังก์ชัน getch ใช้สำหรับรับอักขระจากผู้ใช้นี้ตัวโดยไม่ต้องรอการกด Enter และอักขระที่รับเข้ามาจะไม่แสดงบนหน้าจอ
- #include <stdlib.h> : ฟังก์ชัน stdlib.h เป็นไลบรารีมาตรฐานที่ใช้สำหรับการจัดการหน่วยความจำ, การแปลงข้อมูล, การสุ่มเลข, และฟังก์ชันยูทิลิตี้อื่น ๆ โปรแกรมนี้ใช้ malloc จาก stdlib.h ซึ่งเป็นฟังก์ชันที่ใช้จัดสรรหน่วยความจำให้กับโหนดใหม่ในลิสต์
- #define HeadData -999 : การประกาศ #define เป็นการประกาศมาโครที่กำหนดค่าคงที่ให้กับชื่อในโปรแกรม ในโปรแกรมนี้ HeadData ถูกกำหนดเป็น -999 เพื่อใช้เป็นข้อมูลพิเศษสำหรับโหนดหัว (Head Node) โดยค่า -999 นี้ถูกใช้เพื่อบ่งบอกว่าโหนดนั้นเป็นโหนดหัว ซึ่งจะช่วยในการตรวจสอบว่าโหนดใดเป็นโหนดหัว และไม่อนุญาตให้ลบโหนดนี้

2. การประกาศโครงสร้างของโหนด

```
struct Node { // ประกาศโครงสร้างของโหนด

    int info;

    struct Node *llink;

    struct Node *rlink;

};
```

ฟังก์ชันนี้ใช้สำหรับสร้างโครงสร้างของโหนด โดยแต่ละโหนดจะประกอบด้วยข้อมูล (info) และพอยน์เตอร์สองตัว (llink และ rlink) ที่ชี้ไปยังโหนดก่อนหน้าและโหนดถัดไปตามลำดับ

3. การประกาศพอยน์เตอร์โหนดและตัวแปรต่าง ๆ

```
struct Node *H, *H1, *p, *q; // ประกาศตัวชี้ (Pointer) สำหรับโหนด

int i,j,k,n,data;

char ch;
```

ในส่วนของการประกาศพอยน์เตอร์โหนดและตัวแปรต่าง ๆ มีรายละเอียดดังต่อไปนี้

- การประกาศพอยน์เตอร์โหนด
 - struct Node *H: พอยน์เตอร์ H ถูกใช้เพื่อเป็นพอยน์เตอร์ไปยังโหนดหัว (Head Node) ในลิสต์
 - struct Node *H1: พอยน์เตอร์ H1 ถูกใช้เพื่อเป็นพอยน์เตอร์ช่วยในการเดินทางผ่านโหนดต่าง ๆ ในลิสต์ เพื่อทำการแทรกหรือลบโหนด
 - struct Node *p: พอยน์เตอร์ p ถูกใช้เพื่อเป็นพอยน์เตอร์ชั่วคราวสำหรับโหนดปัจจุบันที่กำลังถูกสร้างหรือลบ
 - struct Node *q: พอยน์เตอร์ q อาจใช้ในส่วนอื่นของโปรแกรมสำหรับการจัดการโหนด (แต่ในโค้ดที่ให้มา q ยังไม่ถูกใช้งานจริง)
- การประกาศตัวแปรอื่น ๆ
 - int i, j, k, n, data: ตัวแปรประเภท int เหล่านี้ใช้สำหรับเก็บข้อมูลตัวเลขทั่วไปในโปรแกรม:
 - i, j, k: ใช้เป็นตัวนับในลูปต่าง ๆ

- n: ใช้เก็บจำนวนของโหนดที่ต้องการสร้างในลิสต์
- data: ใช้เก็บข้อมูลที่ผู้ใช้ป้อนเข้ามาสําหรับการแทรกหรือลบข้อมูลในลิสต์
- char ch: ตัวแปรประเภท char ใช้เก็บตัวอักษรที่ผู้ใช้กดในเมนูเพื่อเลือกว่าจะทำการแทรก (Insert) ลบ (Delete) หรือออกจากโปรแกรม (Exit)

4. การจองพื้นที่หน่วยความจำด้วยฟังก์ชัน Allocate

```
Node *Allocate() { // จัดสรร 1 โหนดจากพื้นที่เก็บข้อมูล

    struct Node *temp;

    temp=(Node*)malloc(sizeof(Node)); // จัดสรรโหนดตามขนาดที่กำหนด

    return(temp);

}
```

ฟังก์ชันนี้ทำหน้าที่จัดสรรหน่วยความจำสำหรับโหนดใหม่ โดยใช้ฟังก์ชัน malloc จากไลบรารี stdlib.h และส่งคืนพอยน์เตอร์ที่ชี้ไปยังหน่วยความจำนั้น

5. การสร้างโหนดหลายโหนดด้วยฟังก์ชัน CreateNNode

```
void CreateNNode(int n) { // สร้าง N โหนด ใส่ข้อมูลและลิงค์ไปยังโหนดอื่นๆ

    int i,temp;

    H1=H; // เริ่มต้น H1 ที่นี่

    for (i=1;i<=n;i++) { // นับ N โหนด

        p=Allocate(); // จัดสรรโหนดใหม่

        temp=1+rand() % 99; // สุ่มตัวเลขระหว่าง 1..99

        p->info=temp; // ใส่ข้อมูลสุ่มในโหนด

        H1->rlink=p; // ลิงค์จากโหนดแรกไปยังโหนดที่สอง

    }
```

5. การสร้างโหนดหลายโหนดด้วยฟังก์ชัน CreateNNode (ต่อ)

```

    p->llink=H1; // LLink ชี้กลับไปโหนดก่อนหน้า

    H1=p; // ให้ H1 ชี้ไปยังโหนดสุดท้าย

    H1->rlink=H; // ตั้งค่า rlink ของ H1 ให้ชี้ไปยังโหนดหัว

    H->llink=H1; // ตั้งค่า LLink ของ H ให้ชี้ไปยัง H1

}

}

```

ฟังก์ชันนี้ใช้สำหรับสร้างโหนดจำนวน n โหนดใน Doubly Circular Linked List โดยแต่ละโหนดจะเก็บข้อมูลเป็นค่าที่สุ่มขึ้นมาระหว่าง 1 ถึง 99 และทำการลิงก์แต่ละโหนดเข้าด้วยกันเป็นลำดับ โดยลิงก์โหนดสุดท้ายกลับไปยังโหนดหัว (Head Node) เพื่อทำให้เป็นลิงก์แบบวงกลม

6. การแสดงข้อมูลโหนดทั้งหมดในลิสต์ด้วยฟังก์ชัน ShowAllNode

```

void ShowAllNode()
{
    printf("H = %x\n",H); // แสดงที่อยู่ของ Pointer H

    p=H; // กำหนดจุดเริ่มต้นของ Pointer p

    i=1; // กำหนดค่าของเคาน์เตอร์เริ่มต้น

    if (p->rlink != H ) // ถ้ามีโหนดมากกว่า 1 โหนด

    {

        p=p->rlink; // ข้าม Pointer ไปยังโหนดแรก

        while (p != H) // ขณะที่ P ไม่เท่ากับ H

        {

            printf("%d) : %x\t",i,p); // แสดงเคาน์เตอร์และ Pointer

```

6. การแสดงข้อมูลโหนดทั้งหมดในลิสต์ด้วยฟังก์ชัน ShowAllNode (ต่อ)

```
printf("LLINK : %x\t",p->llink); // แสดง LLink

    printf("INFO : %d\t",p->info); // แสดงข้อมูลในโหนด

    printf("RLINK : %x\n",p->rlink); // แสดง RLink

    p=p->rlink; // ข้ามไปยังโหนดถัดไป

    i++; // ข้ามเคาน์เตอร์

} // จบ While

} // จบ if

} // จบฟังก์ชัน
```

ฟังก์ชันนี้ใช้สำหรับแสดงข้อมูลทั้งหมดของโหนดในลิงก์ โดยเริ่มต้นจากโหนดแรกและทำการลิงก์ไปจนถึงโหนดสุดท้าย แล้ววนกลับมาที่โหนดหัวเพื่อทำให้เป็นลิงก์แบบวงกลม ข้อมูลที่แสดงประกอบด้วยลิงก์ของโหนดก่อนหน้า (LLINK), ข้อมูลในโหนด (INFO), และลิงก์ของโหนดถัดไป (RLINK)

7. การเพิ่มข้อมูลใหม่หลังโหนดที่กำหนดด้วยฟังก์ชัน InsertAfter

```
void InsertAfter(int data1)

{

    int temp; // ตัวแปรชั่วคราว

    if (H->rlink == H)

        printf("Linked List have no node!!..\n");

    else

    {

        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว
```

7. การเพิ่มข้อมูลใหม่หลังโหนดที่กำหนดด้วยฟังก์ชัน InsertAfter (ต่อ)

```

{
    if (H1->info == data1) // ถ้าพบข้อมูล
    {
        p=Allocate(); // จัดสรรโหนดจากพื้นที่เก็บข้อมูล
        printf("\nใส่ข้อมูล : "); // ป้อนข้อมูลสำหรับการแทรก
        scanf("%d",&temp); // อ่านข้อมูลจากคีย์บอร์ด
        p->info=temp; // ใส่ข้อมูลชั่วคราวในโหนด
        if (H1->rlink == H) // ถ้า H1 เป็นโหนดสุดท้าย
        {
            p->rlink=H; // ให้ p ชี้ไปที่โหนดหัว
            H->llink=p; // ให้ H ชี้ไปที่โหนดสุดท้าย
        }
        else
        {
            p->rlink=H1->rlink; // เปลี่ยน Pointer สำหรับแทรกโหนด (ไกลไปไกล)
            H1->rlink->llink=p; // LLink(RLink(H1))=p
        }
        p->llink=H1; // LLink(p)=H1
        H1->rlink=p; // RLink(H1)=p
    } // จบ if

    H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป

```

7. การเพิ่มข้อมูลใหม่หลังโหนดที่กำหนดด้วยฟังก์ชัน InsertAfter (ต่อ)

```

    } // จบ while

    } // จบ if

} // จบฟังก์ชัน

```

ฟังก์ชันนี้ใช้สำหรับแทรกโหนดใหม่หลังจากโหนดที่มีข้อมูล (info) ตรงกับ data1 โดยหากพบโหนดที่ตรงกันแล้วจะทำการจัดสรรหน่วยความจำสำหรับโหนดใหม่ รับข้อมูลจากผู้ใช้ และทำการลิงก์โหนดใหม่เข้าระหว่างโหนดที่พบและโหนดถัดไป

8. การเพิ่มข้อมูลใหม่ก่อนโหนดที่กำหนดด้วยฟังก์ชัน InsertBefore

```

void InsertBefore(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have no node!!...\n");

    else
    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว
        {
            if (H1->info == data1) // ถ้าพบข้อมูล
            {
                p=Allocate(); // จัดสรรโหนดจากพื้นที่เก็บข้อมูล

                printf("\nใส่ข้อมูล : "); // ป้อนข้อมูลสำหรับการแทรก
            }
        }
    }
}

```

8. การเพิ่มข้อมูลใหม่ก่อนโหนดที่กำหนดด้วยฟังก์ชัน InsertBefore (ต่อ)

```

scanf("%d",&temp); // อ่านข้อมูลจากคีย์บอร์ด

p->info=temp; // ใส่ข้อมูลชั่วคราวในโหนด

if (H1->llink == H) // โหนดแรก
{
    p->llink=H; // LLink(p) ชี้ไปที่โหนดหัว
    H->rlink=p; // RLink(H) ชี้ไปที่ 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

} // จบ if

H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน

```


8. การเพิ่มข้อมูลใหม่ก่อนโหนดที่กำหนดด้วยฟังก์ชัน InsertBefore (ต่อ)

ฟังก์ชันนี้ใช้สำหรับแทรกโหนดใหม่ก่อนโหนดที่มีข้อมูล (info) ตรงกับ data1 โดยหากพบโหนดที่ตรงกันแล้ว จะทำการจัดสรรหน่วยความจำสำหรับโหนดใหม่ รับข้อมูลจากผู้ใช้ และทำการลิงก์โหนดใหม่เข้าระหว่างโหนดก่อนหน้าและโหนดที่พบ

9. การลบโหนดก่อนหน้าโหนดที่กำหนดด้วยฟังก์ชัน DeleteBefore

```
void DeleteBefore(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!..\n");

    else
    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว
        {
            if (H1->info == data1) // ถ้าพบข้อมูล
            {
                if (H1->llink==H) // ถ้าไม่มีโหนดก่อนหน้านี้

                    printf ("No more node before here,Can't delete it!!!\n");

                else
                {
                    p=H1->llink; // ทำเครื่องหมายที่โหนดสำหรับการลบ

                    H1->llink=p->llink; // ตั้งค่า Link ของ H1 ให้ชี้ไปที่ที่อยู่เดียวกับ p
                }
            }
        }
    }
}
```

9. การลบโหนดก่อนหน้าโหนดที่กำหนดด้วยฟังก์ชัน DeleteBefore (ต่อ)

```

        p->llink->rlink=H1;

        free(p); // ปลอยโหนดกลับไปยังพื้นที่เก็บข้อมูล

    } // จบ if2

} // จบ if1

H1=H1->rlink; // ซ้ำม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน

```

ฟังก์ชันนี้ใช้สำหรับลบโหนดที่อยู่ก่อนหน้าโหนดที่มีข้อมูล (info) ตรงกับ data1 โดยหากพบโหนดที่ตรงกันแล้วจะทำการลบโหนดที่อยู่ก่อนหน้าโหนดนั้น และทำการลิงก์โหนดก่อนหน้าของโหนดที่ถูกลบกับโหนดที่ลบ

10. การลบโหนดที่กำหนดด้วยฟังก์ชัน DeleteSelf

```

void DeleteSelf(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!..\n");

    else

    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว

        {

```

10. การลบโหนดที่กำหนดด้วยฟังก์ชัน DeleteSelf (ต่อ)

```

    if (H1->info == data1) // ถ้าพบข้อมูล

    {

        p=H1; // ทำเครื่องหมายที่โหนดสำหรับการลบ

        if(p->llink==H && p->rlink==H) // ถ้ามีเพียงหนึ่งโหนด

        {

            H->llink=H; // ตั้งค่า Pointer ของ HEAD ให้ชี้ไปที่ตัวเอง

            H->rlink=H;

        }

        else

        {

            p->llink->rlink=p->rlink;

            p->rlink->llink=p->llink;

        }

        free(p); // ปลดปล่อยโหนดกลับไปยังพื้นที่เก็บข้อมูล

    } // จบ if1

    H1=H1->rlink; // ข้าม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน

```

10. การลบโหนดที่กำหนดด้วยฟังก์ชัน DeleteSelf (ต่อ)

ฟังก์ชันนี้ใช้สำหรับลบโหนดที่มีข้อมูล (info) ตรงกับ data1 โดยหากพบโหนดที่ตรงกันแล้วจะทำการลบโหนดนั้น และทำการลิงก์โหนดก่อนหน้าและโหนดถัดไปเข้าด้วยกัน

11. การลบโหนดหลังจากโหนดที่กำหนดด้วยฟังก์ชัน DeleteAfter

```
void DeleteAfter(int data1)
{
    int temp; // ตัวแปรชั่วคราว

    if (H->rlink==H)

        printf("Linked List have NO NODE!!..\n");

    else
    {
        H1=H->rlink; // ให้ H1 ชี้ไปที่โหนดแรก

        while (H1->info != HeadInfo) // ค้นหาข้อมูลในขณะที่ H1 วนกลับมาที่โหนดหัว
        {
            if (H1->info == data1) // ถ้าพบข้อมูล
            {
                if (H1->rlink==H) // ถ้าไม่มีโหนดถัดไป

                    printf ("No more node from here,Can't delete it!!!\n");

                else
                {
                    p=H1->rlink; // ทำเครื่องหมายที่โหนดสำหรับการลบ

                    H1->rlink=p->rlink; // ตั้งค่า Link ของ H1 ให้ชี้ไปที่ที่อยู่เดียวกับ p
                }
            }
            H1=H1->rlink;
        }
    }
}
```

11. การลบโหนดหลังจากโหนดที่กำหนดด้วยฟังก์ชัน DeleteAfter (ต่อ)

```
p->rlink->llink=H1;

    free(p); // ปลอยโหนดกลับไปยังพื้นที่เก็บข้อมูล

} // จบ if2

} // จบ if1

H1=H1->rlink; // ซ้ำม H1 ไปยังโหนดถัดไป

} // จบ while

} // จบ if

} // จบฟังก์ชัน
```

ฟังก์ชันนี้ใช้สำหรับลบโหนดที่อยู่หลังจากโหนดที่มีข้อมูล (info) ตรงกับ data1 โดยหากพบโหนดที่ตรงกันแล้ว จะทำการลบโหนดที่อยู่ถัดไปจากโหนดนั้น และทำการลิงก์โหนดที่พบกับโหนดถัดไปของโหนดที่ถูกลบ

12. ฟังก์ชันหลัก (main)

```
int main() // ฟังก์ชันหลัก (MAIN)

{

    p=Allocate(); // สร้างโหนดหัว (HEAD NODE)

    p->info=HeadInfo; // ข้อมูลพิเศษสำหรับโหนดหัว

    p->llink=p; p->rlink=p; // ให้ทั้งสอง Link ชี้กลับมาที่ตัวเอง

    H=p; // ให้ H ชี้ไปที่โหนดหัว

    n=10; // กำหนดจำนวนโหนด

    CreateNNode(n); // เรียกฟังก์ชันสร้าง N โหนด

    printf("PROGRAM DOUBLY CIRCULAR LINKED LIST \n");

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

12. ฟังก์ชันหลัก (main) (ต่อ)

```

printf("All Data in Linked List \n");

ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

ch=' ';

while (ch != 'E')

{

    printf("MENU>> [B:InsertBefore A:InsertAfter\n");

    printf(" O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]\n");

    ch=getch();

    switch (ch)

    {

        case 'B' : printf("\nInsert Before data : " ); // ป้อนข้อมูลสำหรับแทรกก่อน

            scanf("%d",&data);

            InsertBefore(data); // เรียกฟังก์ชันแทรกก่อนข้อมูล

            printf("\nAll Data in Linked List AFTER INSERTED\n");

            ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

            break;

        case 'A' : printf("\nInsert After data : " ); // ป้อนข้อมูลสำหรับแทรกหลัง

            scanf("%d",&data);

            InsertAfter(data); // เรียกฟังก์ชันแทรกหลังข้อมูล

            printf("\nAll Data in Linked List AFTER INSERTED\n");

            ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด
    }
}

```

12. ฟังก์ชันหลัก (main) (ต่อ)

```

        break;

    case 'O' : printf("\nDelete Before data : " ); // ป้อนข้อมูลสำหรับลบก่อน

        scanf("%d",&data);

        DeleteBefore(data); // เรียกฟังก์ชันลบก่อนข้อมูล

        printf("\nAll Data in Linked List AFTER DELETED\n");

        ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

        break;

    case 'X' : printf("\nDelete ItSelf data : " ); // ป้อนข้อมูลสำหรับลบตัวเอง

        scanf("%d",&data);

        DeleteSelf(data); // เรียกฟังก์ชันลบตัวเอง

        printf("\nAll Data in Linked List ITSELF DELETED\n");

        ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

        break;

    case 'D' : printf("\nDelete After data : " ); // ป้อนข้อมูลสำหรับลบหลัง

        scanf("%d",&data);

        DeleteAfter(data); // เรียกฟังก์ชันลบหลังข้อมูล

        printf("\nAll Data in Linked List AFTER DELETED\n");

        ShowAllNode(); // เรียกฟังก์ชันแสดงโหนดทั้งหมด

        break;

} // จบ Switch...case

} // จบ While

```

12. ฟังก์ชันหลัก (main) (ต่อ)

```
return(0);  
} // จบฟังก์ชันหลัก (MAIN)
```

ฟังก์ชัน main() เป็นจุดเริ่มต้นของโปรแกรม โดยโปรแกรมจะสร้างโหนดหัว (Head Node) และสร้างโหนดจำนวน 10 โหนด จากนั้นจะแสดงเมนูให้ผู้ใช้สามารถเพิ่มหรือลบโหนดในลิงก์ได้ตามที่ต้องการโดยการกดปุ่มที่กำหนด (เช่น B, A, O, X, D) และจะแสดงสถานะของลิงก์หลังจากการดำเนินการแต่ละครั้ง

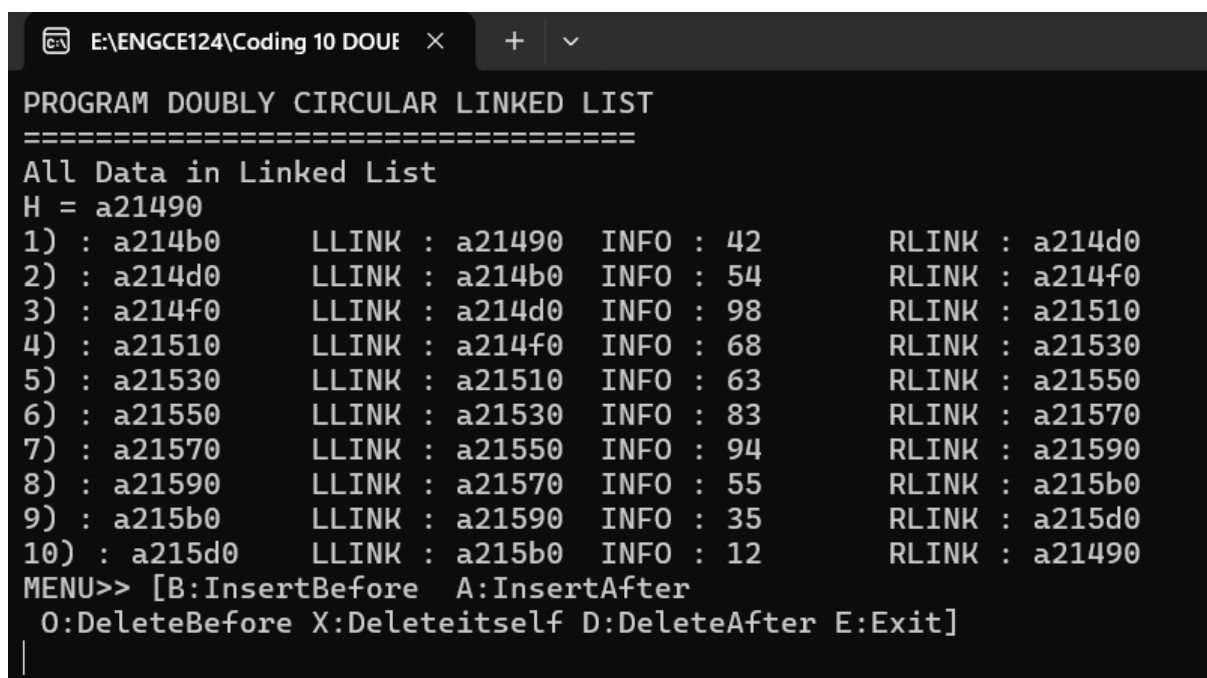
ผลลัพธ์การใช้งานโปรแกรม DOUBLY CIRCULAR LINKED LIST

โครงสร้างข้อมูลแบบ Doubly Circular Linked List เป็นโครงสร้างที่ได้รับความนิยมในการจัดเก็บและจัดการข้อมูล เนื่องจากมีความยืดหยุ่นในการเพิ่มและลบข้อมูล และสามารถเข้าถึงข้อมูลได้จากทั้งสองทิศทาง (ไปข้างหน้าและย้อนกลับ) ในบทความนี้เราจะสำรวจการทำงานของโปรแกรมตัวอย่างที่ใช้โครงสร้างข้อมูลนี้ พร้อมอธิบายผลลัพธ์ที่เกิดขึ้นเมื่อทำการใช้งานโปรแกรม

1. การสร้างลิงก์ลิสต์เริ่มต้น

เมื่อโปรแกรมเริ่มต้นทำงาน โหนดหัว (Head Node) จะถูกสร้างขึ้นโดยมีค่าข้อมูลเป็น -999 ซึ่งเป็นค่าที่ถูกกำหนดไว้ใน #define HeadInfo -999 โหนดหัวนี้จะมีลิงก์ทั้งสอง (llink และ rlink) ชี้กลับมาที่ตัวมันเอง เนื่องจากเป็นโหนดเพียงโหนดเดียวในลิสต์ ณ ขณะนี้

จากนั้นโปรแกรมจะสร้างโหนดเพิ่มเติมอีก 10 โหนด โดยใช้ฟังก์ชัน CreateNNode(int n) ข้อมูลในแต่ละโหนดจะถูกสุ่มขึ้นมาในช่วง 1 ถึง 99 และลิงก์จะถูกจัดให้เชื่อมต่อกันเป็นลำดับโดยที่โหนดสุดท้ายจะเชื่อมกลับไปยังโหนดหัว ทำให้ลิงก์ลิสต์นี้เป็นลิสต์แบบวงกลม



```

E:\ENGCE124\Coding 10 DOUBLY CIRCULAR LINKED LIST
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = a21490
1) : a214b0      LLINK : a21490  INFO : 42      RLINK : a214d0
2) : a214d0      LLINK : a214b0  INFO : 54      RLINK : a214f0
3) : a214f0      LLINK : a214d0  INFO : 98      RLINK : a21510
4) : a21510      LLINK : a214f0  INFO : 68      RLINK : a21530
5) : a21530      LLINK : a21510  INFO : 63      RLINK : a21550
6) : a21550      LLINK : a21530  INFO : 83      RLINK : a21570
7) : a21570      LLINK : a21550  INFO : 94      RLINK : a21590
8) : a21590      LLINK : a21570  INFO : 55      RLINK : a215b0
9) : a215b0      LLINK : a21590  INFO : 35      RLINK : a215d0
10) : a215d0     LLINK : a215b0  INFO : 12      RLINK : a21490
MENU>> [B:InsertBefore A:InsertAfter
         O>DeleteBefore X>Deleteitself D>DeleteAfter E:Exit]
  
```

ในผลลัพธ์นี้ เห็นได้ว่าการแสดงที่อยู่หน่วยความจำของโหนดแต่ละโหนด (p) และค่าของลิงก์ไปยังโหนดก่อนหน้า (LLINK), ข้อมูล (INFO), และลิงก์ไปยังโหนดถัดไป (RLINK) ตามลำดับ โหนดหัว (H) จะชี้ไปยังโหนดแรกที่สร้างขึ้น

2. การแสดงเมนูและเลือกคำสั่ง

หลังจากที่แสดงลิสต์แล้ว โปรแกรมจะนำผู้ใช้เข้าสู่เมนูที่สามารถเลือกคำสั่งต่าง ๆ เพื่อจัดการกับลิงก์ลิสต์ได้ โดยมีคำสั่งต่าง ๆ ได้แก่

- B : เพิ่มโหนดใหม่ก่อนโหนดที่ระบุ
- A : เพิ่มโหนดใหม่หลังโหนดที่ระบุ
- : ลบโหนดก่อนโหนดที่ระบุ
- X : ลบโหนดที่ระบุเอง
- D : ลบโหนดหลังโหนดที่ระบุ
- E : ออกจากโปรแกรม

3. การเพิ่มโหนดใหม่

เมื่อผู้ใช้เลือกคำสั่ง A เพื่อเพิ่มโหนดใหม่หลังโหนดที่ระบุ โปรแกรมจะขอให้ผู้ใช้ป้อนค่าข้อมูลของโหนดที่ต้องการเพิ่มหลัง จากนั้นผู้ใช้ต้องป้อนค่าข้อมูลสำหรับโหนดใหม่

```

E:\ENGCE124\Coding 10 DOUE  X  +  v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = ce1490
1) : ce14b0    LLINK : ce1490  INFO : 42    RLINK : ce14d0
2) : ce14d0    LLINK : ce14b0  INFO : 54    RLINK : ce14f0
3) : ce14f0    LLINK : ce14d0  INFO : 98    RLINK : ce1510
4) : ce1510    LLINK : ce14f0  INFO : 68    RLINK : ce1530
5) : ce1530    LLINK : ce1510  INFO : 63    RLINK : ce1550
6) : ce1550    LLINK : ce1530  INFO : 83    RLINK : ce1570
7) : ce1570    LLINK : ce1550  INFO : 94    RLINK : ce1590
8) : ce1590    LLINK : ce1570  INFO : 55    RLINK : ce15b0
9) : ce15b0    LLINK : ce1590  INFO : 35    RLINK : ce15d0
10) : ce15d0   LLINK : ce15b0  INFO : 12    RLINK : ce1490
MENU>> [B:InsertBefore A:InsertAfter
         O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

Insert After data : 12

Insert data : 100

All Data in Linked List AFTER INSERTED
H = ce1490
1) : ce14b0    LLINK : ce1490  INFO : 42    RLINK : ce14d0
2) : ce14d0    LLINK : ce14b0  INFO : 54    RLINK : ce14f0
3) : ce14f0    LLINK : ce14d0  INFO : 98    RLINK : ce1510
4) : ce1510    LLINK : ce14f0  INFO : 68    RLINK : ce1530
5) : ce1530    LLINK : ce1510  INFO : 63    RLINK : ce1550
6) : ce1550    LLINK : ce1530  INFO : 83    RLINK : ce1570
7) : ce1570    LLINK : ce1550  INFO : 94    RLINK : ce1590
8) : ce1590    LLINK : ce1570  INFO : 55    RLINK : ce15b0
9) : ce15b0    LLINK : ce1590  INFO : 35    RLINK : ce15d0
10) : ce15d0   LLINK : ce15b0  INFO : 12    RLINK : ce15f0
11) : ce15f0   LLINK : ce15d0  INFO : 100   RLINK : ce1490
MENU>> [B:InsertBefore A:InsertAfter
         O:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

```

ในผลลัพธ์นี้จะเห็นว่าโหนดใหม่ที่มีข้อมูล 100 ถูกเพิ่มเข้าหลังโหนดที่มีข้อมูล 12 ลิงก์ (rlink และ llink) ของโหนดทั้งสาม (โหนด 12, 100 และโหนดถัดไป) จะถูกปรับให้เชื่อมต่อกันอย่างถูกต้อง

3. การเพิ่มโหนดใหม่ (ต่อ)

แต่ถ้าหากผู้ใช้เลือกคำสั่ง B เพื่อเพิ่มโหนดใหม่ก่อนโหนดที่ระบุ โปรแกรมจะขอให้ผู้ใช้ป้อนค่าข้อมูลของโหนดที่ต้องการเพิ่มหลัง จากนั้นผู้ใช้ต้องป้อนค่าข้อมูลสำหรับโหนดใหม่

```

E:\ENGCE124\Coding 10 DOUE  X  +  v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = 7e1490
1) : 7e14b0      LLINK : 7e1490  INFO : 42      RLINK : 7e14d0
2) : 7e14d0      LLINK : 7e14b0  INFO : 54      RLINK : 7e14f0
3) : 7e14f0      LLINK : 7e14d0  INFO : 98      RLINK : 7e1510
4) : 7e1510      LLINK : 7e14f0  INFO : 68      RLINK : 7e1530
5) : 7e1530      LLINK : 7e1510  INFO : 63      RLINK : 7e1550
6) : 7e1550      LLINK : 7e1530  INFO : 83      RLINK : 7e1570
7) : 7e1570      LLINK : 7e1550  INFO : 94      RLINK : 7e1590
8) : 7e1590      LLINK : 7e1570  INFO : 55      RLINK : 7e15b0
9) : 7e15b0      LLINK : 7e1590  INFO : 35      RLINK : 7e15d0
10) : 7e15d0     LLINK : 7e15b0  INFO : 12      RLINK : 7e1490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

Insert Before data : 12

Insert data : 100

All Data in Linked List AFTER INSERTED
H = 7e1490
1) : 7e14b0      LLINK : 7e1490  INFO : 42      RLINK : 7e14d0
2) : 7e14d0      LLINK : 7e14b0  INFO : 54      RLINK : 7e14f0
3) : 7e14f0      LLINK : 7e14d0  INFO : 98      RLINK : 7e1510
4) : 7e1510      LLINK : 7e14f0  INFO : 68      RLINK : 7e1530
5) : 7e1530      LLINK : 7e1510  INFO : 63      RLINK : 7e1550
6) : 7e1550      LLINK : 7e1530  INFO : 83      RLINK : 7e1570
7) : 7e1570      LLINK : 7e1550  INFO : 94      RLINK : 7e1590
8) : 7e1590      LLINK : 7e1570  INFO : 55      RLINK : 7e15b0
9) : 7e15b0      LLINK : 7e1590  INFO : 35      RLINK : 7e15f0
10) : 7e15f0     LLINK : 7e15b0  INFO : 100     RLINK : 7e15d0
11) : 7e15d0     LLINK : 7e15f0  INFO : 12      RLINK : 7e1490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

```

ในผลลัพธ์นี้จะเห็นว่าโหนดใหม่ที่มีข้อมูล 100 ถูกเพิ่มเข้าก่อนโหนดที่มีข้อมูล 12 ลิงก์ (rlink และ llink) ของโหนดทั้งสาม (โหนด 100, 12 และโหนดถัดไป) จะถูกปรับให้เชื่อมต่อกันอย่างถูกต้อง

4. การลบโหนด

เมื่อผู้ใช้เลือกคำสั่ง X เพื่อลบโหนดที่ระบุเอง โปรแกรมจะขอให้ผู้ใช้ป้อนค่าข้อมูลของโหนดที่ต้องการลบ จากนั้นโปรแกรมจะค้นหาโหนดนั้นในลิสต์และทำการลบออก

4. การลบโหนด (ต่อ)

```

E:\ENGCE124\Coding 10 DOUE x + v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = 651490
1) : 6514b0      LLINK : 651490  INFO : 42      RLINK : 6514d0
2) : 6514d0      LLINK : 6514b0  INFO : 54      RLINK : 6514f0
3) : 6514f0      LLINK : 6514d0  INFO : 98      RLINK : 651510
4) : 651510      LLINK : 6514f0  INFO : 68      RLINK : 651530
5) : 651530      LLINK : 651510  INFO : 63      RLINK : 651550
6) : 651550      LLINK : 651530  INFO : 83      RLINK : 651570
7) : 651570      LLINK : 651550  INFO : 94      RLINK : 651590
8) : 651590      LLINK : 651570  INFO : 55      RLINK : 6515b0
9) : 6515b0      LLINK : 651590  INFO : 35      RLINK : 6515d0
10) : 6515d0     LLINK : 6515b0  INFO : 12      RLINK : 651490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

Delete ItSelf data : 12

All Data in Linked List ITSELF DELETED
H = 651490
1) : 6514b0      LLINK : 651490  INFO : 42      RLINK : 6514d0
2) : 6514d0      LLINK : 6514b0  INFO : 54      RLINK : 6514f0
3) : 6514f0      LLINK : 6514d0  INFO : 98      RLINK : 651510
4) : 651510      LLINK : 6514f0  INFO : 68      RLINK : 651530
5) : 651530      LLINK : 651510  INFO : 63      RLINK : 651550
6) : 651550      LLINK : 651530  INFO : 83      RLINK : 651570
7) : 651570      LLINK : 651550  INFO : 94      RLINK : 651590
8) : 651590      LLINK : 651570  INFO : 55      RLINK : 6515b0
9) : 6515b0      LLINK : 651590  INFO : 35      RLINK : 651490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

```

ผลลัพธ์จะแสดงว่าโหนดที่มีข้อมูล 12 ถูกลบออกไปจากลิสต์แล้ว ลิงก์ของโหนดก่อนหน้า (35) และโหนดถัดไป (H) จะถูกเชื่อมต่อเข้าด้วยกันอย่างถูกต้อง

แต่ถ้าหากผู้ใช้เลือกคำสั่ง O เพื่อลบโหนดก่อนหน้าโหนดที่กำหนด โปรแกรมจะขอให้ผู้ใช้ป้อนค่าข้อมูลของโหนดก่อนหน้าที่จะต้องการลบ จากนั้นโปรแกรมจะค้นหาโหนดนั้นในลิสต์และทำการลบออก

```

E:\ENGCE124\Coding 10 DOUE x + v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = ad1490
1) : ad14b0      LLINK : ad1490  INFO : 42      RLINK : ad14d0
2) : ad14d0      LLINK : ad14b0  INFO : 54      RLINK : ad14f0
3) : ad14f0      LLINK : ad14d0  INFO : 98      RLINK : ad1510
4) : ad1510      LLINK : ad14f0  INFO : 68      RLINK : ad1530
5) : ad1530      LLINK : ad1510  INFO : 63      RLINK : ad1550
6) : ad1550      LLINK : ad1530  INFO : 83      RLINK : ad1570
7) : ad1570      LLINK : ad1550  INFO : 94      RLINK : ad1590
8) : ad1590      LLINK : ad1570  INFO : 55      RLINK : ad15b0
9) : ad15b0      LLINK : ad1590  INFO : 35      RLINK : ad15d0
10) : ad15d0     LLINK : ad15b0  INFO : 12      RLINK : ad1490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

Delete Before data : 54

All Data in Linked List AFTER DELETED
H = ad1490
1) : ad14d0      LLINK : ad1490  INFO : 54      RLINK : ad14f0
2) : ad14f0      LLINK : ad14d0  INFO : 98      RLINK : ad1510
3) : ad1510      LLINK : ad14f0  INFO : 68      RLINK : ad1530
4) : ad1530      LLINK : ad1510  INFO : 63      RLINK : ad1550
5) : ad1550      LLINK : ad1530  INFO : 83      RLINK : ad1570
6) : ad1570      LLINK : ad1550  INFO : 94      RLINK : ad1590
7) : ad1590      LLINK : ad1570  INFO : 55      RLINK : ad15b0
8) : ad15b0      LLINK : ad1590  INFO : 35      RLINK : ad15d0
9) : ad15d0      LLINK : ad15b0  INFO : 12      RLINK : ad1490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

```

4. การลบโหนด (ต่อ)

ผลลัพธ์จะแสดงว่าโหนดที่มีข้อมูล 42 ถูกลบออกไปจากลิสต์แล้ว ลิงก์ของโหนดก่อนหน้า (H) และโหนดถัดไป (54) จะถูกเชื่อมต่อเข้าด้วยกันอย่างถูกต้อง

แต่ถ้าหากผู้ใช้เลือกคำสั่ง D เพื่อลบโหนดหลังจากโหนดที่กำหนด โปรแกรมจะขอให้ผู้ใช้ป้อนค่าข้อมูลของโหนดหลังจากโหนดที่กำหนด ที่ต้องการลบ จากนั้นโปรแกรมจะค้นหาโหนดนั้นในลิสต์และทำการลบออก

```

E:\ENGCE124\Coding 10 DOUf x + v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = b91490
1) : b914b0 LLINK : b91490 INFO : 42 RLINK : b914d0
2) : b914d0 LLINK : b914b0 INFO : 54 RLINK : b914f0
3) : b914f0 LLINK : b914d0 INFO : 98 RLINK : b91510
4) : b91510 LLINK : b914f0 INFO : 68 RLINK : b91530
5) : b91530 LLINK : b91510 INFO : 63 RLINK : b91550
6) : b91550 LLINK : b91530 INFO : 83 RLINK : b91570
7) : b91570 LLINK : b91550 INFO : 94 RLINK : b91590
8) : b91590 LLINK : b91570 INFO : 55 RLINK : b915b0
9) : b915b0 LLINK : b91590 INFO : 35 RLINK : b915d0
10) : b915d0 LLINK : b915b0 INFO : 12 RLINK : b91490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

Delete After data : 42

All Data in Linked List AFTER DELETED
H = b91490
1) : b914b0 LLINK : b91490 INFO : 42 RLINK : b914f0
2) : b914f0 LLINK : b914b0 INFO : 98 RLINK : b91510
3) : b91510 LLINK : b914f0 INFO : 68 RLINK : b91530
4) : b91530 LLINK : b91510 INFO : 63 RLINK : b91550
5) : b91550 LLINK : b91530 INFO : 83 RLINK : b91570
6) : b91570 LLINK : b91550 INFO : 94 RLINK : b91590
7) : b91590 LLINK : b91570 INFO : 55 RLINK : b915b0
8) : b915b0 LLINK : b91590 INFO : 35 RLINK : b915d0
9) : b915d0 LLINK : b915b0 INFO : 12 RLINK : b91490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

```

ผลลัพธ์จะแสดงว่าโหนดที่มีข้อมูล 54 ถูกลบออกไปจากลิสต์แล้ว ลิงก์ของโหนดก่อนหน้า (42) และโหนดถัดไป (98) จะถูกเชื่อมต่อเข้าด้วยกันอย่างถูกต้อง

6. การออกจากโปรแกรม

เมื่อผู้ใช้เลือกคำสั่ง E โปรแกรมจะสิ้นสุดการทำงานและออกจากโปรแกรม

```

E:\ENGCE124\Coding 10 DOUf x + v
PROGRAM DOUBLY CIRCULAR LINKED LIST
=====
All Data in Linked List
H = 6f1490
1) : 6f14b0 LLINK : 6f1490 INFO : 42 RLINK : 6f14d0
2) : 6f14d0 LLINK : 6f14b0 INFO : 54 RLINK : 6f14f0
3) : 6f14f0 LLINK : 6f14d0 INFO : 98 RLINK : 6f1510
4) : 6f1510 LLINK : 6f14f0 INFO : 68 RLINK : 6f1530
5) : 6f1530 LLINK : 6f1510 INFO : 63 RLINK : 6f1550
6) : 6f1550 LLINK : 6f1530 INFO : 83 RLINK : 6f1570
7) : 6f1570 LLINK : 6f1550 INFO : 94 RLINK : 6f1590
8) : 6f1590 LLINK : 6f1570 INFO : 55 RLINK : 6f15b0
9) : 6f15b0 LLINK : 6f1590 INFO : 35 RLINK : 6f15d0
10) : 6f15d0 LLINK : 6f15b0 INFO : 12 RLINK : 6f1490
MENU>> [B:InsertBefore A:InsertAfter
0:DeleteBefore X:Deleteitself D:DeleteAfter E:Exit]

-----
Process exited after 3.841 seconds with return value 0
Press any key to continue . . .

```

บรรณานุกรม

ChatGPT. (-). Detailed of a Doubly Circular Linked List Program in C. สืบค้น 28 สิงหาคม 2567,
จาก <https://chatgpt.com/>