

## WEEK 2 - Doubly linked list

```
#include <stdio.h>

#include <stdlib.h>

struct node
{
    struct node *Prev;
    int Element;
    struct node *Next;
};

typedef struct node Node;

int IsEmpty(Node *List);
int IsLast(Node *Position);
Node *Find(Node *List, int x);
void InsertBeg(Node *List, int e);
void InsertLast(Node *List, int e);
void InsertMid(Node *List, int p, int e);
void DeleteBeg(Node *List);
void DeleteEnd(Node *List);
void DeleteMid(Node *List, int e);
void Traverse(Node *List);

int main()
{
    Node *List = malloc(sizeof(Node));

    List->Prev = NULL;

    List->Next = NULL;

    Node *Position;

    int ch, e, p;

    printf("1.Insert Beg \n2.Insert Middle \n3.Insert End");

    printf("\n4.Delete Beg \n5.Delete Middle \n6.Delete End");

    printf("\n7.Find \n8.Traverse \n9.Exit\n");

    do
```

```
{  
    printf("Enter your choice : ");  
    scanf("%d", &ch);  
    switch(ch)  
    {  
        case 1:  
            printf("Enter the element : ");  
            scanf("%d", &e);  
            InsertBeg(List, e);  
            break;  
        case 2:  
            printf("Enter the position element : ");  
            scanf("%d", &p);  
            printf("Enter the element : ");  
            scanf("%d", &e);  
            InsertMid(List, p, e);  
            break;  
        case 3:  
            printf("Enter the element : ");  
            scanf("%d", &e);  
            InsertLast(List, e);  
            break;  
        case 4:  
            DeleteBeg(List);  
            break;  
        case 5:  
            printf("Enter the element : ");  
            scanf("%d", &e);  
            DeleteMid(List, e);  
            break;  
        case 6:
```

```

DeleteEnd(List);

break;

case 7:

printf("Enter the element : ");

scanf("%d", &e);

Position = Find(List, e);

if(Position != NULL)

printf("Element found...\n");

else

printf("Element not found...\n");

break;

case 8:

Traverse(List);

break;

}

} while(ch <= 8);

return 0;

}

int IsEmpty(Node *List)

{

if(List->Next == NULL)

return 1;

else

return 0;

}

int IsLast(Node *Position)

{

if(Position->Next == NULL)

return 1;

else

return 0;

```

```

}

Node *Find(Node *List, int x)
{
Node *Position;

Position = List->Next;

while(Position != NULL && Position->Element != x)

Position = Position->Next;

return Position;
}

void InsertBeg(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));

NewNode->Element = e;

if(IsEmpty(List))

NewNode->Next = NULL;

else

{
NewNode->Next = List->Next;

NewNode->Next->Prev = NewNode;

}

NewNode->Prev = List;

List->Next = NewNode;

}

void InsertLast(Node *List, int e)
{
Node *NewNode = malloc(sizeof(Node));

Node *Position;

NewNode->Element = e;

NewNode->Next = NULL;

if(IsEmpty(List))

{

```

```

    NewNode->Prev = List;

    List->Next = NewNode;
}

else
{
    Position = List;

    while(Position->Next != NULL)

        Position = Position->Next;

    Position->Next = NewNode;

    NewNode->Prev = Position;
}

}

void InsertMid(Node *List, int p, int e)
{
    Node *NewNode = malloc(sizeof(Node));

    Node *Position;

    Position = Find(List, p);

    NewNode->Element = e;

    NewNode->Next = Position->Next;

    Position->Next->Prev = NewNode;

    Position->Next = NewNode;

    NewNode->Prev = Position;
}

void DeleteBeg(Node *List)
{
    if(!IsEmpty(List))
    {
        Node *TempNode;

        TempNode = List->Next;

        List->Next = TempNode->Next;

        if(List->Next != NULL)

```

```

TempNode->Next->Prev = List;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...\n");
}
void DeleteEnd(Node *List)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
Position = List;
while(Position->Next != NULL)
Position = Position->Next;
TempNode = Position;
Position->Prev->Next = NULL;
printf("The deleted item is %d\n", TempNode->Element);
free(TempNode);
}
else
printf("List is empty...\n");
}
void DeleteMid(Node *List, int e)
{
if(!IsEmpty(List))
{
Node *Position;
Node *TempNode;
Position = Find(List, e);

```

```

if(!IsLast(Position))
{
    TempNode = Position;
    Position->Prev->Next = Position->Next;
    Position->Next->Prev = Position->Prev;
    printf("The deleted item is %d\n", TempNode->Element);
    free(TempNode);
}
}
else
    printf("List is empty...!\n");
}

void Traverse(Node *List)
{
    if(!IsEmpty(List))
    {
        Node *Position;
        Position = List;
        while(Position->Next != NULL)
        {
            Position = Position->Next;
            printf("%d\t", Position->Element);
        }
        printf("\n");
    }
    else
        printf("List is empty...!\n");
}

```

## Output

Enter your choice :

1.Insert Beg

2.Insert Middle

3.Insert End

4.Delete Beg

5.Delete Middle

6.Delete End

7.Find

8.Traverse

9.Exit

Enter your choice : 1

Enter the element : 40

Enter your choice : 1

Enter the element : 30

Enter your choice : 1

Enter the element : 20

Enter your choice : 1

Enter the element : 10

Enter your choice : 8

10 20 30 40

Enter your choice : 7

Enter the element : 30

Element found...!

Enter your choice : 1

Enter the element : 5

Enter your choice : 8

5 10 20 30 40

Enter your choice : 3

Enter the element : 45

Enter your choice : 8

5 10 20 30 40 45

Enter your choice : 2

Enter the position element : 20



Enter the element : 25

Enter your choice : 8

5 10 20 25 30 40 45

Enter your choice : 4

The deleted item is 5

Enter your choice : 8

10 20 25 30 40 45

Enter your choice : 6

The deleted item is 45

Enter your choice : 8

10 20 25 30 40

Enter your choice : 5

Enter the element : 30

The deleted item is 30

Enter your choice : 8

10 20 25 40

Enter your choice : 9