```c
#include <stdio.h>
#include <malloc.h>

////////////////////////////////////////////////////////////////
//  STRUCTURE DECLARATION
////////////////////////////////////////////////////////////////

struct List
{
    struct List *pPrev;
    int iData;
    struct List *pNext;
};

////////////////////////////////////////////////////////////////
//  FUNCTION PROTOTYPES
////////////////////////////////////////////////////////////////

//
//  Insertion
//
void InsertLast(struct List **ppHead, struct List **ppTail, int iNo);
void InsertFirst(struct List **ppHead, struct List **ppTail, int iNo);
void InsertAtPosition(struct List **ppHead, struct List **ppTail, int iNo, int iPos);

//
//  Deletion
//
int DeleteLast(struct List **ppHead, struct List **ppTail);
int DeleteFirst(struct List **ppHead, struct List **ppTail);
void DeleteAllNodes(struct List **ppHead, struct List **ppTail);
int DeleteAtPosition(struct List **ppHead, struct List **ppTail, int iPos);

//
//  Searching
//
int SearchFirstOccurance(struct List *pHead, struct List *pTail, int iKey);
int SearchLastOccurance(struct List *pHead, struct List *pTail, int iKey);
int SearchAllOccurances(struct List *pHead, struct List *pTail, int iKey);

//
//  Display & Counting
//
void Display(struct List *pHead, struct List *pTail);
void ReverseDisplay(struct List *pHead, struct List *pTail);
int CountNode(struct List *pHead, struct List *pTail);

////////////////////////////////////////////////////////////////
//  FUNCTION DEFINITIONS
////////////////////////////////////////////////////////////////

int main()
{
    int iNo;
    int iPos;
    int iChoice;

    struct List *pFirst = NULL;
    struct List *pLast = NULL;

    while(1)
    {
        printf("\n1.Insert\n2.Delete\n3.Search\n4.Count\n5.Reverse Display\n6.Exit\n");
        printf("Enter your choice:\t");
        scanf("%d", &iChoice);

        switch(iChoice)
```

```
{
case 1:
    while(1)
    {
        printf("\n1.InsertFirst\n2.InsertLast\n3.InsertAtPosition\n4.Back\n");
        printf("Enter your choice again:\t");
        scanf("%d", &iChoice);

        if(iChoice == 4)
            break;

        if(iChoice <= 0 || iChoice > 3)
        {
            printf("Enter valid choice\n");
            continue;
        }

        printf("Enter data to be insert:\t");
        scanf("%d", &iNo);

        switch(iChoice)
        {
        case 1:
            InsertFirst(&pFirst, &pLast, iNo);
            break;
        case 2:
            InsertLast(&pFirst, &pLast, iNo);
            break;
        case 3:
            printf("Enter position:\t");
            scanf("%d", &iPos);
            InsertAtPosition(&pFirst, &pLast, iNo, iPos);
        }

        Display(pFirst, pLast);
    }

    break;

case 2:
    if(NULL == pFirst)
    {
        printf("Linked List Empty, Deletion impossible.\n");
        break;
    }
    while(1)
    {
        printf("\n1.DeleteFirst\n2.DeleteLast\n3.DeleteAtPosition\n4.Back\n");
        printf("Enter your choice again:\t");
        scanf("%d", &iChoice);

        if(iChoice == 4)
            break;

        switch(iChoice)
        {
        case 1:
            iNo = DeleteFirst(&pFirst, &pLast);
            break;
        case 2:
            iNo = DeleteLast(&pFirst, &pLast);
            break;
        case 3:
            printf("Enter position:\t");
            scanf("%d", &iPos);
            iNo = DeleteAtPosition(&pFirst, &pLast, iPos);
            break;
```

```
            default:
                printf("Enter valid choice\n");
                iChoice = 4; // for checking outside for deleted data printing & display
            }

            if(-1 == iNo)
                printf("Linked List Empty\n");
            else if(iChoice != 4 && iNo != -2)
            {
                printf("Deleted data is %d\n", iNo);
                Display(pFirst, pLast);
            }
        }

        break;

    case 3:
        if(NULL == pFirst)
        {
            printf("Linked List Empty, Searching impossible.\n");
            break;
        }
        while(1)
        {
            printf(
            "\n1.SearchFirstOccurance\n2.SearchLastOccurance\n3.SearchAllOccurances\n4.Back\
            n");
            printf("Enter your choice again:\t");
            scanf("%d", &iChoice);

            if(iChoice == 4)
                break;

            if(iChoice <= 0 || iChoice > 3)
            {
                printf("Enter valid choice\n");
                continue;
            }

            Display(pFirst, pLast);

            printf("Enter data to be search:\t");
            scanf("%d", &iNo);

            switch(iChoice)
            {
            case 1:
                iNo = SearchFirstOccurance(pFirst, pLast, iNo);
                if(-1 == iNo)
                    printf("Linked List Empty\n");
                else if(-2 == iNo)
                    printf("Data not found\n");
                else
                    printf("Data found at %d position\n", iNo);
                break;
            case 2:
                iNo = SearchLastOccurance(pFirst, pLast, iNo);
                if(-1 == iNo)
                    printf("Linked List Empty\n");
                else if(-2 == iNo)
                    printf("Data not found\n");
                else
                    printf("Data found at %d position\n", iNo);
                break;
            case 3:
                iNo = SearchAllOccurances(pFirst, pLast, iNo);
                printf("Data found %d times\n", iNo);
```

```
                }
            }

            break;

        case 4:
            Display(pFirst, pLast);
            iNo = CountNode(pFirst, pLast);
            printf("Total node present : %d\n", iNo);
            break;

        case 5:
            Display(pFirst, pLast);
            ReverseDisplay(pFirst, pLast);
            break;

        case 6: //exit
            Display(pFirst, pLast);

            if(pFirst != NULL)
                DeleteAllNodes(&pFirst, &pLast);

            printf("Bye...\n");
            return 0;

        default:
            printf("Enter valid choice.\n");
        }
    }
}

void InsertFirst(struct List **ppHead, struct List **ppTail, int iNo)
{
    struct List *pNewNode = NULL;

    pNewNode = (struct List *)malloc(sizeof(struct List));
    if(NULL == pNewNode)
    {
        printf("memory allocation FAILED\n");
        return;
    }

    pNewNode->iData = iNo;

    if(NULL == *ppHead) // if list initially empty
    {
        *ppHead = pNewNode;
        *ppTail = pNewNode;
        (*ppTail)->pNext = *ppHead;
        (*ppHead)->pPrev = *ppTail;
        return;
    }

    pNewNode->pNext = *ppHead;
    (*ppHead)->pPrev = pNewNode;
    *ppHead = pNewNode;
    (*ppTail)->pNext = *ppHead;
    (*ppHead)->pPrev = *ppTail;
}

void InsertLast(struct List **ppHead, struct List **ppTail, int iNo)
{
    struct List *pNewNode = NULL;

    pNewNode = (struct List *)malloc(sizeof(struct List));
    if(NULL == pNewNode)
    {
```

```
        printf("memory allocation FAILED\n");
        return;
    }

    pNewNode->iData = iNo;

    if(NULL == *ppHead) // if list initially empty
    {
        *ppHead = pNewNode;
        *ppTail = pNewNode;
        (*ppTail)->pNext = *ppHead;
        (*ppHead)->pPrev = *ppTail;
        return;
    }

    (*ppTail)->pNext = pNewNode;
    pNewNode->pPrev = *ppTail;
    *ppTail = pNewNode;
    (*ppTail)->pNext = *ppHead;
    (*ppHead)->pPrev = *ppTail;
}

void InsertAtPosition(struct List **ppHead, struct List **ppTail, int iNo, int iPos)
{
    struct List *pNewNode = NULL;
    struct List *pTemp = NULL;
    int iCount;

    iCount = CountNode(*ppHead, *ppTail);

    if(iPos <= 0 || iPos > iCount + 1)
    {
        printf("Position is invalid\n");
        return;
    }

    if(1 == iPos) // first position
    {
        InsertFirst(ppHead, ppTail, iNo);
        return;
    }

    if(iCount + 1 == iPos) // last position
    {
        InsertLast(ppHead, ppTail, iNo);
        return;
    }

    //
    //  middle position
    //

    pTemp = *ppHead;
    iCount = 1;
    while(iCount < iPos - 1)
    {
        iCount++;
        pTemp = pTemp->pNext;
    }

    pNewNode = (struct List *)malloc(sizeof(struct List));
    if(NULL == pNewNode)
    {
        printf("memory allocation FAILED\n");
        return;
    }
```

```
    pNewNode->iData = iNo;

    pNewNode->pNext = pTemp->pNext;
    pTemp->pNext->pPrev = pNewNode;
    pTemp->pNext = pNewNode;
    pNewNode->pPrev = pTemp;
}

int DeleteFirst(struct List **ppHead, struct List **ppTail)
{
    int iDelData;

    if(NULL == *ppHead)
        return -1;

    iDelData = (*ppHead)->iData;

    if(*ppHead == *ppTail) // only single node present
    {
        (*ppHead)->pNext = NULL;
        (*ppHead)->pPrev = NULL;
        free(*ppHead);
        *ppHead = NULL;
        *ppTail = NULL;
        return iDelData;
    }

    *ppHead = (*ppHead)->pNext;
    (*ppTail)->pNext->pNext = NULL;
    (*ppTail)->pNext->pPrev = NULL;
    free((*ppTail)->pNext);

    (*ppTail)->pNext = *ppHead;
    (*ppHead)->pPrev = *ppTail;

    return iDelData;
}

int DeleteLast(struct List **ppHead, struct List **ppTail)
{
    int iDelData;

    if(NULL == *ppHead)
        return -1;

    iDelData = (*ppTail)->iData;

    if(*ppHead == *ppTail) // only single node present
    {
        (*ppHead)->pNext = NULL;
        (*ppHead)->pPrev = NULL;
        free(*ppHead);
        *ppHead = NULL;
        *ppTail = NULL;
        return iDelData;
    }

    *ppTail = (*ppTail)->pPrev;
    (*ppHead)->pPrev->pNext = NULL;
    (*ppHead)->pPrev->pPrev = NULL;
    free((*ppHead)->pPrev);

    (*ppTail)->pNext = *ppHead;
    (*ppHead)->pPrev = *ppTail;

    return iDelData;
}
```

```c
int DeleteAtPosition(struct List **ppHead, struct List **ppTail, int iPos)
{
    struct List *pTemp = NULL;
    int iCount;

    iCount = CountNode(*ppHead, *ppTail);

    if(iPos <= 0 || iPos > iCount)
    {
        printf("Position is invalid\n");
        return -2;
    }

    if(1 == iPos) // first position
        return DeleteFirst(ppHead, ppTail);

    if(iCount == iPos) // last position
        return DeleteLast(ppHead, ppTail);

    //
    //  middle position
    //

    pTemp = *ppHead;
    iCount = 1;
    while(iCount < iPos)
    {
        iCount++;
        pTemp = pTemp->pNext;
    }

    pTemp->pPrev->pNext = pTemp->pNext;
    pTemp->pNext->pPrev = pTemp->pPrev;
    pTemp->pNext = NULL;
    pTemp->pPrev = NULL;

    iCount = pTemp->iData;
    free(pTemp);

    return iCount;
}

int SearchFirstOccurance(struct List *pHead, struct List *pTail, int iKey)
{
    int iPos;

    if(NULL == pHead) // empty
        return -1;

    iPos = 1;

    do
    {
        if(pHead->iData == iKey)
            break;

        iPos++;
        pHead = pHead->pNext;
    }while(pHead!=pTail->pNext);

    if(pHead == pTail->pNext && iPos != 1) // not found
        return -2;

    return iPos;
}
```

```c
int SearchLastOccurance(struct List *pHead, struct List *pTail, int iKey)
{
    int iPos;
    int iLast;

    if(NULL == pHead) // empty
        return -1;

    iPos = 1;
    iLast = 0;

    do
    {
        if(pHead->iData == iKey)
            iLast = iPos;

        iPos++;
        pHead = pHead->pNext;
    }while(pHead!=pTail->pNext);

    if(0 == iLast) // not found
        return -2;

    return iLast;
}

int SearchAllOccurances(struct List *pHead, struct List *pTail, int iKey)
{
    int iCount;

    iCount = 0;

    if(NULL == pHead) // empty
        return iCount;

    do
    {
        if(pHead->iData == iKey)
            iCount++;

        pHead = pHead->pNext;
    }while(pHead!=pTail->pNext);

    return iCount;
}

int CountNode(struct List *pHead, struct List *pTail)
{
    int iCount;

    iCount = 0;

    if(NULL == pHead) // empty
        return iCount;

    do
    {
        iCount++;
        pHead = pHead->pNext;
    }while(pHead!=pTail->pNext);

    return iCount;
}

void Display(struct List *pHead, struct List *pTail)
{
    printf("\nLinked list is:\n");
```

```
    if(NULL == pHead)
    {
        printf("EMPTY\n");
        return;
    }

    do
    {
        printf("<-|%d|->", pHead->iData);
        pHead = pHead->pNext;
    }while(pHead!=pTail->pNext);

    printf("\n");
}

void DeleteAllNodes(struct List **ppHead, struct List **ppTail)
{
    if(NULL == *ppHead)
        return;

    while(*ppHead != *ppTail) // till single node present
    {
        (*ppHead)->pPrev = NULL;
        *ppHead = (*ppHead)->pNext;
        (*ppTail)->pNext->pNext = NULL;
        free((*ppTail)->pNext);

        (*ppTail)->pNext = *ppHead;
    }

    (*ppHead)->pNext = NULL;
    (*ppHead)->pPrev = NULL;
    free(*ppHead);

    *ppHead = NULL;
    *ppTail = NULL;

    printf("\nDeleted All Nodes Successfully\n");
}

void ReverseDisplay(struct List *pHead, struct List *pTail)
{
    printf("\nReverse Linked list is:\n");

    if(NULL == pTail)
    {
        printf("EMPTY\n");
        return;
    }

    do
    {
        printf("<-|%d|->", pTail->iData);
        pTail = pTail->pPrev;
    }while(pTail!=pHead->pPrev);

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