```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)
```

```
68          {
69          case 1:
70              while(1)
71              {
72                  printf("\n1.InsertFirst\n2.InsertLast\n3.InsertAtPosition\n4.Back\n");
73                  printf("Enter your choice again:\t");
74                  scanf("%d", &iChoice);
75
76                  if(iChoice == 4)
77                      break;
78
79                  if(iChoice <= 0 || iChoice > 3)
80                  {
81                      printf("Enter valid choice\n");
82                      continue;
83                  }
84
85                  printf("Enter data to be insert:\t");
86                  scanf("%d", &iNo);
87
88                  switch(iChoice)
89                  {
90                  case 1:
91                      InsertFirst(&pFirst, &pLast, iNo);
92                      break;
93                  case 2:
94                      InsertLast(&pFirst, &pLast, iNo);
95                      break;
96                  case 3:
97                      printf("Enter position:\t");
98                      scanf("%d", &iPos);
99                      InsertAtPosition(&pFirst, &pLast, iNo, iPos);
100                 }
101
102                 Display(pFirst, pLast);
103             }
104
105             break;
106
107         case 2:
108             if(NULL == pFirst)
109             {
110                 printf("Linked List Empty, Deletion impossible.\n");
111                 break;
112             }
113             while(1)
114             {
115                 printf("\n1.DeleteFirst\n2.DeleteLast\n3.DeleteAtPosition\n4.Back\n");
116                 printf("Enter your choice again:\t");
117                 scanf("%d", &iChoice);
118
119                 if(iChoice == 4)
120                     break;
121
122                 switch(iChoice)
123                 {
124                 case 1:
125                     iNo = DeleteFirst(&pFirst, &pLast);
126                     break;
127                 case 2:
128                     iNo = DeleteLast(&pFirst, &pLast);
129                     break;
130                 case 3:
131                     printf("Enter position:\t");
132                     scanf("%d", &iPos);
133                     iNo = DeleteAtPosition(&pFirst, &pLast, iPos);
134                     break;
```

```
135                 default:
136                     printf("Enter valid choice\n");
137                     iChoice = 4; // for checking outside for deleted data printing &
                        display
138                 }
139 
140                 if(-1 == iNo)
141                     printf("Linked List Empty\n");
142                 else if(iChoice != 4 && iNo != -2)
143                 {
144                     printf("Deleted data is %d\n", iNo);
145                     Display(pFirst, pLast);
146                 }
147             }
148 
149             break;
150 
151         case 3:
152             if(NULL == pFirst)
153             {
154                 printf("Linked List Empty, Searching impossible.\n");
155                 break;
156             }
157             while(1)
158             {
159                 printf(
                    "\n1.SearchFirstOccurance\n2.SearchLastOccurance\n3.SearchAllOccurances\n4
                    .Back\n");
160                 printf("Enter your choice again:\t");
161                 scanf("%d", &iChoice);
162 
163                 if(iChoice == 4)
164                     break;
165 
166                 if(iChoice <= 0 || iChoice > 3)
167                 {
168                     printf("Enter valid choice\n");
169                     continue;
170                 }
171 
172                 Display(pFirst, pLast);
173 
174                 printf("Enter data to be search:\t");
175                 scanf("%d", &iNo);
176 
177                 switch(iChoice)
178                 {
179                 case 1:
180                     iNo = SearchFirstOccurance(pFirst, pLast, iNo);
181                     if(-1 == iNo)
182                         printf("Linked List Empty\n");
183                     else if(-2 == iNo)
184                         printf("Data not found\n");
185                     else
186                         printf("Data found at %d position\n", iNo);
187                     break;
188                 case 2:
189                     iNo = SearchLastOccurance(pFirst, pLast, iNo);
190                     if(-1 == iNo)
191                         printf("Linked List Empty\n");
192                     else if(-2 == iNo)
193                         printf("Data not found\n");
194                     else
195                         printf("Data found at %d position\n", iNo);
196                     break;
197                 case 3:
198                     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)
```

```
266       {
267           printf("memory allocation FAILED\n");
268           return;
269       }
270
271       pNewNode->iData = iNo;
272
273       if(NULL == *ppHead) // if list initially empty
274       {
275           *ppHead = pNewNode;
276           *ppTail = pNewNode;
277           (*ppTail)->pNext = *ppHead;
278           (*ppHead)->pPrev = *ppTail;
279           return;
280       }
281
282       (*ppTail)->pNext = pNewNode;
283       pNewNode->pPrev = *ppTail;
284       *ppTail = pNewNode;
285       (*ppTail)->pNext = *ppHead;
286       (*ppHead)->pPrev = *ppTail;
287   }
288
289   void InsertAtPosition(struct List **ppHead, struct List **ppTail, int iNo, int iPos)
290   {
291       struct List *pNewNode = NULL;
292       struct List *pTemp = NULL;
293       int iCount;
294
295       iCount = CountNode(*ppHead, *ppTail);
296
297       if(iPos <= 0 || iPos > iCount + 1)
298       {
299           printf("Position is invalid\n");
300           return;
301       }
302
303       if(1 == iPos) // first position
304       {
305           InsertFirst(ppHead, ppTail, iNo);
306           return;
307       }
308
309       if(iCount + 1 == iPos) // last position
310       {
311           InsertLast(ppHead, ppTail, iNo);
312           return;
313       }
314
315       //
316       //  middle position
317       //
318
319       pTemp = *ppHead;
320       iCount = 1;
321       while(iCount < iPos - 1)
322       {
323           iCount++;
324           pTemp = pTemp->pNext;
325       }
326
327       pNewNode = (struct List *)malloc(sizeof(struct List));
328       if(NULL == pNewNode)
329       {
330           printf("memory allocation FAILED\n");
331           return;
332       }
```

```
333
334         pNewNode->iData = iNo;
335
336         pNewNode->pNext = pTemp->pNext;
337         pTemp->pNext->pPrev = pNewNode;
338         pTemp->pNext = pNewNode;
339         pNewNode->pPrev = pTemp;
340     }
341
342     int DeleteFirst(struct List **ppHead, struct List **ppTail)
343     {
344         int iDelData;
345
346         if(NULL == *ppHead)
347             return -1;
348
349         iDelData = (*ppHead)->iData;
350
351         if(*ppHead == *ppTail) // only single node present
352         {
353             (*ppHead)->pNext = NULL;
354             (*ppHead)->pPrev = NULL;
355             free(*ppHead);
356             *ppHead = NULL;
357             *ppTail = NULL;
358             return iDelData;
359         }
360
361         *ppHead = (*ppHead)->pNext;
362         (*ppTail)->pNext->pNext = NULL;
363         (*ppTail)->pNext->pPrev = NULL;
364         free((*ppTail)->pNext);
365
366         (*ppTail)->pNext = *ppHead;
367         (*ppHead)->pPrev = *ppTail;
368
369         return iDelData;
370     }
371
372     int DeleteLast(struct List **ppHead, struct List **ppTail)
373     {
374         int iDelData;
375
376         if(NULL == *ppHead)
377             return -1;
378
379         iDelData = (*ppTail)->iData;
380
381         if(*ppHead == *ppTail) // only single node present
382         {
383             (*ppHead)->pNext = NULL;
384             (*ppHead)->pPrev = NULL;
385             free(*ppHead);
386             *ppHead = NULL;
387             *ppTail = NULL;
388             return iDelData;
389         }
390
391         *ppTail = (*ppTail)->pPrev;
392         (*ppHead)->pPrev->pNext = NULL;
393         (*ppHead)->pPrev->pPrev = NULL;
394         free((*ppHead)->pPrev);
395
396         (*ppTail)->pNext = *ppHead;
397         (*ppHead)->pPrev = *ppTail;
398
399         return iDelData;
```

```
}

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;
}
```

```
467
468 int SearchLastOccurance(struct List *pHead, struct List *pTail, int iKey)
469 {
470     int iPos;
471     int iLast;
472
473     if(NULL == pHead) // empty
474         return -1;
475
476     iPos = 1;
477     iLast = 0;
478
479     do
480     {
481         if(pHead->iData == iKey)
482             iLast = iPos;
483
484         iPos++;
485         pHead = pHead->pNext;
486     }while(pHead!=pTail->pNext);
487
488     if(0 == iLast) // not found
489         return -2;
490
491     return iLast;
492 }
493
494 int SearchAllOccurances(struct List *pHead, struct List *pTail, int iKey)
495 {
496     int iCount;
497
498     iCount = 0;
499
500     if(NULL == pHead) // empty
501         return iCount;
502
503     do
504     {
505         if(pHead->iData == iKey)
506             iCount++;
507
508         pHead = pHead->pNext;
509     }while(pHead!=pTail->pNext);
510
511     return iCount;
512 }
513
514 int CountNode(struct List *pHead, struct List *pTail)
515 {
516     int iCount;
517
518     iCount = 0;
519
520     if(NULL == pHead) // empty
521         return iCount;
522
523     do
524     {
525         iCount++;
526         pHead = pHead->pNext;
527     }while(pHead!=pTail->pNext);
528
529     return iCount;
530 }
531
532 void Display(struct List *pHead, struct List *pTail)
533 {
```

```
534       printf("\nLinked list is:\n");
535
536       if(NULL == pHead)
537       {
538           printf("EMPTY\n");
539           return;
540       }
541
542       do
543       {
544           printf("<-|%d|->", pHead->iData);
545           pHead = pHead->pNext;
546       }while(pHead!=pTail->pNext);
547
548       printf("\n");
549   }
550
551   void DeleteAllNodes(struct List **ppHead, struct List **ppTail)
552   {
553       if(NULL == *ppHead)
554           return;
555
556       while(*ppHead != *ppTail) // till single node present
557       {
558           (*ppHead)->pPrev = NULL;
559           *ppHead = (*ppHead)->pNext;
560           (*ppTail)->pNext->pNext = NULL;
561           free((*ppTail)->pNext);
562
563           (*ppTail)->pNext = *ppHead;
564       }
565
566       (*ppHead)->pNext = NULL;
567       (*ppHead)->pPrev = NULL;
568       free(*ppHead);
569
570       *ppHead = NULL;
571       *ppTail = NULL;
572
573       printf("\nDeleted All Nodes Successfully\n");
574   }
575
576   void ReverseDisplay(struct List *pHead, struct List *pTail)
577   {
578       printf("\nReverse Linked list is:\n");
579
580       if(NULL == pTail)
581       {
582           printf("EMPTY\n");
583           return;
584       }
585
586       do
587       {
588           printf("<-|%d|->", pTail->iData);
589           pTail = pTail->pPrev;
590       }while(pTail!=pHead->pPrev);
591
592       printf("\n");
593   }
594
```