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

struct node {
    int value;
    struct node *link;
};
typedef struct node node1;

void in_beg();
void in_end();
void in_bw();
void del_beg();
void del_end();
void del_bw();
void display();
node1 *create();
node1 *start = NULL;

void main() {
    int ch, wish;
    printf("\n\tLinked list implementation\n\t_____\n\n\t"
           "1. Insert at beginning\n\t"
           "2. Insert at end\n\t"
           "3. Insert in between\n\t"
           "4. Delete from beginning\n\t"
           "5. Delete from end\n\t"
           "6. Delete in between\n\t"
           "7. Display\n\t"
           "8. Exit\n");

    do {
        printf("\nEnter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
            case 1: in_beg(); break;
            case 2: in_end(); break;
            case 3: in_bw(); break;
            case 4: del_beg(); break;
            case 5: del_end(); break;
            case 6: del_bw(); break;
            case 7: display(); break;
            case 8: exit(0); break;
            default: printf("\nInvalid choice");
        }

        // Only allow 1 for continue or 0 to exit
        printf("\nDo you wish to continue? (1 = Yes, 0 = No): ");
        scanf("%d", &wish);

        if (wish != 1) {
            // Exit the loop if user doesn't input 1
            break;
        }

    } while (1); // Keep the loop running as long as the user enters 1 for wish
}
```

```
node1 *create() {
    node1 *nptr = (node1*)malloc(sizeof(node1));
    if (nptr == NULL) {
        printf("Memory overflow\n");
        return 0;
    } else {
        return nptr;
    }
}

void in_beg() {
    int val;
    node1 *nptr = create();
    printf("Enter element: ");
    scanf("%d", &val);
    nptr->value = val;
    if (start == NULL) {
        start = nptr;
        nptr->link = NULL;
    } else {
        nptr->link = start;
        start = nptr;
    }
}

void in_end() {
    int val;
    node1 *nptr = create();
    printf("Enter element: ");
    scanf("%d", &val);
    nptr->value = val;
    nptr->link = NULL;

    if (start == NULL) {
        start = nptr;
    } else {
        node1 *temp = start;
        while (temp->link != NULL) {
            temp = temp->link;
        }
        temp->link = nptr;
    }
}

void in_bw() {
    int val, pos, i;
    node1 *nptr = create();
    printf("Enter element and Position to be inserted: ");
    scanf("%d %d", &val, &pos);
    nptr->value = val;

    if (pos == 1) {
        nptr->link = start;
        start = nptr;
    } else {
        node1 *temp = start;
        for (i = 1; i < pos - 1 && temp != NULL; i++) {
```

```
            temp = temp->link;
        }
        if (temp == NULL) {
            printf("Position out of bounds\n");
            return;
        }
        nptr->link = temp->link;
        temp->link = nptr;
    }
}

void display() {
    node1 *temp;
    if (start == NULL) {
        printf("LIST EMPTY\n");
        return;
    }
    temp = start;
    while (temp != NULL) {
        printf("%d ", temp->value);
        temp = temp->link;
    }
    printf("\n");
}

void del_beg() {
    if (start == NULL) {
        printf("LIST EMPTY\n");
        return;
    }
    node1 *temp = start;
    start = start->link;
    free(temp);
}

void del_end() {
    if (start == NULL) {
        printf("LIST EMPTY\n");
        return;
    }

    node1 *temp = start, *prev = NULL;

    if (temp->link == NULL) {  // Only one node
        free(temp);
        start = NULL;
        return;
    }

    while (temp->link != NULL) {
        prev = temp;
        temp = temp->link;
    }
    prev->link = NULL;
    free(temp);
}
```

```
void del_bw() {
    if (start == NULL) {
        printf("LIST EMPTY\n");
        return;
    }

    int pos, i;
    printf("Enter position of the node to be deleted: ");
    scanf("%d", &pos);

    node1 *temp = start, *prev = NULL;

    if (pos == 1) {
        start = start->link;
        free(temp);
        return;
    }

    for (i = 1; i < pos && temp != NULL; i++) {
        prev = temp;
        temp = temp->link;
    }

    if (temp == NULL) {
        printf("Position out of bounds\n");
        return;
    }

    prev->link = temp->link;
    free(temp);
}
```

```
#include <stdio.h>
#include <stdlib.h>

struct link {
   int info;
   struct link *next, *prev;
};
typedef struct link node;

void inb();   // Insert at beginning
void ine();   // Insert at end
void inbw();  // Insert in between
void delb();  // Delete from beginning
void dele();  // Delete from end
void delbw(); // Delete from in between
void dispf(); // Traverse forward
void dispb(); // Traverse backward
void search(); // Search

node *start = NULL;
node *create();

void main() {
   int ch, wish = 1;

   do {
      printf("\n\n\tDoubly linked list implementation\n\t*****************************\n\n\t"
            "1. Insert at beginning\n\t"
            "2. Insert at end\n\t"
            "3. Insert in between\n\t"
            "4. Delete from beginning\n\t"
            "5. Delete from end\n\t"
            "6. Delete from in between\n\t"
            "7. Traverse forward\n\t"
            "8. Traverse backward\n\t"
            "9. Search\n\t"
            "10. Exit\n\n\tEnter your choice : ");
      scanf("%d", &ch);

      switch (ch) {
         case 1: inb(); break;
         case 2: ine(); break;
         case 3: inbw(); break;
         case 4: delb(); break;
         case 5: dele(); break;
         case 6: delbw(); break;
         case 7: dispf(); break;
         case 8: dispb(); break;
         case 9: search(); break;
         case 10: exit(0); break;
         default: printf("\nInvalid choice");
      }

      printf("\nDo you wish to continue? (1/0): ");
      scanf("%d", &wish);

   } while (wish == 1);
}

node *create() {
```

```
    node *nptr = (node *) malloc(sizeof(node));
    if (nptr == NULL) {
        printf("Memory overflow");
        return NULL;
    } else {
        return nptr;
    }
}

void inb() { // Insert at the beginning
    node *nptr = create();
    int e;

    printf("\nEnter item: ");
    scanf("%d", &e);
    nptr->info = e;

    if (start == NULL) {
        start = nptr;
        nptr->next = NULL;
        nptr->prev = NULL;
    } else {
        nptr->next = start;
        nptr->prev = NULL;
        start->prev = nptr;
        start = nptr;
    }
    printf("\nElement inserted\n");
}

void ine() { // Insert at the end
    node *nptr = create();
    node *temp = start;
    int e;

    printf("\nEnter item: ");
    scanf("%d", &e);
    nptr->info = e;

    if (start == NULL) {
        start = nptr;
        nptr->next = NULL;
        nptr->prev = NULL;
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        temp->next = nptr;
        nptr->prev = temp;
        nptr->next = NULL;
    }
    printf("\nElement inserted\n");
}

void inbw() { // Insert in between
    int c = 1, pos, e;
    node *nptr = create();
    node *temp = start;

    printf("\nEnter item: ");
```

```
    scanf("%d", &e);
    nptr->info = e;

    printf("\nEnter position: ");
    scanf("%d", &pos);

    if (pos == 1) { // Inserting at the beginning if pos = 1
        nptr->next = start;
        nptr->prev = NULL;
        if (start != NULL) {
            start->prev = nptr;
        }
        start = nptr;
    } else {
        node *ptr = NULL;
        while (temp != NULL && c < pos) {
            ptr = temp;
            temp = temp->next;
            c++;
        }

        if (temp != NULL) {
            nptr->next = temp;
            temp->prev = nptr;
        }
        ptr->next = nptr;
        nptr->prev = ptr;
    }
    printf("\nElement inserted\n");
}

void delb() { // Delete from beginning
    node *nptr;
    if (start == NULL) {
        printf("\nEmpty list\n");
    } else {
        nptr = start;
        printf("\nElement deleted is: %d", nptr->info);
        start = start->next;
        if (start != NULL) {
            start->prev = NULL;
        }
        free(nptr);
    }
}

void dele() { // Delete from end
    if (start == NULL) {
        printf("\nEmpty list\n");
    } else {
        node *temp = start;
        node *ptr = NULL;

        if (temp->next == NULL) { // Only one node in the list
            printf("\nElement deleted is: %d", temp->info);
            free(temp);
            start = NULL;
        } else {
            while (temp->next != NULL) {
                ptr = temp;
```

```
            temp = temp->next;
        }
        printf("\nElement deleted is: %d", temp->info);
        ptr->next = NULL;
        free(temp);
    }
  }
}

void delbw() { // Delete from in between
  int pos, c = 1;
  node *temp = start;
  node *ptr = NULL;

  printf("\nEnter position: ");
  scanf("%d", &pos);

  if (pos == 1) { // Deleting from the beginning if pos = 1
    delb();
    return;
  }

  while (temp != NULL && c < pos) {
    ptr = temp;
    temp = temp->next;
    c++;
  }

  if (temp == NULL) {
    printf("\nPosition out of range\n");
  } else {
    printf("\nElement deleted is: %d", temp->info);
    ptr->next = temp->next;
    if (temp->next != NULL) {
      temp->next->prev = ptr;
    }
    free(temp);
  }
}

void dispf() { // Traverse forward
  node *temp = start;
  if (temp == NULL) {
    printf("\nEmpty list\n");
  } else {
    printf("\nLinked list is: ");
    while (temp != NULL) {
      printf("\t%d", temp->info);
      temp = temp->next;
    }
    printf("\n");
  }
}
```

```
void dispb() { // Traverse backward
    node *temp = start;
    if (temp == NULL) {
        printf("\nEmpty list\n");
    } else {
        while (temp->next != NULL) {
            temp = temp->next;
        }
        printf("\nLinked list in reverse is: ");
        while (temp != NULL) {
            printf("\t%d", temp->info);
            temp = temp->prev;
        }
        printf("\n");
    }
}

void search() { // Search for an element
    int e, c = 1, found = 0;
    node *temp = start;

    if (temp == NULL) {
        printf("\nEmpty list");
    } else {
        printf("\nEnter element to be searched: ");
        scanf("%d", &e);

        while (temp != NULL) {
            if (temp->info == e) {
                printf("\nElement found at position %d", c);
                found = 1;
                break;
            }
            temp = temp->next;
            c++;
        }
        if (!found) {
            printf("\nElement not found");
        }
    }
}
```