Doubly Linked List Program in C

Doubly Linked List is a variation of Linked list in which navigation is possible in both ways, either forward and backward easily as compared to Single Linked List.

Implementation in C

Live Demo

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

struct node {
   int data;
   int key;

   struct node *next;
   struct node *prev;
};

//this link always point to first Link
struct node *head = NULL;

//this link always point to last Link
struct node *last = NULL;

struct node *current = NULL;

//is list empty
bool isEmpty() {
   return head == NULL;
}

int length() {
   int length = 0;
   struct node *current;

   for(current = head; current != NULL; current = current->next){
```

```
      length++;
   }

   return length;
}

//display the list in from first to last
void displayForward() {

   //start from the beginning
   struct node *ptr = head;

   //navigate till the end of the list
   printf("\n[ ");

   while(ptr != NULL) {
      printf("(%d,%d) ",ptr->key,ptr->data);
      ptr = ptr->next;
   }

   printf(" ]");
}

//display the list from last to first
void displayBackward() {

   //start from the last
   struct node *ptr = last;

   //navigate till the start of the list
   printf("\n[ ");

   while(ptr != NULL) {

      //print data
      printf("(%d,%d) ",ptr->key,ptr->data);

      //move to next item
      ptr = ptr ->prev;

   }

}
```

```
//insert link at the first location
void insertFirst(int key, int data) {

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty()) {
      //make it the last link
      last = link;
   } else {
      //update first prev link
      head->prev = link;
   }

   //point it to old first link
   link->next = head;

   //point first to new first link
   head = link;
}

//insert link at the last location
void insertLast(int key, int data) {

   //create a link
   struct node *link = (struct node*) malloc(sizeof(struct node));
   link->key = key;
   link->data = data;

   if(isEmpty()) {
      //make it the last link
      last = link;
   } else {
      //make link a new last link
      last->next = link;

      //mark old last node as prev of new link
      link->prev = last;
   }

   //point last to new last node
   last = link;
```

```
}

//delete first item
struct node* deleteFirst() {

   //save reference to first link
   struct node *tempLink = head;

   //if only one link
   if(head->next == NULL){
      last = NULL;
   } else {
      head->next->prev = NULL;
   }

   head = head->next;
   //return the deleted link
   return tempLink;
}

//delete link at the last location

struct node* deleteLast() {
   //save reference to last link
   struct node *tempLink = last;

   //if only one link
   if(head->next == NULL) {
      head = NULL;
   } else {
      last->prev->next = NULL;
   }

   last = last->prev;

   //return the deleted link
   return tempLink;
}

//delete a link with given key

struct node* delete(int key) {

   //start from the first link
```

```
   struct node* current = head;
   struct node* previous = NULL;

   //if list is empty
   if(head == NULL) {
      return NULL;
   }

   //navigate through list
   while(current->key != key) {
      //if it is last node

      if(current->next == NULL) {
         return NULL;
      } else {
         //store reference to current link
         previous = current;

         //move to next link
         current = current->next;
      }
   }

   //found a match, update the link
   if(current == head) {
      //change first to point to next link
      head = head->next;
   } else {
      //bypass the current link
      current->prev->next = current->next;
   }

   if(current == last) {
      //change last to point to prev link
      last = current->prev;
   } else {
      current->next->prev = current->prev;
   }

   return current;
}

bool insertAfter(int key, int newKey, int data) {
   //start from the first link
```

```
   struct node *current = head;

   //if list is empty
   if(head == NULL) {
      return false;
   }

   //navigate through list
   while(current->key != key) {

      //if it is last node
      if(current->next == NULL) {
         return false;
      } else {
         //move to next link
         current = current->next;
      }
   }

   //create a link
   struct node *newLink = (struct node*) malloc(sizeof(struct node));
   newLink->key = newKey;
   newLink->data = data;

   if(current == last) {
      newLink->next = NULL;
      last = newLink;
   } else {
      newLink->next = current->next;
      current->next->prev = newLink;
   }

   newLink->prev = current;
   current->next = newLink;
   return true;
}

void main() {
   insertFirst(1,10);
   insertFirst(2,20);
   insertFirst(3,30);
   insertFirst(4,1);
   insertFirst(5,40);
   insertFirst(6,56);
```

```
   printf("\nList (First to Last): ");
   displayForward();

   printf("\n");
   printf("\nList (Last to first): ");
   displayBackward();

   printf("\nList , after deleting first record: ");
   deleteFirst();
   displayForward();

   printf("\nList , after deleting last record: ");
   deleteLast();
   displayForward();

   printf("\nList , insert after key(4) : ");
   insertAfter(4,7, 13);
   displayForward();

   printf("\nList  , after delete key(4) : ");
   delete(4);
   displayForward();
}
```

If we compile and run the above program, it will produce the following result −

Output

```
List (First to Last):
[ (6,56) (5,40) (4,1) (3,30) (2,20) (1,10) ]

List (Last to first):
[ (1,10) (2,20) (3,30) (4,1) (5,40) (6,56) ]
List , after deleting first record:
[ (5,40) (4,1) (3,30) (2,20) (1,10) ]
List , after deleting last record:
[ (5,40) (4,1) (3,30) (2,20) ]
List , insert after key(4) :
[ (5,40) (4,1) (7,13) (3,30) (2,20) ]
List , after delete key(4) :
[ (5,40) (4,13) (3,30) (2,20) ]
```