Conrad Nestor Mativo
dLL Output

Main.c

```
/*
================================================================================
FILE        : main.c
AUTHOR      : Conrad Nestor B. Mativo
DESCRIPTION : A program that  insert (front, rear, at) the elements in to a doubly linked list, delete (front, rear, at)
the elements from that list and display (from front, from rear) all elements in that list.
COPYRIGHT   : 11 February 2024
REVISION HISTORY
Date:                   By:                 Description:
================================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include "dLLheader.h"
/*
================================================================================
FUNCTION    : int main
DESCRIPTION : Executes the main program
ARGUMENTS   : Void
RETURNS     : int - returns the exit code
================================================================================
*/
int main() {

    insertFront(3);
    printList();

    insertFront(2);
    printList();

    insertFront(1);
    printList();

    insertRear(5);
    printList();

    insertAt(0, 0);
    printList();

    removeFront();
    printList();

    removeRear();
```

```
    printList();

    removeRear();
    printList();

    insertRear(6);
    printList();

    insertAt(4, 3);
    printList();

    insertAt(7, 6);
    printList();

    insertAt(8, 2);
    printList();

    removeAt(2);
    printList();

    return 0;
}
```

dLLheader.h

```
/*
================================================================================
FILE        : dLLheader.h
AUTHOR      : Conrad Nestor B. Mativo
DESCRIPTION : A header file that contains the structures and function prototypes
COPYRIGHT   : 05 February 2024
REVISION HISTORY
Date:                   By:                 Description:
================================================================================
*/
#ifndef dLLheader_HH
#define dLLheader_HH

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

struct node {
    int data;
    struct node *prev, *next;
};

struct list {
    int count;
};
```

```
//Function prototypes
struct node* createNode(int data);
void insertFront(int data);
void insertRear(int data);
void insertAt(int data, int position);
void removeFront();
void removeRear();
void removeAt(int position);
void printList();
void printMirror();

#endif // dLLheader_H
```

dLLimplementation.c

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

struct node *head = NULL;
struct node *tail = NULL;

struct node* createNode(int data) {
    struct node* newNode = (struct node*)malloc(sizeof(struct node));
    newNode->data = data;
    newNode->prev = NULL;
    newNode->next = NULL;
    return newNode;
}
/*
================================================================================
FUNCTION    : insertFront
DESCRIPTION : insert element at the front/start/first of the list
ARGUMENTS   : int data - the data to be inserted
RETURNS     : void - returns nothing
================================================================================
*/
void insertFront(int data) {
    struct node* newNode = createNode(data);
    if (head == NULL) {
        head = tail = newNode;
    } else {
        newNode->next = head;
        head->prev = newNode;
        head = newNode;
    }
}
/*
================================================================================
FUNCTION    : insertRear
```

```
DESCRIPTION : insert element at the rear/end/last of the list
ARGUMENTS   : int data - the data to be inserted
RETURNS     : void - returns nothing
==========================================================================
*/
void insertRear(int data) {
   struct node* newNode = createNode(data);
   if (head == NULL) {
      head = tail = newNode;
   } else {
      tail->next = newNode;
      newNode->prev = tail;
      tail = newNode;
   }
}
/*
==========================================================================
FUNCTION    : insertAt
DESCRIPTION : insert element at a specified position in list
ARGUMENTS   : int data - the data to be inserted
              int position - position to be inserted
RETURNS     : void - returns nothing
==========================================================================
*/
void insertAt(int data, int position) {
   if (position < 0) {
      printf("Invalid position\n");
      return;
   }
   if (position == 0) {
      insertFront(data);
   } else {
      struct node* newNode = createNode(data);
      struct node* current = head;
      int i = 0;
      while (current != NULL && i < position - 1) {
         current = current->next;
         i++;
      }
      if (current == NULL) {
         printf("Invalid position\n");
         return;
      }
      newNode->next = current->next;
      newNode->prev = current;
      if (current->next != NULL) {
         current->next->prev = newNode;
      }
      current->next = newNode;
```

```
        if (newNode->next == NULL) {
            tail = newNode;
        }
    }
}
/*
================================================================================
FUNCTION    : removeFront
DESCRIPTION : remove element at the front/start/first of the list
ARGUMENTS   : void - no arguements
RETURNS     : void - returns nothing
================================================================================
*/
void removeFront() {
    if (head == NULL) {
        printf("List is empty\n");
        return;
    }
    struct node* temp = head;
    head = head->next;
    if (head != NULL) {
        head->prev = NULL;
    } else {
        tail = NULL;
    }
    free(temp);
}
/*
================================================================================
FUNCTION    : removeRear
DESCRIPTION : remove element at the rear/end/last of the list
ARGUMENTS   : void - no arguements
RETURNS     : void - returns nothing
================================================================================
*/
void removeRear() {
    if (tail == NULL) {
        printf("List is empty\n");
        return;
    }
    struct node* temp = tail;
    tail = tail->prev;
    if (tail != NULL) {
        tail->next = NULL;
    } else {
        head = NULL;
    }
    free(temp);
}
```

```
/*
==========================================================================
FUNCTION    : removeAt
DESCRIPTION : remove element at a specified position in list
ARGUMENTS   : int position - the position to be inserted
RETURNS     : void - returns nothing
==========================================================================
*/
void removeAt(int position) {
   if (position < 0 || head == NULL) {
      printf("Invalid position or list is empty\n");
      return;
   }
   if (position == 0) {
      removeFront();
   } else {
      struct node* current = head;
      int i = 0;
      while (current != NULL && i < position) {
         current = current->next;
         i++;
      }
      if (current == NULL) {
         printf("Invalid position\n");
         return;
      }
      if (current == tail) {
         removeRear();
      } else {
         current->prev->next = current->next;
         current->next->prev = current->prev;
         free(current);
      }
   }
}
/*
==========================================================================
FUNCTION    : printList
DESCRIPTION : display all elements in the list from front to rear.
ARGUMENTS   : void - no arguements
RETURNS     : void - returns nothing
==========================================================================
*/
void printList() {
   struct node* current = head;
   while (current != NULL) {
      printf("%d ", current->data);
      current = current->next;
   }
```

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

```
*/
int main() {

    insertFront(3);
    printList();

    insertFront(2);
    printList();

    insertFront(1);
    printList();

    insertRear(5);
    printList();

    insertAt(0, 0);
    printList();

    removeFront();
    printList();

    removeRear();
    printList();

    removeRear();
    printList();

    insertRear(6);
    printList();

    insertAt(4, 3);
    printList();

    insertAt(7, 6);
    printList();

    insertAt(8, 2);
    printList();

    removeAt(2);
    printList();
```