Conrad Nestor Mativo
sLL Output

Main.c

```
/*
================================================================================
FILE        : main.c
AUTHOR      : Conrad Nestor B. Mativo
DESCRIPTION : A program that can insert (front, rear, at) the elements into a singly linked list, delete (front, rear, at) the elements from that list and display all elements in that list.
COPYRIGHT   : 05 February 2024
REVISION HISTORY
Date:                   By:                 Description:
================================================================================
*/
#include <stdio.h>
#include "sllheader.h"
/*
================================================================================
FUNCTION    : int main
DESCRIPTION : Executes the main program
ARGUMENTS   : Void
RETURNS     : int - returns the exit code
================================================================================
*/
int main() {
    struct list *L = (struct list *)malloc(sizeof(struct list));
    initializeList(&L);

    printf("INITIAL DATA= 25, 30, 35\n\n");
    // Initial data in the linked list
    insertRear(&L, 25);
    insertRear(&L, 30);
    insertRear(&L, 35);

    // Execute the sequence of operations
    displayAll(&L); // Display initial data
    insertFront(&L, 2); displayAll(&L);
    insertFront(&L, 1); displayAll(&L);
    insertFront(&L, 0); displayAll(&L);
    insertRear(&L, 3); displayAll(&L);
    insertRear(&L, 5); displayAll(&L);
    insertAt(&L, 4, 4); displayAll(&L);
    deleteFront(&L); displayAll(&L);
    deleteRear(&L); displayAll(&L);
    insertAt(&L, 88, 2); displayAll(&L);
    deleteAt(&L, 2); displayAll(&L);
```

```
    return 0;
}
```

sLLheader.h

```
/*
================================================================================
FILE        : sLLheader.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 sLLheader_H
#define sLLheader_H

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

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

struct list {
    int count;
    struct node *head, *tail;
};

void initializeList(struct list *L);
void insertFront(struct list *L, int new_data);
void insertRear(struct list *L, int new_data);
void insertAt(struct list *L, int new_data, int position);
void deleteFront(struct list *L);
void deleteRear(struct list *L);
void deleteAt(struct list *L, int position);
void displayAll(struct list *L);

#endif // sLLheader_H
```

sLLimplementation.c

```
/*
================================================================================
FILE        : sLLimplementation.c
AUTHOR      : Conrad Nestor B. Mativo
DESCRIPTION : A c file containing the function algorthims
COPYRIGHT   : 05 February 2024
REVISION HISTORY
Date:                   By:             Description:
================================================================================
*/
#include <stdio.h>
#include <stdlib.h>
#include "sLLheader.h"
/*
================================================================================
FUNCTION    : initializeList
DESCRIPTION : Initializes the list
ARGUMENTS   : struct list * L - The pointer for the structure of the list;
RETURNS     : void - returns nothing
================================================================================
*/

void initializeList(struct list *L) {
   L->count = 0;
   L->head = NULL;
   L->tail = NULL;
}

/*
================================================================================
FUNCTION    : insertFront
DESCRIPTION : Inserting data to the front of list
ARGUMENTS   : struct list *L - the pointer for the structure of the list
              int new_data - data assigned to the front of the list.
RETURNS     : void - returns nothing
================================================================================
*/

void insertFront(struct list *L, int new_data) {
   struct node *new_node = (struct node *)malloc(sizeof(struct node));
   new_node->data = new_data;
   new_node->next = L->head;
   L->head = new_node;
   if (L->count == 0) {
      L->tail = new_node;
   }
   L->count++;
```

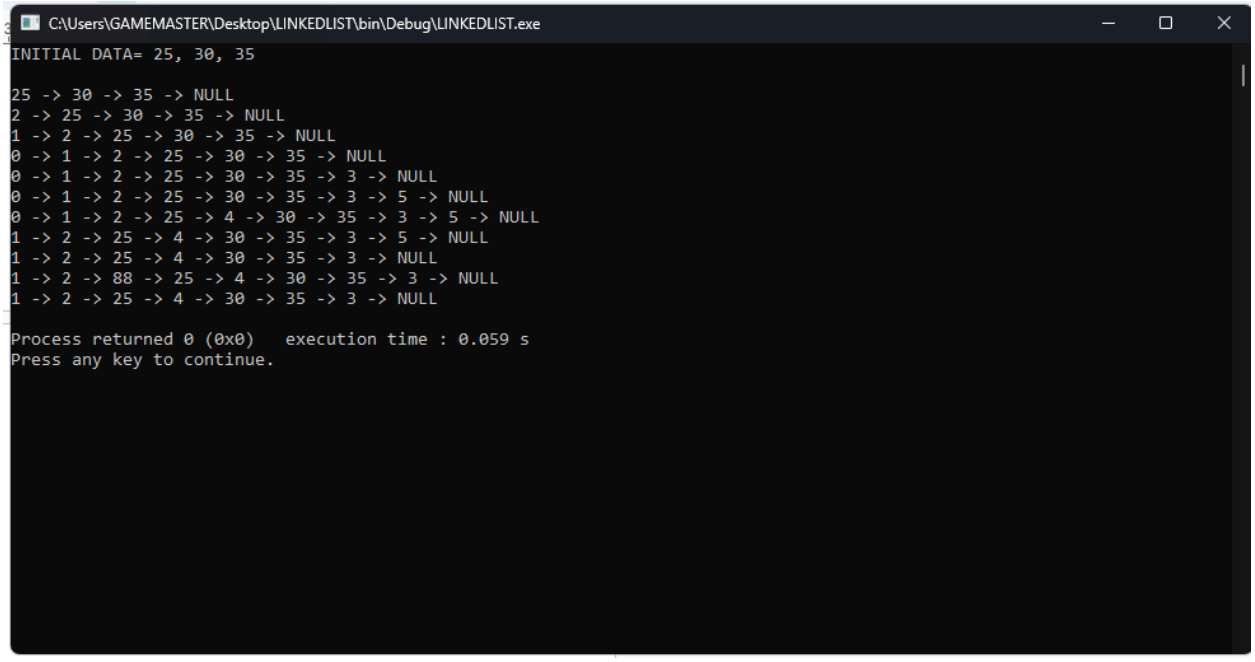
```
}
/*
================================================================================
FUNCTION    : insertRear
DESCRIPTION : Inserting data to the  of list
ARGUMENTS   : struct list *L - The pointer to the structure of the list
              int new_data - data to be inserted to the rear
RETURNS     : void - returns nothing
================================================================================
*/
void insertRear(struct list *L, int new_data) {
   struct node *new_node = (struct node *)malloc(sizeof(struct node));
   new_node->data = new_data;
   new_node->next = NULL;
   if (L->count == 0) {
      L->head = new_node;
      L->tail = new_node;
   } else {
      L->tail->next = new_node;
      L->tail = new_node;
   }
   L->count++;
}
/*
================================================================================
FUNCTION    : insertAt
DESCRIPTION : Inserting data to the specific part of the list
ARGUMENTS   : struct list *L,- The pointer to the structure of the list
              int new_data - data to be inserted
              int position - Position of the current sequence of the list with count++
RETURNS     : void - returns nothing
================================================================================
*/
void insertAt(struct list *L, int new_data, int position) {
   if (position < 0 || position > L->count) {
      printf("Invalid position\n");
      return;
   }

   if (position == 0) {
      insertFront(L, new_data);
   } else if (position == L->count) {
      insertRear(L, new_data);
   } else {
      struct node *temp = L->head;
      for (int i = 0; i < position - 1; i++) {
         temp = temp->next;
      }
      struct node *new_node = (struct node *)malloc(sizeof(struct node));
```

```
        new_node->data = new_data;
        new_node->next = temp->next;
        temp->next = new_node;
        L->count++;
    }
}
/*
================================================================================
FUNCTION    : deleteFront
DESCRIPTION : Deleting the front data of the list
ARGUMENTS   : struct list *L,- The pointer to the structure of the list
RETURNS     : void - returns nothing
================================================================================
*/
void deleteFront(struct list *L) {
    if (L->count == 0) {
        printf("List is empty\n");
        return;
    }
    struct node *temp = L->head;
    L->head = L->head->next;
    free(temp);
    L->count--;
}
/*
================================================================================
FUNCTION    : deleteRear
DESCRIPTION : Deleting the rear data of the list
ARGUMENTS   : struct list *L,- The pointer to the structure of the list
RETURNS     : void - returns nothing
================================================================================
*/
void deleteRear(struct list *L) {
    if (L->count == 0) {
        printf("List is empty\n");
        return;
    }
    struct node *temp = L->head;
    struct node *prev = NULL;
    while (temp->next != NULL) {
        prev = temp;
        temp = temp->next;
    }
    if (prev != NULL) {
        prev->next = NULL;
        L->tail = prev;
    } else {
        L->head = NULL;
        L->tail = NULL;
```

```
    }
    free(temp);
    L->count--;
}
/*
================================================================================
FUNCTION    : deleteAt
DESCRIPTION : Deleting the specific data on the list
ARGUMENTS   : struct list *L,- The pointer to the structure of the list
              int position - Position of the current sequence of the list with count++
RETURNS     : void - returns nothing
================================================================================
*/
void deleteAt(struct list *L, int position) {
    if (position < 0 || position >= L->count) {
        printf("Invalid position\n");
        return;
    }
    if (position == 0) {
        deleteFront(L);
    } else if (position == L->count - 1) {
        deleteRear(L);
    } else {
        struct node *temp = L->head;
        for (int i = 0; i < position - 1; i++) {
            temp = temp->next;
        }
        struct node *to_delete = temp->next;
        temp->next = to_delete->next;
        free(to_delete);
        L->count--;
    }
}
/*
================================================================================
FUNCTION    : displayAll
DESCRIPTION : Displays the List
ARGUMENTS   : struct list *L,- The pointer to the structure of the list
RETURNS     : void - returns nothing
================================================================================
*/
void displayAll(struct list *L) {
    struct node *temp = L->head;
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

OUTPUTS:


C:\Users\GAMEMASTER\Desktop\LINKEDLIST\bin\Debug\LINKEDLIST.exe
INITIAL DATA= 25, 30, 35
25 -> 30 -> 35 -> NULL
2 -> 25 -> 30 -> 35 -> NULL
1 -> 2 -> 25 -> 30 -> 35 -> NULL
0 -> 1 -> 2 -> 25 -> 30 -> 35 -> NULL
0 -> 1 -> 2 -> 25 -> 30 -> 35 -> 3 -> NULL
0 -> 1 -> 2 -> 25 -> 30 -> 35 -> 3 -> 5 -> NULL
0 -> 1 -> 2 -> 25 -> 4 -> 30 -> 35 -> 3 -> 5 -> NULL
1 -> 2 -> 25 -> 4 -> 30 -> 35 -> 3 -> 5 -> NULL
1 -> 2 -> 25 -> 4 -> 30 -> 35 -> 3 -> NULL
1 -> 2 -> 88 -> 25 -> 4 -> 30 -> 35 -> 3 -> NULL
1 -> 2 -> 25 -> 4 -> 30 -> 35 -> 3 -> NULL
Process returned 0 (0x0) execution time : 0.059 s
Press any key to continue.