Q. Create a doubly connected linked list in C. Write a program to perform insertion at the beginning, at an intermediate position, at the end, and deletion. Your program should also have a display function that should display the content of your list after each operation.

Ans.

#include<stdio.h>

#include<stdlib.h>

struct Node{

    int val;

    struct Node\* next;

    struct Node\* prev;

};

struct Node\* add\_start(struct Node\* head, int value){

    if(head == NULL){

        head = (struct Node\*)malloc(sizeof(struct Node));

        if(head == NULL){

            printf("Memory ran out");

        }

        else{

            head->next = NULL;

            head->prev = NULL;

            head->val = value;

        }

        return head;

    }

    else{

        struct Node\* new\_head = (struct Node\*)malloc(sizeof(struct Node));

        if(new\_head == NULL){

            printf("Memory ran out");

        }

        else{

            new\_head->next = head;

            new\_head->prev = NULL;

            new\_head->val = value;

            head->prev = new\_head;

        }

        return new\_head;

    }

}

struct Node\* add\_end(struct Node\* head, int value){

    if(head == NULL){

        head = (struct Node\*)malloc(sizeof(struct Node));

        if(head == NULL){

            printf("Memory ran out");

        }

        else{

            head->next = NULL;

            head->prev = NULL;

            head->val = value;

        }

        return head;

    }

    else{

        struct Node\* store\_head = head;

        struct Node\* store = (struct Node\*)malloc(sizeof(struct Node));

        if(store == NULL){

            printf("Memory ran out");

            return store\_head;

        }

        else{

            while(head->next != NULL){

                head = head->next;

            }

            store->val = value;

            store->next = NULL;

            store->prev = head;

            head->next = store;

        }

        return store\_head;

    }

}

struct Node\* add\_intermediate(struct Node\* head, int pos, int value){

    if(head == NULL){

        head = (struct Node\*)malloc(sizeof(struct Node));

        if(head == NULL){

            printf("Memory ran out");

        }

        else{

            head->next = NULL;

            head->prev = NULL;

            head->val = value;

        }

        return head;

    }

    else{

        struct Node\* store\_head = head;

        struct Node\* store = (struct Node\*)malloc(sizeof(struct Node));

        if(store == NULL){

            printf("Memory ran out");

            return store\_head;

        }

        else{

            store->val = value;

            if (pos <= 1){

                store->next = head;

                store->prev = NULL;

                head->prev = store;

                return store;

            }

            else{

                while((head->next != NULL) && (pos-1>1)){

                    head = head->next;

                    pos--;

                }

                store->next = head->next;

                store->prev = head;

                if (head->next!= NULL){

                    (head->next)->prev = store;

                }

                head->next = store;

            }

        }

        return store\_head;

    }

}

struct Node\* node\_remove(struct Node\* head, int value){

    if(head != NULL && head->val == value){

        printf("\nNode found and deleted\n");

        struct Node\* store = head;

        if (head->next != NULL){

            (head->next)->prev = NULL;

        }

        head = head->next;

        store->next = NULL;

        free(store);

        return head;

    }

    else if (head!= NULL){

        struct Node\* store\_head = head;

        struct Node\* initial = head;

        head = head->next;

        while (head!= NULL)

        {

            if(head->val == value){

                initial->next = head->next;

                if (head->next != NULL){

                    (head->next)->prev = initial;

                    head->next = NULL;

                }

                head->prev = NULL;

                free(head);

                printf("\nNode found and deleted!\n");

                return store\_head;

            }

            initial = head;

            head = head->next;

        }

        printf("\nNode was not found!\n");

        return store\_head;

    }

    else{

        printf("\nList is empty!\n");

        return head;

    }

}

void display(struct Node\* head){

    if(head == NULL){

        printf("\nList is empty!\n");

    }

    else{

        printf("\nLinked List: ");

        while(head != NULL){

            printf("%d ", head->val);

            head = head->next;

        }

        printf("\n");

    }

}

int main(){

    struct Node\* head = NULL;

    char ch;

    int v, pos;

    do

    {

        printf("\n1. Add at the beginning of Linked List: ");

        printf("\n2. Add at the end of Linked List: ");

        printf("\n3. Add in an intermediate position of Linked List: ");

        printf("\n4. Display the Linked List: ");

        printf("\n5. Delete an element from the Linked List: ");

        printf("\n6. Exit");

        printf("\nEnter your choice(1-6): ");

        fflush(stdin);

        scanf("%c", &ch);

        switch(ch)

        {

        case '1':

            printf("\nEnter value: ");

            fflush(stdin);

            scanf("%d", &v);

            head = add\_start(head, v);

            display(head);

            break;

        case '2':

            printf("\nEnter value: ");

            fflush(stdin);

            scanf("%d", &v);

            head = add\_end(head, v);

            display(head);

            break;

        case '3':

            printf("\nEnter position from the start: ");

            fflush(stdin);

            scanf("%d", &pos);

            printf("Enter value: ");

            fflush(stdin);

            scanf("%d", &v);

            head = add\_intermediate(head, pos, v);

            display(head);

            break;

        case '4':

            display(head);

            break;

        case '5':

            printf("\nEnter value to delete: ");

            fflush(stdin);

            scanf("%d", &v);

            head = node\_remove(head, v);

            display(head);

            break;

        case '6':

            printf("\nPROGRAM TERMINATED!!");

            break;

        default :

            printf("\nWrong Entry!!\n");

            break;

        }

    }while(ch!='6');

    return EXIT\_SUCCESS;

}

Output



