ASSIGNMENT IX:

Doubly Linked List Implementation

*U19CS012 [D-12]*

Implement the following operations in context to singly linked list:

1) Creation & Display of Doubly Linked List

2) Insertion (at beginning, middle and end)

3) Deletion (from beginning, middle and end)

4.) Search for a Specific Element

5.) Find the Maximum and Minimum Element in Doubly Linked List

Code:

*#include* <stdio.h>

*// For Exit Function*

*#include* <stdlib.h>

*// Structure for Each Node*

struct node

{

    struct node \*prev;

    int data;

    struct node \*next;

};

*// head Pointer -> head of Linked List*

struct node \*head = NULL;

*//Helper Functions*

*// 1 -> Creation of Linked List*

*// Creation of the Doubly Linked List*

void CREATION\_DLL();

*// Display of the Whole Doubly Linked List*

void DISPLAY\_DLL();

*// Returns the Length of Doubly Linked List*

int LENGTH\_DLL();

*// 2 -> Insertion in Linked List*

*// Insert at the Beginning of Linked List*

void Insert\_Begin();

*// Insert at the End of Linked List*

void Insert\_End();

*// Insert in the Middle Of the Linked List*

void Insert\_Middle();

*// 3 -> Deletion in the Linked List*

*// Delete at the Beginning of Linked List*

void Delete\_Begin();

*// Delete at the End of Linked List*

void Delete\_End();

*// Delete in the Middle Of the Linked List*

void Delete\_Middle();

*//Deletes Node at Particular Position*

void Delete\_Position();

*// Deletes all Nodes with Particular Value*

void Delete\_Value();

*// 4 -> Search in the Doubly Linked List*

void Search();

*// 5 -> Find the Max and Min in Doubly Linked List*

void Max\_Min();

void main()

{

    printf("\nDOUBLY LINKED LIST\n");

    printf(" 1 -> Create a Doubly Linked List\n");

    printf(" 2 -> Display the Doubly Linked List\n");

    printf(" 3 -> Insert at the Beginning of Doubly Linked List\n");

    printf(" 4 -> Insert at the End of Doubly Linked List\n");

    printf(" 5 -> Insert at Middle of Doubly Linked List\n");

    printf(" 6 -> Delete from Beginning of Doubly Linked List\n");

    printf(" 7 -> Delete from the End of Doubly Linked List\n");

    printf(" 8 -> Delete at Middle of Doubly Linked List\n");

    printf(" 9 -> Search in Doubly Linked List\n");

    printf(" 10 -> Find Maximum and Minimum in Doubly Linked List\n");

    printf(" 11 -> Exit\n");

    int choice;

*while* (1)

    {

        printf("Enter your choice : ");

        scanf("%d", &choice);

*switch* (choice)

        {

*case* 1:

            CREATION\_DLL();

*break*;

*case* 2:

            DISPLAY\_DLL();

*break*;

*case* 3:

            Insert\_Begin();

*break*;

*case* 4:

            Insert\_End();

*break*;

*case* 5:

*// Insert at Middle of DlL*

            Insert\_Middle();

*break*;

*case* 6:

            Delete\_Begin();

*break*;

*case* 7:

            Delete\_End();

*break*;

*case* 8:

*// Delete at Middle of Dll*

            Delete\_Middle();

*break*;

*case* 9:

            Search();

*break*;

*case* 10:

            Max\_Min();

*break*;

*case* 11:

            exit(0);

*break*;

*default*:

            printf("Enter a Valid Choice!");

*break*;

        }

    }

}

*// Creation of the Doubly Linked List*

void CREATION\_DLL()

{

    struct node \*ptr;

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

    }

*else*

    {

        int Element;

        printf("Enter Value of Node : ");

        scanf("%d", &Element);

*if* (head == NULL)

        {

*// List is Empty*

            ptr->next = NULL;

            ptr->prev = NULL;

            ptr->data = Element;

*// Point the new Node to head*

            head = ptr;

        }

*else*

        {

*// Insert at Beginning by Default*

            ptr->data = Element;

            ptr->prev = NULL;

            ptr->next = head;

            head->prev = ptr;

            head = ptr;

        }

        printf("Node Inserted Successfully!\n");

    }

}

*// Display of the Whole Doubly Linked List*

void DISPLAY\_DLL()

{

    struct node \*ptr;

    printf("DOUBLY LINKED LIST : \n");

    ptr = head;

*if* (ptr == NULL)

    {

        printf("List is Empty! List has No Nodes!\n");

*return*;

    }

*else*

    {

        printf("NULL <=> ");

*while* (ptr != NULL)

        {

            printf("%d <=> ", ptr->data);

            ptr = ptr->next;

        }

        printf("NULL\n");

    }

}

*// Display the Length of Linked List*

int LENGTH\_DLL()

{

    struct node \*ptr;

    ptr = head;

*if* (ptr == NULL)

    {

*// Empty List*

*return* 0;

    }

*else*

    {

        int cnt = 0;

*// ptr is Pointing to Head*

*while* (ptr != NULL)

        {

            cnt++;

            ptr = ptr->next;

        }

*return* cnt;

    }

}

*// Insert at the Beginning of Linked List*

void Insert\_Begin()

{

    struct node \*ptr;

    int Element;

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

    }

*else*

    {

        printf("Enter Value of Node : ");

        scanf("%d", &Element);

*// If the Linked List is Empty*

*if* (head == NULL)

        {

            ptr->next = NULL;

            ptr->prev = NULL;

            ptr->data = Element;

            head = ptr;

        }

*else*

        {

*// If the Linked List is Not Empty*

*// Insert at Beginning*

            ptr->data = Element;

            ptr->prev = NULL;

            ptr->next = head;

            head->prev = ptr;

            head = ptr;

        }

        printf("Node Inserted Succesfully!\n");

    }

}

*// Insert at the End of Linked List*

void Insert\_End()

{

    struct node \*ptr, \*temp;

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

    }

*else*

    {

        int Element;

        printf("Enter Value of Node : ");

        scanf("%d", &Element);

        ptr->data = Element;

*if* (head == NULL)

        {

*// If List is Empty*

            ptr->next = NULL;

            ptr->prev = NULL;

            head = ptr;

        }

*else*

        {

*// Traverse till End of List*

            temp = head;

*while* (temp->next != NULL)

            {

                temp = temp->next;

            }

*// Link Management*

            temp->next = ptr;

            ptr->prev = temp;

            ptr->next = NULL;

        }

    }

    printf("Node Inserted Succesfully!\n");

}

*// Insert in the Middle Of the Linked List*

void Insert\_Middle()

{

    struct node \*ptr;

    int pos;

    struct node \*temp;

    int Element, i;

    ptr = (struct node \*)malloc(sizeof(struct node));

*if* (ptr == NULL)

    {

        printf("No Memory Space on Device!\n");

    }

*else*

    {

        temp = (struct node \*)malloc(sizeof(struct node));

        printf("Enter the Position for the New Node to be Inserted : ");

        scanf("%d", &pos);

*// pos = 1 -> Insertion at Beginning of LL*

*// pos = len + 1 -> Insertion at Ending of LL*

*// Length of the Linked List*

        int len = LENGTH\_DLL();

*if* (pos <= 0 || pos > len + 1)

        {

            printf("Enter Valid Postion for Insertion!\n");

*return*;

        }

*// Creation of New Node { NULL<-[?]->NULL }*

        printf("Enter the Data to be stored in Node : ");

        scanf("%d", &Element);

        ptr->data = Element;

*if* (pos == 1)

        {

*// At the Beginning of Linked List*

            ptr->prev = NULL;

            ptr->next = head;

            head->prev = ptr;

            head = ptr;

        }

*else* *if* (pos == len + 1)

        {

*// Traverse till End of List*

            temp = head;

*while* (temp->next != NULL)

            {

                temp = temp->next;

            }

*// Link Management*

            temp->next = ptr;

            ptr->prev = temp;

            ptr->next = NULL;

        }

*else*

        {

*for* (i = 1, temp = head; i < pos - 1; i++)

            {

                temp = temp->next;

            }

*// Link Management*

*// [temp] -> [temp->next]*

            ptr->next = temp->next;

            ptr->prev = temp;

            temp->next = ptr;

            (temp->next)->prev = ptr;

        }

        printf("Node Inserted Succesfully!\n");

    }

}

*// Delete at the Beginning of Linked List*

void Delete\_Begin()

{

    struct node \*ptr;

*if* (head == NULL)

    {

        printf("List is Empty! UnderFlow Condition!\n");

    }

*else* *if* (head->next == NULL)

    {

*// Only One Element in the Linked List*

        head = NULL;

*// Release the Memory*

        free(head);

        printf("Node Deleted Succesfully!\n");

    }

*else*

    {

*// To Hold Address of Node to be Deleted*

        ptr = head;

*// Delete Front Logic*

        head = head->next;

        head->prev = NULL;

*// Release the Memory*

        free(ptr);

        printf("Node Deleted Successfully!\n");

    }

}

*// Delete at the End of Linked List*

void Delete\_End()

{

    struct node \*ptr;

*if* (head == NULL)

    {

        printf("List is Empty! UnderFlow Condition!\n");

*return*;

    }

*else* *if* (head->next == NULL)

    {

*// Only One Element in the Linked List*

        head = NULL;

*// Release the Memory*

        free(head);

        printf("Node Deleted Successfully!\n");

*return*;

    }

*else*

    {

*if* (ptr == NULL)

        {

            printf("No Memory Space on Device!\n");

*return*;

        }

        ptr = head;

*while* (ptr->next != NULL)

        {

            ptr = ptr->next;

        }

        (ptr->prev)->next = ptr->next;

        free(ptr);

        printf("Node Deleted Successfully!\n");

    }

}

*// Delete in the Middle Of the Linked List*

void Delete\_Middle()

{

*if* (head == NULL)

    {

        printf("List is Empty! No Deletion Possible!!\n");

*return*;

    }

*else*

    {

        int ch = 0;

        printf("Delete A Node By : \n");

        printf(" 1 -> Position\n");

        printf(" 2 -> Value\n");

        printf("Enter Your Choice : ");

        scanf("%d", &ch);

*switch* (ch)

        {

*case* 1:

            Delete\_Position();

*break*;

*case* 2:

            Delete\_Value();

*break*;

*default*:

            printf("Enter a Valid Choice!\n");

*break*;

        }

    }

}

*//Deletes Node at Particular Position*

void Delete\_Position()

{

    int i, pos;

    struct node \*temp, \*ptr;

    printf("Enter the Position of the Node to be Deleted : ");

    scanf("%d", &pos);

*// pos = 1 -> Deletion at Beginning of LL*

*// pos = len -> Deletion at Ending of LL*

*// Length of the Linked Lst*

    int len = LENGTH\_DLL();

*if* (pos <= 0 || pos > len)

    {

        printf("Enter Valid Postion for Deletion!\n");

*return*;

    }

*if* (pos == 1)

    {

        Delete\_Begin();

    }

*else* *if* (pos == len)

    {

        Delete\_End();

    }

*else*

    {

        ptr = head;

*for* (i = 1; i < pos; i++)

        {

            ptr = ptr->next;

        }

*// point the prev of {element to be deleted} to "next of deleted"*

*// []     []     []*

*//       ptr*

        printf("ptr data : %d\n", ptr->data);

        temp = ptr->prev;

        temp->next = ptr->next;

        (ptr->next)->prev = temp;

        printf("Node Deleted Successfully!\n");

        free(ptr);

    }

}

*// Deletes all Nodes with Particular Value*

void Delete\_Value()

{

    int value;

    struct node \*temp, \*ptr;

    printf("Enter the Value of the Node to be Deleted : ");

    scanf("%d", &value);

    int flag = 0;

*if* (head == NULL)

    {

        printf("List is Empty!No Deletions Possible\n");

*return*;

    }

*else*

    {

*// Head Pointer*

        ptr = head;

*while* (ptr != NULL)

        {

*// If the Value of Node = Value of Node to be Deleted*

*if* (ptr->data == value)

            {

*if* (ptr == head)

                {

                    Delete\_Begin();

                    flag = 1;

                }

*else*

                {

*// Element to be deleted is Not the Last Node*

*if* (ptr->next != NULL)

                    {

                        (ptr->prev)->next = ptr->next;

                        (ptr->next)->prev = ptr->prev;

                        printf("Node Deleted Successfully!\n");

                        free(ptr);

                    }

*else*

                    {

                        Delete\_End();

                    }

                    flag = 1;

                }

            }

*// ptr now points to next node*

            ptr = ptr->next;

        }

*if* (flag == 0)

        {

            printf("Node with Given Value Does Not Exist! OR Deleted Earlier!\n");

        }

*else*

        {

            printf("Node with Given Value Found and Deleted Successfully!\n");

        }

    }

}

*// 4 -> Search in the Doubly Linked List*

void Search()

{

*// Temp Pointer for Traversal*

    struct node \*ptr;

    ptr = head;

*if* (ptr == NULL)

    {

        printf("List is Empty! Search Can't Be Performed!\n");

    }

*else*

    {

        int ele, pos = 0;

        int flag = 1;

        printf("Enter Element to be Searched in List : ");

        scanf("%d", &ele);

*while* (ptr != NULL)

        {

*if* (ptr->data == ele)

            {

                printf("Element Found at Position %d !!\n", pos + 1);

                flag = 0;

*break*;

            }

            pos++;

            ptr = ptr->next;

        }

*if* (flag == 1)

        {

            printf("Element Not Found !!\n");

        }

    }

}

*// 5 -> Find the Max and Min in Doubly Linked List*

void Max\_Min()

{

    struct node \*ptr;

    ptr = head;

*if* (ptr == NULL)

    {

        printf("List is Empty!Can't Find Max/Min Element!\n");

*return*;

    }

*else*

    {

        int mnn = ptr->data;

        int mxn = ptr->data;

*while* (ptr != NULL)

        {

*if* (ptr->data > mxn)

            {

                mxn = ptr->data;

            }

*else* *if* (ptr->data < mnn)

            {

                mnn = ptr->data;

            }

            ptr = ptr->next;

        }

        printf("Minimum Element in Doubly Linked List : %d\n", mnn);

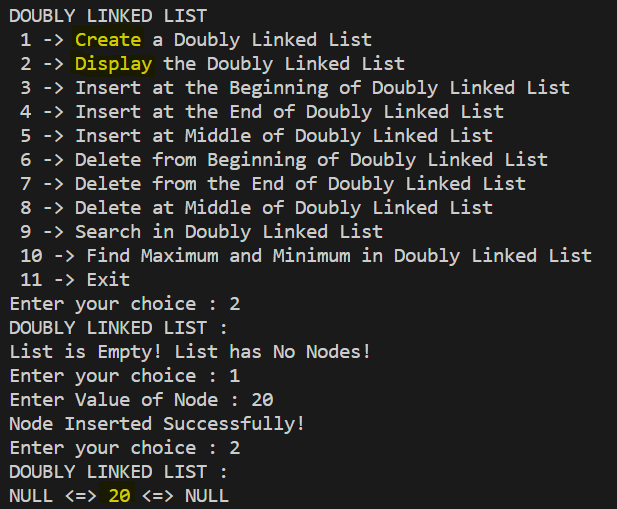
        printf("Maximum Element in Doubly Linked List : %d\n", mxn);

    }

}

Test Cases:

A.) Creation of Linked List

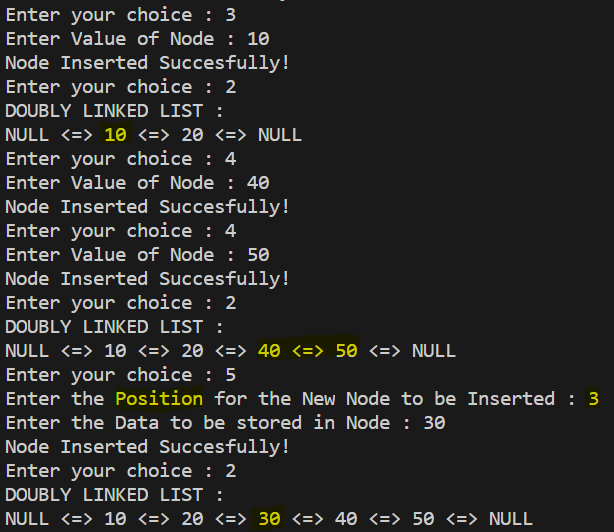


B.) Insertion of Linked List

1.) Insert 10 at Front of Linked List

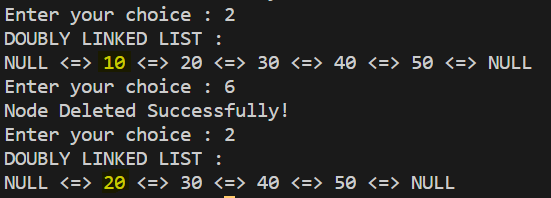
2.) Insert 40 & 50 at End of Linked List

3.) Insert 30 at Middle of Linked List at Position 3

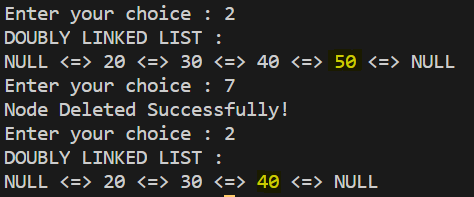


C.) Deletion of Linked List

1.) Delete from Beginning of Linked List



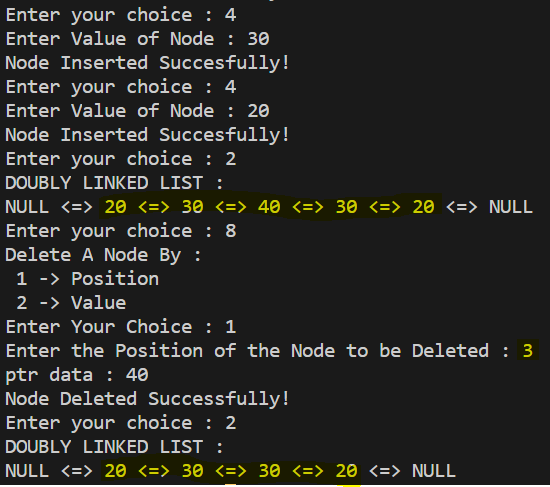
2.) Delete from End of Linked List



3.) Delete from Middle of Linked List

{Lets Add Some Extra Nodes at End of Linked List: 30 & 20}

A.) Delete by Position {we have Deleted 3rd Position i.e. 40}



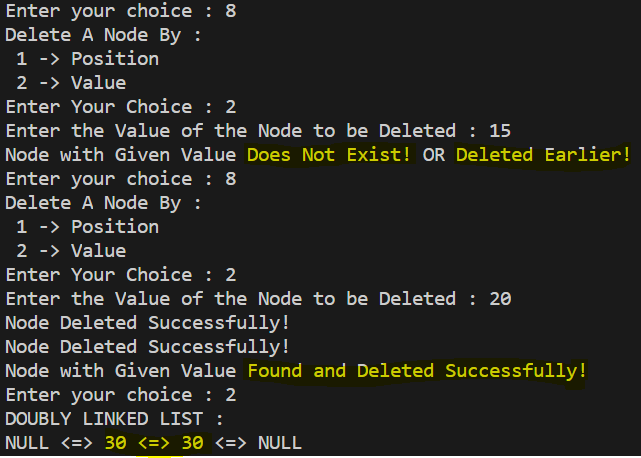
B.) Delete by Value

1.) Deletion of Value that is Not in Linked List {i.e. 15}

{Node with Given Value Does Not Exist! OR Deleted Earlier!}

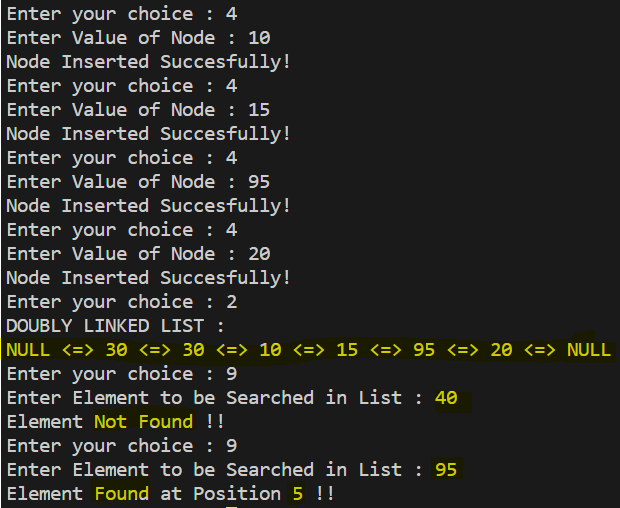
2.) Deletion of Value that is in Linked List {i.e. 20}

{Node with Given Value Found and Deleted Successfully!}



D.) Search in Linked List

{Lets Add Some Extra Nodes at End of Linked List: 10,15,95 & 20 & Search for 40 [Not Found Case] & 95 [Recently Added Element]}



E.) Maximum and Minimum of Linked List

{We can clearly observe from List that **95** is the Highest and **10** is the Lowest}

