**Week 5**

**5. C Program to implement Doubly Linked List and its Operations**

#include <stdio.h>

#include <stdlib.h>

struct **Node**

{

int data;

struct Node\* prev;

struct Node\* next;

};

struct Node\* head = NULL;

*// Create a new node*

struct Node\* **createNode**(int value)

{

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

newNode->data = value;

newNode->prev = NULL;

newNode->next = NULL;

return newNode;

}

*// Insert at beginning*

void **insertBegin**(int value)

{

struct Node\* newNode = createNode(value);

if (head == NULL)

{

head = newNode;

} else {

newNode->next = head;

head->prev = newNode;

head = newNode;

}

printf("Inserted %d at Beginning\n", value);

}

*// Insert at end*

void **insertEnd**(int value)

{

struct Node\* newNode = createNode(value);

if (head == NULL)

{

head = newNode;

return;

}

struct Node\* temp = head;

while (temp->next != NULL)

temp = temp->next;

temp->next = newNode;

newNode->prev = temp;

printf("Inserted %d at End\n", value);

}

*// Insert at specific position (1-based index)*

void **insertAtPosition**(int pos, int value)

{

if (pos <= 1)

{

insertBegin(value);

return;

}

struct Node\* temp = head;

int count = 1;

while (temp != NULL && count < pos - 1)

{

temp = temp->next;

count++;

}

if (temp == NULL)

{

insertEnd(value);

return;

}

struct Node\* newNode = createNode(value);

newNode->next = temp->next;

newNode->prev = temp;

if (temp->next != NULL)

temp->next->prev = newNode;

temp->next = newNode;

printf("Inserted %d at position %d\n", value, pos);

}

*// Delete from beginning*

void **deleteBegin**()

{

if (head == NULL)

{

printf("List is empty\n");

return;

}

struct Node\* temp = head;

head = head->next;

if (head != NULL)

head->prev = NULL;

printf("Deleted %d from beginning\n", temp->data);

free(temp);

}

*// Delete from end*

void **deleteEnd**()

{

if (head == NULL)

{

printf("List is empty\n");

return;

}

struct Node\* temp = head;

if (temp->next == NULL)

{

head = NULL;

} else {

while (temp->next != NULL)

temp = temp->next;

temp->prev->next = NULL;

}

printf("Deleted %d from end\n", temp->data);

free(temp);

}

*// Delete a node with a given value*

void **deleteValue**(int value)

{

struct Node\* temp = head;

while (temp != NULL && temp->data != value)

temp = temp->next;

if (temp == NULL)

{

printf("Value not found\n");

return;

}

if (temp->prev != NULL)

temp->prev->next = temp->next;

else

head = temp->next; // head node

if (temp->next != NULL)

temp->next->prev = temp->prev;

printf("Deleted node with value %d\n", temp->data);

free(temp);

}

*// Display forward*

void **displayForward**()

{

struct Node\* temp = head;

if (temp == NULL)

{

printf("List is empty\n");

return;

}

printf("List (forward): ");

while (temp != NULL)

{

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

temp = temp->next;

}

printf("NULL\n");

}

*// Display backward*

void **displayBackward**()

{

struct Node\* temp = head;

if (temp == NULL)

{

printf("List is empty\n");

return;

}

while (temp->next != NULL)

temp = temp->next;

printf("List (backward): ");

while (temp != NULL)

{

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

temp = temp->prev;

}

printf("NULL\n");

}

*// Main menu*

int **main**()

{

int choice, value, position;

while (1)

{

printf("\n--- Doubly Linked List Menu ---\n");

printf("1. Insert at Beginning\n2. Insert at End\n3. Insert at Position\n");

printf("4. Delete from Beginning\n5. Delete from End\n6. Delete by Value\n");

printf("7. Display Forward\n8. Display Backward\n9. Exit\n");

printf("Enter Ur Choice: ");

scanf("%d", &choice);

switch (choice)

{

case 1:

printf("Enter Value: ");

scanf("%d", &value);

insertBegin(value);

break;

case 2:

printf("Enter Value: ");

scanf("%d", &value);

insertEnd(value);

break;

case 3:

printf("Enter Position and value: ");

scanf("%d%d", &position, &value);

insertAtPosition(position, value);

break;

case 4:

deleteBegin();

break;

case 5:

deleteEnd();

break;

case 6:

printf("Enter value to Delete: ");

scanf("%d", &value);

deleteValue(value);

break;

case 7:

displayForward();

break;

case 8:

displayBackward();

break;

case 9:

printf("Exit from Program\n");

exit(0);

default:

printf("Invalid Choice\n");

}

}

return 0;

}

**Output:**











