**Q2. Write function prototypes for the following operations:**

**• Create a new node**

**• Insert at beginning**

**• Insert at end**

**• Insert after a specific position**

**• Delete from beginning**

**• Delete from end**

**• Delete a specific node**

**• Search an element**

**• Display the list**

#include <stddef.h>

#include <stdio.h>

#include <stdlib.h>

// declare struct about Node

struct Node {

int data;

struct Node \*next;

};

struct Node \*head = NULL;

struct Node \*CreateNode(int value);

// function declaration

void insertAtbeg(int data);

void insertAtend(int data);

// insert at position

void insertAtbefore(int data, int before);

void insertAtafter(int data, int after);

void insertAtpos(int data, int position);

// delete list

void DeleteFromBeg();

void DeleteFromEnd();

void DeleteAfterNode(int after);

void DeleteByVal(int value);

// display list

void Search(int value);

void Display();

// main

int main() {

// variable declare

int choice = 0, value, key;

while (choice != 12) {

printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\* MAIN MENU (C) \*\*\*\*\*\*\*\*\*\*\*\*");

printf("\n1 : Add a Node at Beginning");

printf("\n2 : Add a Node at the End");

printf("\n3 : Add a Node after a Node");

printf("\n4 : Add a Node before a Node");

printf("\n5 : Add a Node at position");

printf("\n6 : Delete a Node from the Beginning");

printf("\n7 : Delete a Node from the End");

printf("\n8 : Delete a Node after a Node");

printf("\n9 : Delete a Node by value");

printf("\n10: Search an element");

printf("\n11: Display the Linked List");

printf("\n12: END the Operation !!!");

printf("\n\nEnter your choice : ");

scanf("%d", &choice);

switch (choice) {

case 1: // calling insertAtBeg(value) function to add a Node in the

// Beginning of the LL

printf("Enter Value to Add : ");

scanf("%d", &value);

insertAtbeg(value);

break;

case 2: // calling insertAtEnd(value) function to add a Node in the End of

// the LL

printf("Enter Value to Add : ");

scanf("%d", &value);

insertAtend(value);

break;

case 3: // calling insertAfterNode(value,key) function to add a Node after a

// Node in the LL

printf("Enter Value to Add : ");

scanf("%d", &value);

printf("Enter the Value of Node coming before the new Node : ");

scanf("%d", &key);

insertAtafter(value, key);

break;

case 4: // calling insertBeforeNode(value,key) function to add a Node before

// a Node in the LL

printf("Enter Value to Add : ");

scanf("%d", &value);

printf("Enter the Value of Node coming after the new Node : ");

scanf("%d", &key);

insertAtbefore(value, key);

break;

case 5: // calling insertAtPos(value,key) function to add a Node on given

// postion in the LL

printf("Enter Value to Add : ");

scanf("%d", &value);

printf("Enter the postion at-which it is to be added : ");

scanf("%d", &key);

insertAtpos(value, key);

break;

case 6: // calling DeleteFromBeg() function to delete a Node from the

// Beginning of the LL

DeleteFromBeg();

break;

case 7: // calling DeleteFromEnd() function to delete a Node from the End of

// the LL

DeleteFromEnd();

break;

case 8: // calling DeleteAfterNode(key) function to delete a Node after a

// given Node

printf("Enter the value of Node coming before the Node : ");

scanf("%d", &key);

DeleteAfterNode(key);

break;

case 9: // calling DeleteByVal(value) function to delete a Node by its value

printf("Enter the value of Node you want to delete : ");

scanf("%d", &value);

DeleteByVal(value);

break;

case 10: // calling Search(value) function to search for an element

printf("Enter the value you want to search : ");

scanf("%d", &value);

Search(value);

break;

case 11: // calling Display() function to Display the LL

Display();

break;

case 12: // exiting from the program

printf("\nExit !!!");

break;

default:

printf("\nInvalid option. Try again.");

break;

}

}

return 0;

}

// newNode

struct Node \*CreateNode(int value) {

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

newNode->data = value;

newNode->next = NULL;

return newNode;

};

// insert at beg

void insertAtbeg(int data) {

struct Node \*new\_Node = CreateNode(data);

new\_Node->next = head;

head = new\_Node;

printf("\nNode Added at the beginning.");

}

// insert at end

void insertAtend(int data) {

if (head == NULL) {

insertAtbeg(data);

printf("\nNode Added at the beginning.");

return;

}

struct Node \*new\_Node = CreateNode(data);

struct Node \*ptr = head;

while (ptr != NULL) {

ptr = ptr->next;

};

ptr->next = new\_Node;

printf("\nNode Added at the end.");

}

// insert at position

void insertAtbefore(int data, int before) {

struct Node \*new\_Node = CreateNode(data);

if (head == NULL) {

insertAtbeg(data);

return;

} else if (head->data == before) {

new\_Node->next = head;

head = new\_Node;

} else {

struct Node \*ptr = head;

while (ptr->next != NULL && ptr->next->data != before) {

ptr = ptr->next;

}

new\_Node->next = ptr->next;

ptr->next = new\_Node;

printf("\nNode Added before the Node having value %d.", before);

}

}

void insertAtafter(int data, int after) {

if (head == NULL) {

insertAtbeg(data);

return;

}

struct Node \*new\_Node = CreateNode(data);

struct Node \*ptr = head;

while (ptr != NULL && ptr->data != after) {

ptr = ptr->next;

}

new\_Node->next = ptr->next;

ptr->next = new\_Node;

printf("\nNode Added after the Node having value %d.", after);

}

void insertAtpos(int data, int position) {

if (head == NULL && position == 0) {

insertAtbeg(data);

return;

}

struct Node \*new\_Node = CreateNode(data);

struct Node \*ptr = head;

for (int i = 0; i < position; i++) {

if (ptr == NULL) {

printf("\nInvalid Position !!");

return;

}

ptr = ptr->next;

}

new\_Node->next = ptr->next;

ptr->next = new\_Node;

printf("\nNode Added at position %d.", position);

}

// delete list

// function to delete a Node from the beginning of the LL

void DeleteFromBeg() {

if (head == NULL) {

printf("\nEmpty List !!!");

return;

}

struct Node \*ptr = head;

head = head->next;

free(ptr);

printf("\nNode Deleted from the Beginning.");

};

// function to delete a Node from the end of the LL

void DeleteFromEnd() {

if (head == NULL) {

printf("\nEmpty List !!!");

return;

}

struct Node \*ptr = head;

while (ptr->next->next != NULL) {

ptr = ptr->next;

}

free(ptr->next);

ptr->next = NULL;

printf("\nNode Deleted from the End.");

};

// function to delete a Node after a given Node in the LL

void DeleteAfterNode(int after) {

struct Node \*ptr = head;

while (ptr != NULL && ptr->data != after) {

ptr = ptr->next;

}

if (ptr == NULL || ptr->next == NULL) {

printf("\nValue not found !!\nTry again !!!");

return;

}

struct Node \*temp = ptr->next;

ptr->next = temp->next;

free(temp);

printf("\nNode deleted after the Node having value %d.", after);

};

void DeleteByVal(int value) {

if (head == NULL) {

printf("\nEmpty List !!!");

return;

}

struct Node \*ptr = head;

while (ptr->next != NULL && ptr->next->data != value) {

ptr = ptr->next;

}

if (ptr->next == NULL) {

printf("\nNode with value %d not found in the LL !! ", value);

return;

}

struct Node \*temp = ptr->next;

ptr->next = temp->next;

free(temp);

printf("\nNode with value %d deleted from the LL.", value);

};

// display list

void Search(int value) {

struct Node \*ptr = head;

int pos = 1;

while (ptr != NULL) {

if (ptr->data == value) {

printf("\n%d found at Node having position %d\n", ptr->data, pos);

return;

}

ptr = ptr->next;

pos++;

}

printf("\nNode with value %d not found in the LL !\n", value);

}; // function to Search an element from the LL

void Display() {

int count = 0; // counting total Nodes

struct Node \*ptr = head;

if (ptr == NULL) {

printf("List is empty.\n");

return;

}

printf("\nLinked List: ");

while (ptr != NULL) {

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

count++; // increasing count by one

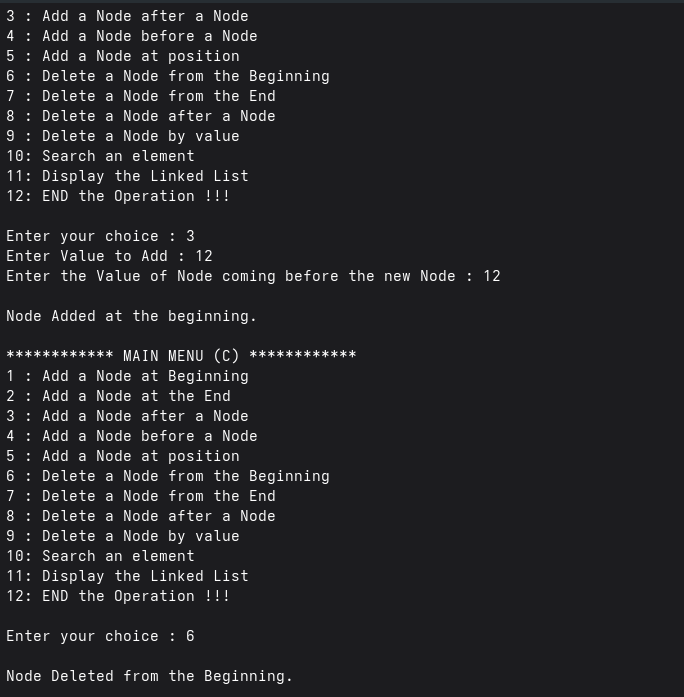
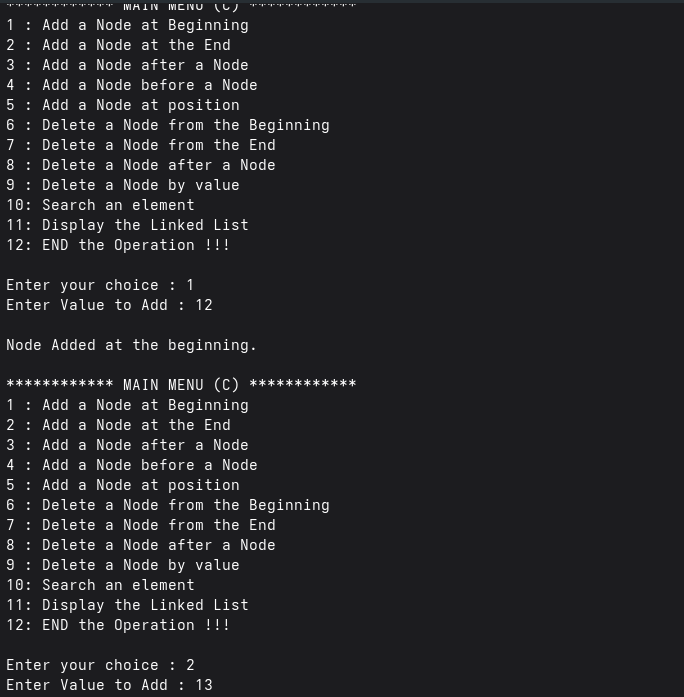
ptr = ptr->next;

}

printf("END\n");

printf("Total number of Nodes : %d", count);

}; // function to display the LL





**Create a program for the implementation of a Singly Linked List with all operations in C++.**

**#include <iostream> // header file**

**using namespace std;**

**class Node // class for the node structure**

**{**

**public:**

**int data;**

**Node \*next;**

**Node(int value) {**

**data = value;**

**next = nullptr;**

**}**

**};**

**class LinkedList // class for the LL functions**

**{**

**private:**

**Node \*head; // head pointer**

**public:**

**LinkedList() { head = nullptr; }**

**~LinkedList() // Free memory when program ends**

**{**

**Node \*current = head;**

**while (current != nullptr) {**

**Node \*temp = current;**

**current = current->next;**

**delete temp;**

**}**

**}**

**void**

**InsertAtBeg(int value) // function to add a node at the beginning of the LL**

**{**

**Node \*newNode = new Node(value);**

**newNode->next = head;**

**head = newNode;**

**cout << "\nNode added at the beginning.";**

**}**

**void InsertAtEnd(int value) // function to add a node at the end of the LL**

**{**

**if (head == nullptr) {**

**InsertAtBeg(value);**

**return;**

**}**

**Node \*newNode = new Node(value);**

**Node \*ptr = head;**

**while (ptr->next != nullptr) {**

**ptr = ptr->next;**

**}**

**ptr->next = newNode;**

**cout << "\nNode added at the end.";**

**}**

**void InsertAfterNode(**

**int value,**

**int after) // function to add a node after a given node in the LL**

**{**

**if (head == nullptr) {**

**InsertAtBeg(value);**

**return;**

**}**

**Node \*ptr = head;**

**while (ptr != nullptr && ptr->data != after) {**

**ptr = ptr->next;**

**}**

**if (ptr == nullptr) {**

**cout << "\nNode with value " << after << " not found!";**

**return;**

**}**

**Node \*newNode = new Node(value);**

**newNode->next = ptr->next;**

**ptr->next = newNode;**

**cout << "\nNode added after " << after << ".";**

**}**

**void InsertBeforeNode(**

**int value,**

**int before) // function to add a node before a given node in the LL**

**{**

**if (head == nullptr) {**

**InsertAtBeg(value);**

**return;**

**}**

**if (head->data == before) {**

**InsertAtBeg(value);**

**return;**

**}**

**Node \*ptr = head;**

**while (ptr->next != nullptr && ptr->next->data != before) {**

**ptr = ptr->next;**

**}**

**if (ptr->next == nullptr) {**

**cout << "\nNode with value " << before << " not found!";**

**return;**

**}**

**Node \*newNode = new Node(value);**

**newNode->next = ptr->next;**

**ptr->next = newNode;**

**cout << "\nNode added before " << before << ".";**

**}**

**void InsertAtPos(**

**int value,**

**int pos) // function to add a node at a specific position in the LL**

**{**

**if (pos <= 0) {**

**cout << "\nInvalid position!";**

**return;**

**}**

**if (pos == 1) {**

**InsertAtBeg(value);**

**return;**

**}**

**Node \*ptr = head;**

**for (int i = 1; i < pos - 1 && ptr != nullptr; i++) {**

**ptr = ptr->next;**

**}**

**if (ptr == nullptr) {**

**cout << "\nInvalid position!";**

**return;**

**}**

**Node \*newNode = new Node(value);**

**newNode->next = ptr->next;**

**ptr->next = newNode;**

**cout << "\nNode added at position " << pos << ".";**

**}**

**void DeleteFromBeg() // function to delete a node from the beginning of the LL**

**{**

**if (head == nullptr) {**

**cout << "\nEmpty list!";**

**return;**

**}**

**Node \*temp = head;**

**head = head->next;**

**delete temp;**

**cout << "\nNode deleted from beginning.";**

**}**

**void DeleteFromEnd() // function to delete a node from the end of the LL**

**{**

**if (head == nullptr) {**

**cout << "\nEmpty list!";**

**return;**

**}**

**if (head->next == nullptr) {**

**delete head;**

**head = nullptr;**

**cout << "\nNode deleted from end.";**

**return;**

**}**

**Node \*ptr = head;**

**while (ptr->next->next != nullptr) {**

**ptr = ptr->next;**

**}**

**delete ptr->next;**

**ptr->next = nullptr;**

**cout << "\nNode deleted from end.";**

**}**

**void DeleteAfterNode(**

**int after) // function to delete a node after a given node in the LL**

**{**

**Node \*ptr = head;**

**while (ptr != nullptr && ptr->data != after) {**

**ptr = ptr->next;**

**}**

**if (ptr == nullptr || ptr->next == nullptr) {**

**cout << "\nValue not found or no node after " << after << ".";**

**return;**

**}**

**Node \*temp = ptr->next;**

**ptr->next = temp->next;**

**delete temp;**

**cout << "\nNode deleted after " << after << ".";**

**}**

**void DeleteByVal(int value) // function to delete a node by its value**

**{**

**if (head == nullptr) {**

**cout << "\nEmpty list!";**

**return;**

**}**

**if (head->data == value) {**

**Node \*temp = head;**

**head = head->next;**

**delete temp;**

**cout << "\nNode with value " << value << " deleted.";**

**return;**

**}**

**Node \*ptr = head;**

**while (ptr->next != nullptr && ptr->next->data != value) {**

**ptr = ptr->next;**

**}**

**if (ptr->next == nullptr) {**

**cout << "\nNode with value " << value << " not found!";**

**return;**

**}**

**Node \*temp = ptr->next;**

**ptr->next = temp->next;**

**delete temp;**

**cout << "\nNode with value " << value << " deleted.";**

**}**

**void Search(int value) // function to search a value in the LL**

**{**

**Node \*ptr = head;**

**int pos = 1;**

**while (ptr != nullptr) {**

**if (ptr->data == value) {**

**cout << "\n" << value << " found at position " << pos << ".";**

**return;**

**}**

**ptr = ptr->next;**

**pos++;**

**}**

**cout << "\nNode with value " << value << " not found!";**

**}**

**void Display() // function to display all values in the LL**

**{**

**if (head == nullptr) {**

**cout << "\nList is empty.";**

**return;**

**}**

**Node \*ptr = head;**

**int count = 0;**

**cout << "\nLinked List: ";**

**while (ptr != nullptr) {**

**cout << ptr->data << " -> ";**

**ptr = ptr->next;**

**count++;**

**}**

**cout << "END\n";**

**cout << "Total number of nodes: " << count;**

**}**

**};**

**int main() {**

**LinkedList list;**

**int choice = 0, value, key;**

**while (choice != 12) {**

**cout << "\n\n\*\*\*\*\*\*\*\*\*\*\*\* MAIN MENU (C++) \*\*\*\*\*\*\*\*\*\*\*\*";**

**cout << "\n1 : Add a node at Beginning";**

**cout << "\n2 : Add a node at the End";**

**cout << "\n3 : Add a node after a Node";**

**cout << "\n4 : Add a node before a Node";**

**cout << "\n5 : Add a node at position";**

**cout << "\n6 : Delete a node from the Beginning";**

**cout << "\n7 : Delete a node from the End";**

**cout << "\n8 : Delete a node after a Node";**

**cout << "\n9 : Delete a node by Value";**

**cout << "\n10: Search an element";**

**cout << "\n11: Display the Linked List";**

**cout << "\n12: END the Operation !!!";**

**cout << "\n\nEnter your choice: ";**

**cin >> choice;**

**switch (choice) {**

**case 1:**

**cout << "Enter Value to Add: ";**

**cin >> value;**

**list.InsertAtBeg(value);**

**break;**

**case 2:**

**cout << "Enter Value to Add: ";**

**cin >> value;**

**list.InsertAtEnd(value);**

**break;**

**case 3:**

**cout << "Enter Value to Add: ";**

**cin >> value;**

**cout << "Enter the value of node after which to insert: ";**

**cin >> key;**

**list.InsertAfterNode(value, key);**

**break;**

**case 4:**

**cout << "Enter Value to Add: ";**

**cin >> value;**

**cout << "Enter the value of node before which to insert: ";**

**cin >> key;**

**list.InsertBeforeNode(value, key);**

**break;**

**case 5:**

**cout << "Enter Value to Add: ";**

**cin >> value;**

**cout << "Enter position: ";**

**cin >> key;**

**list.InsertAtPos(value, key);**

**break;**

**case 6:**

**list.DeleteFromBeg();**

**break;**

**case 7:**

**list.DeleteFromEnd();**

**break;**

**case 8:**

**cout << "Enter value of node after which to delete: ";**

**cin >> key;**

**list.DeleteAfterNode(key);**

**break;**

**case 9:**

**cout << "Enter value to delete: ";**

**cin >> value;**

**list.DeleteByVal(value);**

**break;**

**case 10:**

**cout << "Enter value to search: ";**

**cin >> value;**

**list.Search(value);**

**break;**

**case 11:**

**list.Display();**

**break;**

**case 12:**

**cout << "\nExit !!!";**

**break;**

**default:**

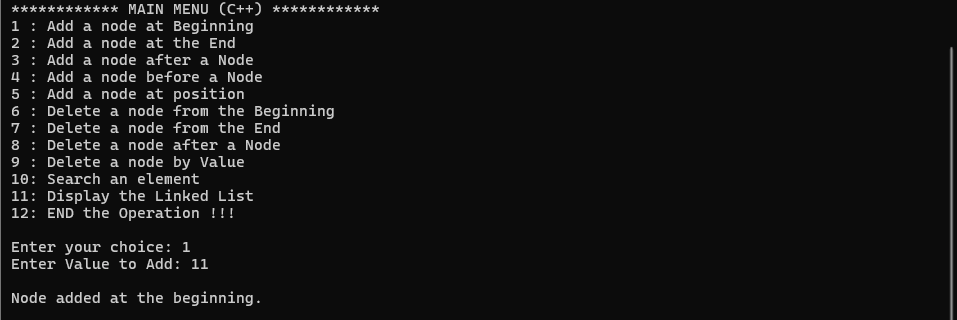
**cout << "\nInvalid option. Try again.";**

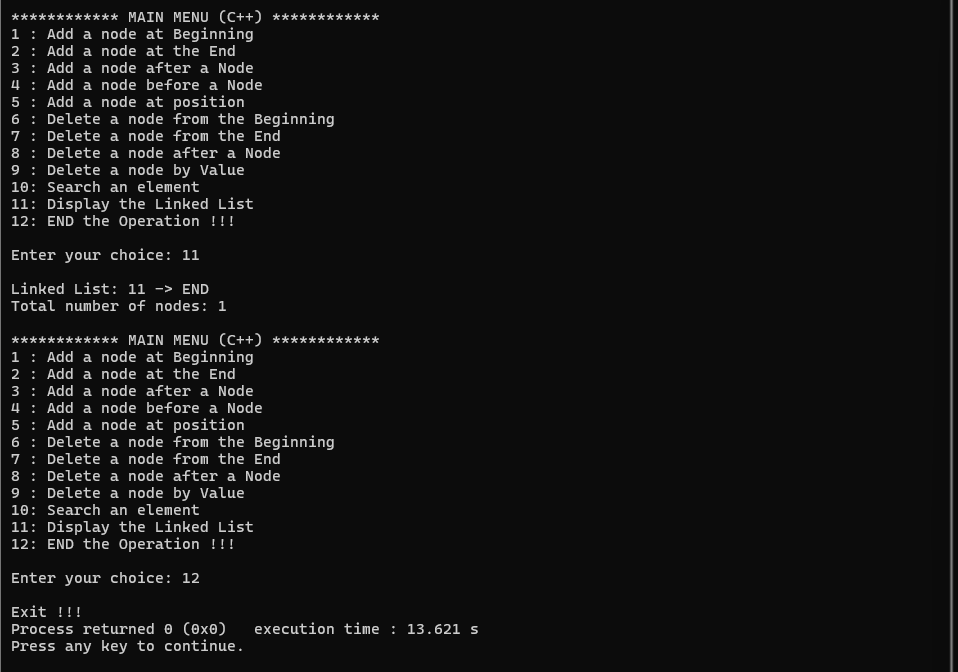
**}**

**}**

**return 0;**

**}**

****

  
  
  
**Q7. Write the code snippet to define a Node structure for a Doubly Linked List in C/C++.**

**Q8. Write function prototypes for the following operations:**

**• Create a new node**

**• Insert at beginning**

**• Insert at end**

**• Insert after a specific position**

**• Delete from beginning**

**• Delete from end**

**• Delete a specific node**

**• Search an element**

**• Display forward**

**• Display backward**

**#include <stdio.h>**

**#include <stdlib.h> // Header File for malloc(),calloc(),realloc()**

**struct node // Structure for the nodes we create**

**{**

**int data;**

**struct node \*next;**

**struct node \*prev;**

**};**

**struct node \*head = NULL; // Head pointer which indicates first node**

**struct node \*tail = NULL; // Tail pointer which indicates last node**

**struct node \*CreateNode(**

**int value); // function to create a new node, NOTE : this func. won't be**

**// called directly by the users but by the other functions**

**void InsertAtBeg(**

**int value); // function to insert a new node at the beginning of the LL**

**void InsertAtEnd(**

**int value); // function to insert a new node at the end of the LL**

**void InsertAfterNode(**

**int value,**

**int after); // function to insert a new node after a given node in the LL**

**void InsertBeforeNode(**

**int value,**

**int before); // function to insert a new node before a given node in the LL**

**void InsertAtPos(**

**int value,**

**int pos); // function to insert a new node at given position in the LL**

**void DeleteFromBeg(); // function to delete a node from the beginning of the LL**

**void DeleteFromEnd(); // function to delete a node from the end of the LL**

**void DeleteAtPos(int pos); // function to delete a node by its position**

**void Search(int value); // function to Search an element in the LL**

**void Display(); // function to display the created LL**

**void DisplayReverse(); // function to display the created LL in reverse order**

**int main() {**

**int choice = 0, value, key;**

**while (choice != 12) {**

**printf("\n\n\*\*\*\*\*\*\*\*\*\*\*\* MAIN MENU | DOUBLY LL | (C) \*\*\*\*\*\*\*\*\*\*\*\*");**

**printf("\n1 : Add a node at Beginning");**

**printf("\n2 : Add a node at the End");**

**printf("\n3 : Add a node after a Node");**

**printf("\n4 : Add a node before a node");**

**printf("\n5 : Add a node by position");**

**printf("\n6 : Delete a node from the Beginning");**

**printf("\n7 : Delete a node from the End");**

**printf("\n8 : Delete a node by its position");**

**printf("\n9 : Search for an element");**

**printf("\n10: Display the Linked List");**

**printf("\n11: Display the Linked List in reverse order");**

**printf("\n12: END the Operation !!!");**

**printf("\n\nEnter your choice : ");**

**scanf("%d", &choice);**

**switch (choice) {**

**case 1: // calling InsertAtBeg(value) function to add a node in the**

**// Beginning of the LL**

**printf("Enter Value to Add : ");**

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

**InsertAtBeg(value);**

**break;**

**case 2: // calling InsertAtEnd(value) function to add a node in the End of**

**// the LL**

**printf("Enter Value to Add : ");**

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

**InsertAtEnd(value);**

**break;**

**case 3: // calling InsertAfterNode(value,after) function to add a node after**

**// a node in the LL**

**printf("Enter Value to Add : ");**

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

**printf("Enter the Value of node coming before the new node : ");**

**scanf("%d", &key);**

**InsertAfterNode(value, key);**

**break;**

**case 4: // calling InsertBeforeNode(value,before) function to add a node**

**// before a node in the LL**

**printf("Enter Value to Add : ");**

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

**printf("Enter the Value of node coming after the new node : ");**

**scanf("%d", &key);**

**InsertBeforeNode(value, key);**

**break;**

**case 5:**

**printf("Enter Value to Add : ");**

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

**printf("Enter postion at-which node is to be added : ");**

**scanf("%d", &key);**

**InsertAtPos(value, key);**

**break;**

**case 6: // calling DeleteFromBeg() function to delete a node from the**

**// Beginning of the LL**

**DeleteFromBeg();**

**break;**

**case 7: // calling DeleteFromEnd() function to delete a node from the End of**

**// the LL**

**DeleteFromEnd();**

**break;**

**case 8:**

**printf("Enter pos of the node you want to delete : ");**

**scanf("%d", &key);**

**DeleteAtPos(key);**

**break;**

**case 9:**

**printf("Enter Value to find in the LL : ");**

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

**Search(value);**

**break;**

**case 10: // calling Display() function to Display the created LL**

**printf("\nLinked List: ");**

**Display();**

**break;**

**case 11: // calling DisplayReverse() function to Display the created LL in**

**// reverse order**

**printf("\nLinked List in reverse order : ");**

**DisplayReverse();**

**break;**

**case 12: // exiting from the program**

**printf("\nExit !!!");**

**break;**

**default:**

**printf("\nInvalid option. Try again.");**

**break;**

**}**

**}**

**return 0;**

**}**

**struct node \*CreateNode(int value) // Creating a new node NOTE : this function**

**// won't be called directly by the user**

**{**

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

**newNode->data = value;**

**newNode->next = NULL;**

**newNode->prev = NULL;**

**return newNode;**

**}**

**void InsertAtBeg(int value) // Inserting at beginning**

**{**

**struct node \*new\_node = CreateNode(value);**

**if (head == NULL) {**

**head = new\_node;**

**tail = new\_node;**

**return;**

**}**

**new\_node->next = head;**

**head->prev = new\_node;**

**head = new\_node;**

**printf("\nNode Added at the beginning.");**

**}**

**void InsertAtEnd(int value) // Inserting at end**

**{**

**struct node \*new\_node = CreateNode(value);**

**if (head == NULL) {**

**head = new\_node;**

**tail = new\_node;**

**return;**

**}**

**tail->next = new\_node;**

**new\_node->prev = tail;**

**tail = new\_node;**

**printf("\nNode Added at the end.");**

**}**

**void InsertAfterNode(int value, int after) {**

**if (head == NULL) {**

**InsertAtBeg(value);**

**return;**

**}**

**struct node \*ptr = head;**

**while (ptr != NULL && ptr->data != after) {**

**ptr = ptr->next;**

**}**

**if (ptr == NULL) {**

**printf("Node with value %d not found.\n", after);**

**return;**

**}**

**struct node \*new\_node = CreateNode(value);**

**new\_node->next = ptr->next;**

**new\_node->prev = ptr;**

**if (ptr->next != NULL) {**

**ptr->next->prev = new\_node;**

**} else {**

**tail = new\_node;**

**}**

**ptr->next = new\_node;**

**printf("\nNode Added after the Node having value %d.", after);**

**}**

**void InsertBeforeNode(int value, int before) {**

**if (head->data == before) {**

**InsertAtBeg(value);**

**return;**

**}**

**struct node \*ptr = head;**

**while (ptr != NULL && ptr->data != before) {**

**ptr = ptr->next;**

**}**

**if (ptr == NULL) {**

**printf("Node with value %d not found.\n", before);**

**return;**

**}**

**struct node \*new\_node = CreateNode(value);**

**new\_node->next = ptr;**

**new\_node->prev = ptr->prev;**

**if (ptr->prev != NULL) {**

**ptr->prev->next = new\_node;**

**}**

**ptr->prev = new\_node;**

**if (ptr->prev == NULL) {**

**head = new\_node;**

**}**

**printf("\nNode Added before the Node having value %d.", before);**

**}**

**void InsertAtPos(int value, int pos) {**

**if (pos == 0) {**

**InsertAtBeg(value);**

**return;**

**}**

**struct node \*new\_node = CreateNode(value);**

**struct node \*ptr = head;**

**for (int i = 0; i < pos - 2 && ptr != NULL; i++) {**

**if (ptr == NULL) {**

**printf("\nInvalid Position !!");**

**return;**

**}**

**ptr = ptr->next;**

**}**

**new\_node->next = ptr->next;**

**new\_node->prev = ptr;**

**if (ptr->next != NULL) {**

**ptr->next->prev = new\_node;**

**}**

**ptr->next = new\_node;**

**printf("\nNew node Added at postion %d", pos);**

**}**

**void DeleteFromBeg() {**

**if (head == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**struct node \*ptr = head;**

**head = head->next;**

**if (head != NULL) {**

**head->prev = NULL;**

**} else {**

**tail = NULL;**

**}**

**free(ptr);**

**printf("\nNode Deleted from the Beginning.");**

**}**

**void DeleteFromEnd() {**

**if (head == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**if (head->next == NULL) {**

**free(head);**

**head = NULL;**

**tail = NULL;**

**} else {**

**struct node \*ptr = tail;**

**tail = tail->prev;**

**tail->next = NULL;**

**free(ptr);**

**}**

**printf("\nNode Deleted from the End.");**

**}**

**void DeleteAtPos(int pos) {**

**if (head == NULL) {**

**printf("List is Empty.\n");**

**return;**

**}**

**if (pos == 0) {**

**DeleteFromBeg();**

**return;**

**}**

**struct node \*ptr = head;**

**for (int i = 0; i < pos - 1 && ptr != NULL; i++) {**

**ptr = ptr->next;**

**}**

**if (ptr == NULL) {**

**printf("\nInvalid Position.");**

**return;**

**}**

**ptr->prev->next = ptr->next;**

**if (ptr->next != NULL) {**

**ptr->next->prev = ptr->prev;**

**}**

**free(ptr);**

**printf("\nNode Deleted from position %d.", pos);**

**}**

**void Search(int value) {**

**struct node \*ptr = head;**

**int pos = 1;**

**while (ptr != NULL) {**

**if (ptr->data == value) {**

**printf("\nNode with Value %d found at position %d\n", ptr->data, pos);**

**return;**

**}**

**ptr = ptr->next;**

**pos++;**

**}**

**printf("\nNode with value %d not found in the LL !\n", value);**

**}**

**void Display() {**

**struct node \*ptr = head;**

**int count = 0;**

**if (ptr == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**while (ptr != NULL) {**

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

**ptr = ptr->next;**

**count++;**

**}**

**printf("END\n");**

**printf("Total nodes: %d\n", count);**

**}**

**void DisplayReverse() {**

**struct node \*ptr = tail;**

**int count = 0;**

**if (ptr == NULL) {**

**printf("List is empty.\n");**

**return;**

**}**

**while (ptr != NULL) {**

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

**ptr = ptr->prev;**

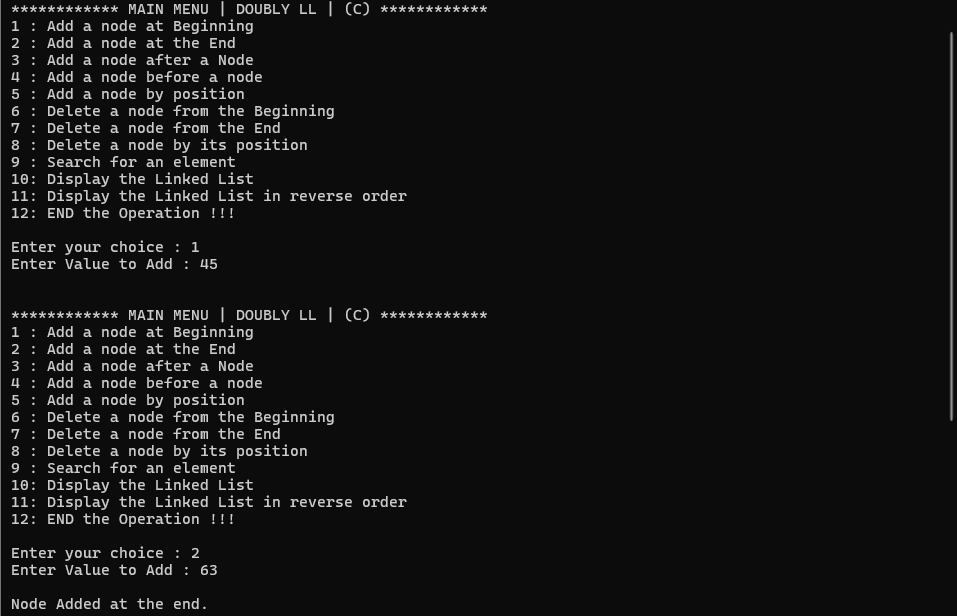
**count++;**

**}**

**printf("NULL\n");**

**printf("Total nodes: %d\n", count);**

**}**

****

