**Name: Sneha Roy , Section : B , Roll : 42**

**Enrollment No.: 12024006015099**

**Week 2 Assignment**

**Topic: Implementation of Array**

1. Write a C program to read a 2D array (with most of the elements as 0s) and then represent the same array as Sparse Metrics.

**Answer:**

#include<stdio.h>

int main(){

  int r, c, zeroCount = 0, count = 0;

  printf("Enter number of row : ");

  scanf("%d", &r);

  printf("Enter number of column : ");

  scanf("%d", &c);

  int arr[r][c];

  printf("Enter elements for array : \n");

  for(int i = 0; i < r; i++)

    for(int j = 0; j < c; j++)

      scanf("%d", &arr[i][j]);

  for(int i = 0; i < r; i++)

    for(int j = 0; j < c; j++){

      if(arr[i][j] == 0) zeroCount++;

      else count++;

    }

  if(zeroCount > count) {

    int ans[3][count], iCol = 0;

    for(int i = 0; i < r; i++)

    for(int j = 0; j < c; j++){

      if(arr[i][j] != 0) {

        ans[0][iCol] = i;

        ans[1][iCol] = j;

        ans[2][iCol] = arr[i][j];

        iCol++;

      }

    }

    printf("\n\n");

  for(int i = 0; i < 3; i++){

    for(int j = 0; j < count; j++)

      printf("%d   ", ans[i][j]);

      printf("\n");

  }

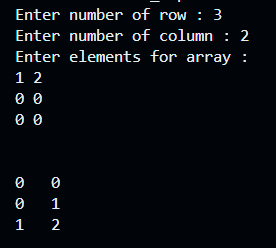
  }

  else printf("Not A Sparse Matrix");

  return 0;

}

**Output:**

****

2. Write a C program to pass an array to a function using Call by Value, update the array values in the function, print the array elements both in the function and in the calling function.

**Answer:**

#include<stdio.h>

void Change (int arr[], int n){

  for(int i = 0 ; i < n; i++) arr[i] += 10;

  for(int i = 0 ; i < n; i++) printf("%d  ,", arr[i]);

}

int main(){

  int n, arr[100];

  printf("Enter size of array : ");

  scanf("%d", &n);

  printf("Enter Elements : ");

  for(int i = 0 ; i < n; i++) scanf("%d", &arr[i]);

  for(int i = 0 ; i < n; i++) printf("%d ,  ", arr[i]);

  printf("\n\n");

  Change(arr, n);

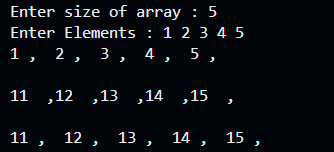
  printf("\n\n");

  for(int i = 0 ; i < n; i++) printf("%d ,  ", arr[i]);

  return 0;

}

**Output:**

****

4. Write a program that reads two 2D metrices from the console, verifies if metrics multiplication is possible or not. Then multiplies the metrices and prints the 3rd metrics.

**Answer:**

#include<stdio.h>

int main(){

  int r1, c1, r2, c2;

    printf("Enter row and column of the 1st matrix : ");

    scanf("%d %d", &r1, &c1);

    printf("Enter row and column of the 2nd matrix : ");

    scanf("%d %d", &r2, &c2);

    if(c1!=r2) printf("Given two matrices can't be multiplied.");

    else if (c1==r2){

        int arr[r1][c1], brr[r2][c2], crr[r1][c2];

        printf("Enter the 1st matrix : \n");

        for(int i = 0; i < r1; i++)//Taking Inputs

            for(int j = 0; j < c1; j++) scanf("%d", &arr[i][j]);

        printf("\nEnter the 2nd matrix : \n");

        for(int i = 0; i < r2; i++)//Taking Inputs

            for(int j = 0; j < c2; j++) scanf("%d", &brr[i][j]);

        for(int i = 0; i < r1; i++)//Assume all element in result matrix is '0'

            for(int j = 0; j < c2; j++) crr[i][j]=0;

        for(int i = 0; i < r1; i++)

            for(int j = 0; j < c2; j++)

                for(int k = 0; k<c1; k++)

                    crr[i][j] = crr[i][j] + (arr[i][k] \* brr[k][j]);

        printf("Result is : \n");

        for(int i = 0; i < r1; i++){//Print the result matrix

            for(int j = 0; j < c2; j++) printf("%d    ", crr[i][j]);

            printf("\n");

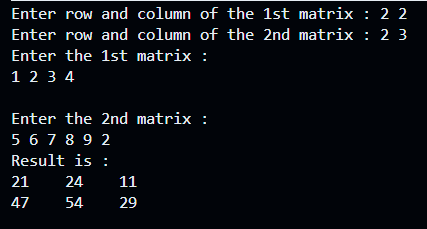
        }

    }

  return 0;

}

**Output:**

****

5. Write a program that reads a 2D metrics and checks if the metrics is a symmetric metrics or not.

**Answer:**

#include<stdio.h>

int main(){

  int n, check = 1 ;

  printf("Enter number of row / column : ");

  scanf("%d", &n);

  int arr[n][n];

  printf("Enter elements for array : \n");

  for(int i = 0; i < n; i++)

    for(int j = 0; j < n; j++)

      scanf("%d", &arr[i][j]);

  for(int i = 0; i < n; i++)

    for(int j = i; j < n; j++)

      if(arr[i][j] != arr[j][i]) {

        check = 0;

        break;

      }

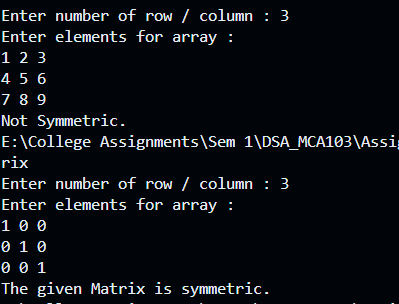
  if(check) printf("The given Matrix is symmetric.");

  else printf("Not Symmetric.");

  return 0;

}

**Output:**



6. Write a program to display n number of elements. Memory should be allocated dynamically using malloc ( ).

**Answer:**

#include <stdio.h>

#include <stdlib.h>

int main(){

  int n;

  printf("Enter number of elements you want : ");

  scanf("%d", &n);

  int\* ptr = (int\*) malloc(n \* sizeof(int));

  int\* p = ptr;

  printf("Enter Elements : ");

  for(int i = 1; i <= n; i++){

    scanf("%d", &(\*p));

    p++;

  }

  p = ptr;

  printf("\n\n");

  for(int i = 1; i <= n; i++){

    printf("%d \n",\*p);

    p++;

  }

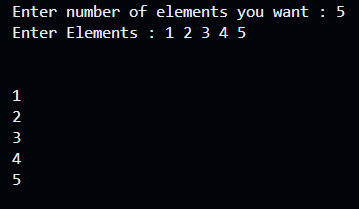
  free(ptr);

  ptr = NULL;

  return 0;

}

**Output:**

****

7. Write a program to display n number of elements. Memory should be allocated dynamically using calloc( ).

**Answer:**

#include <stdio.h>

#include <stdlib.h>

int main(){

  int n;

  printf("Enter number of elements you want : ");

  scanf("%d", &n);

  int\* ptr = (int\*) calloc(n , sizeof(int));

  int\* p = ptr;

  printf("Enter Elements : ");

  for(int i = 1; i <= n; i++){

    scanf("%d", &(\*p));

    p++;

  }

  p = ptr;

  printf("\n\n");

  for(int i = 1; i <= n; i++){

    printf("%d \n",\*p);

    p++;

  }

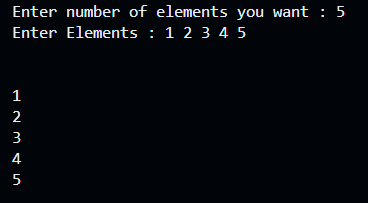
  free(ptr);

  ptr = NULL;

  return 0;

}

**Output:**

****

8. Write a program to allocate memory using malloc ( ) and then reallocate the previously allocated memory using realloc( ). Display the elements which have been taken after reallocation.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

int main(){

  int n, a;

  printf("Enter number of elements you want : ");

  scanf("%d", &n);

  int\* ptr = (int\*) malloc(n \* sizeof(int));

  printf("Enter number you want to increase : ");

  scanf("%d", &a);

  n = n + a;

  ptr = realloc (ptr, n \* sizeof(int));

  int\* p = ptr;

  printf("Enter Elements : ");

  for(int i = 1; i <= n; i++){

    scanf("%d", &(\*p));

    p++;

  }

  p = ptr;

  printf("\n\n");

  for(int i = 1; i <= n; i++){

    printf("%d \n",\*p);

    p++;

  }

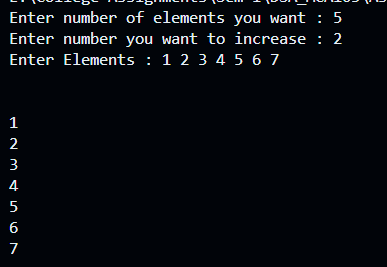
  free(ptr);

  ptr = NULL;

  return 0;

}

**Output:**

****

9. Write a program to allocate memory using calloc( ) and then reallocate the previously allocated memory using realloc( ). Display the elements which have been taken after reallocation.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

int main(){

  int n, a;

  printf("Enter number of elements you want : ");

  scanf("%d", &n);

  int\* ptr = (int\*) calloc(n, sizeof(int));

  printf("Enter number you want to increase : ");

  scanf("%d", &a);

  n = n + a;

  ptr = realloc (ptr, n \* sizeof(int));

  int\* p = ptr;

  printf("Enter Elements : ");

  for(int i = 1; i <= n; i++){

    scanf("%d", &(\*p));

    p++;

  }

  p = ptr;

  printf("\n\n");

  for(int i = 1; i <= n; i++){

    printf("%d \n",\*p);

    p++;

  }

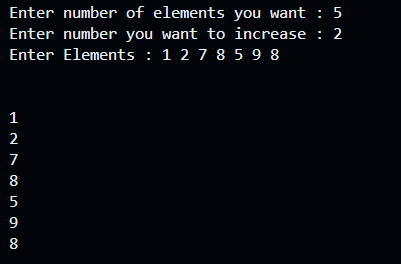
  free(ptr);

  ptr = NULL;

  return 0;

}

**Output:**

****

10. Write a C program to search an element in an Array using dynamic memory allocation

**Answer:**

#include <stdio.h>

#include <stdlib.h>

int main(){

  int n, key, check = 0;

  printf("Enter number of elements you want : ");

  scanf("%d", &n);

  int\* ptr = (int\*) malloc(n \* sizeof(int));

  int\* p = ptr;

  printf("Enter Elements : ");

  for(int i = 1; i <= n; i++){

    scanf("%d", &(\*p));

    p++;

  }

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

  scanf("%d", &key);

  p = ptr;

  printf("\n\n");

  for(int i = 1; i <= n; i++){

    if(key == \*p){

      check = 1;

      break;

    }

    p++;

  }

  free(ptr);

  ptr = NULL;

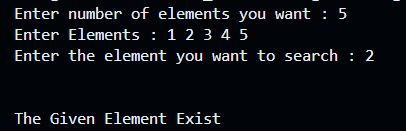
  if(check) printf("The Given Element Exist");

  else printf("The Given Element Exist");

  return 0;

}

**Output:**



**Week 3 Assignment**

**Topic: Linked List**

1) Write a Menu driven C program to accomplish the following functionalities in single linked list.

a) Create a single linked list. b) Display the elements of a single linked list.

c) Insert a node at the beginning of a single linked list.

d) Insert a node at the end of a single linked list.

e) Insert a node before a given node of a single linked list.

f) Insert a node after a given node of a single linked list.

g) Delete a node from the beginning of a single linked list.

h) Delete a node from the end of a single linked list.

i) Delete a node after a given node of a single linked list.

j) Delete the entire single linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

struct Node {

    int data;

    struct Node\* next;

};

struct Node\* head = NULL;

// Function to create a single linked list

void createList(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    if (head == NULL) {

        head = newNode;

    } else {

        struct Node\* temp = head;

        while (temp->next != NULL) {

            temp = temp->next;

        }

        temp->next = newNode;

    }

}

// Function to display elements of a single linked list

void displayList() {

    struct Node\* temp = head;

    if (temp == NULL) {

        printf("List is empty.\n");

        return;

    }

    while (temp != NULL) {

        printf("%d -> ", temp->data);

        temp = temp->next;

    }

    printf("NULL\n");

}

// Function to insert a node at the beginning

void insertAtBeginning(int data) {

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

    newNode->data = data;

    newNode->next = head;

    head = newNode;

}

// Function to insert a node at the end

void insertAtEnd(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    if (head == NULL) {

        head = newNode;

        return;

    }

    struct Node\* temp = head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

}

// Function to insert a node before a given node

void insertBeforeNode(int target, int data) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

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

    newNode->data = data;

    if (head->data == target) {

        newNode->next = head;

        head = newNode;

        return;

    }

    struct Node\* temp = head;

    while (temp->next != NULL && temp->next->data != target) {

        temp = temp->next;

    }

    if (temp->next == NULL) {

        printf("Node not found.\n");

    } else {

        newNode->next = temp->next;

        temp->next = newNode;

    }

}

// Function to insert a node after a given node

void insertAfterNode(int target, int data) {

    struct Node\* temp = head;

    while (temp != NULL && temp->data != target) {

        temp = temp->next;

    }

    if (temp == NULL) {

        printf("Node not found.\n");

        return;

    }

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

    newNode->data = data;

    newNode->next = temp->next;

    temp->next = newNode;

}

// Function to delete a node from the beginning

void deleteFromBeginning() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = head;

    head = head->next;

    free(temp);

}

// Function to delete a node from the end

void deleteFromEnd() {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    if (head->next == NULL) {

        free(head);

        head = NULL;

        return;

    }

    struct Node\* temp = head;

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

        temp = temp->next;

    }

    free(temp->next);

    temp->next = NULL;

}

// Function to delete a node after a given node

void deleteAfterNode(int target) {

    struct Node\* temp = head;

    while (temp != NULL && temp->data != target) {

        temp = temp->next;

    }

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

        printf("Node not found or no node exists after the given node.\n");

        return;

    }

    struct Node\* nodeToDelete = temp->next;

    temp->next = temp->next->next;

    free(nodeToDelete);

}

// Function to delete the entire list

void deleteList() {

    struct Node\* temp;

    while (head != NULL) {

        temp = head;

        head = head->next;

        free(temp);

    }

    printf("Entire list deleted.\n");

}

// Main function with menu

int main() {

    int choice, data, target;

    while (1) {

        printf("\nMenu:\n");

        printf("1. Create a single linked list\n");

        printf("2. Display the elements\n");

        printf("3. Insert at the beginning\n");

        printf("4. Insert at the end\n");

        printf("5. Insert before a given node\n");

        printf("6. Insert after a given node\n");

        printf("7. Delete from the beginning\n");

        printf("8. Delete from the end\n");

        printf("9. Delete after a given node\n");

        printf("10. Delete the entire list\n");

        printf("11. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter data to insert: ");

                scanf("%d", &data);

                createList(data);

                break;

            case 2:

                displayList();

                break;

            case 3:

                printf("Enter data to insert at the beginning: ");

                scanf("%d", &data);

                insertAtBeginning(data);

                break;

            case 4:

                printf("Enter data to insert at the end: ");

                scanf("%d", &data);

                insertAtEnd(data);

                break;

            case 5:

                printf("Enter the target node data before which to insert: ");

                scanf("%d", &target);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                insertBeforeNode(target, data);

                break;

            case 6:

                printf("Enter the target node data after which to insert: ");

                scanf("%d", &target);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                insertAfterNode(target, data);

                break;

            case 7:

                deleteFromBeginning();

                break;

            case 8:

                deleteFromEnd();

                break;

            case 9:

                printf("Enter the target node data after which to delete: ");

                scanf("%d", &target);

                deleteAfterNode(target);

                break;

            case 10:

                deleteList();

                break;

            case 11: exit(0);

            default: printf("Invalid choice. Try again.\n");

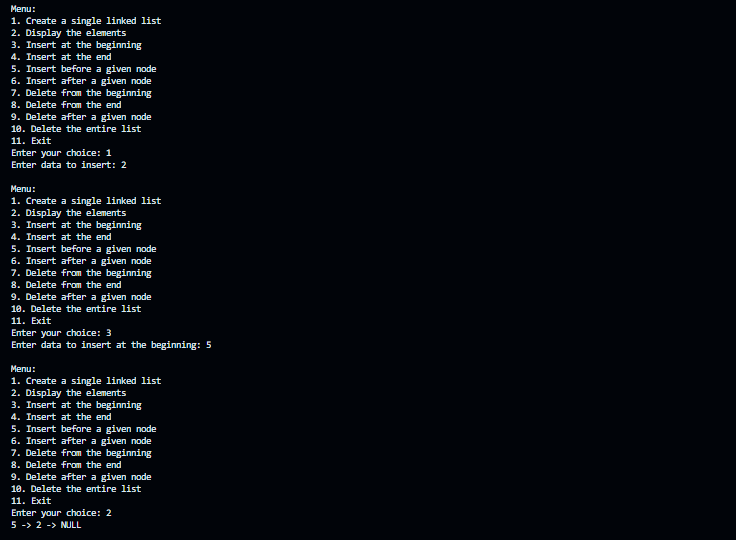
        }

    }

    return 0;

}

**Output:**



2) Write a Menu driven C program to accomplish the following functionalities in circular linked list.

a) Create a circular linked list.

b) Display the elements of a circular linked list.

c) Insert a node at the beginning of a circular linked list.

d) Insert a node at the end of a circular linked list.

e) Delete a node from the beginning of a circular linked list.

f) Delete a node from the end of a circular linked list.

g) Delete a node after a given node of a circular linked list.

h) Delete the entire circular linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the circular linked list

struct Node {

  int data;

  struct Node \*next;

};

// Function to create a circular linked list with a single node

struct Node\* createNode(int data) {

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

  newNode->data = data;

  newNode->next = newNode;

  return newNode;

}

// Function to display all elements in the circular linked list

void display(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return;

  }

  struct Node\* temp = last->next;

  do {

    printf("%d -> ", temp->data);

    temp = temp->next;

  } while (temp != last->next);

  printf("\n");

}

// Function to insert a node at the beginning of the circular linked list

struct Node\* insertAtBeginning(struct Node\* last, int data) {

  struct Node\* newNode = createNode(data);

  if (last == NULL) {

    last = newNode;

  } else {

    newNode->next = last->next;

    last->next = newNode;

  }

  return last;

}

// Function to insert a node at the end of the circular linked list

struct Node\* insertAtEnd(struct Node\* last, int data) {

  struct Node\* newNode = createNode(data);

  if (last == NULL) {

    return newNode;

  }

  newNode->next = last->next;

  last->next = newNode;

  last = newNode;

  return last;

}

// Function to delete a node from the beginning of the circular linked list

struct Node\* deleteFromBeginning(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  if (last == temp) {

    free(temp);

    return NULL;

  }

  last->next = temp->next;

  free(temp);

  return last;

}

// Function to delete a node from the end of the circular linked list

struct Node\* deleteFromEnd(struct Node\* last) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  if (last == temp) {

    free(last);

    return NULL;

  }

  while (temp->next != last) {

    temp = temp->next;

  }

  temp->next = last->next;

  free(last);

  last = temp;

  return last;

}

// Function to delete a node after a given node in the circular linked list

struct Node\* deleteAfterNode(struct Node\* last, int value) {

  if (last == NULL) {

    printf("List is empty.\n");

    return NULL;

  }

  struct Node\* temp = last->next;

  do {

    if (temp->data == value) {

      struct Node\* nodeToDelete = temp->next;

      if (nodeToDelete == last) {

        last = temp;

      }

      temp->next = nodeToDelete->next;

      free(nodeToDelete);

      return last;

    }

    temp = temp->next;

  } while (temp != last->next);

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

    return last;

}

// Function to delete the entire circular linked list

struct Node\* deleteList(struct Node\* last) {

  if (last == NULL) return NULL;

  struct Node\* current = last->next;

  while (current != last) {

    struct Node\* temp = current;

    current = current->next;

    free(temp);

  }

  free(last);

  printf("Entire list deleted.\n");

  return NULL;

}

int main() {

  struct Node\* last = NULL;

  int choice, data, value;

  do {

    printf("\nCircular Linked List Operations:\n");

    printf("1. Create circular linked list\n");

    printf("2. Display elements\n");

    printf("3. Insert at beginning\n");

    printf("4. Insert at end\n");

    printf("5. Delete from beginning\n");

    printf("6. Delete from end\n");

    printf("7. Delete after a node\n");

    printf("8. Delete entire list\n");

    printf("9. Exit\n");

    printf("Enter your choice: ");

    scanf("%d", &choice);

    switch (choice) {

      case 1:

        printf("Enter data to create list: ");

        scanf("%d", &data);

        last = createNode(data);

        break;

      case 2:

        display(last);

        break;

      case 3:

        printf("Enter data to insert at beginning: ");

        scanf("%d", &data);

        last = insertAtBeginning(last, data);

        break;

      case 4:

        printf("Enter data to insert at end: ");

        scanf("%d", &data);

        last = insertAtEnd(last, data);

        break;

      case 5:

        last = deleteFromBeginning(last);

        break;

      case 6:

        last = deleteFromEnd(last);

        break;

      case 7:

        printf("Enter value after which to delete: ");

        scanf("%d", &value);

        last = deleteAfterNode(last, value);

        break;

      case 8:

        last = deleteList(last);

        break;

      case 9:

        printf("Exiting program.\n");

        break;

      default:

        printf("Invalid choice. Try again.\n");

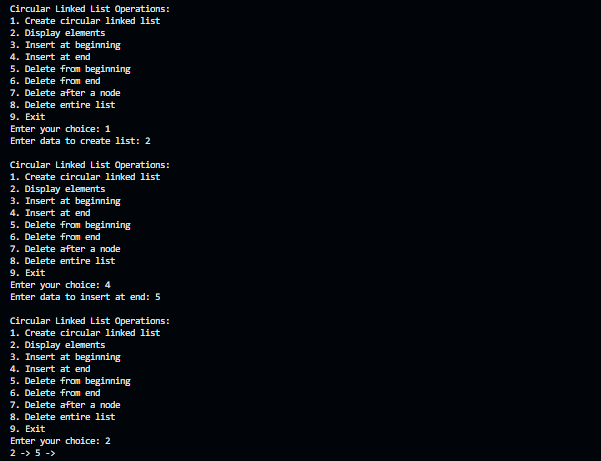
    }

  } while (choice != 9);

  return 0;

}

**Output:**



**Week 4 Assignment**

**Topic: Linked List**

1) Write a Menu driven C program to accomplish the following functionalities in doubly linked list.

a) Create a doubly linked list.

b) Display the elements of a doubly linked list.

c) Insert a node at the beginning of a doubly linked list.

d) Insert a node at the end of a doubly linked list.

e) Insert a node before a given node of a doubly linked list.

f) Insert a node after a given node of a doubly linked list.

g) Delete a node from the beginning of a doubly linked list.

h) Delete a node from the end of a doubly linked list.

i) Delete a node after a given node of a doubly linked list.

j) Delete the entire doubly linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the doubly linked list

struct Node {

    int data;

    struct Node \*next;

    struct Node \*prev;

};

// Function to create a node

struct Node\* createNode(int data) {

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

    newNode->data = data;

    newNode->next = NULL;

    newNode->prev = NULL;

    return newNode;

}

// Function to display all elements in the doubly linked list

void display(struct Node\* head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = head;

    while (temp != NULL) {

        printf("%d ", temp->data);

        temp = temp->next;

    }

    printf("\n");

}

// Function to insert a node at the beginning of the doubly linked list

struct Node\* insertAtBeginning(struct Node\* head, int data) {

    struct Node\* newNode = createNode(data);

    if (head != NULL) {

        head->prev = newNode;

    }

    newNode->next = head;

    return newNode;

}

// Function to insert a node at the end of the doubly linked list

struct Node\* insertAtEnd(struct Node\* head, int data) {

    struct Node\* newNode = createNode(data);

    if (head == NULL) {

        return newNode;

    }

    struct Node\* temp = head;

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->next = newNode;

    newNode->prev = temp;

    return head;

}

// Function to insert a node before a given node by value

struct Node\* insertBeforeNode(struct Node\* head, int value, int data) {

    struct Node\* temp = head;

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

        temp = temp->next;

    }

    if (temp == NULL) {

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

        return head;

    }

    struct Node\* newNode = createNode(data);

    newNode->next = temp;

    newNode->prev = temp->prev;

    if (temp->prev != NULL) {

        temp->prev->next = newNode;

    } else {

        head = newNode;

    }

    temp->prev = newNode;

    return head;

}

// Function to insert a node after a given node by value

struct Node\* insertAfterNode(struct Node\* head, int value, int data) {

    struct Node\* temp = head;

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

        temp = temp->next;

    }

    if (temp == NULL) {

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

        return head;

    }

    struct Node\* newNode = createNode(data);

    newNode->next = temp->next;

    newNode->prev = temp;

    if (temp->next != NULL) {

        temp->next->prev = newNode;

    }

    temp->next = newNode;

    return head;

}

// Function to delete a node from the beginning of the doubly linked list

struct Node\* deleteFromBeginning(struct Node\* head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return NULL;

    }

    struct Node\* temp = head;

    head = head->next;

    if (head != NULL) {

        head->prev = NULL;

    }

    free(temp);

    return head;

}

// Function to delete a node from the end of the doubly linked list

struct Node\* deleteFromEnd(struct Node\* head) {

    if (head == NULL) {

        printf("List is empty.\n");

        return NULL;

    }

    struct Node\* temp = head;

    if (temp->next == NULL) {

        free(temp);

        return NULL;

    }

    while (temp->next != NULL) {

        temp = temp->next;

    }

    temp->prev->next = NULL;

    free(temp);

    return head;

}

// Function to delete a node after a given node by value

struct Node\* deleteAfterNode(struct Node\* head, int value) {

    struct Node\* temp = head;

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

        temp = temp->next;

    }

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

        printf("Node with value %d not found or has no next node.\n", value);

        return head;

    }

    struct Node\* nodeToDelete = temp->next;

    temp->next = nodeToDelete->next;

    if (nodeToDelete->next != NULL) {

        nodeToDelete->next->prev = temp;

    }

    free(nodeToDelete);

    return head;

}

// Function to delete the entire doubly linked list

struct Node\* deleteList(struct Node\* head) {

    struct Node\* temp = head;

    while (temp != NULL) {

        struct Node\* next = temp->next;

        free(temp);

        temp = next;

    }

    printf("Entire list deleted.\n");

    return NULL;

}

int main() {

    struct Node\* head = NULL;

    int choice, data, value;

    do {

        printf("\nDoubly Linked List Operations:\n");

        printf("1. Create doubly linked list\n");

        printf("2. Display elements\n");

        printf("3. Insert at beginning\n");

        printf("4. Insert at end\n");

        printf("5. Insert before a node\n");

        printf("6. Insert after a node\n");

        printf("7. Delete from beginning\n");

        printf("8. Delete from end\n");

        printf("9. Delete after a node\n");

        printf("10. Delete entire list\n");

        printf("11. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter data to create list: ");

                scanf("%d", &data);

                head = insertAtEnd(head, data);

                break;

            case 2:

                display(head);

                break;

            case 3:

                printf("Enter data to insert at beginning: ");

                scanf("%d", &data);

                head = insertAtBeginning(head, data);

                break;

            case 4:

                printf("Enter data to insert at end: ");

                scanf("%d", &data);

                head = insertAtEnd(head, data);

                break;

            case 5:

                printf("Enter value before which to insert: ");

                scanf("%d", &value);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                head = insertBeforeNode(head, value, data);

                break;

            case 6:

                printf("Enter value after which to insert: ");

                scanf("%d", &value);

                printf("Enter data to insert: ");

                scanf("%d", &data);

                head = insertAfterNode(head, value, data);

                break;

            case 7:

                head = deleteFromBeginning(head);

                break;

            case 8:

                head = deleteFromEnd(head);

                break;

            case 9:

                printf("Enter value after which to delete: ");

                scanf("%d", &value);

                head = deleteAfterNode(head, value);

                break;

            case 10:

                head = deleteList(head);

                break;

            case 11:

                printf("Exiting program.\n");

                break;

            default:

                printf("Invalid choice. Try again.\n");

        }

    } while (choice != 11);

    return 0;

}

**Output:**



2) Write a Menu driven C program to accomplish the following functionalities in circular doubly linked list.

a) Create a circular doubly linked list.

b) Display the elements of a circular doubly linked list.

c) Insert a node at the beginning of a circular doubly linked list.

d) Insert a node at the end of a circular doubly linked list.

e) Delete a node from the beginning of a circular doubly linked list.

f) Delete a node from the end of a circular doubly linked list.

g) Delete a node after a given node of a circular doubly linked list.

h) Delete the entire circular doubly linked list.

**Answer:**

#include <stdio.h>

#include <stdlib.h>

// Define the structure for a node in the circular doubly linked list

struct Node {

    int data;

    struct Node \*next;

    struct Node \*prev;

};

// Function to create a new node

struct Node\* createNode(int data) {

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

    newNode->data = data;

    newNode->next = newNode;

    newNode->prev = newNode;

    return newNode;

}

// Function to display all elements in the circular doubly linked list

void display(struct Node\* last) {

    if (last == NULL) {

        printf("List is empty.\n");

        return;

    }

    struct Node\* temp = last->next;

    do {

        printf("%d ", temp->data);

        temp = temp->next;

    } while (temp != last->next);

    printf("\n");

}

// Function to insert a node at the beginning of the circular doubly linked list

struct Node\* insertAtBeginning(struct Node\* last, int data) {

    struct Node\* newNode = createNode(data);

    if (last == NULL) {

        return newNode;

    }

    newNode->next = last->next;

    newNode->prev = last;

    last->next->prev = newNode;

    last->next = newNode;

    return last;

}

// Function to insert a node at the end of the circular doubly linked list

struct Node\* insertAtEnd(struct Node\* last, int data) {

    struct Node\* newNode = createNode(data);

    if (last == NULL) {

        return newNode;

    }

    newNode->next = last->next;

    newNode->prev = last;

    last->next->prev = newNode;

    last->next = newNode;

    last = newNode;

    return last;

}

// Function to delete a node from the beginning of the circular doubly linked list

struct Node\* deleteFromBeginning(struct Node\* last) {

    if (last == NULL) {

        printf("List is empty.\n");

        return NULL;

    }

    struct Node\* temp = last->next;

    if (last == temp) {

        free(temp);

        return NULL;

    }

    last->next = temp->next;

    temp->next->prev = last;

    free(temp);

    return last;

}

// Function to delete a node from the end of the circular doubly linked list

struct Node\* deleteFromEnd(struct Node\* last) {

    if (last == NULL) {

        printf("List is empty.\n");

        return NULL;

    }

    struct Node\* temp = last;

    if (last->next == last) {

        free(last);

        return NULL;

    }

    last->prev->next = last->next;

    last->next->prev = last->prev;

    last = last->prev;

    free(temp);

    return last;

}

// Function to delete a node after a given node by value

struct Node\* deleteAfterNode(struct Node\* last, int value) {

    if (last == NULL) {

        printf("List is empty.\n");

        return NULL;

    }

    struct Node\* temp = last->next;

    do {

        if (temp->data == value) {

            struct Node\* nodeToDelete = temp->next;

            if (nodeToDelete == last) {

                last = temp;

            }

            temp->next = nodeToDelete->next;

            nodeToDelete->next->prev = temp;

            free(nodeToDelete);

            return last;

        }

        temp = temp->next;

    } while (temp != last->next);

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

    return last;

}

// Function to delete the entire circular doubly linked list

struct Node\* deleteList(struct Node\* last) {

    if (last == NULL) return NULL;

    struct Node\* current = last->next;

    while (current != last) {

        struct Node\* temp = current;

        current = current->next;

        free(temp);

    }

    free(last);

    printf("Entire list deleted.\n");

    return NULL;

}

int main() {

    struct Node\* last = NULL;

    int choice, data, value;

    do {

        printf("\nCircular Doubly Linked List Operations:\n");

        printf("1. Create circular doubly linked list\n");

        printf("2. Display elements\n");

        printf("3. Insert at beginning\n");

        printf("4. Insert at end\n");

        printf("5. Delete from beginning\n");

        printf("6. Delete from end\n");

        printf("7. Delete after a node\n");

        printf("8. Delete entire list\n");

        printf("9. Exit\n");

        printf("Enter your choice: ");

        scanf("%d", &choice);

        switch (choice) {

            case 1:

                printf("Enter data to create list: ");

                scanf("%d", &data);

                last = createNode(data);

                break;

            case 2:

                display(last);

                break;

            case 3:

                printf("Enter data to insert at beginning: ");

                scanf("%d", &data);

                last = insertAtBeginning(last, data);

                break;

            case 4:

                printf("Enter data to insert at end: ");

                scanf("%d", &data);

                last = insertAtEnd(last, data);

                break;

            case 5:

                last = deleteFromBeginning(last);

                break;

            case 6:

                last = deleteFromEnd(last);

                break;

            case 7:

                printf("Enter value after which to delete: ");

                scanf("%d", &value);

                last = deleteAfterNode(last, value);

                break;

            case 8:

                last = deleteList(last);

                break;

            case 9:

                printf("Exiting program.\n");

                break;

            default:

                printf("Invalid choice. Try again.\n");

        }

    } while (choice != 9);

    return 0;

}

**Output:**

