LIST OF PROGRAMS AND SOLUTIONS

Assignment –1

1. Write a C program to print an array.
2. Write a C program to check whether a given string is Palindrome or not.
3. Write a C program to convert temperature from degree Centigrade to Fahrenheit.
4. Write a C program to sort an array.
5. Write a C program to print the largest and second largest element of the array.
6. Write a C program to display Fibonacci series.
7. 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.

8. Write a program that reads a 2D metrics and checks if the metrics is a symmetric metrics or not.
9. Write a C program to print reverse array.
10. Write a C program to check the sum of all elements of an array.
11. Write a C program to check duplicate number in an array.

//1. Print an Array

```
#include <stdio.h>

int main() {
    int arr[] = {10, 20, 30, 40, 50};
    int n = sizeof(arr)/sizeof(arr[0]);

    printf("Array elements: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Code Input:
int arr[] = {10, 20, 30, 40, 50};
Code Output:
Array elements: 10 20 30 40 50

// 2. Check Palindrome String

```
#include <stdio.h>
#include <string.h>

int main() {
    char str[100], rev[100];
    printf("Enter a string: ");
    scanf("%s", str);

    strcpy(rev, str);
    strrev(rev);  // You can use custom reverse if strrev is not supported

    if(strcmp(str, rev) == 0)
        printf("Palindrome");
    else
        printf("Not a Palindrome");
    return 0;
}
```

Code Input:
Enter a string: madam
Code Output: Palindrome

Code Input:
Enter a string: hello
Code Output: Not a Palindrome

//3. Convert °C to °F

```
#include <stdio.h>

int main() {
    float celsius, fahrenheit;
```

```
    printf("Enter temperature in Celsius: ");
    scanf("%f", &celsius);

    fahrenheit = (celsius * 9/5) + 32;
    printf("Temperature in Fahrenheit: %.2f", fahrenheit);
    return 0;
}
```

Input:
Enter temperature in Celsius: 25
Output:
Temperature in Fahrenheit: 77.00

//4. Sort an Array

```
#include <stdio.h>

int main() {
    int arr[] = {5, 2, 8, 1, 9};
    int n = sizeof(arr)/sizeof(arr[0]);
    int temp;

    for(int i = 0; i < n-1; i++) {
        for(int j = i+1; j < n; j++) {
            if(arr[i] > arr[j]) {
                temp = arr[i];
                arr[i] = arr[j];
                arr[j] = temp;
            }
        }
    }

    printf("Sorted array: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Input Array:
int arr[] = {5, 2, 8, 1, 9};
Output:
Sorted array: 1 2 5 8 9

//5. Largest and Second Largest

```
#include <stdio.h>

int main() {
    int arr[] = {12, 45, 1, 67, 23};
    int n = sizeof(arr)/sizeof(arr[0]);
```

```
    int first = arr[0], second = -1;

    for(int i = 1; i < n; i++) {
        if(arr[i] > first) {
            second = first;
            first = arr[i];
        } else if(arr[i] > second && arr[i] != first) {
            second = arr[i];
        }
    }

    printf("Largest: %d\nSecond Largest: %d\n", first, second);
    return 0;
}
```

Input Array:

```
int arr[] = {12, 45, 1, 67, 23};
```

Output:

```
Largest: 67
Second Largest: 45
```

//6. Fibonacci Series

```
#include <stdio.h>

int main() {
    int n, a = 0, b = 1, next;
    printf("Enter number of terms: ");
    scanf("%d", &n);

    printf("Fibonacci Series: ");
    for(int i = 0; i < n; i++) {
        printf("%d ", a);
        next = a + b;
        a = b;
        b = next;
    }
    return 0;
}
```

Input:

```
Enter number of terms: 6
```

Output:

```
Fibonacci Series: 0 1 1 2 3 5
```

//7. Matrix Multiplication

```
#include <stdio.h>

int main() {
    int a[10][10], b[10][10], mul[10][10];
    int r1, c1, r2, c2;

    printf("Enter rows and cols of matrix A: ");
    scanf("%d%d", &r1, &c1);
```

```
    printf("Enter rows and cols of matrix B: ");
    scanf("%d%d", &r2, &c2);

    if(c1 != r2) {
        printf("Matrix multiplication not possible.\n");
        return 0;
    }

    printf("Enter matrix A:\n");
    for(int i = 0; i < r1; i++)
        for(int j = 0; j < c1; j++)
            scanf("%d", &a[i][j]);

    printf("Enter matrix B:\n");
    for(int i = 0; i < r2; i++)
        for(int j = 0; j < c2; j++)
            scanf("%d", &b[i][j]);

    // Multiplying
    for(int i = 0; i < r1; i++) {
        for(int j = 0; j < c2; j++) {
            mul[i][j] = 0;
            for(int k = 0; k < c1; k++) {
                mul[i][j] += a[i][k] * b[k][j];
            }
        }
    }

    printf("Resultant Matrix:\n");
    for(int i = 0; i < r1; i++) {
        for(int j = 0; j < c2; j++) {
            printf("%d ", mul[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

Sample Input:

```
Matrix A (2x3):
1 2 3
4 5 6

Matrix B (3x2):
1 4
2 5
3 6
```

Output:

```
Resultant Matrix:
14 32
32 77
```

//8. Symmetric Matrix

```
#include <stdio.h>
```

```
int main() {
   int a[10][10], n, isSymmetric = 1;

   printf("Enter order of square matrix: ");
   scanf("%d", &n);

   printf("Enter matrix elements:\n");
   for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
         scanf("%d", &a[i][j]);

   for(int i = 0; i < n; i++)
      for(int j = 0; j < n; j++)
         if(a[i][j] != a[j][i]) {
            isSymmetric = 0;
            break;
         }

   if(isSymmetric)
      printf("Matrix is symmetric.\n");
   else
      printf("Matrix is not symmetric.\n");

   return 0;
}
```

User Input:

```
Enter order of square matrix: 3

Matrix:
1 2 3
2 4 5
3 5 6
```

Output:

```
Matrix is symmetric.
```

Another Input (non-symmetric):

```
1 2 3
2 4 5
9 5 6
```

Output:

```
Matrix is not symmetric.
```

//9. Reverse Array

```
#include <stdio.h>

int main() {
   int arr[] = {1, 2, 3, 4, 5};
   int n = sizeof(arr)/sizeof(arr[0]);

   printf("Reversed array: ");
   for(int i = n - 1; i >= 0; i--) {
```

```
        printf("%d ", arr[i]);
    }
    return 0;
}
```

Input:

```
int arr[] = {1, 2, 3, 4, 5};
```

Output:

```
Reversed array: 5 4 3 2 1
```

//10. Sum of Array Elements

```
#include <stdio.h>

int main() {
    int arr[] = {4, 7, 2, 9, 5};
    int n = sizeof(arr)/sizeof(arr[0]), sum = 0;

    for(int i = 0; i < n; i++) {
        sum += arr[i];
    }

    printf("Sum of elements: %d", sum);
    return 0;
}
```

Input:

```
int arr[] = {4, 7, 2, 9, 5};
```

Output:

```
Sum of elements: 27
```

//11. Find Duplicates in Array

```
#include <stdio.h>

int main() {
    int arr[] = {2, 5, 3, 2, 6, 5, 7};
    int n = sizeof(arr)/sizeof(arr[0]);
    int found = 0;

    printf("Duplicate elements: ");
    for(int i = 0; i < n; i++) {
        for(int j = i+1; j < n; j++) {
            if(arr[i] == arr[j]) {
                printf("%d ", arr[i]);
                break;
            }
```

```
        }
    }
    return 0;
}
```

Input:

```
int arr[] = {2, 5, 3, 2, 6, 5, 7};
```

Output:

```
Duplicate elements: 2 5
```

Assignment -2

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.
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.
3. Write a C program to pass an array to a function using Call by Reference, update the array
   values in the function, print the array elements both in the function and in the calling function.
4. Write a program to display n number of elements. Memory should be allocated dynamically
   using malloc( ).
5. Write a program to display n number of elements. Memory should be allocated dynamically
   using calloc( ).
6. 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.
7. 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.
8. Write a C program to search an element in an Array using dynamic memory allocation.

//1. Represent Sparse Matrix

```
#include <stdio.h>

int main() {
   int i, j, rows, cols, k = 1;
   int mat[10][10], sparse[100][3];

   printf("Enter rows and columns: ");
   scanf("%d%d", &rows, &cols);

   printf("Enter matrix elements:\n");
   for(i = 0; i < rows; i++)
      for(j = 0; j < cols; j++)
         scanf("%d", &mat[i][j]);

   // First row of sparse matrix
   sparse[0][0] = rows;
   sparse[0][1] = cols;
   sparse[0][2] = 0; // will hold number of non-zero elements

   for(i = 0; i < rows; i++) {
      for(j = 0; j < cols; j++) {
         if(mat[i][j] != 0) {
            sparse[k][0] = i;
            sparse[k][1] = j;
            sparse[k][2] = mat[i][j];
            k++;
         }
      }
   }
   sparse[0][2] = k - 1;

   printf("\nSparse Matrix:\n");
   for(i = 0; i < k; i++)
      printf("%d %d %d\n", sparse[i][0], sparse[i][1], sparse[i][2]);

   return 0;
}
```

Sample Input:

```
3 3
0 0 5
0 0 0
1 0 0
```

Output:

```
Sparse Matrix:
3 3 2
0 2 5
2 0 1
```

//2. Call by Value (Array cannot be modified)

```
#include <stdio.h>

void update(int arr[], int size) {
   printf("Inside function:\n");
   for(int i = 0; i < size; i++) {
      arr[i] += 10;
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int arr[] = {1, 2, 3, 4, 5};
   int size = sizeof(arr)/sizeof(arr[0]);

   update(arr, size); // actually still works as pointer gets passed

   printf("Inside main:\n");
   for(int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }

   return 0;
}
```

Output:

```
Inside function:
11 12 13 14 15
Inside main:
11 12 13 14 15
```

//3. Call by Reference

```
#include <stdio.h>

void updateArray(int *arr, int size) {
   printf("Inside function:\n");
   for(int i = 0; i < size; i++) {
      arr[i] *= 2;
      printf("%d ", arr[i]);
   }
   printf("\n");
}

int main() {
   int arr[] = {2, 4, 6, 8};
   int size = sizeof(arr)/sizeof(arr[0]);

   updateArray(arr, size);

   printf("Inside main:\n");
   for(int i = 0; i < size; i++) {
      printf("%d ", arr[i]);
   }
```

```
    return 0;
}
```
Output:

```
Inside function:
4 8 12 16
Inside main:
4 8 12 16
```

4. Dynamic Allocation using malloc()

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr, n;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    arr = (int *)malloc(n * sizeof(int));

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

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

    free(arr);
    return 0;
}
```
Sample Input:

```
Enter number of elements: 3
1 2 3
```
Output:

```
You entered:
1 2 3
```

//5. Dynamic Allocation using calloc()

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
   int *arr, n;

   printf("Enter number of elements: ");
   scanf("%d", &n);

   arr = (int *)calloc(n, sizeof(int));

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

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

   free(arr);
   return 0;
}
```

Sample Input:

```
4
10 20 30 40
```

Output:

```
You entered:
10 20 30 40
```

//6. malloc() and realloc()

```
#include <stdio.h>
#include <stdlib.h>

int main() {
   int *arr, n;

   printf("Enter initial number of elements: ");
   scanf("%d", &n);

   arr = (int *)malloc(n * sizeof(int));

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

   printf("Enter new size: ");
   scanf("%d", &n);

   arr = (int *)realloc(arr, n * sizeof(int));

   printf("Enter new elements:\n");
   for(int i = 0; i < n; i++)
      scanf("%d", &arr[i]);

   printf("Final array:\n");
   for(int i = 0; i < n; i++)
```

```
        printf("%d ", arr[i]);
    free(arr);
    return 0;
}
```

Sample Input/Output:

```
Enter initial number of elements: 2
10 20
Enter new size: 4
30 40 50 60
Final array:
30 40 50 60
```

//7. calloc() and realloc()

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    int *arr, n;

    printf("Enter initial number of elements: ");
    scanf("%d", &n);

    arr = (int *)calloc(n, sizeof(int));

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

    printf("Enter new size: ");
    scanf("%d", &n);

    arr = (int *)realloc(arr, n * sizeof(int));

    printf("Enter new elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

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

    free(arr);
    return 0;
}
```

Sample Output:

```
Final array:
11 22 33 44 55
```

//8. Search Element Using Dynamic Allocation

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main() {
    int *arr, n, key, found = 0;

    printf("Enter number of elements: ");
    scanf("%d", &n);

    arr = (int *)malloc(n * sizeof(int));
    printf("Enter elements:\n");
    for(int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    printf("Enter element to search: ");
    scanf("%d", &key);

    for(int i = 0; i < n; i++) {
        if(arr[i] == key) {
            printf("Element found at index %d\n", i);
            found = 1;
            break;
        }
    }

    if(!found)
        printf("Element not found.\n");

    free(arr);
    return 0;
}
```

Sample Input:

5
10 20 30 40 50
Search 30

Output:

Element found at index 2

Assignment - 3

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.

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.

//1) Singly Linked List – Menu Driven Program

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
   int data;
   struct Node* next;
};

struct Node* head = NULL;

// Create and Insert at End
void create(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;
   }
}

void display() {
   if (head == NULL) {
      printf("List is empty.\n");
      return;
   }
   struct Node* temp = head;
   printf("List: ");
   while (temp != NULL) {
      printf("%d -> ", temp->data);
      temp = temp->next;
   }
   printf("NULL\n");
}

void insertAtBeginning(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
}

void insertAtEnd(int data) {
   create(data);
}

void insertBefore(int value, int data) {
   struct Node* temp = head, *prev = NULL;

   if (head == NULL) return;
```

```
    while (temp && temp->data != value) {
        prev = temp;
        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;

    if (prev == NULL)
        head = newNode;
    else
        prev->next = newNode;
}

void insertAfter(int value, int data) {
    struct Node* temp = head;

    while (temp && temp->data != value)
        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;
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    head = head->next;
    free(temp);
}

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;
}

void deleteAfter(int value) {
    struct Node* temp = head;

    while (temp && temp->data != value)
        temp = temp->next;

    if (temp == NULL || temp->next == NULL) {
        printf("Cannot delete.\n");
        return;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;
    free(toDelete);
}

void deleteEntireList() {
    struct Node* temp;
    while (head) {
        temp = head;
        head = head->next;
        free(temp);
    }
}

int main() {
    int choice, value, target;

    while (1) {
        printf("\n--- Menu ---\n");
        printf("1. Create\n2. Display\n3. Insert at Beginning\n4. Insert at End\n5. Insert Before\n");
        printf("6. Insert After\n7. Delete from Beginning\n8. Delete from End\n9. Delete After\n10.
Delete Entire List\n11. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter data: ");
                scanf("%d", &value);
                create(value);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Enter data: ");
                scanf("%d", &value);
                insertAtBeginning(value);
```

```
            break;
        case 4:
            printf("Enter data: ");
            scanf("%d", &value);
            insertAtEnd(value);
            break;
        case 5:
            printf("Enter target node value: ");
            scanf("%d", &target);
            printf("Enter new node data: ");
            scanf("%d", &value);
            insertBefore(target, value);
            break;
        case 6:
            printf("Enter target node value: ");
            scanf("%d", &target);
            printf("Enter new node data: ");
            scanf("%d", &value);
            insertAfter(target, value);
            break;
        case 7:
            deleteFromBeginning();
            break;
        case 8:
            deleteFromEnd();
            break;
        case 9:
            printf("Enter value after which to delete: ");
            scanf("%d", &target);
            deleteAfter(target);
            break;
        case 10:
            deleteEntireList();
            printf("All nodes deleted.\n");
            break;
        case 11:
            return 0;
        default:
            printf("Invalid choice!\n");
    }
  }
}
```

Input sequence (example):

1. Create (10)
1. Create (20)
4. Insert at End (30)
3. Insert at Beginning (5)
5. Insert Before (20, 15)
6. Insert After (15, 17)
2. Display
7. Delete from Beginning
8. Delete from End
9. Delete After (15)
2. Display
10. Delete Entire List
2. Display
11. Exit

Expected Output:

--- Menu ---
1. Create
2. Display

Enter choice: 1
Enter data: 10

Enter choice: 1
Enter data: 20

Enter choice: 4
Enter data: 30

Enter choice: 3
Enter data: 5

Enter choice: 5
Enter target node value: 20
Enter new node data: 15

Enter choice: 6
Enter target node value: 15
Enter new node data: 17

Enter choice: 2
List: 5 -> 10 -> 15 -> 17 -> 20 -> 30 -> NULL

Enter choice: 7
(deletes 5)

Enter choice: 8
(deletes 30)

Enter choice: 9
Enter value after which to delete: 15
(deletes 17)

Enter choice: 2
List: 10 -> 15 -> 20 -> NULL

Enter choice: 10
All nodes deleted.

Enter choice: 2
List is empty.

//2. Circular Linked List – Menu Driven Program

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* next;
};
```

```
struct Node* tail = NULL;

void create(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;

   if (tail == NULL) {
      tail = newNode;
      tail->next = tail;
   } else {
      newNode->next = tail->next;
      tail->next = newNode;
      tail = newNode;
   }
}

void display() {
   if (tail == NULL) {
      printf("List is empty.\n");
      return;
   }

   struct Node* temp = tail->next;
   printf("List: ");
   do {
      printf("%d -> ", temp->data);
      temp = temp->next;
   } while (temp != tail->next);
   printf("(back to head)\n");
}

void insertAtBeginning(int data) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = data;

   if (tail == NULL) {
      tail = newNode;
      tail->next = tail;
   } else {
      newNode->next = tail->next;
      tail->next = newNode;
   }
}

void insertAtEnd(int data) {
   create(data);
}

void deleteFromBeginning() {
   if (tail == NULL) return;

   struct Node* head = tail->next;
   if (head == tail) {
      free(head);
      tail = NULL;
   } else {
      tail->next = head->next;
      free(head);
```

```
    }
}

void deleteFromEnd() {
    if (tail == NULL) return;

    struct Node *temp = tail->next, *prev = NULL;

    if (tail->next == tail) {
        free(tail);
        tail = NULL;
        return;
    }

    while (temp->next != tail->next) {
        prev = temp;
        temp = temp->next;
    }

    prev->next = tail->next;
    free(tail);
    tail = prev;
}

void deleteAfter(int value) {
    if (tail == NULL) return;

    struct Node* temp = tail->next;

    do {
        if (temp->data == value) {
            struct Node* delNode = temp->next;
            if (delNode == tail)
                tail = temp;
            temp->next = delNode->next;
            free(delNode);
            return;
        }
        temp = temp->next;
    } while (temp != tail->next);

    printf("Node not found.\n");
}

void deleteAll() {
    if (tail == NULL) return;

    struct Node* temp = tail->next;
    while (temp != tail) {
        struct Node* next = temp->next;
        free(temp);
        temp = next;
    }
    free(tail);
    tail = NULL;
}

int main() {
```

```
    int choice, value;

    while (1) {
        printf("\n--- Circular Linked List Menu ---\n");
        printf("1. Create\n2. Display\n3. Insert at Beginning\n4. Insert at End\n5. Delete from
Beginning\n");
        printf("6. Delete from End\n7. Delete After Node\n8. Delete Entire List\n9. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                create(value);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 4:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 5:
                deleteFromBeginning();
                break;
            case 6:
                deleteFromEnd();
                break;
            case 7:
                printf("Enter node value after which to delete: ");
                scanf("%d", &value);
                deleteAfter(value);
                break;
            case 8:
                deleteAll();
                printf("List deleted.\n");
                break;
            case 9:
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }
}
```

Input sequence (example):

1. Create (100)
1. Create (200)
4. Insert at End (300)

3. Insert at Beginning (50)
2. Display
5. Delete from Beginning
6. Delete from End
7. Delete After (100)
2. Display
8. Delete Entire List
2. Display
9. Exit

Expected Output:

--- Circular Linked List Menu ---
1. Create
2. Display
...
Enter choice: 1
Enter value: 100

Enter choice: 1
Enter value: 200

Enter choice: 4
Enter value: 300

Enter choice: 3
Enter value: 50

Enter choice: 2
List: 50 -> 100 -> 200 -> 300 -> (back to head)

Enter choice: 5
(deletes 50)

Enter choice: 6
(deletes 300)

Enter choice: 7
Enter node value after which to delete: 100
(deletes 200)

Enter choice: 2
List: 100 -> (back to head)

Enter choice: 8
List deleted.

Enter choice: 2
List is empty.

Assignment –4

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.

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

//1. Doubly Linked List – Full C Code

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;

void create(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = NULL;

    if (head == NULL)
        head = newNode;
    else {
        struct Node* temp = head;
        while (temp->next != NULL)
            temp = temp->next;

        temp->next = newNode;
        newNode->prev = temp;
    }
}

void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    struct Node* temp = head;
    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = head;

    if (head != NULL)
        head->prev = newNode;

    head = newNode;
}

void insertAtEnd(int value) {
```

```
    create(value);
}

void insertBefore(int target, int value) {
    struct Node* temp = head;

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

    if (temp == NULL) {
        printf("Target node not found.\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = temp;
    newNode->prev = temp->prev;

    if (temp->prev != NULL)
        temp->prev->next = newNode;
    else
        head = newNode;

    temp->prev = newNode;
}

void insertAfter(int target, int value) {
    struct Node* temp = head;

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

    if (temp == NULL) {
        printf("Target node not found.\n");
        return;
    }

    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = temp->next;
    newNode->prev = temp;

    if (temp->next != NULL)
        temp->next->prev = newNode;

    temp->next = newNode;
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    head = head->next;
```

```
    if (head != NULL)
        head->prev = NULL;

    free(temp);
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;

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

    if (temp->prev != NULL)
        temp->prev->next = NULL;
    else
        head = NULL;

    free(temp);
}

void deleteAfter(int target) {
    struct Node* temp = head;

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

    if (temp == NULL || temp->next == NULL) {
        printf("Cannot delete after this node.\n");
        return;
    }

    struct Node* toDelete = temp->next;
    temp->next = toDelete->next;

    if (toDelete->next != NULL)
        toDelete->next->prev = temp;

    free(toDelete);
}

void deleteAll() {
    struct Node* temp = head;
    while (temp != NULL) {
        struct Node* next = temp->next;
        free(temp);
        temp = next;
    }
    head = NULL;
    printf("All nodes deleted.\n");
}

int main() {
    int choice, value, target;
```

```
    while (1) {
        printf("\n--- Doubly Linked List Menu ---\n");
        printf("1. Create\n2. Display\n3. Insert at Beginning\n4. Insert at End\n5. Insert
Before\n");
        printf("6. Insert After\n7. Delete from Beginning\n8. Delete from End\n9. Delete
After\n10. Delete All\n11. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                create(value);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 4:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 5:
                printf("Enter target node value: ");
                scanf("%d", &target);
                printf("Enter value to insert before it: ");
                scanf("%d", &value);
                insertBefore(target, value);
                break;
            case 6:
                printf("Enter target node value: ");
                scanf("%d", &target);
                printf("Enter value to insert after it: ");
                scanf("%d", &value);
                insertAfter(target, value);
                break;
            case 7:
                deleteFromBeginning();
                break;
            case 8:
                deleteFromEnd();
                break;
            case 9:
                printf("Enter value after which to delete: ");
                scanf("%d", &target);
                deleteAfter(target);
                break;
            case 10:
                deleteAll();
                break;
            case 11:
```

```
            return 0;
        default:
            printf("Invalid choice.\n");
    }
  }
}
```

Example Input Steps:

1. Create (10)
1. Create (20)
3. Insert at Beginning (5)
4. Insert at End (30)
5. Insert Before (20, 15)
6. Insert After (15, 17)
2. Display
7. Delete from Beginning
8. Delete from End
9. Delete After (15)
2. Display
10. Delete All
2. Display
11. Exit

Expected Output:

--- Doubly Linked List Menu ---
Enter value: 10
Enter value: 20
Enter value: 5
Enter value: 30
Enter target node value: 20
Enter value to insert before it: 15
Enter target node value: 15
Enter value to insert after it: 17

Doubly Linked List: 5 <-> 10 <-> 15 <-> 17 <-> 20 <-> 30 <-> NULL

(Deletes 5 from beginning)
(Deletes 30 from end)
(Deletes 17 which is after 15)

Doubly Linked List: 10 <-> 15 <-> 20 <-> NULL

(All nodes deleted)

List is empty.

//2. Circular Doubly Linked List – Full C Code

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
    int data;
    struct Node* prev;
    struct Node* next;
```

```
};

struct Node* head = NULL;

void create(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;

    if (head == NULL) {
        newNode->next = newNode->prev = newNode;
        head = newNode;
    } else {
        struct Node* tail = head->prev;

        newNode->next = head;
        newNode->prev = tail;
        tail->next = newNode;
        head->prev = newNode;
    }
}

void display() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    printf("Circular Doubly Linked List: ");
    do {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    } while (temp != head);
    printf("(back to head)\n");
}

void insertAtBeginning(int value) {
    create(value);
    head = head->prev; // move head to the new node
}

void insertAtEnd(int value) {
    create(value);
}

void deleteFromBeginning() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if (head->next == head) {
        free(head);
        head = NULL;
    } else {
        struct Node* tail = head->prev;
        struct Node* temp = head;
```

```
        head = head->next;
        tail->next = head;
        head->prev = tail;

        free(temp);
    }
}

void deleteFromEnd() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    if (head->next == head) {
        free(head);
        head = NULL;
    } else {
        struct Node* tail = head->prev;
        struct Node* newTail = tail->prev;

        newTail->next = head;
        head->prev = newTail;

        free(tail);
    }
}

void deleteAfter(int target) {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    do {
        if (temp->data == target) {
            struct Node* delNode = temp->next;

            if (delNode == head) {
                printf("Cannot delete head using this method.\n");
                return;
            }

            temp->next = delNode->next;
            delNode->next->prev = temp;
            free(delNode);
            return;
        }
        temp = temp->next;
    } while (temp != head);

    printf("Node not found.\n");
}

void deleteAll() {
    if (head == NULL) {
        printf("List already empty.\n");
```

```
        return;
    }

    struct Node* current = head->next;
    while (current != head) {
        struct Node* next = current->next;
        free(current);
        current = next;
    }

    free(head);
    head = NULL;

    printf("All nodes deleted.\n");
}

int main() {
    int choice, value, target;

    while (1) {
        printf("\n--- Circular Doubly Linked List Menu ---\n");
        printf("1. Create\n2. Display\n3. Insert at Beginning\n4. Insert at End\n");
        printf("5. Delete from Beginning\n6. Delete from End\n7. Delete After Node\n");
        printf("8. Delete Entire List\n9. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value: ");
                scanf("%d", &value);
                create(value);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Enter value: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 4:
               printf("Enter value: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 5:
                deleteFromBeginning();
                break;
            case 6:
                deleteFromEnd();
                break;
            case 7:
                printf("Enter node value after which to delete: ");
                scanf("%d", &target);
                deleteAfter(target);
                break;
```

```
            case 8:
                deleteAll();
                break;
            case 9:
                return 0;
            default:
                printf("Invalid choice.\n");
        }
    }
}
```

Sample Input Sequence:

1. Create (10)
1. Create (20)
3. Insert at Beginning (5)
4. Insert at End (30)
2. Display
5. Delete from Beginning
6. Delete from End
7. Delete After (10)
2. Display
8. Delete Entire List
2. Display
9. Exit

Output:

```
--- Circular Doubly Linked List Menu ---
Enter value: 10
Enter value: 20
Enter value: 5
Enter value: 30

Circular Doubly Linked List: 5 <-> 10 <-> 20 <-> 30 <-> (back to head)
(After deleting from beginning)
(After deleting from end)
(After deleting node after 10)
Circular Doubly Linked List: 10 <-> 30 <-> (back to head)
All nodes deleted.
 List is empty.
```

Assignment –5

1. Write a Menu driven C program to accomplish the following functionalities in Queue using an Array:

a. Insert an element into the queue using an array (Enqueue Operation).
b. Delete an element from the queue using an array (Dequeue Operation).
c. Return the value of the FRONT element of the queue (without deleting it from the queue) using an array (Peep operation).
d. Display the elements of a queue using an array.

2. Write a Menu driven C program to accomplish the following functionalities in Queue using an Array:

e. Insert an element into the queue using a Linked List (Enqueue Operation)
f. Delete an element from the queue using a Linked List (Dequeue Operation).
g. Return the value of the FRONT element of the queue (without deleting it from the queue) using a Linked List (Peep operation).
h. Display the elements of a queue using a Linked List.

3. Write a Menu driven C program to accomplish the following functionalities in Circular Queue using Array:

i. Insert an element into the circular queue.
j. Delete an element from the circular queue.
k. Return the value of the FRONT element of the circular queue (without deleting it from the queue).
l. Display the elements of a circular queue using the circular queue.

//1.Queue Using Array – Full C Code

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

void enqueue(int value) {
    if (rear == SIZE - 1) {
        printf("Queue is full (Overflow).\n");
        return;
    }
    if (front == -1) front = 0;
    rear++;
    queue[rear] = value;
    printf("%d inserted into the queue.\n", value);
}

void dequeue() {
    if (front == -1 || front > rear) {
        printf("Queue is empty (Underflow).\n");
        return;
    }
    printf("Deleted element: %d\n", queue[front]);
    front++;
}

void peep() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Front element: %d\n", queue[front]);
}

void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- Queue Using Array Menu ---\n");
        printf("1. Enqueue (Insert)\n");
        printf("2. Dequeue (Delete)\n");
        printf("3. Peep (Front Element)\n");
        printf("4. Display Queue\n");
        printf("5. Exit\n");
```

```
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                peep();
                break;
            case 4:
                display();
                break;
            case 5:
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

Input Sequence (Example):

1. Enqueue (10)
1. Enqueue (20)
1. Enqueue (30)
4. Display
3. Peep
2. Dequeue
4. Display
5. Exit

Output:

```
Enter value to insert: 10
10 inserted into the queue.

Enter value to insert: 20
20 inserted into the queue.

Enter value to insert: 30
30 inserted into the queue.

Queue elements: 10 20 30

Front element: 10
Deleted element: 10
Queue elements: 20 30
```

//2. Queue Using Linked List – Full C Code

```
#include <stdio.h>
#include <stdlib.h>
```

```
// Define node structure
struct Node {
    int data;
    struct Node* next;
};

// Define front and rear pointers
struct Node* front = NULL;
struct Node* rear = NULL;

// Enqueue operation
void enqueue(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }

    printf("%d inserted into the queue.\n", value);
}

// Dequeue operation
void dequeue() {
    if (front == NULL) {
        printf("Queue is empty (Underflow).\n");
        return;
    }

    struct Node* temp = front;
    printf("Deleted element: %d\n", front->data);
    front = front->next;

    if (front == NULL)
        rear = NULL;

    free(temp);
}

// Peep operation
void peep() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    printf("Front element: %d\n", front->data);
}

// Display operation
void display() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
```

```
    }

    struct Node* temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

// Main menu
int main() {
    int choice, value;

    while (1) {
        printf("\n--- Queue Using Linked List Menu ---\n");
        printf("1. Enqueue (Insert)\n");
        printf("2. Dequeue (Delete)\n");
        printf("3. Peep (Front Element)\n");
        printf("4. Display Queue\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                enqueue(value);
                break;
            case 2:
                dequeue();
                break;
            case 3:
                peep();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

Input Sequence (Example):

1. Enqueue (10)
1. Enqueue (20)
1. Enqueue (30)
4. Display
3. Peep
2. Dequeue

4. Display
5. Exit

Output:

10 inserted into the queue.
20 inserted into the queue.
30 inserted into the queue.

Queue elements: 10 20 30

Front element: 10

Deleted element: 10

Queue elements: 20 30

Exiting program...

//3. Circular Queue Using Array – Full C Code

```
#include <stdio.h>
#define SIZE 5

int queue[SIZE];
int front = -1, rear = -1;

// Check if queue is full
int isFull() {
    return (front == (rear + 1) % SIZE);
}

// Check if queue is empty
int isEmpty() {
    return (front == -1);
}

// Insert (Enqueue)
void enqueue(int value) {
    if (isFull()) {
        printf("Queue is full (Overflow).\n");
        return;
    }

    if (isEmpty()) {
        front = rear = 0;
    } else {
        rear = (rear + 1) % SIZE;
    }

    queue[rear] = value;
    printf("%d inserted into circular queue.\n", value);
}

// Delete (Dequeue)
void dequeue() {
    if (isEmpty()) {
```

```
        printf("Queue is empty (Underflow).\n");
        return;
    }

    printf("Deleted element: %d\n", queue[front]);

    if (front == rear) {
        front = rear = -1; // Queue becomes empty
    } else {
        front = (front + 1) % SIZE;
    }
}

// Peep (View Front)
void peep() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Front element: %d\n", queue[front]);
}

// Display
void display() {
    if (isEmpty()) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Circular Queue elements: ");
    int i = front;
    while (1) {
        printf("%d ", queue[i]);
        if (i == rear)
            break;
        i = (i + 1) % SIZE;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\n--- Circular Queue Using Array Menu ---\n");
        printf("1. Enqueue (Insert)\n");
        printf("2. Dequeue (Delete)\n");
        printf("3. Peep (Front Element)\n");
        printf("4. Display Queue\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
```

```
            scanf("%d", &value);
            enqueue(value);
            break;
        case 2:
            dequeue();
            break;
        case 3:
            peep();
            break;
        case 4:
            display();
            break;
        case 5:
            printf("Exiting program...\n");
            return 0;
        default:
            printf("Invalid choice! Try again.\n");
    }
  }
}
```

Sample Input:

1. Enqueue 10
1. Enqueue 20
1. Enqueue 30
1. Enqueue 40
1. Enqueue 50
4. Display
2. Dequeue
2. Dequeue
1. Enqueue 60
1. Enqueue 70
4. Display
3. Peep
5. Exit

Output:

10 inserted into circular queue.
20 inserted into circular queue.
30 inserted into circular queue.
40 inserted into circular queue.
50 inserted into circular queue.
Circular Queue elements: 10 20 30 40 50

Deleted element: 10
Deleted element: 20

60 inserted into circular queue.
70 inserted into circular queue.

Circular Queue elements: 30 40 50 60 70
Front element: 30
Exiting program...

Assignment – 6

1. Write a Menu driven C program to accomplish the following functionalities in Stack using an Array:
   a. Insert an element into the stack using an array (Push Operation).
   b. Delete an element from the stack using an array (Pop Operation).
   c. Return the value of the topmost element of the stack (without deleting it from the stack) using an array.
   d. Display the elements of a stack using an array.

2. Write a Menu driven C program to accomplish the following functionalities in Stack using Linked List:

   a. Insert an element into the stack using a Linked List (Push Operation).
   b. Delete an element from the stack using a Linked List (Pop Operation).
   c. Return the value of the topmost element of the stack (without deleting it from the stack) using a Linked List.
   d. Display the elements of the stack using a Linked List.

3. Write a program to convert an infix expression into its equivalent postfix notation.
4. Write a program to convert an infix expression into its equivalent prefix notation.
5. Write a program to evaluate a postfix expression.
6. Write a program to evaluate a prefix expression.
7. Write a program to print the Fibonacci series using recursion.
8. Write a program to solve the tower of Hanoi problem using recursion.

//1. Stack Using Array – Full C Code

```
#include <stdio.h>
#define SIZE 5

int stack[SIZE];
int top = -1;

// Push operation
void push(int value) {
    if (top == SIZE - 1) {
        printf("Stack is full (Overflow).\n");
    } else {
        stack[++top] = value;
        printf("%d pushed to the stack.\n", value);
    }
}

// Pop operation
void pop() {
    if (top == -1) {
        printf("Stack is empty (Underflow).\n");
    } else {
        printf("Popped element: %d\n", stack[top--]);
    }
}

// Peek operation
void peek() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Top element: %d\n", stack[top]);
    }
}

// Display operation
void display() {
    if (top == -1) {
        printf("Stack is empty.\n");
    } else {
        printf("Stack elements (Top to Bottom): ");
        for (int i = top; i >= 0; i--) {
            printf("%d ", stack[i]);
        }
        printf("\n");
    }
}

// Main menu
int main() {
    int choice, value;

    while (1) {
        printf("\n--- Stack Using Array Menu ---\n");
        printf("1. Push (Insert)\n");
        printf("2. Pop (Delete)\n");
        printf("3. Peek (Top Element)\n");
```

```
        printf("4. Display Stack\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                peek();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

Sample Input Sequence:

1. Push 10
1. Push 20
1. Push 30
4. Display
3. Peek
2. Pop
4. Display
5. Exit

Output:

10 pushed to the stack.
20 pushed to the stack.
30 pushed to the stack.
Stack elements (Top to Bottom): 30 20 10

Top element: 30

Popped element: 30
Stack elements (Top to Bottom): 20 10

Exiting program...

//2. Stack Using Linked List – Full C Code

```
#include <stdio.h>
#include <stdlib.h>
```

```
// Define node structure
struct Node {
    int data;
    struct Node* next;
};

struct Node* top = NULL;

// Push operation
void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    if (!newNode) {
        printf("Memory allocation failed.\n");
        return;
    }
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("%d pushed to the stack.\n", value);
}

// Pop operation
void pop() {
    if (top == NULL) {
        printf("Stack is empty (Underflow).\n");
        return;
    }
    struct Node* temp = top;
    printf("Popped element: %d\n", top->data);
    top = top->next;
    free(temp);
}

// Peek operation
void peek() {
    if (top == NULL) {
        printf("Stack is empty.\n");
    } else {
        printf("Top element: %d\n", top->data);
    }
}

// Display operation
void display() {
    if (top == NULL) {
        printf("Stack is empty.\n");
        return;
    }
    struct Node* temp = top;
    printf("Stack elements (Top to Bottom): ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
```

```
// Main function with menu
int main() {
    int choice, value;

    while (1) {
        printf("\n--- Stack Using Linked List Menu ---\n");
        printf("1. Push (Insert)\n");
        printf("2. Pop (Delete)\n");
        printf("3. Peek (Top Element)\n");
        printf("4. Display Stack\n");
        printf("5. Exit\n");

        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                pop();
                break;
            case 3:
                peek();
                break;
            case 4:
                display();
                break;
            case 5:
                printf("Exiting program...\n");
                return 0;
            default:
                printf("Invalid choice! Try again.\n");
        }
    }
}
```

Input Steps:

1. Push 10
1. Push 20
1. Push 30
4. Display
3. Peek
2. Pop
4. Display
5. Exit

Output:

10 pushed to the stack.
20 pushed to the stack.
30 pushed to the stack.

Stack elements (Top to Bottom): 30 20 10

Top element: 30

Popped element: 30

Stack elements (Top to Bottom): 20 10

Exiting program...

//3. Convert Infix to Postfix

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define SIZE 100
char stack[SIZE];
int top = -1;

void push(char ch) {
    stack[++top] = ch;
}

char pop() {
    return stack[top--];
}

int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}

void infixToPostfix(char infix[]) {
    char postfix[SIZE];
    int j = 0;
    for (int i = 0; infix[i]; i++) {
        char ch = infix[i];
        if (isalnum(ch))
            postfix[j++] = ch;
        else if (ch == '(')
            push(ch);
        else if (ch == ')') {
            while (stack[top] != '(')
                postfix[j++] = pop();
            pop(); // remove '('
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch))
                postfix[j++] = pop();
            push(ch);
        }
    }
    while (top != -1)
        postfix[j++] = pop();
    postfix[j] = '\0';
    printf("Postfix: %s\n", postfix);
```

```
}

int main() {
    char infix[SIZE];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixToPostfix(infix);
    return 0;
}
```

Input:

Enter infix expression: a+b*c

Output:

Postfix: abc*+

// 4. Convert Infix to Prefix

```
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define SIZE 100
char stack[SIZE];
int top = -1;

void push(char ch) { stack[++top] = ch; }
char pop() { return stack[top--]; }

int precedence(char op) {
    if (op == '^') return 3;
    if (op == '*' || op == '/') return 2;
    if (op == '+' || op == '-') return 1;
    return 0;
}

void reverse(char str[]) {
    int len = strlen(str);
    for (int i = 0; i < len / 2; i++) {
        char tmp = str[i];
        str[i] = str[len - i - 1];
        str[len - i - 1] = tmp;
    }
}

void infixToPrefix(char infix[]) {
    reverse(infix);
    for (int i = 0; infix[i]; i++) {
        if (infix[i] == '(') infix[i] = ')';
        else if (infix[i] == ')') infix[i] = '(';
    }

    char prefix[SIZE], ch;
    int j = 0;
    for (int i = 0; infix[i]; i++) {
        ch = infix[i];
        if (isalnum(ch)) prefix[j++] = ch;
```

```
        else if (ch == '(') push(ch);
        else if (ch == ')') {
            while (top != -1 && stack[top] != '(')
                prefix[j++] = pop();
            pop(); // remove '('
        } else {
            while (top != -1 && precedence(stack[top]) >= precedence(ch))
                prefix[j++] = pop();
            push(ch);
        }
    }
    while (top != -1) prefix[j++] = pop();
    prefix[j] = '\0';
    reverse(prefix);
    printf("Prefix: %s\n", prefix);
}

int main() {
    char infix[SIZE];
    printf("Enter infix expression: ");
    scanf("%s", infix);
    infixToPrefix(infix);
    return 0;
}
```

Input:

Enter infix expression: a+b*c

Output:

Prefix: +a*bc

//5. Evaluate Postfix Expression

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#define SIZE 100
int stack[SIZE], top = -1;

void push(int x) { stack[++top] = x; }
int pop() { return stack[top--]; }

int evaluatePostfix(char* expr) {
    int i, op1, op2;
    for (i = 0; expr[i]; i++) {
        if (isdigit(expr[i])) {
            push(expr[i] - '0');
        } else {
            op2 = pop();
            op1 = pop();
            switch (expr[i]) {
                case '+': push(op1 + op2); break;
                case '-': push(op1 - op2); break;
                case '*': push(op1 * op2); break;
                case '/': push(op1 / op2); break;
```

```
            }
        }
    }
    return pop();
}

int main() {
    char expr[SIZE];
    printf("Enter postfix expression: ");
    scanf("%s", expr);
    printf("Result: %d\n", evaluatePostfix(expr));
    return 0;
}
```

Input:

Enter postfix expression: 53+82-*

Explanation:

Evaluates as:

3 + = 8

8 8 2 - * = 8 * 6 = 48

Output:

Result: 48

6. Evaluate Prefix Expression

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>

#define SIZE 100
int stack[SIZE], top = -1;

void push(int x) { stack[++top] = x; }
int pop() { return stack[top--]; }

int evaluatePrefix(char* expr) {
    int i, op1, op2;
    for (i = strlen(expr) - 1; i >= 0; i--) {
        if (isdigit(expr[i])) {
            push(expr[i] - '0');
        } else {
            op1 = pop();
            op2 = pop();
            switch (expr[i]) {
                case '+': push(op1 + op2); break;
                case '-': push(op1 - op2); break;
                case '*': push(op1 * op2); break;
                case '/': push(op1 / op2); break;
            }
        }
    }
    return pop();
}

int main() {
```

```
    char expr[SIZE];
    printf("Enter prefix expression: ");
    scanf("%s", expr);
    printf("Result: %d\n", evaluatePrefix(expr));
    return 0;
}
```

Input:

Enter prefix expression: -+534

Explanation:

+ 5 3 = 8

- 8 4 = 4

Output:

Result: 4

// 7. Fibonacci Series Using Recursion

```
#include <stdio.h>

int fibonacci(int n) {
    if (n == 0) return 0;
    if (n == 1) return 1;
    return fibonacci(n-1) + fibonacci(n-2);
}

int main() {
    int n;
    printf("Enter number of terms: ");
    scanf("%d", &n);
    printf("Fibonacci series: ");
    for (int i = 0; i < n; i++)
        printf("%d ", fibonacci(i));
    printf("\n");
    return 0;
}
```

Input:

Enter number of terms: 7

Output:

Fibonacci series: 0 1 1 2 3 5 8

// 8. Tower of Hanoi Using Recursion

```
#include <stdio.h>

void towerOfHanoi(int n, char from, char to, char aux) {
    if (n == 1) {
        printf("Move disk 1 from %c to %c\n", from, to);
        return;
    }
```

```
    towerOfHanoi(n-1, from, aux, to);
    printf("Move disk %d from %c to %c\n", n, from, to);
    towerOfHanoi(n-1, aux, to, from);
}

int main() {
    int n;
    printf("Enter number of disks: ");
    scanf("%d", &n);
    towerOfHanoi(n, 'A', 'C', 'B');
    return 0;
}
```

Input:

Enter number of disks: 3

Output:

Move disk 1 from A to C
Move disk 2 from A to B
Move disk 1 from C to B
Move disk 3 from A to C
Move disk 1 from B to A
Move disk 2 from B to C
Move disk 1 from A to C