PROBLEM-1: Write a C program to design a 5D array, store integer values in it and print them.

PROGRAM:

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

int main()
{
   int a, b, c, d, e;
   printf("\nEnter the 5 different sizes of array: ");
   scanf("%d\n%d\n%d\n%d\n%d", &a, &b, &c, &d, &e);
   int *****arr = (int *****)malloc(a * sizeof(int ****));
   for (int i = 0; i < a; i++)
   {
      arr[i] = (int ****)malloc(b * sizeof(int ***));
      for (int j = 0; j < b; j++)
      {
         arr[i][j] = (int ***)malloc(c * sizeof(int **));
         for (int k = 0; k < c; k++)
         {
            arr[i][j][k] = (int **)malloc(d * sizeof(int *));
            for (int l = 0; l < d; l++)
            {
               arr[i][j][k][l] = (int *)malloc(e * sizeof(int));
               for (int m = 0; m < e; m++)
               {
                  arr[i][j][k][l][m] = i + j + k + l + m;
               }
            }
         }
      }
   }
   int n = 0;
   printf("\nElements in array: ");
   for (int i = 0; i < a; i++)
      for (int j = 0; j < b; j++)
         for (int k = 0; k < c; k++)
            for (int l = 0; l < d; l++)
               for (int m = 0; m < e; m++)
                  printf("%d  ", arr[i][j][k][l][m]);
   printf("\n");
   return 0;
}
```

OUTPUT:
Enter the 5 different sizes of array: 2 1 2 3 1

Elements in array: 0 1 2 1 2 3 1 2 3 2 3 4

PROBLEM-2: Write a C program to design a Singly Linked List and perform Insertion at end, Insertion at Beginning, Deletion at Begin, Deletion at End, Insertion at Index, Deletion at Index, Deletion of entire List and Display Linked List operation on it. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next;
} node;

node *head = NULL;
int data;
int count = 0;

void beginsert()
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   else
   {
      printf("\nEnter Data:");
      scanf("%d", &data);
      newnode->data = data;
      newnode->next = head;
      head = newnode;
      printf("Node inserted at position: 1\nData: %d", data);
      count++;
      return;
   }
}

void lastinsert()
{
   node *newnode = (node *)malloc(sizeof(node));
```

```
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        printf("\nEnter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = NULL;
        if (head == NULL)
        {
            head = newnode;
        }
        else
        {
            node *temp = head;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
        }
        count++;
        printf("Node inserted at position: %d\nData: %d", count, data);
        return;
    }
}

void randominsert()
{
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        int pos;
        printf("\nEnter the position where you want to insert the node: ");
        scanf("%d", &pos);
        if (pos > 0 && pos <= count + 1)
        {
            printf("Enter data: ");
            scanf("%d", &data);
            newnode->data = data;
```

```
            newnode->next = NULL;
            if (head == NULL || pos <= 1)
            {
               newnode->next = head;
               head = newnode;
            }
            else
            {
               node *temp = head;
               int n = 1;
               while (temp->next != NULL && n < pos - 1)
               {
                  temp = temp->next;
                  n++;
               }
               newnode->next = temp->next;
               temp->next = newnode;
            }
            count++;
            printf("Node inserted at position: ");
            if (pos <= 1 || head == NULL)
               printf("1");
            else if (pos < count)
               printf("%d", pos);
            else
               printf("%d", count);
            printf("\nData: %d", data);
            return;
         }
         else
         {
            printf("\nInvalid Position...");
            return;
         }
      }
   }

   void deletebegin()
   {
      if (head == NULL)
      {
         printf("\nList is empty...");
         return;
      }
      else
      {
         data = head->data;
         node *temp = head;
```

```
        head = head->next;
        free(temp);
        printf("\nNode has been deleted at position: 1\nData: %d", data);
        count--;
    }
}

void deletelast()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        node *temp = head;
        node *ptr = temp;
        while (temp->next != NULL)
        {
            ptr = temp;
            temp = temp->next;
        }
        data = temp->data;
        if (temp == head)
            head = NULL;
        else
            ptr->next = NULL;
        free(temp);
        printf("\nNode has been deleted at position: %d\nData: %d", count, data);
        count--;
    }
}

void deleterandom()
{
    if (head != NULL)
    {
        int pos;
        printf("\nEnter position of the node you want to delete: ");
        scanf("%d", &pos);
        if (pos < 1)
        {
            printf("\nA Invalid Postion!! Delete process doesn't occured...");
            return;
        }
        else if (pos > 1)
        {
```

```
            node *temp = head;
            int n = 1;
            while (temp->next != NULL && n < pos - 1)
            {
               temp = temp->next;
               n++;
            }
            if (n == pos - 1 && temp->next != NULL)
            {
               node *ptr = temp->next;
               temp->next = ptr->next;
               data = ptr->data;
               free(ptr);
               count--;
               printf("Node has been deleted at position: %d\nData: %d", pos, data);
               return;
            }
            else
            {
               printf("A Invalid Postion!! Delete process doesn't occured...");
               return;
            }
         }
         else
         {
            node *temp = head;
            data = head->data;
            head = head->next;
            free(temp);
            count--;
            printf("Node has been deleted at position: 1\nData: %d", data);
            return;
         }
      }
      else
      {
         printf("\nList is empty...");
         return;
      }
   }

   void deletelist()
   {
      if (head == NULL)
      {
         printf("\nList is empty...");
         return;
      }
```

```
    else
    {
        while (head != NULL)
        {
            node *temp = head;
            head = head->next;
            free(temp);
        }
        printf("\nEntire Linked List has been deleted...");
        count = 0;
    }
}

void display()
{
    if (head == NULL)
    {
        printf("\nList is not created yet or deleted...");
        return;
    }
    else
    {
        node *temp = head;
        printf("\nValues in List: [  ");
        while (temp != NULL)
        {
            printf("%d  ", temp->data);
            temp = temp->next;
        }
        printf("]");
    }
}

int main()
{
    printf("\n-------->Main Menu<--------");
    printf("\n1.Insert at Beginning.          2.Insert at Last.\n3.Insert at a specified Index.
4.Delete at Beginning.\n5.Delete at Last.          6.Delete at specified Index.\n7.Delete entire
list          8.Show.\n9.Exit.\n");
    int choice;
    while (choice != 9)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT a node at BEGINNING...");
```

```
            beginsert();
            display();
            break;
        case 2:
            printf("You choose to INSERT a node at LAST...");
            lastinsert();\
            display();
            break;
        case 3:
            printf("You choose to INSERT a node at SPECIFIED INDEX...");
            randominsert();
            display();
            break;
        case 4:
            printf("You choose to DELETE a node at BEGINNING...");
            deletebegin();
            display();
            break;
        case 5:
            printf("You choose to DELETE a node at LAST...");
            deletelast();
            display();
            break;
        case 6:
            printf("You choose to DELETE a node at SPECIFIED INDEX...");
            deleterandom();
            display();
            break;
        case 7:
            printf("You choose to DELETE ENTIRE LIST...");
            deletelist();
            display();
            break;
        case 8:
            printf("You choose to DISPLAY ENTIRE LIST...");
            display();
            break;
        case 9:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("Invalid choice...");
        }
    }
    return 0;
}
```

OUTPUT:

```
-------->Main Menu<--------
1.Insert at Beginning.          2.Insert at Last.
3.Insert at a specified Index.  4.Delete at Beginning.
5.Delete at Last.               6.Delete at specified Index.
7.Delete entire list            8.Show.
9.Exit.

Enter your choice: 1
You choose to INSERT a node at BEGINNING...
Enter Data:23
Node inserted at position: 1
Data: 23
Values in List: [  23  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position where you want to insert the node: 2
Enter data: 43
Node inserted at position: 2
Data: 43
Values in List: [  23  43  ]
Enter your choice: 2
You choose to INSERT a node at LAST...
Enter data: 17
Node inserted at position: 3
Data: 17
Values in List: [  23  43  17  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position where you want to insert the node: 2
Enter data: 67
Node inserted at position: 2
Data: 67
Values in List: [  23  67  43  17  ]
Enter your choice: 6
You choose to DELETE a node at SPECIFIED INDEX...
Enter position of the node you want to delete: 3
Node has been deleted at position: 3
Data: 43
Values in List: [  23  67  17  ]
Enter your choice: 4
You choose to DELETE a node at BEGINNING...
Node has been deleted at position: 1
Data: 23
Values in List: [  67  17  ]
Enter your choice: 5
You choose to DELETE a node at LAST...
Node has been deleted at position: 2
```

```
Data: 17
Values in List: [  67  ]
Enter your choice: 7
You choose to DELETE ENTIRE LIST...
Entire Linked List has been deleted...
List is not created yet or deleted...
Enter your choice: 9

CODE EXITED SUCCESSFULLY...
```

PROBLEM-3: Write a program in C to reverse a singly linked list. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next;
} node;

node *head = NULL;

void display()
{
   node *temp = head;
   printf("Values in Linked List:  [  ");
   while (temp != NULL)
   {
      printf("%d  ", temp->data);
      temp = temp->next;
   }
   printf("]");
}

int main()
{
   int size, data;
   printf("\nEnter size of linked list: ");
   scanf("%d", &size);
   node *temp;
   printf("Enter values in linked list: ");
   for (int i = 1; i <= size; i++)
   {
      scanf("%d", &data);
      node *newnode = (node *)malloc(sizeof(node));
```

```
        newnode->data = data;
        newnode->next = NULL;
        if (i == 1)
        {
            head = newnode;
            temp = head;
        }
        else
        {
            temp->next = newnode;
            temp = temp->next;
        }
    }
    display();
    node *tmp = head;
    node *ttm = NULL;
    node *ptr;
    while (tmp != NULL)
    {
        ptr = tmp->next;
        tmp->next = ttm;
        ttm = tmp;
        tmp = ptr;
    }
    head = ttm;
    printf("\nLinked List has been reverse...\n");
    display();
    printf("\n");
    return 0;
}
```

OUTPUT:

```
Enter size of linked list: 5
Enter values in linked list: 12 23 34 45 56
Values in Linked List:  [  12  23  34  45  56  ]
Linked List has been reverse...
Values in Linked List:  [  56  45  34  23  12  ]
```

PROBLEM-4: Write a C program to design a Doubly Linked List and perform Insertion at end, Insertion at Beginning, Deletion at Begin, Deletion at End, Insertion at Index, Deletion at Index, Deletion of entire List, Reverse operation, and Display Linked List operation on it. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next;
   struct node *prev;
} node;

node *head = NULL;
int data;
int count = 0;

void beginsert();
void lastinsert();
void randominsert();
void deletebegin();
void deletelast();
void deleterandom();
void display();
void deletelist();

void beginsert()
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   else
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      newnode->data = data;
      newnode->next = head;
      if (head != NULL)
         head->prev = newnode;
      newnode->prev = NULL;
      head = newnode;
      count++;
```

```
        printf("Node is inserted at position: 1\nData: %d", data);
    }
}

void lastinsert()
{
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        printf("\nEnter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = NULL;
        newnode->prev = NULL;
        if (head == NULL)
            head = newnode;
        else
        {
            node *temp = head;
            while (temp->next != NULL)
            {
                temp = temp->next;
            }
            temp->next = newnode;
            newnode->prev = temp;
        }
        count++;
        printf("Node is inserted at position: %d\nData: %d", count, newnode->data);
    }
}

void randominsert()
{
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        int pos;
        printf("\nEnter position node will be inserted: ");
```

```
        scanf("%d", &pos);
        printf("Enter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = NULL;
        newnode->prev = NULL;
        if (pos <= 1 || head == NULL)
        {
           newnode->next = head;
           if (head != NULL)
              head->prev = newnode;
           head = newnode;
        }
        else
        {
           node *temp = head;
           int n = 1;
           while (temp->next != NULL && n < pos - 1)
           {
              temp = temp->next;
              n++;
           }
           newnode->next = temp->next;
           newnode->prev = temp;
           if (temp->next != NULL)
              temp->next->prev = newnode;
           temp->next = newnode;
        }
        count++;
        printf("Node is inserted at position: ");
        if (pos <= 1)
           printf("1");
        else if (pos < count)
           printf("%d", pos);
        else
           printf("%d", count);
        printf("\nData: %d", data);
        return;
     }
  }

  void deletebegin()
  {
     if (head == NULL)
     {
        printf("\nList is empty...");
        return;
     }
```

```
    else
    {
        node *temp = head;
        data = head->data;
        head = head->next;
        if (head != NULL)
            head->prev = NULL;
        free(temp);
        count--;
        printf("\nNode has been deleted at position: 1\nData: %d", data);
    }
}

void deletelast()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        node *temp = head;
        node *ptr = temp;
        while (temp->next != NULL)
        {
            ptr = temp;
            temp = temp->next;
        }
        ptr->next = NULL;
        data = temp->data;
        if (temp == head)
        {
            head = NULL;
        }
        free(temp);
        count--;
        printf("\nNode has been deleted at position: %d\nData: %d", count + 1, data);
    }
}

void deleterandom()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
```

```
else
{
   int pos;
   printf("\nEnter the position you want to delete: ");
   scanf("%d", &pos);
   if (pos < 1)
   {
      printf("Invalid position...");
      return;
   }
   else
   {
      node *temp = head;
      if (pos == 1)
      {
         data = head->data;
         head = head->next;
         if (head != NULL)
            head->prev = NULL;
         free(temp);
         count--;
      }
      else
      {
         int n = 1;
         while (temp->next != NULL && n < pos)
         {
            temp = temp->next;
            n++;
         }
         if (n == pos)
         {
            data = temp->data;
            temp->prev->next = temp->next;
            if (temp->next != NULL)
               temp->next->prev = temp->prev;
            free(temp);
            count--;
         }
         else
         {
            printf("Invalid position...");
            return;
         }
      }
      printf("Node has been deleted at positoin: %d\nData: %d", pos, data);
      return;
   }
```

```
    }
}

void display()
{
    if (head == NULL)
    {
        printf("\nList is not created yet or deleted...");
        return;
    }
    else
    {
        node *temp = head;
        printf("\nValues in List: [  ");
        while (temp->next != NULL)
        {
            printf("%d  ", temp->data);
            temp = temp->next;
        }
        printf("%d  ]", temp->data);
    }
}

void reverse()
{
    node *temp = head;
    node *ttm = NULL;
    node *ptr;
    display();
    while (temp != NULL)
    {
        ptr = temp->next;
        temp->next = ttm;
        if (ttm != NULL)
            ttm->prev = temp;
        if (ptr != NULL)
            ptr->prev = ttm;
        ttm = temp;
        temp = ptr;
    }
    head = ttm;
    printf("\nLinked List has been reversed...");
    display();
}

void deletelist()
{
    if (head == NULL)
```

```
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        while (head != NULL)
        {
            node *temp = head;
            head = head->next;
            free(temp);
        }
        printf("\nWhole List is deleted...");
    }
}

int main()
{
    printf("\n-------->Main Menu<--------");
    printf("\n1.Insert at Beginning.          2.Insert at Last.\n3.Insert at a specified Index.
4.Delete at Beginning.\n5.Delete at Last.          6.Delete at specified Index.\n7.Delete entire
list          8.Show.\n9.Reverse Linked List.          10.Exit.\n");
    int choice;
    while (choice != 10)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT a node at BEGINNING...");
            beginsert();
            display();
            break;
        case 2:
            printf("You choose to INSERT a node at LAST...");
            lastinsert();
            display();
            break;
        case 3:
            printf("You choose to INSERT a node at SPECIFIED INDEX...");
            randominsert();
            display();
            break;
        case 4:
            printf("You choose to DELETE a node at BEGINNING...");
            deletebegin();
            display();
```

```
            break;
        case 5:
            printf("You choose to DELETE a node at LAST...");
            deletelast();
            display();
            break;
        case 6:
            printf("You choose to DELETE a node at SPECIFIED INDEX...");
            deleterandom();
            display();
            break;
        case 7:
            printf("You choose to DELETE ENTIRE LIST...");
            deletelist();
            display();
            break;
        case 8:
            printf("You choose to DISPLAY ENTIRE LIST...");
            display();
            break;
        case 9:
            printf("You choose to REVERSE ENTIRE LINKED LIST...");
            reverse();
            break;
        case 10:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("Invalid choice...");
        }
    }
    return 0;
}
```

OUTPUT:

```
--------->Main Menu<--------
1.Insert at Beginning.              2.Insert at Last.
3.Insert at a specified Index.      4.Delete at Beginning.
5.Delete at Last.                   6.Delete at specified Index.
7.Delete entire list                8.Show.
9.Reverse Linked List.              10.Exit.

Enter your choice: 1
You choose to INSERT a node at BEGINNING...
Enter data: 23
Node is inserted at position: 1
Data: 23
Values in List: [  23  ]
```

```
Enter your choice: 2
You choose to INSERT a node at LAST...
Enter data: 45
Node is inserted at position: 2
Data: 45
Values in List: [  23  45  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter position node will be inserted: 1
Enter data: 26
Node is inserted at position: 1
Data: 26
Values in List: [  26  23  45  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter position node will be inserted: 2
Enter data: 16
Node is inserted at position: 2
Data: 16
Values in List: [  26  16  23  45  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter position node will be inserted: 5
Enter data: 17
Node is inserted at position: 5
Data: 17
Values in List: [  26  16  23  45  17  ]
Enter your choice: 9
You choose to REVERSE ENTIRE LINKED LIST...
Values in List: [  26  16  23  45  17  ]
Linked List has been reversed...
Values in List: [  17  45  23  16  26  ]
Enter your choice: 6
You choose to DELETE a node at SPECIFIED INDEX...
Enter the position you want to delete: 3
Node has been deleted at positoin: 3
Data: 23
Values in List: [  17  45  16  26  ]
Enter your choice: 4
You choose to DELETE a node at BEGINNING...
Node has been deleted at position: 1
Data: 17
Values in List: [  45  16  26  ]
Enter your choice: 5
You choose to DELETE a node at LAST...
Node has been deleted at position: 3
Data: 26
Values in List: [  45  16  ]
```

```
Enter your choice: 7
You choose to DELETE ENTIRE LIST...
Whole List is deleted...
List is not created yet or deleted...
Enter your choice: 10

CODE EXITED SUCCESSFULLY...
```

PROBLEM-5: Write a C program to design a Circular Linked List and perform Insertion at end, Insertion at Beginning, Deletion at Begin, Deletion at End, Insertion at Index, Deletion at Index, Deletion of entire List, and Display Circular Linked List operation. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next;
} node;

node *head = NULL;
node *last = NULL;
int count = 0;
int data;

void beginsert();
void lastinsert();
void randominsert();
void deletebegin();
void deletelast();
void deleterandom();
void display();
void deletelist();

void beginsert()
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   else
   {
      printf("\nEnter data: ");
```

```
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = newnode;
        if (head == NULL)
        {
           head = last = newnode;
        }
        else
        {
           newnode->next = head;
           last->next = newnode;
           head = newnode;
        }
        count++;
        printf("Node is inserted at position: 1\nData: %d", data);
    }
}

void lastinsert()
{
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        printf("\nEnter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = newnode;
        if (head == NULL)
        {
           head = last = newnode;
        }
        else
        {
           newnode->next = head;
           last->next = newnode;
           last = newnode;
        }
        count++;
        printf("Node is inserted at position: %d\nData: %d", count, data);
    }
}

void randominsert()
```

```
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   else
   {
      int pos;
      printf("\nEnter the position node will be inserted: ");
      scanf("%d", &pos);
      printf("Enter data: ");
      scanf("%d", &data);
      newnode->data = data;
      newnode->next = newnode;
      if (head == NULL)
         last = head = newnode;
      else if (pos <= 1)
      {
         newnode->next = head;
         last->next = newnode;
         head = newnode;
      }
      else
      {
         node *temp = head;
         int n = 1;
         while (temp->next != head && n < pos - 1)
         {
            temp = temp->next;
            n++;
         }
         newnode->next = temp->next;
         temp->next = newnode;
         if (temp == last)
            last = newnode;
      }
      count++;
      printf("Node is inserted at position: ");
      if (pos <= 1)
         printf("1");
      else if (pos <= count)
         printf("%d", pos);
      else
         printf("%d", count);
      printf("\nData: %d", data);
   }
```

```
}

void deletebegin()
{
   if (head == NULL)
   {
      printf("\nList is empty...");
      return;
   }
   else
   {
      data = head->data;
      node *temp = head;
      if (head == last)
      {
         head = last = NULL;
      }
      else
      {
         head = head->next;
         last->next = head;
      }
      free(temp);
      count--;
      printf("\nOne node has been deleted at position: 1\nData: %d", data);
   }
}

void deletelast()
{
   if (head == NULL)
   {
      printf("\nList is empty...");
      return;
   }
   else
   {
      node *temp = head;
      if (last == head)
         last = head = NULL;
      else
      {
         while (temp->next != last)
            temp = temp->next;
         last = temp;
         temp = temp->next;
         last->next = head;
      }
```

```
        free(temp);
        printf("\nOne node has been deleted at position: %d\nData: %d", count, data);
        count--;
    }
}

void deleterandom()
{
    if (head == NULL)
    {
        printf("\nList is empty...\n");
        return;
    }
    else
    {
        printf("\nEnter positon which you want to delete: ");
        int pos;
        scanf("%d", &pos);
        if (pos < 1)
        {
            printf("Invalid position...");
            return;
        }
        else if (pos == 1)
        {
            node *temp = head;
            data = head->data;
            if (head == last)
                head = last = NULL;
            else
            {
                head = head->next;
                last->next = head;
            }
            free(temp);
            count--;
            printf("Node has been deleted at position: 1\nData: %d", data);
        }
        else
        {
            node *temp = head;
            int n = 1;
            while (temp->next != head && n < pos - 1)
            {
                temp = temp->next;
                n++;
            }
            if (n == pos - 1 && temp->next != head)
```

```
            {
                if (temp->next == last)
                    last = temp;
                node *ptm = temp->next;
                temp->next = temp->next->next;
                data = ptm->data;
                free(ptm);
                count--;
                printf("Node has been deleted at position: %d\nData: %d", pos, data);
            }
            else
            {
                printf("Invalid position...");
                return;
            }
        }
    }
}

void display()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        node *temp = head;
        printf("\nValues in linked list: [  ");
        do
        {
            printf("%d  ", temp->data);
            temp = temp->next;
        } while (temp != head);
        printf("]");
    }
}

void deletelist()
{
    while (head != last)
    {
        node *temp = head;
        head = head->next;
        free(temp);
        ;
    }
```

```
    free(head);
    head = last = NULL;
    printf("\nEntire Linked List has been deleted...");
    count = 0;
}

int main()
{
    printf("\n-------->Main Menu<--------");
    printf("\n1.Insert at Beginning.          2.Insert at Last.\n3.Insert at a specified Index.
4.Delete at Beginning.\n5.Delete at Last.          6.Delete at specified Index.\n7.Delete entire
list          8.Show.\n9.Exit.\n");
    int choice;
    while (choice != 9)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT a node at BEGINNING...");
            beginsert();
            display();
            break;
        case 2:
            printf("You choose to INSERT a node at LAST...");
            lastinsert();
            display();
            break;
        case 3:
            printf("You choose to INSERT a node at SPECIFIED INDEX...");
            randominsert();
            display();
            break;
        case 4:
            printf("You choose to DELETE a node at BEGINNING...");
            deletebegin();
            display();
            break;
        case 5:
            printf("You choose to DELETE a node at LAST...");
            deletelast();
            display();
            break;
        case 6:
            printf("You choose to DELETE a node at SPECIFIED INDEX...");
            deleterandom();
            display();
```

```
            break;
        case 7:
            printf("You choose to DELETE ENTIRE LIST...");
            deletelist();
            display();
            break;
        case 8:
            printf("You choose to DISPLAY ENTIRE LIST...");
            display();
            break;
        case 9:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("Invalid choice...");
        }
    }
    return 0;
}
```

OUTPUT:

```
-------->Main Menu<--------
1.Insert at Beginning.              2.Insert at Last.
3.Insert at a specified Index.      4.Delete at Beginning.
5.Delete at Last.                   6.Delete at specified Index.
7.Delete entire list                8.Show.
9.Exit.

Enter your choice: 1
You choose to INSERT a node at BEGINNING...
Enter data: 24
Node is inserted at position: 1
Data: 24
Values in linked list: [  24  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position node will be inserted: 2
Enter data: 37
Node is inserted at position: 2
Data: 37
Values in linked list: [  24  37  ]
Enter your choice: 2
You choose to INSERT a node at LAST...
Enter data: 18
Node is inserted at position: 3
Data: 18
Values in linked list: [  24  37  18  ]
Enter your choice: 3
```

```
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position node will be inserted: 2
Enter data: 35
Node is inserted at position: 2
Data: 35
Values in linked list: [  24  35  37  18  ]
Enter your choice: 6
You choose to DELETE a node at SPECIFIED INDEX...
Enter positon which you want to delete: 3
Node has been deleted at position: 3
Data: 37
Values in linked list: [  24  35  18  ]
Enter your choice: 4
You choose to DELETE a node at BEGINNING...
One node has been deleted at position: 1
Data: 24
Values in linked list: [  35  18  ]
Enter your choice: 5
You choose to DELETE a node at LAST...
One node has been deleted at position: 2
Data: 24
Values in linked list: [  35  ]
Enter your choice: 7
You choose to DELETE ENTIRE LIST...
Entire Linked List has been deleted...
List is empty...
Enter your choice: 9

CODE EXITED SUCCESSFULLY...
```

PROBLEM-6: Write a C program to design a Circular Doubly Linked List and perform Insertion at end, Insertion at Beginning, Deletion at Begin, Deletion at End, Insertion at Index, Deletion at Index, Deletion of entire List, and Display Circular Doubly Linked List operation. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next, *prev;
} node;

node *head = NULL;
node *last = NULL;
int count = 0;
```

```
int data;

void beginsert();
void lastinsert();
void randominsert();
void deletebegin();
void deletelast();
void deleterandom();
void display();

void beginsert()
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   else
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      newnode->data = data;
      newnode->next = newnode;
      newnode->prev = newnode;
      if (head == NULL)
      {
         head = last = newnode;
      }
      else
      {
         newnode->next = head;
         newnode->prev = last;
         last->next = newnode;
         head->prev = newnode;
         head = newnode;
      }
      count++;
      printf("Node is inserted at position: 1\nData: %d", data);
   }
}

void lastinsert()
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
```

```
        return;
    }
    else
    {
        printf("\nEnter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = newnode;
        newnode->prev = newnode;
        if (head == NULL)
        {
            head = last = newnode;
        }
        else
        {
            newnode->next = head;
            newnode->prev = last;
            last->next = newnode;
            head->prev = newnode;
            last = newnode;
        }
        count++;
        printf("Node is inserted at position: %d\nData: %d", count, data);
    }
}

void randominsert()
{
    node *newnode = (node *)malloc(sizeof(node));
    if (newnode == NULL)
    {
        printf("\nOVERFLOW...");
        return;
    }
    else
    {
        int pos;
        printf("\nEnter the position node will be inserted: ");
        scanf("%d", &pos);
        printf("Enter data: ");
        scanf("%d", &data);
        newnode->data = data;
        newnode->next = newnode;
        newnode->prev = newnode;
        if (head == NULL)
            last = head = newnode;
        else if (pos <= 1)
        {
```

```
            newnode->next = head;
            newnode->prev = last;
            last->next = newnode;
            head->prev = newnode;
            head = newnode;
        }
        else
        {
            node *temp = head;
            int n = 1;
            while (temp->next != head && n < pos - 1)
            {
                temp = temp->next;
                n++;
            }
            newnode->next = temp->next;
            newnode->prev = temp;
            temp->next = newnode;
            newnode->next->prev = newnode;
            if (temp == last)
                last = newnode;
        }
        count++;
        printf("Node is inserted at position: ");
        if (pos <= 1)
            printf("1");
        else if (pos <= count)
            printf("%d", pos);
        else
            printf("%d", count);
        printf("\nData: %d", data);
    }
}

void deletebegin()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        data = head->data;
        node *temp = head;
        if (head == last)
        {
            head = last = NULL;
```

```
        }
        else
        {
            head = head->next;
            head->prev = last;
            last->next = head;
        }
        free(temp);
        count--;
        printf("\nOne node has been deleted at position: 1\nData: %d", data);
    }
}

void deletelast()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
    else
    {
        node *temp = head;
        if (last == head)
            last = head = NULL;
        else
        {
            while (temp->next != last)
                temp = temp->next;
            last = temp;
            temp = temp->next;
            last->next = head;
            head->prev = last;
        }
        data = temp->data;
        free(temp);
        printf("\nOne node has been deleted at position: %d\nData: %d", count, data);
        count--;
    }
}

void deleterandom()
{
    if (head == NULL)
    {
        printf("\nList is empty...");
        return;
    }
```

```
else
{
   printf("\nEnter positon which you want to delete: ");
   int pos;
   scanf("%d", &pos);
   if (pos < 1)
   {
      printf("Invalid position...");
      return;
   }
   else if (pos == 1)
   {
      data = head->data;
      node *temp = head;
      if (head == last)
      {
         head = last = NULL;
      }
      else
      {
         head = head->next;
         head->prev = last;
         last->next = head;
      }
      free(temp);
      count--;
      printf("Node has been deleted at position: 1\nData: %d", data);
   }
   else
   {
      node *temp = head;
      int n = 1;
      while (temp->next != head && n < pos - 1)
      {
         temp = temp->next;
         n++;
      }
      if (n == pos - 1 && temp->next != head)
      {
         if (temp->next == last)
            last = temp;
         node *ptm = temp->next;
         temp->next = temp->next->next;
         temp->next->prev = temp;
         data = ptm->data;
         free(ptm);
         count--;
         printf("Node has been deleted at position: %d\nData: %d", pos, data);
```

```
            }
            else
            {
               printf("Invalid position...");
               return;
            }
         }
      }
}

void display()
{
   if (head == NULL)
   {
      printf("\nList is empty...");
      return;
   }
   else
   {
      node *temp = head;
      printf("\nValues in linked list: [  ");
      do
      {
         printf("%d  ", temp->data);
         temp = temp->next;
      } while (temp != head);
      printf("]");
   }
}

void deletelist()
{
   while (head != last)
   {
      node *temp = head;
      head = head->next;
      free(temp);
      ;
   }
   free(head);
   head = last = NULL;
   printf("\nEntire Linked List has been deleted...");
   count = 0;
}

int main()
{
   printf("\n-------->Main Menu<--------");
```

```
    printf("\n1.Insert at Beginning.          2.Insert at Last.\n3.Insert at a specified Index.
4.Delete at Beginning.\n5.Delete at Last.          6.Delete at specified Index.\n7.Delete entire
list          8.Show.\n9.Exit.\n");
    int choice;
    while (choice != 9)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT a node at BEGINNING...");
            beginsert();
            display();
            break;
        case 2:
            printf("You choose to INSERT a node at LAST...");
            lastinsert();
            display();
            break;
        case 3:
            printf("You choose to INSERT a node at SPECIFIED INDEX...");
            randominsert();
            display();
            break;
        case 4:
            printf("You choose to DELETE a node at BEGINNING...");
            deletebegin();
            display();
            break;
        case 5:
            printf("You choose to DELETE a node at LAST...");
            deletelast();
            display();
            break;
        case 6:
            printf("You choose to DELETE a node at SPECIFIED INDEX...");
            deleterandom();
            display();
            break;
        case 7:
            printf("You choose to DELETE ENTIRE LIST...");
            deletelist();
            display();
            break;
        case 8:
            printf("You choose to DISPLAY ENTIRE LIST...");
            display();
```

```
            break;
        case 9:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("Invalid choice...");
        }
    }
    return 0;
}
```

OUTPUT:

```
-------->Main Menu<--------
1.Insert at Beginning.              2.Insert at Last.
3.Insert at a specified Index.      4.Delete at Beginning.
5.Delete at Last.                   6.Delete at specified Index.
7.Delete entire list                8.Show.
9.Exit.

Enter your choice: 1
You choose to INSERT a node at BEGINNING...
Enter data: 23
Node is inserted at position: 1
Data: 23
Values in linked list: [  23  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position node will be inserted: 2
Enter data: 16
Node is inserted at position: 2
Data: 16
Values in linked list: [  23  16  ]
Enter your choice: 2
You choose to INSERT a node at LAST...
Enter data: 18
Node is inserted at position: 3
Data: 18
Values in linked list: [  23  16  18  ]
Enter your choice: 3
You choose to INSERT a node at SPECIFIED INDEX...
Enter the position node will be inserted: 3
Enter data: 38
Node is inserted at position: 3
Data: 38
Values in linked list: [  23  16  38  18  ]
Enter your choice: 6
You choose to DELETE a node at SPECIFIED INDEX...
Enter positon which you want to delete: 3
```

```
Node has been deleted at position: 3
Data: 38
Values in linked list: [  23  16  18  ]
Enter your choice: 5
You choose to DELETE a node at LAST...
One node has been deleted at position: 3
Data: 18
Values in linked list: [  23  16  ]
Enter your choice: 4
You choose to DELETE a node at BEGINNING...
One node has been deleted at position: 1
Data: 23
Values in linked list: [  16  ]
Enter your choice: 7
You choose to DELETE ENTIRE LIST...
Entire Linked List has been deleted...
List is empty...
Enter your choice: 9

CODE EXITED SUCCESSFULLY...
```

PROBLEM-7: Write a C program to implement Last in First out (LIFO) data structure and perform Push, Pop, Peek operations on it. Use array to implement the same.

PROGRAM:

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

int *stack;
int top = -1;
int max, data;

void push();
void pop();
void peek();
int isFull();
int isEmpty();

int isEmpty()
{
   if (top == -1)
   {
      printf("\nSTACK IS EMPTY...");
      return 1;
   }
   else
      return 0;
}
```

```
int isFull()
{
   if (top == max - 1)
   {
      printf("\nSTACK IS FULL...");
      return 1;
   }
   else
      return 0;
}

void push()
{
   if (!isFull())
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      stack[++top] = data;
      printf("%d has been pushed...", data);
   }
}

void pop()
{
   if (!isEmpty())
   {
      data = stack[top--];
      printf("\n%d has been popped...", data);
   }
}

void peek()
{
   if (!isEmpty())
   {
      printf("\nTop Element of the stack = %d", stack[top]);
   }
}

int main()
{
   int a;
   printf("\nEnter size of stack: ");
   scanf("%d", &max);
   stack = (int *)malloc(max * sizeof(int));
   int choice;
   printf("---->Main Menu<----\n1.Push    2.Pop\n3.Peek    4.Exit\n");
```

```
    while (choice != 4)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to PUSH a data to stack...");
            push();
            peek();
            break;
        case 2:
            printf("You choose to POP a data from stack...");
            pop();
            peek();
            break;
        case 3:
            printf("You choose to show PEEK data of stack...");
            peek();
            break;
        case 4:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter size of stack: 5
---->Main Menu<----
1.Push    2.Pop
3.Peek    4.Exit

Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 2
2 has been pushed...
Top Element of the stack = 2
Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 3
3 has been pushed...
Top Element of the stack = 3
Enter your choice: 2
You choose to POP a data from stack...
3 has been popped...
Top Element of the stack = 2
```

```
Enter your choice: 2
You choose to POP a data from stack...
2 has been popped...
STACK IS EMPTY...
Enter your choice: 4

CODE EXITED SUCCESSFULLY...
```

PROBLEM-8: Write a C program to implement Push, Pop, Peek operations of Stack using Linked List.

PROGRAM:

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

typedef struct node{
   int data;
   struct node *next;
}node;

node *head = NULL;
int max, data, top = 0;

void push();
void pop();
void peek();
int isFull();
int isEmpty();

int isEmpty()
{
   if (head == NULL)
   {
      printf("\nSTACK IS EMPTY...");
      return 1;
   }
   else
      return 0;
}

int isFull()
{
   if (top == max)
   {
      printf("\nSTACK IS FULL...");
      return 1;
   }
   else
```

```
    return 0;
}

void push()
{
   if (!isFull())
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      node *newnode = (node*)malloc(sizeof(node));
      newnode->data = data;
      newnode->next = head;
      head = newnode;
      top++;
      printf("%d has been pushed...", data);
   }
}

void pop()
{
   if (!isEmpty())
   {
      data = head->data;
      node *temp = head;
      head = head->next;
      free(temp);
      top--;
      printf("\n%d has been popped...", data);
   }
}

void peek()
{
   if (!isEmpty())
   {
      printf("\nTop Element of the stack = %d", head->data);
   }
}

int main()
{
   int a;
   printf("\nEnter size of stack: ");
   scanf("%d", &max);
   int choice;
   printf("---->Main Menu<----\n1.Push    2.Pop\n3.Peek    4.Exit\n");
   while (choice != 4)
   {
```

```
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to PUSH a data to stack...");
            push();
            peek();
            break;
        case 2:
            printf("You choose to POP a data from stack...");
            pop();
            peek();
            break;
        case 3:
            printf("You choose to show PEEK data of stack...");
            peek();
            break;
        case 4:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter size of stack: 5
---->Main Menu<----
1.Push    2.Pop
3.Peek    4.Exit

Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 2
2 has been pushed...
Top Element of the stack = 2
Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 3
3 has been pushed...
Top Element of the stack = 3
Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 4
4 has been pushed...
Top Element of the stack = 4
Enter your choice: 1
```

```
You choose to PUSH a data to stack...
Enter data: 5
5 has been pushed...
Top Element of the stack = 5
Enter your choice: 1
You choose to PUSH a data to stack...
Enter data: 6
6 has been pushed...
Top Element of the stack = 6
Enter your choice: 1
You choose to PUSH a data to stack...
STACK IS FULL...
Top Element of the stack = 6
Enter your choice: 2
You choose to POP a data from stack...
6 has been popped...
Top Element of the stack = 5
Enter your choice: 2
You choose to POP a data from stack...
5 has been popped...
Top Element of the stack = 4
Enter your choice: 2
You choose to POP a data from stack...
4 has been popped...
Top Element of the stack = 3
Enter your choice: 2
You choose to POP a data from stack...
3 has been popped...
Top Element of the stack = 2
Enter your choice: 2
You choose to POP a data from stack...
2 has been popped...
STACK IS EMPTY...
Enter your choice: 2
You choose to POP a data from stack...
STACK IS EMPTY...
STACK IS EMPTY...
Enter your choice: 4

CODE EXITED SUCCESSFULLY...
```

PROBLEM-9: Write a C program to reverse an array using stack. You need to implement the stack first then use that for reversal. You may create a header file of your own to perform the task.

PROGRAM:

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

```
int *stack;
int top = -1;
int max, data;

void push(int data);
int pop();
int isFull();
int isEmpty();
void reverse(int arr[]);
void display(int arr[]);

int isEmpty()
{
   if (top == -1)
   {
      return 1;
   }
   else
      return 0;
}

int isFull()
{
   if (top == max - 1)
   {
      return 1;
   }
   else
      return 0;
}

void push(int data)
{
   if (!isFull())
   {
      stack[++top] = data;
   }
}

int pop()
{
   if (!isEmpty())
   {
      return stack[top--];
   }
}

void reverse(int arr[]){
```

```
    stack = (int*)malloc(max*sizeof(int));
    for(int i = 0; i<max; i++){
        push(arr[i]);
    }
    for(int i = 0; i<max; i++){
        arr[i] = pop();
    }
    free(stack);
    printf("\nArray has been reverse...");
}

void display(int arr[]){
    printf("\nValues in array:  [  ");
    for(int i = 0; i<max; i++){
        printf("%d  ",arr[i]);
    }
    printf("]");
}

int main(){
    printf("\nEnter Size of array: ");
    scanf("%d",&max);
    int *arr = (int*)malloc(max*sizeof(int));
    printf("Insert element in array: ");
    for(int i = 0; i<max; i++)
    scanf("%d",&arr[i]);
    display(arr);
    reverse(arr);
    display(arr);
    printf("\n");
}
```

OUTPUT:

```
Enter Size of array: 5
Insert element in array: 1 2 3 4 5

Values in array:  [  1  2  3  4  5  ]
Array has been reverse...
Values in array:  [  5  4  3  2  1  ]
```

PROBLEM-10: Write a C program to convert an equation to Postfix and Prefix with the help of a Stack. Implement the Stack using Linked List.

PROGRAM:

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

```
char prefix[100];

typedef struct node
{
   char data;
   struct node *next;
} node;

node *head = NULL;
int top = -1;

int isEmpty()
{
   if (head == NULL)
   {
      printf("\nUNDERFLOW...");
      return 1;
   }
   return 0;
}

void push(char x)
{
   node *newnode = (node *)malloc(sizeof(node));
   if (newnode == NULL)
   {
      printf("\nOVERFLOW...");
      return;
   }
   newnode->data = x;
   newnode->next = head;
   head = newnode;
   top++;
}

char pop()
{
   if (!isEmpty())
   {
      char data = head->data;
      node *temp = head;
      head = head->next;
      free(temp);
      top--;
      return data;
   }
}
```

```
int is_operator(char x)
{
   if (x == '^' || x == '*' || x == '/' || x == '+' || x == '-' || x == '(' || x == ')')
      return 1;
   else
      return 0;
}

char peek()
{
   if (head == NULL)
      return '#';
   else
      return head->data;
}

void revstr(char *str)
{
   int len;
   int start, end;
   char temp;
   len = strlen(str);
   start = 0;
   end = len - 1;
   for (int i = 0; i < len / 2; i++)
   {
      temp = str[end];
      str[end] = str[start];
      str[start] = temp;
      start++;
      end--;
   }
}

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

int associatvity_righttoleft(char x)
```

```
{
   if (x == '^')
      return 1;
   else
      return 0;
}

void Infix_Postfix(char *e)
{
   while (*e != '\0')
   {
      if (isalnum(*e))
      {
         printf("%c ", *e);
      }
      else if (*e == '('
         push(*e);
      else if (*e == ')')
      {
         char x;
         while ((x = pop()) != '('
         {
            printf("%c ", x);
         }
      }
      else
      {
         while (priority(*e) <= priority(peek()))
         {
            if (associatvity_righttoleft(*e))
               break;
            else
               printf("%c ", pop());
         }
         push(*e);
      }
      e++;
   }
   while (top != -1)
   {
      printf("%c ", pop());
   }
}

void infix_prefix(char *e)
{
   int i = -1;
   while (*e != '\0')
```

```
    {
        if (*e == ')')
        {
            push(*e);
        }
        else if (*e == '(')
        {
            char x;
            while ((x = pop()) != ')')
            {
                prefix[++i] = x;
            }
        }
        else if (isalnum(*e))
        {
            prefix[++i] = *e;
        }
        else if (*e == '^')
        {
            while (peek() == '^')
            {
                prefix[++i] = pop();
            }
            push(*e);
        }
        else if (is_operator(*e))
        {
            while (priority(*e) < priority(peek()))
                prefix[++i] = pop();
            push(*e);
        }
        else
            return;
        e++;
    }
    while (top != -1)
    {
        prefix[++i] = pop();
    }
    revstr(prefix);
    printf("%s", prefix);
}

int main()
{
    char exp[100];
    int flag;
    do
```

```
    {
        printf("\nEnter a equation: ");
        scanf("%s", exp);
        flag = 0;
        char *e;
        e = exp;
        int cop = 0, ccp = 0;
        printf("\nEquation in INFIX NOTATION: ");
        while (*e != '\0')
        {
            printf("%c", *e);
            if (*e == '(')
            {
                ++cop;
            }
            if (*e == ')')
            {
                ++ccp;
            }
            if (!isalnum(*e) && *e != '^' && *e != '*' && *e != '/' && *e != '+' && *e != '-' && *e !=
            '(' && *e != ')')
            {
                printf("\nYour enter equation is invalid...\n");
                flag = 1;
                break;
            }
            e++;
        }
        if (cop != ccp)
        {
            printf("\nYour enter equation is invalid.\n");
            flag = 1;
        }
    } while (flag);
    printf("\nAbove equation in POSTFIX NOTATION: ");
    Infix_Postfix(exp);
    revstr(exp);
    printf("\nAbove equation in PREFIX NOTATION: ");
    infix_prefix(exp);
    printf("\n");
    return 0;
}
```

OUTPUT:
Enter a equation: a+b-c/d(e*c-d)/g^j^i

Equation in INFIX NOTATION: a+b-c/d(e*c-d)/g^j^i
Above equation in POSTFIX NOTATION: a b + c d e c * d - / g j i ^ ^ / -
Above equation in PREFIX NOTATION: -+ab//cd-*ecd^g^ji

PROBLEM-11: Write a C program to perform First in First out Data Structure. You need to perform Enqueue and Dequeue operation on this data structure. You may create a header file of your own to perform the task.

PROGRAM:

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

int *queue, max, rear = -1, front = -1, data;

int isEmpty()
{
   if (front > rear || front == -1)
   {
      printf("\nQueue is not created...\n");
      return 1;
   }
   else
      return 0;
}

int isFull()
{
   if (rear == (max - 1))
   {
      printf("\nQueue reached it's maximum limit...");
      return 1;
   }
   else
      return 0;
}

void display()
{
   if (!isEmpty())
   {
      printf("\nValues in Queue: [  ");
      for (int i = front; i <= rear; i++)
         printf("%d  ", queue[i]);
      printf("]");
   }
```

```
}

void enqueue()
{
   if (!isFull())
   {
      printf("\nEnter value: ");
      scanf("%d", &data);
      queue[++rear] = data;
      if (rear == 0)
         front = 0;
      printf("One element has been added...\nData: %d",data);
   }
}

void dequeue()
{
   if (!isEmpty())
   {
      data = queue[front++];
      printf("\nOne item is deleted...\nData: %d", data);
   }
}

int main()
{
   int choice;
   printf("\nEnter how many node you need in the queue: ");
   scanf("%d", &max);
   queue = (int *)malloc(max * sizeof(int));
   printf("-------->Main Menu<--------\n1.Enqueue.\n2.Dequeue.\n3.Display.\n4.Exit.\n");
   while (choice != 4)
   {
      printf("\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("You choose to ENQUEUE...");
         enqueue();
         display();
         break;
      case 2:
         printf("You choose to DEQUEUE...");
         dequeue();
         display();
         break;
      case 3:
```

```
            printf("You choose to DISPLAY QUEUE...");
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nYou enter a invalid choice...\n");
            break;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter how many node you need in the queue: 5
-------->Main Menu<--------
1.Enqueue.
2.Dequeue.
3.Display.
4.Exit.

Enter your choice: 1
You choose to ENQUEUE...
Enter value: 23
One element has been added...
Data: 23
Values in Queue: [  23  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter value: 27
One element has been added...
Data: 27
Values in Queue: [  23  27  ]
Enter your choice: 2
You choose to DEQUEUE...
One item is deleted...
Data: 23
Values in Queue: [  27  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter value: 28
One element has been added...
Data: 28
Values in Queue: [  27  28  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter value: 26
One element has been added...
```

```
Data: 26
Values in Queue: [  27  28  26  ]
Enter your choice: 2
You choose to DEQUEUE...
One item is deleted...
Data: 27
Values in Queue: [  28  26  ]
Enter your choice: 2
You choose to DEQUEUE...
One item is deleted...
Data: 28
Values in Queue: [  26  ]
Enter your choice: 2
You choose to DEQUEUE...
One item is deleted...
Data: 26
Queue is not created...

Enter your choice: 2
You choose to DEQUEUE...
Queue is not created...

Queue is not created...

Enter your choice: 4
```

PROBLEM-12: Write a C program to implement circular queue using array. You need to perform Enqueue and Dequeue operation on this data structure. You may create a header file of your own to perform the task.

PROGRAM:

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

int *queue, max, front = -1, rear = -1, data;

int isFull()
{
   if (front == (rear + 1) % max)
   {
      printf("\nQUEUE IS FULL...");
      return 1;
   }
   else
      return 0;
}

int isEmpty()
```

```
{
   if (front == -1)
   {
      printf("\nQUEUE IS EMPTY...");
      return 1;
   }
   else
      return 0;
}

void enqueue()
{
   if (!isFull())
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      rear = (rear + 1) % max;
      queue[rear] = data;
      if (front == -1)
         front = 0;
      printf("%d has been added to queue...", data);
   }
}

void dequeue()
{
   if (!isEmpty())
   {
      data = queue[front];
      if (rear == front)
         rear = front = -1;
      else
         front = (front + 1) % max;
      printf("\n%d has been removed from queue...", data);
   }
}

void display()
{
   if (!isEmpty())
   {
      int i = front;
      printf("\nValues in queue:  [  ");
      while (i != rear)
      {
         printf("%d  ", queue[i]);
         i = (i + 1) % max;
      }
```

```
        printf("%d  ]", queue[i]);
    }
}

int main()
{
    printf("\nEnter the size of queue: ");
    scanf("%d", &max);
    queue = (int *)malloc(max * sizeof(int));
    int choice;
    printf("-------->Main Menu<--------\n1.Enqueue.\n2.Dequeue.\n3.Display.\n4.Exit.");
    while (choice != 4)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to ENQUEUE...");
            enqueue();
            display();
            break;
        case 2:
            printf("You choose to DEQUEUE...");
            dequeue();
            display();
            break;
        case 3:
            printf("You choose to DISPLAY QUEUE...");
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("\nYou enter a invalid choice...\n");
            break;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter the size of queue: 3
-------->Main Menu<--------
1.Enqueue.
2.Dequeue.
3.Display.
4.Exit.
```

```
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 23
23 has been added to queue...
Values in queue:  [  23  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 24
24 has been added to queue...
Values in queue:  [  23  24  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 25
25 has been added to queue...
Values in queue:  [  23  24  25  ]
Enter your choice: 1
You choose to ENQUEUE...
QUEUE IS FULL...
Values in queue:  [  23  24  25  ]
Enter your choice: 2
You choose to DEQUEUE...
23 has been removed from queue...
Values in queue:  [  24  25  ]
Enter your choice: 2
You choose to DEQUEUE...
24 has been removed from queue...
Values in queue:  [  25  ]
Enter your choice: 2
You choose to DEQUEUE...
25 has been removed from queue...
QUEUE IS EMPTY...
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 26
26 has been added to queue...
Values in queue:  [  26  ]
Enter your choice: 4
```

PROBLEM-13: Write a C program to implement circular queue using linked list. You need to perform Enqueue and Dequeue operation on this data structure. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node
{
   int data;
   struct node *next;
} node;

node *head = NULL;

int max, count = 0, data;

int isEmpty()
{
   if (head == NULL)
   {
      printf("\nQUEUE IS EMPTY...");
      return 1;
   }
   else
      return 0;
}

int isFull()
{
   if (count == max)
   {
      printf("\nQUEUE IS FULL...");
      return 1;
   }
   else
      return 0;
}

void enqueue()
{
   if (!isFull())
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      node *newnode = (node *)malloc(sizeof(node));
      newnode->data = data;
```

```
        newnode->next = newnode;
        if (head == NULL)
            head = newnode;
        else
        {
            node *temp = head;
            while (temp->next != head)
                temp = temp->next;
            newnode->next = head;
            temp->next = newnode;
        }
        count++;
        printf("%d has been added to queue...", data);
    }
}

void dequeue()
{
    if (!isEmpty())
    {
        data = head->data;
        node *temp = head;
        if (head->next == head)
            head = NULL;
        else
        {
            while (temp->next != head)
                temp = temp->next;
            temp->next = head->next;
            temp = head;
            head = head->next;
        }
        free(temp);
        count--;
        printf("\n%d has been removed from queue...", data);
    }
}

void display()
{
    if (!isEmpty())
    {
        node *temp = head;
        printf("\nValues in queue: [  ");
        while (temp->next != head)
        {
            printf("%d  ", temp->data);
            temp = temp->next;
```

```
        }
        printf("%d  ]", temp->data);
    }
}

int main()
{
    printf("\nEnter the size of queue: ");
    scanf("%d", &max);
    int choice;
    printf("-------->Main Menu<--------\n1.Enqueue.\n2.Dequeue.\n3.Display.\n4.Exit.");
    while (choice != 4)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to ENQUEUE...");
            enqueue();
            display();
            break;
        case 2:
            printf("You choose to DEQUEUE...");
            dequeue();
            display();
            break;
        case 3:
            printf("You choose to DISPLAY QUEUE...");
            display();
            break;
        case 4:
            printf("\nCODE EXITED SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("\nYou enter a invalid choice...\n");
            break;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter the size of queue: 3
-------->Main Menu<--------
1.Enqueue.
2.Dequeue.
3.Display.
```

```
4.Exit.
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 2
2 has been added to queue...
Values in queue: [  2  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 3
3 has been added to queue...
Values in queue: [  2  3  ]
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 4
4 has been added to queue...
Values in queue: [  2  3  4  ]
Enter your choice: 1
You choose to ENQUEUE...
QUEUE IS FULL...
Values in queue: [  2  3  4  ]
Enter your choice: 2
You choose to DEQUEUE...
2 has been removed from queue...
Values in queue: [  3  4  ]
Enter your choice: 2
You choose to DEQUEUE...
3 has been removed from queue...
Values in queue: [  4  ]
Enter your choice: 2
You choose to DEQUEUE...
4 has been removed from queue...
QUEUE IS EMPTY...
Enter your choice: 2
You choose to DEQUEUE...
QUEUE IS EMPTY...
QUEUE IS EMPTY...
Enter your choice: 1
You choose to ENQUEUE...
Enter data: 2
2 has been added to queue...
Values in queue: [  2  ]
Enter your choice: 4

CODE EXITED SUCCESSFULLY...
```

PROBLEM-14: Write a C program to implement Deque using array. Perform the operations add to front, add to rear, delete from front, delete from rear, and display. You may create a header file of your own to perform the task.

PROGRAM:

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

int max, *queue, front = -1, rear = -1, data;

int isEmpty()
{
   if (front == -1)
   {
      printf("\nQUEUE IS EMPTY...");
      return 1;
   }
   else
      return 0;
}

int isFull()
{
   if (front == (rear + 1) % max)
   {
      printf("\nQUEUE IS FULL...");
      return 1;
   }
   else
      return 0;
}

void insert_at_front()
{
   if (!isFull())
   {
      printf("\nEnter data: ");
      scanf("%d", &data);
      if (front == -1)
      {
         front = rear = 0;
         queue[front] = data;
      }
      else if (front == 0)
      {
         front = max - 1;
         queue[front] = data;
      }
```

```
        else
        {
            queue[--front] = data;
        }
        printf("%d has been ADDED at FRONT of QUEUE...", data);
    }
}

void insert_at_rear()
{
    if (!isFull())
    {
        printf("\nEnter data: ");
        scanf("%d", &data);
        rear = (rear + 1) % max;
        queue[rear] = data;
        if (front == -1)
            front = 0;
    }
    printf("%d has been ADDED at REAR of QUEUE...", data);
}

void delete_at_front()
{
    if (!isEmpty())
    {
        data = queue[front];
        if (front == rear)
            front = rear = -1;
        else
            front = (front + 1) % max;
        printf("\n%d has been REMOVED from FRONT of QUEUE...", data);
    }
}

void delete_at_rear()
{
    if (!isEmpty())
    {
        data = queue[rear];
        if (front == rear)
            front = rear = -1;
        else
        {
            if (rear == 0)
                rear = max - 1;
            else
                rear--;
```

```
        }
        printf("\n%d has been REMOVED from REAR of QUEUE...", data);
    }
}

void display()
{
    if (!isEmpty())
    {
        printf("\nValues in queue:  [  ");
        int i;
        for (i = front; i != rear; i = (i + 1) % max)
        {
            printf("%d  ", queue[i]);
        }
        printf("%d  ]", queue[i]);
    }
}

int main()
{
    printf("\nEnter size of queue: ");
    scanf("%d", &max);
    queue = (int *)malloc(max * sizeof(int));
    int choice;
    printf("----->Main Menu<-----\n1.Insert at front\n2.Insert at rear\n3.Delete at front\n4.Delete at
rear\n5.Show\n6.Exit");
    while (choice != 6)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT AT FRONT...");
            insert_at_front();
            display();
            break;
        case 2:
            printf("You choose to INSERT AT REAR...");
            insert_at_rear();
            display();
            break;
        case 3:
            printf("You choose to DELETE FROM FRONT...");
            delete_at_front();
            display();
            break;
```

```
        case 4:
            printf("You choose to DELETE FROM REAR...");
            delete_at_rear();
            display();
            break;
        case 5:
            printf("You choose to DISPLAY QUEUE...");
            display();
            break;
        case 6:
            printf("\nCODE EXITD SUCCESSFULLY...\n");
            exit(0);
        default:
            printf("\nEnter a valid choice...\n");
            break;
        }
    }
    return 0;
}
```

OUTPUT:

```
Enter size of queue: 4
----->Main Menu<-----
1.Insert at front
2.Insert at rear
3.Delete at front
4.Delete at rear
5.Show
6.Exit
Enter your choice: 1
You choose to INSERT AT FRONT...
Enter data: 2
2 has been ADDED at FRONT of QUEUE...
Values in queue:  [  2  ]
Enter your choice: 2
You choose to INSERT AT REAR...
Enter data: 5
5 has been ADDED at REAR of QUEUE...
Values in queue:  [  2  5  ]
Enter your choice: 1
You choose to INSERT AT FRONT...
Enter data: 23
23 has been ADDED at FRONT of QUEUE...
Values in queue:  [  23  2  5  ]
Enter your choice: 2
You choose to INSERT AT REAR...
Enter data: 14
14 has been ADDED at REAR of QUEUE...
```

```
Values in queue:  [  23  2  5  14  ]
Enter your choice: 1
You choose to INSERT AT FRONT...
QUEUE IS FULL...
Values in queue:  [  23  2  5  14  ]
Enter your choice: 3
You choose to DELETE FROM FRONT...
23 has been REMOVED from FRONT of QUEUE...
Values in queue:  [  2  5  14  ]
Enter your choice: 4
You choose to DELETE FROM REAR...
14 has been REMOVED from REAR of QUEUE...
Values in queue:  [  2  5  ]
Enter your choice: 4
You choose to DELETE FROM REAR...
5 has been REMOVED from REAR of QUEUE...
Values in queue:  [  2  ]
Enter your choice: 3
You choose to DELETE FROM FRONT...
2 has been REMOVED from FRONT of QUEUE...
QUEUE IS EMPTY...
Enter your choice: 6

CODE EXITD SUCCESSFULLY...
```

PROBLEM-15: Write a C program to implement Deque using linked list. Perform the operations add to front, add to rear, delete from front, delete from rear, and display. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct node{
   int data;
   struct node *next;
}node;

node *head = NULL;
node *last = NULL;
int max, count = 0, data;

int isEmpty(){
   if(head == NULL){
      printf("\nQUEUE IS EMPTY...");
      return 1;
   }
   else return 0;
```

```
}

int isFull(){
   if(count == max){
      printf("\nQUEUE IS FULL...");
      return 1;
   }
   else return 0;
}

void insert_at_front(){
   if(!isFull()){
      node *newnode = (node*)malloc(sizeof(node));
      printf("\nEnter data: ");
      scanf("%d",&data);
      newnode->data = data;
      newnode->next = newnode;
      if(head == NULL)
      head = last = newnode;
      else{
         newnode->next = head;
         last->next = newnode;
         head = newnode;
      }
      count++;
      printf("%d has been ADDED at FRONT of QUEUE...", data);
   }
}

void insert_at_rear(){
   if(!isFull()){
      node *newnode = (node*)malloc(sizeof(node));
      printf("\nEnter data: ");
      scanf("%d",&data);
      newnode->data = data;
      newnode->next = newnode;
      if(head == NULL)
         head = last = newnode;
      else{
         newnode->next = head;
         last->next = newnode;
         last = newnode;
      }
      count++;
      printf("%d has been ADDED at REAR of QUEUE...", data);
   }
}
```

```
void delete_at_front(){
   if(!isEmpty()){
      data = head->data;
      node *temp = head;
      if(head == head->next)
      head = last = NULL;
      else{
         head = head->next;
         last->next = head;
      }
      free(temp);
      printf("\n%d has been REMOVED from FRONT of QUEUE...", data);
      count--;
   }
}

void delete_at_rear(){
   if(!isEmpty()){
      data = last->data;
      node *temp = head;
      if(head == head->next)
      head = last = NULL;
      else{
         while(temp->next!=last){
            temp = temp->next;
         }
         last = temp;
         temp = temp->next;
         last->next = head;
      }
      free(temp);
      count--;
      printf("\n%d has been REMOVED from REAR of QUEUE...", data);
   }
}

void display(){
   if(!isEmpty()){
      node *temp = head;
      printf("\nValues in queue:  [  ");
      do{
         printf("%d  ",temp->data);
         temp = temp->next;
      }while(temp != head);
      printf("]");
   }
}
```

```
int main()
{
   printf("\nEnter size of queue: ");
   scanf("%d", &max);
   int choice;
   printf("----->Main Menu<-----\n1.Insert at front\n2.Insert at rear\n3.Delete at front\n4.Delete at
rear\n5.Show\n6.Exit");
   while (choice != 6)
   {
      printf("\nEnter your choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
         printf("You choose to INSERT AT FRONT...");
         insert_at_front();
         display();
         break;
      case 2:
         printf("You choose to INSERT AT REAR...");
         insert_at_rear();
         display();
         break;
      case 3:
         printf("You choose to DELETE FROM FRONT...");
         delete_at_front();
         display();
         break;
      case 4:
         printf("You choose to DELETE FROM REAR...");
         delete_at_rear();
         display();
         break;
      case 5:
         printf("You choose to DISPLAY QUEUE...");
         display();
         break;
      case 6:
         printf("\nCODE EXITD SUCCESSFULLY...\n");
         exit(0);
      default:
         printf("\nEnter a valid choice...\n");
         break;
      }
   }
   return 0;
}
```

OUTPUT:

```
Enter size of queue: 4
----->Main Menu<-----
1.Insert at front
2.Insert at rear
3.Delete at front
4.Delete at rear
5.Show
6.Exit
Enter your choice: 1
You choose to INSERT AT FRONT...
Enter data: 2
2 has been ADDED at FRONT of QUEUE...
Values in queue:  [  2  ]
Enter your choice: 2
You choose to INSERT AT REAR...
Enter data: 5
5 has been ADDED at REAR of QUEUE...
Values in queue:  [  2  5  ]
Enter your choice: 1
You choose to INSERT AT FRONT...
Enter data: 4
4 has been ADDED at FRONT of QUEUE...
Values in queue:  [  4  2  5  ]
Enter your choice: 2
You choose to INSERT AT REAR...
Enter data: 7
7 has been ADDED at REAR of QUEUE...
Values in queue:  [  4  2  5  7  ]
Enter your choice: 1
You choose to INSERT AT FRONT...
QUEUE IS FULL...
Values in queue:  [  4  2  5  7  ]
Enter your choice: 3
You choose to DELETE FROM FRONT...
4 has been REMOVED from FRONT of QUEUE...
Values in queue:  [  2  5  7  ]
Enter your choice: 4
You choose to DELETE FROM REAR...
7 has been REMOVED from REAR of QUEUE...
Values in queue:  [  2  5  ]
Enter your choice: 3
You choose to DELETE FROM FRONT...
2 has been REMOVED from FRONT of QUEUE...
Values in queue:  [  5  ]
Enter your choice: 4
You choose to DELETE FROM REAR...
5 has been REMOVED from REAR of QUEUE...
```

```
QUEUE IS EMPTY...
Enter your choice: 1
You choose to INSERT AT FRONT...
Enter data: 2
2 has been ADDED at FRONT of QUEUE...
Values in queue:  [  2  ]
Enter your choice: 6

CODE EXITD SUCCESSFULLY...
```

PROBLEM-16: Write a C program to implement a Binary Tree and perform Add Node, Delete Node, and Breadth First Traversal operation on it. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct tnode
{
   int data;
   struct tnode *right, *left;
} tnode;

typedef struct node
{
   tnode *key;
   struct node *next;
} queue;

queue *head = NULL;

int isEmpty()
{
   if (head == NULL)
      return 1;
   else
      return 0;
}

void push(tnode *key)
{
   queue *newnode = (queue *)malloc(sizeof(queue));
   newnode->key = key;
   newnode->next = NULL;
   if (head == NULL)
      head = newnode;
   else
   {
      queue *temp = head;
      while (temp->next != NULL)
         temp = temp->next;
      temp->next = newnode;
   }
}

tnode *pop()
{
```

```
    if (!isEmpty())
    {
        tnode *key = head->key;
        queue *temp = head;
        head = head->next;
        free(temp);
        return key;
    }
}

void clearQ()
{
    while (head != NULL)
    {
        pop();
    }
    return;
}

tnode *newnode(int data)
{
    tnode *newnode = (tnode *)malloc(sizeof(tnode));
    newnode->data = data;
    newnode->right = newnode->left = NULL;
    return newnode;
}

tnode *insert(tnode *root, int data)
{
    if (root == NULL)
    {
        root = newnode(data);
        return root;
    }
    else
    {
        clearQ();
        ;
        push(root);
        while (!isEmpty())
        {
            tnode *n = pop();
            if (n->left == NULL)
            {
                n->left = newnode(data);
                return root;
            }
            else
```

```
            {
               push(n->left);
            }
            if (n->right == NULL)
            {
               n->right = newnode(data);
               return root;
            }
            else
            {
               push(n->right);
            }
         }
      }
}

void removenode(tnode *root, tnode *n)
{
   if (root == NULL)
   {
      return;
   }
   if (root == n)
   {
      free(n);
      root = NULL;
      return;
   }
   if (root->left == n)
   {
      free(n);
      root->left = NULL;
      return;
   }
   if (root->right == n)
   {
      free(n);
      root->right = NULL;
      return;
   }
   removenode(root->left, n);
   removenode(root->right, n);
}

tnode *deletenode(tnode *root, int data)
{
   if (root == NULL)
   {
```

```
        return NULL;
    }
    if (root->left == NULL && root->right == NULL)
    {
        if (root->data == data)
        {
            free(root);
            root = NULL;
            return root;
            printf("Node has been deleted from Binary tree...");
        }
        printf("Data not present...");
        return root;
    }
    clearQ();
    push(root);
    tnode *current, *keynode = NULL;
    while (!isEmpty())
    {
        current = pop();
        if (current->data == data)
        {
            keynode = current;
        }
        if (current->left != NULL)
        {
            push(current->left);
        }
        if (current->right != NULL)
        {
            push(current->right);
        }
    }
    if (keynode != NULL)
    {
        keynode->data = current->data;
        removenode(root, current);
        printf("Node has been deleted from Binary tree...");
    }
    else
    {
        printf("Data not present...");
    }
    return root;
}

tnode *levelorder(tnode *root)
{
```

```
    if (root == NULL)
    {
        return NULL;
    }
    clearQ();
    push(root);
    while (!isEmpty())
    {
        tnode *n = pop();
        printf("%d  ", n->data);
        if (n->left != NULL)
        {
            push(n->left);
        }
        if (n->right != NULL)
        {
            push(n->right);
        }
    }
    return root;
}

int main()
{
    tnode *root = NULL;
    int data;
    printf("\n--->Main Menu<---\n1.Insert node in binary tree.\n2.Delete node from binary tree.\
n3.Level order traversal.\n4.Exit.");
    int choice = 0;
    while (choice != 5)
    {
        printf("\nEnter your choice: ");
        scanf("%d", &choice);
        switch (choice)
        {
        case 1:
            printf("You choose to INSERT a node in Binary tree...");
            printf("\nEnter data: ");
            scanf("%d", &data);
            root = insert(root, data);
            printf("A node has been ADDED to Binary tree...");
            break;
        case 2:
            printf("You choose to DELETE a node from Binary tree...");
            printf("\nEnter data: ");
            scanf("%d", &data);
            root = deletenode(root, data);
            break;
```

```
        case 3:
            printf("You choose to DISPLAY tree using LEVEL ORDER TRAVERSAL...");
            printf("\nValues in Binary Tree:  [  ");
            levelorder(root);
            printf("]");
            break;
        case 4:
            printf("\nCODE SUCCESSFULLY TERMINATED...\n");
            exit(0);
        default:
            printf("\nEnter valid choice...\n");
            break;
        }
    }
}
```

OUTPUT:

```
--->Main Menu<---
1.Insert node in binary tree.
2.Delete node from binary tree.
3.Level order traversal.
4.Exit.
Enter your choice: 1
You choose to INSERT a node in Binary tree...
Enter data: 2
A node has been ADDED to Binary tree...
Enter your choice: 1
You choose to INSERT a node in Binary tree...
Enter data: 3
A node has been ADDED to Binary tree...
Enter your choice: 1
You choose to INSERT a node in Binary tree...
Enter data: 4
A node has been ADDED to Binary tree...
Enter your choice: 1
You choose to INSERT a node in Binary tree...
Enter data: 5
A node has been ADDED to Binary tree...
Enter your choice: 1
You choose to INSERT a node in Binary tree...
Enter data: 6
A node has been ADDED to Binary tree...
Enter your choice: 3
You choose to DISPLAY tree using LEVEL ORDER TRAVERSAL...
Values in Binary Tree:  [  2  3  4  5  6  ]
Enter your choice: 2
You choose to DELETE a node from Binary tree...
Enter data: 3
```

```
Node has been deleted from Binary tree...
Enter your choice: 3
You choose to DISPLAY tree using LEVEL ORDER TRAVERSAL...
Values in Binary Tree: [ 2 6 4 5 ]
Enter your choice: 4

CODE SUCCESSFULLY TERMINATED...
```

PROBLEM-17: Write a C program to implement a Binary Search Tree and perform Add Node, Delete Node. Also perform the Inorder, Preorder, and Postorder traversal on it. You may create a header file of your own to perform the task.

PROGRAM:

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

typedef struct tnode
{
   int data;
   struct tnode *right;
   struct tnode *left;
} tnode;

tnode *insert(tnode *, int);
void Inorder(tnode *);
void Preorder(tnode *);
void Postorder(tnode *);
tnode *minValueNode();
tnode *deleteNode(tnode *, int);

int main()
{
   tnode *root = NULL;
   int choice, item, n, i;
   do
   {
      printf("\n\nBinary Tree Operations\n");
      printf("\n1. Insert node in Binaery tree.");
      printf("\n2. Traverse in Inorder.");
      printf("\n3. Traverse in Preorder.");
      printf("\n4. Traverse in Postorder.");
      printf("\n5. Delete a node.");
      printf("\n6. Exit\n");
      printf("\nEnter Choice: ");
      scanf("%d", &choice);
      switch (choice)
      {
      case 1:
```

```
            printf("\nEnter data for node: ");
            scanf("%d", &item);
            root = insert(root, item);
            break;
        case 2:
            printf("\nBinary tree Traversal in INORDER \n");
            Inorder(root);
            break;
        case 3:
            printf("\nBinary tree Traversal in PREORDER \n");
            Preorder(root);
            break;
        case 4:
            printf("\nBinary tree Traversal in POSTORDER \n");
            Postorder(root);
            break;
        case 5:
            printf("\nEnter which node you want to delete: ");
            scanf("%d", &item);
            deleteNode(root, item);
            break;
        case 6:
            printf("\n\nTerminating \n\n");
            break;
        default:
            printf("\n\nInvalid Option !!! Try Again !! \n\n");
            break;
        }
    } while (choice != 6);
    return 0;
}

tnode *newnode(int value)
{
    tnode *newnode = (tnode *)malloc(sizeof(tnode));
    newnode->left = newnode->right = NULL;
    newnode->data = value;
}

tnode *insert(tnode *root, int item)
{
    if (root == NULL)
        return newnode(item);
    else
    {
        if (item < root->data)
            root->left = insert(root->left, item);
        else if (item > root->data)
```

```
            root->right = insert(root->right, item);
        else
            printf("Duplicate Element!! Not Allowed...");
        return root;
    }
}

void Inorder(tnode *root)
{
    if (root != NULL)
    {
        Inorder(root->left);
        printf(" %d ", root->data);
        Inorder(root->right);
    }
}

void Preorder(tnode *root)
{
    if (root != NULL)
    {
        printf(" %d ", root->data);
        Preorder(root->left);
        Preorder(root->right);
    }
}

void Postorder(tnode *root)
{
    if (root != NULL)
    {
        Postorder(root->left);
        Postorder(root->right);
        printf(" %d ", root->data);
    }
}

tnode *minValueNode(tnode *node)
{
    tnode *current = node;
    while (current && current->left != NULL)
        current = current->left;
    return current;
}

tnode *deleteNode(tnode *root, int key)
{
    if (root == NULL)
```

```
        return root;
    if (key < root->data)
        root->left = deleteNode(root->left, key);
    else if (key > root->data)
        root->right = deleteNode(root->right, key);
    else
    {
        if (root->left == NULL)
        {
            tnode *temp = root->right;
            free(root);
            return temp;
        }
        else if (root->right == NULL)
        {
            tnode *temp = root->left;
            free(root);
            return temp;
        }
        tnode *temp = minValueNode(root->right);
        root->data = temp->data;
        root->right = deleteNode(root->right, temp->data);
    }
    return root;
}
```

OUTPUT:

```
Binary Tree Operations

1. Insert node in Binaery tree.
2. Traverse in Inorder.
3. Traverse in Preorder.
4. Traverse in Postorder.
5. Delete a node.
6. Exit

Enter Choice: 1
Enter data for node: 23

Enter Choice: 1
Enter data for node: 2

Enter Choice: 1
Enter data for node: 17

Enter Choice: 1
Enter data for node: 29
```

```
Enter Choice: 1
Enter data for node: 1

Enter Choice: 1
Enter data for node: 42

Enter Choice: 1
Enter data for node: 25

Enter Choice: 3
Binary tree Traversal in PREORDER
 23  2  1  17  29  25  42
Enter Choice: 2
Binary tree Traversal in INORDER
 1  2  17  23  25  29  42
Enter Choice: 4
Binary tree Traversal in POSTORDER
 1  17  2  25  42  29  23
Enter Choice: 5
Enter which node you want to delete: 25

Enter Choice: 2
Binary tree Traversal in INORDER
 1  2  17  23  29  42
Enter Choice: 6

CODE IS SUCCESSFULLY TERMINATED...
```

PROBLEM:18: Write a C program to implement Bubble Sort.

PROGRAM:

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

int main()
{
   printf("\nEnter size of array: ");
   int size;
   scanf("%d", &size);
   int *arr = (int *)malloc(size * sizeof(int));
   printf("\nEnter elements in array: ");
   for (int i = 0; i < size; i++)
   {
      scanf("%d", &arr[i]);
   }
   printf("\nElements in array before sorting: [  ");
   for (int i = 0; i < size; i++)
   {
```

```
        printf("%d  ", arr[i]);
    }
    printf("]\n");
    int i, j;
    for(i=0; i<size-1; i++){
        for(j = 0; j<size-i-1; j++){
            if(arr[j]>arr[j+1]){
                int temp = arr[j];
                arr[j] = arr[j+1];
                arr[j+1] = temp;
            }
        }
    }
    printf("\nElements in array after sorting: [  ");
    for (int i = 0; i < size; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("]\n");
}
```

OUTPUT:
Enter size of array: 6

Enter elements in array: 10 12 2 5 7 25

Elements in array before sorting: [10 12 2 5 7 25]

Elements in array after sorting: [2 5 7 10 12 25]

PROBLEM-19: Write a C program to implement Selection Sort.

PROGRAM:

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

int main()
{
    printf("\nEnter size of array: ");
    int size;
    scanf("%d", &size);
    int *arr = (int *)malloc(size * sizeof(int));
    printf("\nEnter elements in array: ");
    for (int i = 0; i < size; i++)
    {
        scanf("%d", &arr[i]);
    }
    printf("\nElements in array before sorting: [  ");
```

```
    for (int i = 0; i < size; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("]\n");
    int i, j, position, swap;
    for (i = 0; i < (size - 1); i++) {
        position = i;
        for (j = i + 1; j < size; j++) {
            if (arr[position] > arr[j])
                position = j;
        }
        if (position != i) {
            swap = arr[i];
            arr[i] = arr[position];
            arr[position] = swap;
        }
    }
    printf("\nElements in array after sorting: [  ");
    for (int i = 0; i < size; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("]\n");
}
```

OUTPUT:

```
Enter size of array: 6

Enter elements in array: 12 24 2 10 56 1

Elements in array before sorting: [  12  24  2  10  56  1  ]

Elements in array after sorting: [  1  2  10  12  24  56  ]
```

PROBLEM-20: Write a C program to implement Insertion Sort.

PROGRAM:

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

int main(){
    printf("\nEnter size of array: ");
    int size;
    scanf("%d", &size);
    int *arr = (int*)malloc(size*sizeof(int));
    printf("\nEnter elements in array: ");
    for(int i = 0; i<size; i++){
```

```
    scanf("%d",&arr[i]);
  }
  printf("\nElements in array before sorting: [  ");
  for(int i = 0; i<size; i++){
    printf("%d  ",arr[i]);
  }
  printf("]\n");
  int i, j;
  for(i = 1; i<size; i++){
    int key = arr[i];
    for(j = i-1; arr[j]>key && j>=0; j--){
      arr[j+1] = arr[j];
    }
    arr[j+1] = key;
  }
  printf("\nElements in array after sorting: [  ");
  for(int i = 0; i<size; i++){
    printf("%d  ",arr[i]);
  }
  printf("]\n");
}
```

OUTPUT:
Enter size of array: 6

Enter elements in array: 12 24 2 10 56 1

Elements in array before sorting: [12 24 2 10 56 1]

Elements in array after sorting: [1 2 10 12 24 56]

PROBLEM-21: Write a C program to implement Shell Sort.

PROGRAM:

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

int main()
{
  printf("\nEnter size of array: ");
  int size;
  scanf("%d", &size);
  int *arr = (int *)malloc(size * sizeof(int));
  printf("\nEnter elements in array: ");
  for (int i = 0; i < size; i++)
  {
    scanf("%d", &arr[i]);
  }
```

```
    printf("\nElements in array before sorting: [  ");
    for (int i = 0; i < size; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("]\n");
    int gap, i, j, key;
    for (gap = size / 2; gap >= 1; gap = gap / 2)
    {
        for (i = gap; i < size; i++)
        {
            key = arr[i];
            for (j = i-gap; j >= 0 && arr[j] > key; j = j - gap)
            {
                arr[j + gap] = arr[j];
            }
            arr[j + gap] = key;
        }
    }
    printf("\nElements in array after sorting: [  ");
    for (int i = 0; i < size; i++)
    {
        printf("%d  ", arr[i]);
    }
    printf("]\n");
}
```

OUTPUT:

Enter size of array: 6

Enter elements in array: 12 24 2 10 56 1

Elements in array before sorting: [12 24 2 10 56 1]

Elements in array after sorting: [1 2 10 12 24 56]