DSU Program's Codes By Rajan Sir

1.Array Implementation: Linear Search

```
#include <stdio.h>

int search(int arr[], int N, int x)
{
   for (int i = 0; i < N; i++)
      if (arr[i] == x)
         return i;
   return -1;
}

int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int x = 10;
   int result = search(arr, 5, x);
   (result == -1)
      ? printf("Element is not present in array")
      : printf("Element is present at index %d", result);
   return 0;
}
```

---

2. Array Implementation: Binary Search

```
#include <stdio.h>
int binarySearch(int arr[], int low, int high, int x)
{
   while (low <= high) {
      int mid = (low + high ) / 2;
      if (arr[mid] == x)
         return mid;
      if (arr[mid] < x)
         low = mid + 1;
      else
         high = mid - 1;
   }
   return -1;
}
int main(void)
{
   int arr[] = { 2, 3, 4, 10, 40 };
   int n = sizeof(arr) / sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n - 1, x);
  if(result == -1) printf("Element is not present in array");
  else printf("Element is present at index %d",result);
```

```
}
```

3.Linked List: Singly Linked List Implementation

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

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

struct Node* head = NULL;

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

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

int main() {
   int choice, value;


   while (1) {
      printf("\n1. Insert\n2. Display\n3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("Enter the value to insert: ");
            scanf("%d", &value);
            insert(value);
            break;
         case 2:
            display();
```

```
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice! Try again.\n");
    }
  }

}
```

---

4. Doubly Linked List Implementation

```
#include <stdio.h>

#include <stdlib.h>

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

struct Node* head = NULL;

// Function to insert at the beginning
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;
  printf("Inserted %d at the beginning.\n", value);
}

// Function to delete from the end
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;
    }

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

// Function to display the list
void display() {
    struct Node* temp = head;
    if (temp == NULL) {
        printf("List is empty.\n");
        return;
    }

    printf("Doubly Linked List: ");
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

int main() {
    int choice, value;


    while (1) {
        printf("\n1. Insert at Beginning\n2. Delete from End\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtBeginning(value);
                break;
            case 2:
                deleteFromEnd();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
```

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

}
```

---

5. Circular Linked List Implementation

```
#include <stdio.h>

#include <stdlib.h>

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

struct Node* head = NULL;

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

    if (head == NULL) {
        head = newNode;
        newNode->next = head;
    } else {
        while (temp->next != head) {
            temp = temp->next;
        }
        temp->next = newNode;
        newNode->next = head;
    }
    printf("Inserted %d at the end.\n", value);
}

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

    struct Node* temp = head;

    if (head->next == head) {
```

```
        head = NULL;
    } else {
        struct Node* last = head;
        while (last->next != head) {
            last = last->next;
        }
        head = head->next;
        last->next = head;
    }

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


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

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

void main() {
    int choice, value;


    while (1) {
        printf("\n1. Insert at End\n2. Delete from Beginning\n3. Display\n4. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insertAtEnd(value);
                break;
            case 2:
                deleteFromBeginning();
                break;
            case 3:
```

```
            display();
            break;
         case 4:
            exit(0);
         default:
            printf("Invalid choice! Try again.\n");
      }
   }

}
```

6.Stack: Array Implementation

```
#include <stdio.h>

#include <stdlib.h>

#define SIZE 4
int top = -1, inp_array[SIZE];
int main()
{

   while (1)
   {
      printf("\nPerform operations on the stack:");
      printf("\n1.Push the element\n2.Pop the element\n3.Show\n4.End");
      printf("\n\nEnter the choice: ");
      scanf("%d", &choice);

      switch (choice)
      {
      case 1:
         push();
         break;
      case 2:
         pop();
         break;
      case 3:
         show();
         break;
      case 4:
         exit(0);

      default:
         printf("\nInvalid choice!!");
      }
   }
}
```

```
void push()
{
   int x;

   if (top == SIZE - 1)
   {
      printf("\nOverflow!!");
   }
   else
   {
      printf("\nEnter the element to be added onto the stack: ");
      scanf("%d", &x);
      top = top + 1;
      inp_array[top] = x;
   }
}

void pop()
{
   if (top == -1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nPopped element: %d", inp_array[top]);
      top = top - 1;
   }
}

void show()
{
   if (top == -1)
   {
      printf("\nUnderflow!!");
   }
   else
   {
      printf("\nElements present in the stack: \n");
      for (int i = top; i >= 0; --i)
         printf("%d\n", inp_array[i]);
   }
}
```

7.Queue: Array Implementation (Enque & Deque & Display)

```
#include <stdio.h>
# define SIZE 100
```

```
int inp_arr[SIZE];
int Rear = - 1;
int Front = - 1;
int main()
{
   int ch;
   while (1)
   {
      printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");

      printf("Enter your choice of operations : ");
      scanf("%d", &ch);
      switch (ch)
      {
         case 1:
         enqueue();
         break;
         case 2:
         dequeue();
         break;
         case 3:
         show();
         break;
         case 4:
         exit(0);
         default:
         printf("Incorrect choice \n");
      }
   }
}

void enqueue()
{
   int insert_item;
   if (Rear == SIZE - 1)
     printf("Overflow \n");
   else
   {
      if (Front == - 1)

      Front = 0;
      printf("Element to be inserted in the Queue\n : ");
      scanf("%d", &insert_item);
      Rear = Rear + 1;
      inp_arr[Rear] = insert_item;
   }
}

void dequeue()
```

```
{
   if (Front == - 1 || Front > Rear)
   {
      printf("Underflow \n");
      return ;
   }
   else
   {
      printf("Element deleted from the Queue: %d\n", inp_arr[Front]);
      Front = Front + 1;
   }
}

void show()
{

   if (Front == - 1)
      printf("Empty Queue \n");
   else
   {
      printf("Queue: \n");
      for (int i = Front; i <= Rear; i++)
         printf("%d ", inp_arr[i]);
      printf("\n");
   }
}
```

Insert & display value in array

```
#include <stdio.h>

int main()
{
   int arr[100] = { 0 };
   int i, x, pos, n = 10;
   for (i = 0; i < 10; i++)
      arr[i] = i + 1;

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

   x = 50;

   pos = 5;

   n++;
   // shift elements forward
   for (i = n - 1; i >= pos; i--)
```

```
        arr[i] = arr[i - 1];

    // insert x at pos
    arr[pos - 1] = x;

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

    return 0;
}
```