## 1.Write a c program to implementation of static stack using array with push() & display() function.

```
#include <stdio.h>

#define MAX 5

int stack[MAX];

int top = -1;

void push(int val) {

   if (top == MAX - 1)

      printf("Stack Overflow!\n");

   else {

      top++;

      stack[top] = val;

      printf("%d pushed.\n", val);      }        }

void display() {

   if (top == -1)

      printf("Stack Empty!\n");

   else {

      int i;   // ? declare before using in the for loop

      printf("Stack elements: ");

      for (i = top; i >= 0; i--)

         printf("%d ", stack[i]);

      printf("\n");  }        }

int main() {

   int choice, val;

   do {

      printf("\n1. Push  2. Display  3. Exit\nEnter choice: ");

      scanf("%d", &choice);

      switch (choice) {

         case 1:
                printf("Enter value: ");
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);
    return 0;
}
```

---

**2. Write a C program to sort an array using bubble sort.**

```
#include <stdio.h>
int main() {
    int arr[100], n, i, j, temp;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    // Bubble sort algorithm
    for (i = 0; i < n - 1; i++) {
        for (j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
    return 0;
}
```

---

**3. Write a c program to evaluate a postfix expression with init() & push() function.**

```
#include <stdio.h>
#include <ctype.h>  // for isdigit()
#define MAX 50
int stack[MAX];
int top;
// initialize stack
void init() {
    top = -1;
}
// push function
void push(int val) {
    if (top == MAX - 1)
        printf("Stack Overflow!\n");
    else
```

```
        stack[++top] = val;
}
// pop function
int pop() {
    if (top == -1) {
        printf("Stack Underflow!\n");
        return -1;
    } else
        return stack[top--];
}
// evaluate postfix expression
int evaluate(char postfix[]) {
    int i;
    char ch;
    int val1, val2, result;
    for (i = 0; postfix[i] != '\0'; i++) {
        ch = postfix[i];
        if (isdigit(ch)) {
            push(ch - '0'); // convert char to int
        } else {
            val2 = pop();
            val1 = pop();
            switch (ch) {
                case '+': result = val1 + val2; break;
                case '-': result = val1 - val2; break;
                case '*': result = val1 * val2; break;
                case '/': result = val1 / val2; break;
                default:
                    printf("Invalid operator: %c\n", ch);
                return -1;
            }
            push(result);
        }
    }
    return pop(); // final result
}
// main function
int main() {
    char postfix[50];
    int result;
    init();
    printf("Enter a postfix expression (single-digit operands): ");
    scanf("%s", postfix);
    result = evaluate(postfix);
    printf("Result = %d\n", result);
    return 0; }
```

---

**4. Write a c program to implementation of static Linked List with create() & insert() function.**

```
#include <stdio.h>
#define MAX 10
// structure for a node
struct Node {
    int data;
    int next;
} node[MAX];
int head = -1;  // head pointer
int avail = 0;  // points to next free node
```

```
// initialize available list
void init() {
    for (int i = 0; i < MAX - 1; i++)
        node[i].next = i + 1;
    node[MAX - 1].next = -1;
}
// allocate a new node from static memory
int getNode() {
    if (avail == -1) {
        printf("No more free nodes available!\n");
        return -1;
    }
    int newNode = avail;
    avail = node[avail].next;
    return newNode;
}
// create linked list
void create(int n) {
    int temp, newNode, val;
    head = getNode();
    printf("Enter data for node 1: ");
    scanf("%d", &val);
    node[head].data = val;
    node[head].next = -1;
    temp = head;
    for (int i = 2; i <= n; i++) {
        newNode = getNode();
        if (newNode == -1) return;
        printf("Enter data for node %d: ", i);
        scanf("%d", &val);
        node[newNode].data = val;
        node[newNode].next = -1;
        node[temp].next = newNode;
        temp = newNode;
    }
}
// insert new node at end
void insert(int val) {
    int newNode = getNode();
    if (newNode == -1) return;
    node[newNode].data = val;
    node[newNode].next = -1;
    if (head == -1)
        head = newNode;
    else {
        int temp = head;
        while (node[temp].next != -1)
            temp = node[temp].next;
        node[temp].next = newNode;
    }
}
// display linked list
void display() {
    if (head == -1) {
        printf("List is empty!\n");
        return;
    }
    int temp = head;
    printf("Linked List: ");
    while (temp != -1) {
        printf("%d ", node[temp].data);
        temp = node[temp].next;
    }
    printf("\n");
}
int main() {
    int n, val, choice;
    init();
    printf("How many nodes to create initially? ");
    scanf("%d", &n);
    create(n);
    display();
    do {
        printf("\n1. Insert  2. Display  3. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                insert(val);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);
    return 0;
}
```

---

**5. Write a C program to implement insertion sort.**

```
#include <stdio.h>
int main() {
    int arr[100], n, i, j, key;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++)
        scanf("%d", &arr[i]);
    // Insertion sort algorithm
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        // Move elements greater than key one position ahead
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
    printf("Sorted array:\n");
    for (i = 0; i < n; i++)
```

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

---

## 6. Write a c program Static implementation of queue check isFull() & insert() function.

```
#include <stdio.h>
#define MAX 5
int queue[MAX];
int front = -1, rear = -1;
// check if queue is full
int isFull() {
    return rear == MAX - 1;
}
// insert element into queue
void insert(int val) {
    if (isFull()) {
        printf("Queue Overflow!\n");
        return;
    }
    if (front == -1) // first element
        front = 0;
    rear++;
    queue[rear] = val;
    printf("%d inserted into queue.\n", val);
}
// display queue elements
void display() {
    if (front == -1 || front > rear) {
        printf("Queue is empty!\n");
        return;
    }
    printf("Queue elements: ");
    for (int i = front; i <= rear; i++)
        printf("%d ", queue[i]);
    printf("\n");
}
int main() {
    int choice, val;
    do {
        printf("\n1. Insert  2. Display  3. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &val);
                insert(val);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 3);
    return 0;
}
```

---

**7. Write a c program to implementation of static stack using array with check isEmpty() & pop() function.**

```
#include <stdio.h>
#define MAX 5
int stack[MAX];
int top = -1;
// check if stack is empty
int isEmpty() {
    return top == -1;
}
// push element onto stack
void push(int val) {
    if (top == MAX - 1)
        printf("Stack Overflow!\n");
    else {
        top++;
        stack[top] = val;
        printf("%d pushed onto stack.\n", val);
    }
}
// pop element from stack
int pop() {
    if (isEmpty()) {
        printf("Stack Underflow!\n");
        return -1;
    } else {
        int val = stack[top];
        top--;
        return val;
    }
}
// display stack elements
void display() {
    if (isEmpty()) {
        printf("Stack is empty!\n");
        return;
    }
    printf("Stack elements: ");
    for (int i = top; i >= 0; i--)
        printf("%d ", stack[i]);
    printf("\n");
}
int main() {
    int choice, val;
    do {
        printf("\n1. Push  2. Pop  3. Display  4. Exit\nEnter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &val);
                push(val);
                break;
            case 2:
                val = pop();
                if (val != -1)
                    printf("%d popped from stack.\n", val);
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Exiting...\n");
                break;
            default:
                printf("Invalid choice!\n");
        }
    } while (choice != 4);
    return 0;
}
```

---

**8. Write a program to implement quick sort.**

```
#include <stdio.h>
// Function to swap two integers
void swap(int *a, int *b) {
   int temp = *a;
   *a = *b;
   *b = temp;
}
// Partition function
int partition(int arr[], int low, int high) {
   int pivot = arr[high];  // choose last element as pivot
   int i = low - 1;        // index of smaller element
   for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return i + 1;
}
// Quick sort function
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);   // sort left subarray
        quickSort(arr, pi + 1, high);  // sort right subarray
    }
}
// Display function
void display(int arr[], int n) {
    for (int i = 0; i < n; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
// Main function
int main() {
    int arr[100], n;
    printf("Enter number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (int i = 0; i < n; i++)
```

```
        scanf("%d", &arr[i]);
    quickSort(arr, 0, n - 1);
    printf("Sorted array:\n");
    display(arr, n);
    return 0;
}
```

---

**9. Write a c program to implementation of Singly Linked List with create() & display() function.**

```
#include <stdio.h>
#include <stdlib.h>
// Structure for a node
struct Node {
    int data;
    struct Node *next;
};
struct Node *head = NULL; // Head pointer for the linked list
// Function to create a linked list
void create(int n) {
    struct Node *newNode, *temp;
    int data, i;
    if (n <= 0) {
        printf("Invalid number of nodes.\n");
        return;
    }
    for (i = 1; i <= n; i++) {
        newNode = (struct Node*)malloc(sizeof(struct Node));
        if (newNode == NULL) {
            printf("Memory allocation failed!\n");
            return;
        }
        printf("Enter data for node %d: ", i);
        scanf("%d", &data);
        newNode->data = data;
        newNode->next = NULL;
        if (head == NULL) {
            head = newNode;  // First node becomes head
        } else {
            temp = head;
            while (temp->next != NULL)
                temp = temp->next;
            temp->next = newNode;  // Add new node at the end
        }
    }
    printf("Linked list created successfully.\n");
}
// Function to display the linked list
void display() {
    struct Node *temp = head;
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }
    printf("Linked List: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
// Main function
int main() {
    int n, choice;
    while (1) {
        printf("\n--- Singly Linked List Menu ---\n");
        printf("1. Create Linked List\n");
        printf("2. Display Linked List\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter number of nodes: ");
                scanf("%d", &n);
                create(n);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
    return 0;   }
```

**10. Write a c program to implementation of Dynamic queue with insert() & display() function.**

```
#include <stdio.h>
#include <stdlib.h>
// Define structure for a queue node
struct Node {
    int data;
    struct Node *next;
};
// Define front and rear pointers
struct Node *front = NULL;
struct Node *rear = NULL;
// Function to insert (enqueue) an element into the queue
void insert(int value) {
    struct Node *newNode = (struct Node*) malloc(sizeof(struct Node));
    if (newNode == NULL) {
        printf("Memory allocation failed!\n");
        return;
    }
    newNode->data = value;
    newNode->next = NULL;
    if (rear == NULL) {
        // Queue is empty
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("%d inserted into the queue.\n", value);
```

```
}
// Function to display (traverse) the queue
void display() {
    if (front == NULL) {
        printf("Queue is empty.\n");
        return;
    }
    struct Node *temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}
// Main function
int main() {
    int choice, value;
    while (1) {
        printf("\n--- Dynamic Queue Menu ---\n");
        printf("1. Insert\n");
        printf("2. Display\n");
        printf("3. Exit\n");
        printf("Enter your choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                printf("Enter value to insert: ");
                scanf("%d", &value);
                insert(value);
                break;
            case 2:
                display();
                break;
            case 3:
                printf("Exiting program.\n");
                return 0;
            default:
                printf("Invalid choice! Please try again.\n");
        }
    }
}
```

---

**11. Write a program for linear search.**

```
#include <stdio.h>
int main() {
    int arr[100], n, i, key, found = 0;
    printf("Enter the number of elements: ");
    scanf("%d", &n);
    printf("Enter %d elements:\n", n);
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter the element to search: ");
    scanf("%d", &key);
    for (i = 0; i < n; i++) {
        if (arr[i] == key) {
            printf("Element %d found at position %d.\n", key, i + 1);
            found = 1;
            break;
        }
    }
    if (!found) {
        printf("Element %d not found in the array.\n", key);
    }
    return 0;
}
```

---

**12. Write a c program to implementation of dynamic stack using linked list with push() & display() function.**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node* next;
};
struct Node* top = NULL;
// Push function
void push(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = top;
    top = newNode;
    printf("%d pushed to stack\n", value);
}
// Display function
void display() {
    struct Node* temp = top;

    if (temp == NULL) {

        printf("Stack is empty\n");

        return;

    }

    printf("Stack elements: ");

    while (temp != NULL) {

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

        temp = temp->next;

    }

    printf("\n");

}

int main() {

    push(10);

    push(20);

    push(30);

    display();

    return 0;

}
```

---

**13. Write a c program Static implementation of queue check init() & isEmpty() function**

```
#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int front, rear;

void init() {

    front = -1;

    rear = -1;

}
```

```
// Check if queue is empty
int isEmpty() {
   return (front == -1 && rear == -1);
}
int main() {
   init();  // initialize queue
   if (isEmpty())
      printf("Queue is empty\n");
   else
      printf("Queue is not empty\n");
   return 0;
}
```

---

**14. Write C program to implement a Doubly linked list with create() and display() operation**

```
#include <stdio.h>
#include <stdlib.h>
struct Node {
   int data;
   struct Node* prev;
   struct Node* next;
};
struct Node* head = NULL;
// Create (insert at end)
void create(int value) {
   struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   newNode->prev = NULL;
   newNode->next = NULL;
   if (head == NULL) {
       head = newNode;
   } else {
       struct Node* temp = head;
       while (temp->next != NULL)
           temp = temp->next;
       temp->next = newNode;
       newNode->prev = temp;
   }
   printf("%d inserted\n", value);
}
// Display 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("\n");
}
int main() {
   create(10);
   create(20);
   create(30);
   display();    return 0;   }
```

**15. Write C program to Static implementation of queue to check isEmpty() & insert() function.**

```
#include <stdio.h>

#define SIZE 5

int queue[SIZE];

int front = -1, rear = -1;

// Check if queue is empty

int isEmpty() {

   return (front == -1 && rear == -1);

}

// Insert (enqueue)

void insert(int value) {

   if (rear == SIZE - 1) {

      printf("Queue is Full\n");

      return;

   }

   if (isEmpty()) {

      front = rear = 0;

   } else {

      rear++;

   }

   queue[rear] = value;

   printf("%d inserted\n", value);

}

int main() {

   if (isEmpty())

      printf("Queue is empty\n");

   insert(10);

   insert(20);
```

```
    insert(30);
    if (!isEmpty())
        printf("Queue is not empty now\n");
    return 0;
}
```

---

16. Write C program to Static implementation of stack with init() & insert() Function.

```
#include <stdio.h>
#define SIZE 5
int stack[SIZE];
int top;
// Initialize stack
void init() {
    top = -1;
}
// Insert (push)
void insert(int value) {
    if (top == SIZE - 1) {
        printf("Stack is Full\n");
        return;
    }
    stack[++top] = value;
    printf("%d inserted\n", value);
}
int main() {
    int choice, value;
    init();   // initialize stack
    while (1) {
        printf("\n--- STACK MENU ---\n");
        printf("1. Insert (Push)\n");
        printf("2. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        if (choice == 1) {
            printf("Enter value: ");
            scanf("%d", &value);
            insert(value);
        }
        else if (choice == 2) {
            break;
        }
        else {
            printf("Invalid choice\n");
        }
    }
    return 0;
}
```

---

**17. Write C program to Dynamic implementation of queue with delete() & display()** function.

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

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

```
struct Node *front = NULL, *rear = NULL;


// Predefined values
int values[] = {10, 20, 30, 40, 50};
int n = 5;


// Initialize queue with predefined values
void initQueue() {
    for (int i = 0; i < n; i++) {
        struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = values[i];
        newNode->next = NULL;


        if (rear == NULL) {
            front = rear = newNode;
        } else {
            rear->next = newNode;
            rear = newNode;
        }
    }
    printf("Queue initialized with predefined values.\n");
}


// Delete (dequeue)
void delete() {
    if (front == NULL) {
        printf("Queue is empty\n");
        return;
    }
    struct Node* temp = front;
    printf("%d deleted\n", temp->data);

    front = front->next;
    if (front == NULL)
        rear = NULL;

    free(temp);
}

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

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

int main() {
    initQueue();  // initialize queue with predefined values
    int choice;
    while (1) {
        printf("\n--- QUEUE MENU ---\n");
        printf("1. Delete\n");
        printf("2. Display\n");
        printf("3. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);

        switch (choice) {
            case 1: delete(); break;
            case 2: display(); break;
            case 3: return 0;
            default: printf("Invalid choice\n");
        }
    }
}
```