```
#include <stdio.h>
int main()
{
        int i, low, high, mid, n, key, array[100];
        printf("Enter number of elementsn");
        scanf("%d",&n);

        printf("Enter %d integersn", n);
        for(i = 0; i < n; i++)
        scanf("%d",&array[i]);

        printf("Enter value to findn");
        scanf("%d", &key);

        low = 0;
        high = n - 1;
        mid = (low+high)/2;
        while (low <= high)
         {
                if(array[mid] < key)
                {
                        low = mid + 1;
                }
                else if (array[mid] == key)
                {
                        printf("%d found at location %d.n", key, mid+1);
                        break;
            }
                else
                {
                        high = mid - 1;
                        mid = (low + high)/2;
                }
        }

        if(low > high)
        {
                printf("Not found! %d isn't present in the list.n", key);
        }

        return 0;
}
```

```c
// C program for implementation of Bubble sort
#include <stdio.h>

void swap(int *xp, int *yp)
{
    int temp = *xp;
    *xp = *yp;
    *yp = temp;
}

// A function to implement bubble sort
void bubbleSort(int arr[], int n)
{
   int i, j;
   for (i = 0; i < n-1; i++)

       // Last i elements are already in place
       for (j = 0; j < n-i-1; j++)
           if (arr[j] > arr[j+1])
              swap(&arr[j], &arr[j+1]);
}

/* Function to print an array */
void printArray(int arr[], int size)
{
    int i;
    for (i=0; i < size; i++)
        printf("%d ", arr[i]);
    printf("\n");
}

// Driver program to test above functions
int main()
{
    int arr[] = {64, 34, 25, 12, 22, 11, 90};
    int n = sizeof(arr)/sizeof(arr[0]);
    bubbleSort(arr, n);
    printf("Sorted array: \n");
    printArray(arr, n);
    return 0;
}
```

```
/*
 * C program to input N numbers and store them in an array.
 * Do a linear search for a given key and report success
 * or failure.
 */
#include <stdio.h>

void main()
{  int num;

    int i,  keynum, found = 0;

    printf("Enter the number of elements ");
    scanf("%d", &num);
    int array[num];
    printf("Enter the elements one by one \n");
    for (i = 0; i < num; i++)
    {
        scanf("%d", &array[i]);
    }

    printf("Enter the element to be searched ");
    scanf("%d", &keynum);
    /*  Linear search begins */
    for (i = 0; i < num ; i++)
    {
        if (keynum == array[i] )
        {
            found = 1;
            break;
        }
    }
    if (found == 1)
        printf("Element is present in the array at position %d",i+1);
    else
        printf("Element is not present in the array\n");
}
```

```
// Linked list operations in C

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

// Create a node
struct Node {
  int item;
  struct Node *next;
};

void insertAtBeginning(struct Node **ref, int data) {
  // Allocate memory to a node
  struct Node *new_node = (struct Node*)malloc(sizeof(struct Node));

  // insert the item
  new_node->item = data;
  new_node->next = (*ref);

  // Move head to new node
  (*ref) = new_node;
}

// Insert a node after a node
void insertAfter(struct Node* node, int data) {
  if (node == NULL) {
    printf("the given previous node cannot be NULL");
    return;
  }

  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  new_node->item = data;
  new_node->next = node->next;
  node->next = new_node;
}

void insertAtEnd(struct Node** ref, int data) {
  struct Node* new_node = (struct Node*)malloc(sizeof(struct Node));
  struct Node* last = *ref;

  new_node->item = data;
  new_node->next = NULL;

  if (*ref == NULL) {
    *ref = new_node;
    return;
  }

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

  last->next = new_node;
  return;
}

void deleteNode(struct Node** ref, int key) {
  struct Node *temp = *ref, *prev;

  if (temp != NULL && temp->item == key) {
```

```
    *ref = temp->next;
    free(temp);
    return;
  }
  // Find the key to be deleted
  while (temp != NULL && temp->item != key) {
    prev = temp;
    temp = temp->next;
  }

  // If the key is not present
  if (temp == NULL) return;

  // Remove the node
  prev->next = temp->next;

  free(temp);
}

// Print the linked list
void printList(struct Node* node) {
  while (node != NULL) {
    printf(" %d ", node->item);
    node = node->next;
  }
}

// Driver program
int main() {
  struct Node* head = NULL;

  insertAtEnd(&head, 1);
  insertAtBeginning(&head, 2);
  insertAtBeginning(&head, 3);
  insertAtEnd(&head, 4);
  insertAfter(head, 5);

  printf("Linked list: ");
  printList(head);

  printf("\nAfter deleting an element: ");
  deleteNode(&head, 3);
  printList(head);
}
```

```
// A complete working C program to demonstrate all insertion methods in Linked List
#include <stdio.h>
#include <stdlib.h>

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

/* Given a reference (pointer to pointer) to the head of a list and
an int, inserts a new node on the front of the list. */
void push(struct Node** head_ref, int new_data)
{
        /* 1. allocate node */
        struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

        /* 2. put in the data */
        new_node->data = new_data;

        /* 3. Make next of new node as head */
        new_node->next = (*head_ref);

        /* 4. move the head to point to the new node */
        (*head_ref) = new_node;
}

/* Given a node prev_node, insert a new node after the given
prev_node */
void insertAfter(struct Node* prev_node, int new_data)
{
        /*1. check if the given prev_node is NULL */
        if (prev_node == NULL)
        {
        printf("the given previous node cannot be NULL");
        return;
        }

        /* 2. allocate new node */
        struct Node* new_node =(struct Node*) malloc(sizeof(struct Node));

        /* 3. put in the data */
        new_node->data = new_data;

        /* 4. Make next of new node as next of prev_node */
        new_node->next = prev_node->next;

        /* 5. move the next of prev_node as new_node */
        prev_node->next = new_node;
}

/* Given a reference (pointer to pointer) to the head
of a list and an int, appends a new node at the end */
void append(struct Node** head_ref, int new_data)
{
        /* 1. allocate node */
        struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));

        struct Node *last = *head_ref; /* used in step 5*/
```

```
        /* 2. put in the data */
        new_node->data = new_data;

        /* 3. This new node is going to be the last node, so make next of
                it as NULL*/
        new_node->next = NULL;

        /* 4. If the Linked List is empty, then make the new node as head */
        if (*head_ref == NULL)
        {
        *head_ref = new_node;
        return;
        }

        /* 5. Else traverse till the last node */
        while (last->next != NULL)
                last = last->next;

        /* 6. Change the next of last node */
        last->next = new_node;
        return;
}

// This function prints contents of linked list starting from head
void printList(struct Node *node)
{
while (node != NULL)
{
        printf(" %d ", node->data);
        node = node->next;
}
}

int main()
{
/* Start with the empty list */
struct Node* head = NULL;

// Insert 6. So linked list becomes 6->NULL
append(&head, 6);

// Insert 7 at the beginning. So linked list becomes 7->6->NULL
push(&head, 7);

// Insert 1 at the beginning. So linked list becomes 1->7->6->NULL
push(&head, 1);

// Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
append(&head, 4);

// Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
insertAfter(head->next, 8);

printf("\n Created Linked list is: ");
printList(head);
return 0;
}
```

```
// A simple C program for traversal of a linked list
#include <stdio.h>
#include <stdlib.h>

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

// This function prints contents of linked list starting from
// the given node
void printList(struct Node* n)
{
        while (n != NULL) {
                printf(" %d ", n->data);
                n = n->next;
        }
}

int main()
{
        struct Node* head = NULL;
        struct Node* second = NULL;
        struct Node* third = NULL;

        // allocate 3 nodes in the heap
        head = (struct Node*)malloc(sizeof(struct Node));
        second = (struct Node*)malloc(sizeof(struct Node));
        third = (struct Node*)malloc(sizeof(struct Node));

        head->data = 1; // assign data in first node
        head->next = second; // Link first node with second

        second->data = 2; // assign data to second node
        second->next = third;

        third->data = 3; // assign data to third node
        third->next = NULL;

        printList(head);

        return 0;
}
```

```
/*
 * C Program to Implement a Queue using an Array
 */
#include <stdio.h>

#define Size 50

// Function Declaration

void insert();
void delete();
void display();

int queue_array[Size];
int rear = - 1;
int front = - 1;

main()
{
    int choice;
    while (1)
    {
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        } /* End of switch */
    } /* End of while */
} /* End of main() */

void insert()
{
    int item;
    if (rear == Size - 1)
    printf("Queue Overflow \n");
    else
    {
        if (front == - 1)
        /*If queue is initially empty */
        front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &item);
```

```
        rear = rear + 1;
        queue_array[rear] = item;
    }
} /* End of insert() */

void delete()
{
    if (front == - 1 || front > rear)
    {
        printf("Queue Underflow \n");
        return ;
    }
    else
    {
        printf("Element deleted from queue is : %d\n", queue_array[front]);
        front = front + 1;
    }
} /* End of delete() */

void display()
{
    int i;
    if (front == - 1)
        printf("Queue is empty \n");
    else
    {
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
} /* End of display() */
```

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
    int data;
    struct node *next;
};

struct node *front;
struct node *rear;

void insert();
void delete();
void display();

void main ()
{
    int choice;
    while(choice != 4)
    {
        printf("Chose one of the Options: \n");
        printf("\n1.insert an element\n2.Delete an element\n3.Display the
queue\n4.Exit\n");
        printf("\nEnter your choice ?");
        scanf("%d",& choice);

        switch(choice)
        {
            case 1:
            insert();
            break;
            case 2:
            delete();
            break;
            case 3:
            display();
            break;
            case 4:
            exit(0);
            break;
            default:
            printf("\nEnter valid choice??\n");
        }
    }
}

void insert()
{
    struct node *ptr;
    int item;

    ptr = (struct node *) malloc (sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW\n");
        return;
    }
    else
    {
```

```
        printf("\nEnter value?\n");
        scanf("%d",&item);
        ptr -> data = item;
        if(front == NULL)
        {
            front = ptr;
            rear = ptr;
            front -> next = NULL;
            rear -> next = NULL;
        }
        else
        {
            rear -> next = ptr;
            rear = ptr;
            rear->next = NULL;
        }
    }
}

void delete ()
{
    struct node *ptr;
    if(front == NULL)
    {
        printf("\nUNDERFLOW\n");
        return;
    }
    else
    {
        ptr = front;
        front = front -> next;
        free(ptr);
    }
}

void display()
{
    struct node *ptr;
    ptr = front;
    if(front == NULL)
    {
        printf("\nEmpty queue\n");
    }
    else
    {   printf("\nprinting values .....\n");
        while(ptr != NULL)
        {
            printf("\n%d\n",ptr -> data);
            ptr = ptr -> next;
        }
    }
}
```

```
// A simple C program to introduce
// a linked list
#include <stdio.h>
#include <stdlib.h>

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

// Program to create a simple linked
// list with 3 nodes
int main()
{
        struct Node* head = NULL;
        struct Node* second = NULL;
        struct Node* third = NULL;

        // allocate 3 nodes in the heap
        head = (struct Node*)malloc(sizeof(struct Node));
        second = (struct Node*)malloc(sizeof(struct Node));
        third = (struct Node*)malloc(sizeof(struct Node));

        head->data = 1; // assign data in first node
        head->next = second; // Link first node with

        second->data = 2;

        // Link second node with the third node
        second->next = third;

        +---+---+        +---+---+       +----+----+     */

        third->data = 3; // assign data to third node
        third->next = NULL;


        return 0;
}
```

```
#include <stdio.h>
int stack[100],i,j,choice=0,n,top=-1, size;
void push();
void pop();
void show();

void main ()
{

    printf("Enter the number of elements in the stack ");
    scanf("%d",&n);
    printf("*********Stack operations using array*********");

    while(choice != 4)
    {
        printf("Chose one from the below options...\n");
        printf("\n1.Push\n2.Pop\n3.Show\n4.Exit");
        printf("\n Enter your choice \n");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                show();
                break;
            }
            case 4:
            {
                printf("Exiting....");
                break;
            }
            default:
            {
                printf("Please Enter valid choice ");
            }
        };
    }
}

// Function Declaration's

void push ()
{
    int val;
    if (top == n )
    printf("\n Overflow");
    else
    {
```

```
        printf("Enter the value: ");
        scanf("%d",&val);
        top = top +1;
        stack[top] = val;
    }
}

void pop ()
{
    if(top == -1)
    printf("Underflow");
    else
    top = top -1;
}

void show()
{
    for (i=top;i>=0;i--)
    {
        printf("%d\n",stack[i]);
    }
    if(top == -1)
    {
        printf("Stack is empty \n");
    }
}
```