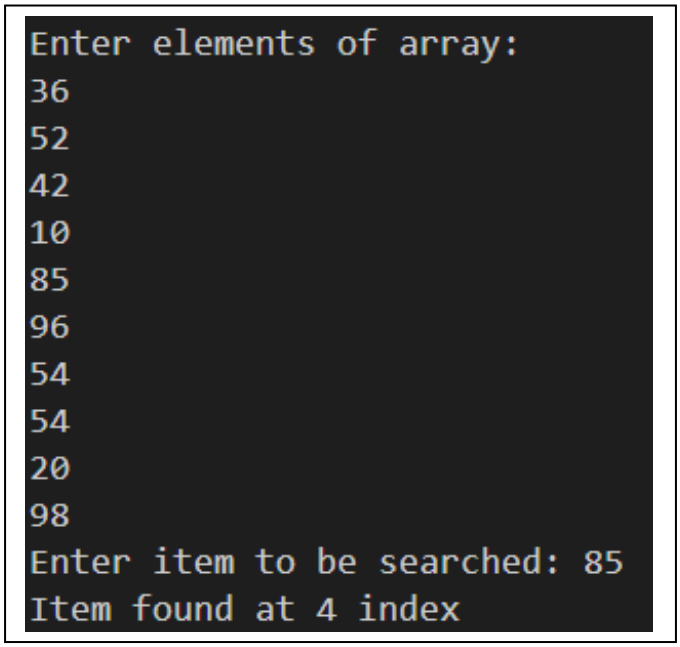
// WAP to perform linear search in an array

```
#include<stdio.h>
int main(){
    int arr[100];
    int item,i,loc=-1,n;
    printf("Enter number of elements you want to give in array: ");
    scanf("%d",&n);
    printf("Enter elements: \n");
    for(int i=0;i<n;i++){
        scanf("%d",&arr[i]); }
    printf("Enter item to be searched: ");
    scanf("%d",&item);
    for(i=0;i<n;i++){
        if(arr[i]==item){
            loc=i;  }     }
    if(loc==-1){
        printf("Item not found");
    }
    else{
        printf("Item found at %d index location", loc);
    }
}
```

```
Enter number of elements you want to give in array: 4
Enter elements:
12
58
32
56
Enter item to be searched: 32
Item found at 2 index location
```

// WAP to perform binary search in an array (Non – recursive)

```
#include <stdio.h>
void main() {
    int arr[10];
    int item, n, left = 0, right = 9, mid, loc = -1, i;
    printf("Enter elements of array: \n");
    for (i = 0; i < 10; i++) {
        scanf("%d", &arr[i]);
    }
    printf("Enter item to be searched: ");
    scanf("%d", &item);
    while (left < right) {
        mid = (int)(left + right) / 2;
        if (item == arr[mid]) {
            loc = mid;
            break; }
        else if (item > arr[mid]) {
            left = mid + 1; }
        else {
            right = mid - 1; } }
    if (loc == -1) {
        printf("Item not found");
    }
    else {
        printf("Item found at %d index", loc);
    }
}
```



// WAP to perform binary search in an array (recursively)

```
#include<stdio.h>
int binarySearch(int arr[], int l, int r, int item){
    if(l>r){
        return -1; }
    else{
        int mid=(int)(l+r)/2;
        if(arr[mid]==item){
            return mid; }
        else if(arr[mid]>item)
            return binarySearch(arr,l,mid-1,item);
        else
            return binarySearch(arr,mid+1,r,item);
    }
}
int main(){
    int item;
    int arr[]={2,3,4,10,40};
    printf("Enter item to be searched: ");
    scanf("%d",&item);
    int n=sizeof(arr)/sizeof(arr[0]);
    int result=binarySearch(arr,0,n-1,item);
    if(result==-1)
        printf("Item not found");
    else
        printf("Item found at %d index", result);
}
```

```
Enter item to be searched: 10
Item found at 3 index
```

// WAP to perform selection sort in an array

```
#include <stdio.h>
int main()
{
    int arr[100], n, i, j,temp;
    printf("Enter number of elements you want to give to array: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for (i = 0; i < n; i++){
        scanf("%d", &arr[i]); }

    for(i=0;i<n-1;i++){
        for(j=i+1;j<n;j++){
            if(arr[i]>arr[j]){
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
            }
        }
    }
    printf("\nSorted array is: ");
    for(i=0;i<n;i++){
        printf("%d, ",arr[i]);
    }
}
```

```
Enter number of elements you want to give to array: 3
Enter elements:
52
48
92
Sorted array is: 48, 52, 92,
```

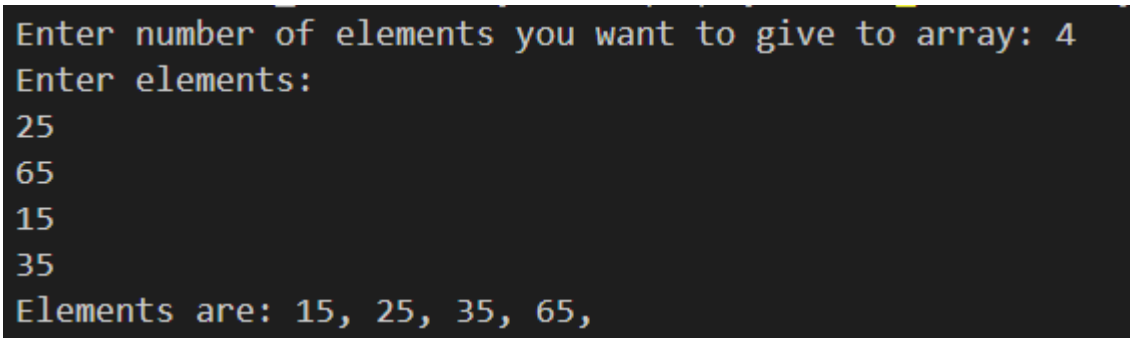
// WAP to perform bubble sort in an array

```
#include <stdio.h>
int main()
{
    int i, n, j, noswap, temp;
    int arr[100];
    printf("Enter no.of elements you want to give to array: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for (i = 0; i < n; i++) {
        scanf("%d", &arr[i]); }
    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 is: ");
    for (i = 0; i < n; i++) {
        printf("%d, ", arr[i]);
    }
}
```

```
Enter no.of elements you want to give to array: 3
Enter elements:
21
11
36
Sorted array is: 11, 21, 36,
```

// WAP to perform insertion sort in an array

```
#include <stdio.h>
int main()
{
    int arr[100], n, i, j, key;
    printf("Enter number of elements you want to give to array: ");
    scanf("%d", &n);
    printf("Enter elements: \n");
    for (i = 0; i < n; i++)    {
        scanf("%d", &arr[i]);
    }
    for (i = 1; i < n; i++) {
        key = arr[i];
        j=i-1;
        while (key < arr[j] && j >= 0) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
    printf("Elements are: ");
    for (i = 0; i < n; i++) {
        printf("%d, ", arr[i]);  }
}
```



// WAP to perform merge sort in an array

```
#include <stdio.h>
void merge(int arr[], int l, int mid, int r)
{
    int n1 = mid - l + 1;
    int n2 = r - mid;      // r-mid+1-1
    int a[n1];
    int b[n2];
    for (int i = 0; i < n1; i++)  {
        a[i] = arr[l + i];
    }
    for (int i = 0; i < n2; i++)  {
        b[i] = arr[mid + 1 + i];
    }
    int i = 0, j = 0;
    int k = l;
    while (i < n1 && j < n2)  {
        if (a[i] < b[j])  {
            arr[k] = a[i];
            k++;
            i++;
        }
        else  {
            arr[k] = b[j];
            k++;
            j++;
        }
    }
    while (i < n1) {
```

```
        arr[k] = a[i];
        k++;
        i++;
    }
    while (j < n2) {
        arr[k] = b[j];
        k++;
        j++;
    }
}
void mergeSort(int arr[], int l, int r) {
    if (l < r) {
        int mid = (l + r) / 2;
        mergeSort(arr, l, mid);
        mergeSort(arr, mid + 1, r);
        merge(arr, l, mid, r);
    }
}
int main()
{
    int arr[] = {50, 40, 30, 20, 10};
    mergeSort(arr, 0, 4);
    for (int i = 0; i < 5; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

```
10 20 30 40 50
```

// WAP to perform quick sort in an array

```
#include<stdio.h>
void swap(int arr[],int i,int j){
    int temp=arr[i];
    arr[i]=arr[j];
    arr[j]=temp; }
int partition(int arr[],int l,int r){
    int pivot=arr[r];
    int i=l-1;
    for(int j=l;j<r;j++){
        if(arr[j]<pivot){
            i++;
            swap(arr,i,j); } }
    swap(arr,i+1,r);
    return i+1;   }
void quickSort(int arr[],int l,int r){
    if(l<r){
        int pi=partition(arr,l,r);
        quickSort(arr,l,pi-1);
        quickSort(arr,pi+1,r);  }
}
int main() {
    int arr[5]={36,25,86,12,65};
    quickSort(arr,0,4);
    for(int i=0;i<5;i++){
        printf("%d ",arr[i]); }
    printf("\n");
    return 0;  }
```

```
12 25 36 65 86
```

// WAP to implement linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
struct node *head = NULL, *tail = NULL;
// struct node *head = NULL;
void create(int x)
{
    struct node *nn;
    nn = (struct node *)malloc(sizeof(struct node));
    nn->data = x;
    nn->next = NULL;
    // head = nn;
    head = tail = nn;
}
void insert_at_begin(int x){
    struct node *nn;
    nn = (struct node *)malloc(sizeof(struct node));
    nn->data = x;
    nn->next = NULL;
    if (head == NULL)   {
        create(x);
    }   else
    {
        nn->next = head;
        head = nn;
```

```
    }
}

void display(){
    struct node *temp = head;
    if (temp == NULL)  {
        printf("List is empty");
    }
    else  {
        while (temp != NULL)  {
            printf("%d->", temp->data);
            temp = temp->next;
        }
    }
}

void insert_at_end(int x){
    struct node *nn, *temp = head;
    nn = (struct node *)malloc(sizeof(struct node));
    nn->data = x;
    nn->next = NULL;
    if (head == NULL) {
        create(x);
    }
    else {
        // while(temp->next!=NULL){
        //     temp=temp->next;
        // }
        // temp->next=nn;
```

```
        tail->next = nn;
        tail = nn;
    }
}
void insert_position(int x, int p){
    int i;
    struct node *nn, *temp = head;
    nn = (struct node *)malloc(sizeof(struct node));
    nn->data = x;
    nn->next = NULL;
    if (head == NULL) {
        create(x);
    }
    else if (p == 1) {
        insert_at_begin(x);
    }
    else {
        for (i = 1; i < p - 1; i++)  {
            temp = temp->next;
        }
        nn->next = temp->next;
        temp->next = nn;
        if (nn->next == NULL)
        {
            tail = nn;
        }
    }
}
void delete_begin(){
```

```
    struct node *temp = head;
    if (head == NULL)
    {
        printf("List is empty");
    }
    else if (head->next == NULL)
    { // condition having one element in the list
        head = tail = NULL;
        printf("%d deleted", temp->data);
        free(temp); // free up memeory(space)
    }
    else  {
        head = head->next;
        printf("%d deleted", temp->data);
        free(temp);
    }
}
void delete_end(){
    struct node *temp = head;
    if (head == NULL)
    {
        printf("List is empty");
    }
    else if (head->next == NULL)
    { // condition having one element in the list
        head = tail = NULL;
        printf("%d deleted", temp->data);
        free(temp); // free up memeory(space)
    }
```

```
    else
    {
        while (temp->next != tail)
        {
            temp = temp->next;
        }
        temp->next = NULL; // temp is the current node
        printf("%d deleted", tail->data);
        free(tail);
        tail = temp;
    }
}
void delete_position(int p){
    struct node *temp = head, *temp1;
    if (head == NULL)
    {
        printf("List is empty");
    }
    else if (p == 1) {
        delete_begin();
    }
    else
    {
        for (int i = 1; i < p; i++)
        {
            temp1 = temp;
            temp = temp->next;
        }
        temp1->next = temp->next;
```

```
        printf("%d deleted", temp->data);
        if (temp->next == NULL)
        {
            tail = temp1;
        }
        free(temp);
    }
}

int count()
{
    int count = 0;
    struct node *temp = head;
    while (temp != NULL)
    {
        count++;
        temp = temp->next;
    }
    return count;
}

int search(int x)
{
    int p = 1;
    struct node *temp = head;
    while (temp != NULL)
    {
        if (temp->data == x)
        {
```

```
            return p;
        }
        p++;
        temp = temp->next;
    }
    return -1;
}

void main()
{
    int ch, val, pos;

    printf("\n 1. Create");
    printf("\n 2. Insert at beginning");
    printf("\n 3. Display");
    printf("\n 4. Exit");
    printf("\n 5. Insert at end");
    printf("\n 6. Insert at a position");
    printf("\n 7. Delete at beginning");
    printf("\n 8. Delete at end");
    printf("\n 9. Delete at a position");
    printf("\n 10. Count");
    printf("\n 11. Search");
    while (1)
    {
        printf("\nEnter choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
```

```
case 1:
    printf("Enter value for creating: ");
    scanf("%d", &val);
    create(val);
    break;
case 2:
    printf("Enter value for insertion: ");
    scanf("%d", &val);
    insert_at_begin(val);
    break;
case 3:
    display();
    break;
case 4:
    exit(0);
case 5:
    printf("Enter a value to insert at end: ");
    scanf("%d", &val);
    insert_at_end(val);
    break;
case 6:
    printf("Enter position in which you want to insert: ");
    scanf("%d", &pos);
    printf("Enter value to be inserted at %d position: ", pos);
    scanf("%d", &val);
    insert_position(val, pos);
    break;
case 7:
    delete_begin();
```

```
            break;
        case 8:
            delete_end();
            break;
        case 9:
            printf("Enter position to delete: ");
            scanf("%d", &pos);
            delete_position(pos);
            break;
        case 10:
            printf("%d", count());
            break;
        case 11:
            printf("Enter element to search: ");
            scanf("%d", &val);
            printf("%d", search(val));
            break;
        default:
            printf("Enter a valid input");
            break;
        }
    }
}
```

```
 1. Create
 2. Insert at beginning
 3. Display
 4. Exit
 5. Insert at end
 6. Insert at a position
 7. Delete at beginning
 8. Delete at end
 9. Delete at a position
 10. Count
 11. Search
Enter choice: 1
Enter value for creating: 50

Enter choice: 2
Enter value for insertion: 40

Enter choice: 5
Enter a value to insert at end: 60

Enter choice: 3
40->50->60->
Enter choice: 6
Enter position in which you want to insert: 3
Enter value to be inserted at 3 position: 55

Enter choice: 3
40->50->55->60->
Enter choice: 9
Enter position to delete: 2
50 deleted
Enter choice: 3
40->55->60->
Enter choice: 4
PS C:\Users\GAUTAM\Documents\A d drive content new\dsa c++>
```

// WAP to implement stack using array

```
#include <stdio.h>
#include <stdlib.h>
#define mxsize 50
int stack[mxsize];
int top;
void push(int item){
    if (top >= mxsize) {
        printf("Overflow");
    }
    else {
        top++;
        printf("Enter item to be inserted: ");
        scanf("%d", &item);
        stack[top] = item;
        printf("Item inserted\n");
    }
}
void pop(){
    if (top < 0) {
        printf("Underflow");
    }
    else {
        printf("Item deleted %d\n", stack[top]);
        top--;
    }
}
void display(){
    if (top >= 0){
```

```
        printf("Stack elements are: \n");
        for (int i = top; i > 0; i--) {
            printf("%d", stack[i]);
            printf("\n");
        }
    }
    else {
        printf("Stack is empty");
    }
}
void main()
{
    int ch;
    int item;
    printf("\n1. Insert data: ");
    printf("\n2. Delete data: ");
    printf("\n3. Display: ");
    printf("\n4. Exit: \n");
    while (1)  {
        printf("Enter choice: ");
        scanf("%d", &ch);

        switch (ch)
        {
        case 1:
            push(item);
            break;
        case 2:
            pop();
```

```
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
        default:
            printf("Please give a valid input");
            break;
        }
    }
}
```

```
1. Insert data:
2. Delete data:
3. Display:
4. Exit:
Enter choice: 1
Enter item to be inserted: 100
Item inserted
Enter choice: 1
Enter item to be inserted: 90
Item inserted
Enter choice: 1
Enter item to be inserted: 80
Item inserted
Enter choice: 3
Stack elements are:
80
90
100
Enter choice: 2
Item deleted 80
Enter choice: 3
Stack elements are:
90
100
Enter choice: 4
```

// WAP to implement stack using linked list

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int data;
    struct node *next;
};
struct node *head=NULL;
void push(int val){
    struct node *nn=(struct node *)malloc(sizeof(struct node));
    nn->data=val;
    nn->next=head;
    head=nn;
    printf("Item inserted\n");
}
void pop(){
    if(head==NULL){
        printf("Stack empty or underflow");
    }
    else{
        printf("Item deleted %d\n",head->data);
        head=head->next;  }
}
void display(){
    struct node *ptr;
    if(head == NULL){
        printf("Stack is empty");
    }
    else{
```

```
        ptr=head;
        printf("Stack elements are: \n");
        while(ptr!=NULL){
            printf("%d\n",ptr->data);
            ptr=ptr->next;  }  }
}
void main(){
    int ch;
    int item;
    printf("\n1. Insert");
    printf("\n2. Delete");
    printf("\n3. Display");
    printf("\n4. Exit\n");
    while(1){
        printf("Enter choice: ");
        scanf("%d",&ch);
        switch (ch) {
        case 1:
            printf("Enter item to be inserted: ");
            scanf("%d",&item);
            push(item);
            break;
        case 2:
            pop();  break;
        case 3:
            display();  break;
        case 4:
            exit(0);  break;
        default:     break;
```

```
    }
  }
}
```

```
1. Insert data:
2. Delete data:
3. Display:
4. Exit:
Enter choice: 1
Enter item to be inserted: 100
Item inserted
Enter choice: 1
Enter item to be inserted: 90
Item deleted 80
Enter choice: 3
Stack elements are:
90
3. Display
4. Exit
Enter choice: 1
Enter item to be inserted: 50
Item inserted
Enter choice: 1
Enter item to be inserted: 60
Item inserted
Enter choice: 1
Enter item to be inserted: 80
Item inserted
Enter choice: 3
Stack elements are:
80
60
50
Enter choice: 2
Item deleted 80
Enter choice: 3
Stack elements are:
60
50
Enter choice: 4
```

// WAP to implement queue using array

```
#include <stdio.h>
#include <stdlib.h>
#define mxsize 50
int queue[mxsize];
int front = -1;
int rear = -1;
void enqueue(int item){
    if (rear >= mxsize) {
        printf("Overflow");
    }
    else {
        if (front == -1) {
            front = 0;
        }
        rear++;
        printf("Enter item to be inserted: ");
        scanf("%d", &item);
        queue[rear] = item;
        printf("Item inserted\n");
    }
}
void dequeue(){
    if (front == -1 || front > rear){
        printf("Underflow");
    }
    else {
        printf("Item deleted %d\n", queue[front]);
        front++;
```

```
    }
}
void display(){
    if (front == -1){
        printf("\nQueue is empty");
    }
    else {
        printf("Queue elements are: \n");
        for (int i = front; i <= rear; i++)
        {
            printf("%d", queue[i]);
            printf("\n");
        }
    }
}
void main(){
    int ch;
    int item;
    printf("\n1. Insert data: ");
    printf("\n2. Delete data: ");
    printf("\n3. Display: ");
    printf("\n4. Exit: \n");
    while (1) {
        printf("Enter choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            enqueue(item);   break;
```

```
        case 2:
            dequeue();  break;
        case 3:
            display();  break;
        case 4:
            exit(0);  default:
            printf("Please give a valid input\n");
            break;
        }
    }
}
```

```
1. Insert data:
2. Delete data:
3. Display:
4. Exit:
Enter choice: 1
Enter item to be inserted: 10
Item inserted
Enter choice: 1
Enter item to be inserted: 11
Item inserted
Enter choice: 1
Enter item to be inserted: 12
Item inserted
Enter choice: 3
Queue elements are:
10
11
12
Enter choice: 2
Item deleted 10
Enter choice: 3
Queue elements are:
11
12
Enter choice: 4
PS C:\Users\GAUTAM\Documents\A
```

// WAP to implement queue using LinkedList

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node *next;
};
struct node *front;
struct node *rear;
void insert(int x){
    struct node *nn=(struct node *)malloc(sizeof(struct node));
    if (nn == NULL) {
        printf("\nOVERFLOW\n");
        return;
    }
    else  {
        nn->data = x;
        if (front == NULL)
        {
            front = nn;
            rear = nn;
            front->next = NULL;
            rear->next = NULL;
        }
        else {
            rear->next = nn;
            rear = nn;
            rear->next = NULL;
        }
```

```
    }
}
void delete(){
    struct node *nn;
    if (front == NULL){
        printf("\nUNDERFLOW\n");
        return;
    }
    else {
        nn = front;
        printf("%d deleted\n",front->data);
        front = front->next;
        free(nn);
    }
}
void display(){
    struct node *ptr;
    ptr = front;
    if (front == NULL){
        printf("\nQueue is empty\n");
    }
    else {
        printf("\nQueue elements are: \n");
        while (ptr != NULL)
        {
            printf("%d\n", ptr->data);
            ptr = ptr->next;
        }
    }
```

```
}
void main(){
    int ch,item;
    printf("1.insert\n");
    printf("2.Delete\n");
    printf("3.Display");
    printf("\n4.Exit\n");
    while (1) {
        printf("Enter your choice: ");
        scanf("%d", &ch);
        switch (ch)
        {
        case 1:
            printf("Enter item to be inserted: ");
            scanf("%d",&item);
            insert(item);
            break;
        case 2:
            delete ();
            break;
        case 3:
            display();
            break;
        case 4:
            exit(0);
            break;
        default:
            printf("\nPlease enter a valid input\n");
            break;
```

```
    }
  }
}
```

```
1.insert
2.Delete
3.Display
4.Exit
Enter your choice: 1
Enter item to be inserted: 10
Enter your choice: 1
Enter item to be inserted: 20
Enter your choice: 1
Enter item to be inserted: 30
Enter your choice: 3

Queue elements are:
10
20
30
Enter your choice: 2
10 deleted
Enter your choice: 3

Queue elements are:
20
30
Enter your choice: 4
```