Inserting and Appending in a Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};

    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
    void Append(struct Array *arr,int x)
    {
        if(arr->length<arr->size)
            arr->A[arr->length++]=x;
    }
    void Insert(struct Array *arr,int index,int x)
   {
    int i;

    if(index>=0 && index <=arr->length)
    {
        for(i=arr->length;i>index;i--)
            arr->A[i]=arr->A[i-1];
        arr->A[index]=x;
        arr->length++;
    }
   }

int main()
{

struct Array arr1={{2,3,4,5,6},10,5};
Append(&arr1,10);
Insert(&arr1,0,12);
Display(arr1);
    return 0;
}
```

Deleting from Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }

int Delete(struct Array *arr,int index)
{
    int x=0;
    int i;
    if(index>=0 && index<arr->length)
    {
        x=arr->A[index];
        for(i=index;i<arr->length-1;i++)
            arr->A[i]=arr->A[i+1];
        arr->length--;
        return x;
    }
    return 0;
}

int main()
{

    struct Array arr1={{2,3,4,5,6},10,5};
    printf("%d",Delete(&arr1,0));
    Display(arr1);
    return 0;
}
```

Searching in a Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }

int LinearSearch(struct Array *arr,int key)
{
    int i;
    for(i=0;i<arr->length;i++)
    {
        if(key==arr->A[i])
        {
            swap(&arr->A[i],&arr->A[0]);
            return i;
        }
    }
    return -1;
}


int main()
{

    struct Array arr1={{2,23,14,5,6,9,8,12},10,8};
    printf("%d",LinearSearch(&arr1,14));
    Display(arr1);
    return 0;
}
```

Binary Search in Array

```
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }

int BinarySearch(struct Array arr,int key)
{
    int l,mid,h;
    l=0;
    h=arr.length-1;
    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==arr.A[mid])
            return mid;
        else if(key<arr.A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
return -1;
}

int RBinSearch(int a[],int l,int h,int key)
{

    int mid=0;
    if(l<=h)
    {
```

```
        mid=(l+h)/2;
        if(key==a[mid])
            return mid;
        else if(key<a[mid])
            return RBinSearch(a,l,mid-1,key);
    }
    else
        return RBinSearch(a,mid+1,h,key);
return -1;
}

int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",BinarySearch(arr1,16));
    Display(arr1);
    return 0;
}
```

Get Set Max Min on Array

```c
#include<stdio.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }
int Get(struct Array arr,int index)
{
    if(index>=0 && index<arr.length)
        return arr.A[index];
    return -1;
}

void Set(struct Array *arr,int index,int x)
{
    if(index>=0 && index<arr->length)
    arr->A[index]=x;
}

int Max(struct Array arr)
{
    int max=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]>max)
            max=arr.A[i];
    }
    return max;
}
```

```c
int Min(struct Array arr)
{
    int min=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]<min)
            min=arr.A[i];
    }
    return min;
}

int Sum(struct Array arr)
{
    int s=0;
    int i;
    for(i=0;i<arr.length;i++)
        s+=arr.A[i];
    return s;
}
float Avg(struct Array arr)
{
    return (float)Sum(arr)/arr.length;
}
int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",Sum(arr1));
    Display(arr1);
    return 0;
}
```

Reversing an Array

```c
#include<stdio.h>
#include<stdlib.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }
 void swap(int *x,int *y)
 {
     int temp=*x;
     *x=*y;
     *y=temp;
 }
void Reverse(struct Array *arr)
{
    int *B;
    int i,j;

    B=(int *)malloc(arr->length*sizeof(int));
    for(i=arr->length-1,j=0;i>=0;i--,j++)
        B[j]=arr->A[i];
    for(i=0;i<arr->length;i++)
        arr->A[i]=B[i];

}

void Reverse2(struct Array *arr)
{
    int i,j;
    for(i=0,j=arr->length-1;i<j;i++,j--)
    {
        swap(&arr->A[i],&arr->A[j]);
    }
}
int main()
{
```

```
    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    Reverse(&arr1);
    Display(arr1);
    return 0;
}
```

Checking if Array is Sorted

```c
#include<stdio.h>
#include<stdlib.h>
struct Array
{
    int A[10];
    int size;
    int length;
};
    void Display(struct Array arr)
    {
        int i;
        printf("\nElements are\n");
        for(i=0;i<arr.length;i++)
            printf("%d ",arr.A[i]);
    }

int isSorted(struct Array arr)
{
    int i;
    for(i=0;i<arr.length-1;i++)
    {
        if(arr.A[i]>arr.A[i+1])
            return 0;
    }
    return 1;
}

int main()
{

    struct Array arr1={{2,3,9,16,18,21,28,32,35},10,9};
    printf("%d",isSorted(arr1));
    Display(arr1);
    return 0;
}
```

Merging 2 Arrays

```
struct Array
{
    int A[10];
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

struct Array* Merge(struct Array *arr1,struct Array *arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
            if(arr1->A[i]<arr2->A[j])
                arr3->A[k++]=arr1->A[i++];
            else
                arr3->A[k++]=arr2->A[j++];
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];
    arr3->length=arr1->length+arr2->length;
    arr3->size=10;

    return arr3;
}

int main()
{
    struct Array arr1={{2,9,21,28,35},10,5};
    struct Array arr1={{2,3,16,18,28},10,5};
    struct Array *arr3;
```

```
arr3=Merge(&arr1,&arr2);
Display(*arr3);

return 0;
}
```

Set Operations on Arrays

```
struct Array
{
    int A[10];
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

struct Array* Union(struct Array *arr1,struct Array *arr2)
{
    int i,j,k;
    i=j=k=0;


struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            arr3->A[k++]=arr2->A[j++];
        else
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];

    arr3->length=k;
    arr3->size=10;

    return arr3;
```

```
}

struct Array* Intersection(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            i++;
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else if(arr1->A[i]==arr2->A[j])
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Difference(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array *)malloc(sizeof(struct
Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else
        {
            i++;
```

```
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];


    arr3->length=k;
    arr3->size=10;

    return arr3;
}

int main()
{
    struct Array arr1={{2,9,21,28,35},10,5};
    struct Array arr1={{2,3,9,18,28},10,5};
    struct Array *arr3;

arr3=Union(&arr1,&arr2);
Display(*arr3);

return 0;
}
```

Array Menu using C

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

struct Array
{
    int *A;
    int size;
    int length;
};

void Display(struct Array arr)
{
    int i;
    printf("\nElements are\n");
    for(i=0;i<arr.length;i++)
        printf("%d ",arr.A[i]);
}

void Append(struct Array *arr,int x)
{
    if(arr->length<arr->size)
        arr->A[arr->length++]=x;

}

void Insert(struct Array *arr,int index,int x)
{
    int i;
    if(index>=0 && index <=arr->length)
    {
        for(i=arr->length;i>index;i--)
            arr->A[i]=arr->A[i-1];
        arr->A[index]=x;
        arr->length++;

    }
}
```

```
int Delete(struct Array *arr,int index)
{
    int x=0;
    int i;

    if(index>=0 && index<arr->length)
    {
        x=arr->A[index];
        for(i=index;i<arr->length-1;i++)
            arr->A[i]=arr->A[i+1];
        arr->length--;
        return x;
    }

    return 0;
}

void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int LinearSearch(struct Array *arr,int key)
{
    int i;
    for(i=0;i<arr->length;i++)
    {
        if(key==arr->A[i])
        {
            swap(&arr->A[i],&arr->A[0]);
            return i;
        }
    }
    return -1;
}
```

```
int BinarySearch(struct Array arr,int key)
{
    int l,mid,h;
    l=0;
    h=arr.length-1;

    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==arr.A[mid])
            return mid;
        else if(key<arr.A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
    return -1;
}

int RBinSearch(int a[],int l,int h,int key)
{
    int mid;

    if(l<=h)
    {
        mid=(l+h)/2;
        if(key==a[mid])
            return mid;
        else if(key<a[mid])
            return RBinSearch(a,l,mid-1,key);
        else
            return RBinSearch(a,mid+1,h,key);
    }
    return -1;
}

int Get(struct Array arr,int index)
{
        if(index>=0 && index<arr.length)
            return arr.A[index];
```

```
    return -1;
}

void Set(struct Array *arr,int index,int x)
{
    if(index>=0 && index<arr->length)
        arr->A[index]=x;
}

int Max(struct Array arr)
{
    int max=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]>max)
            max=arr.A[i];
    }
    return max;
}
int Min(struct Array arr)
{
    int min=arr.A[0];
    int i;
    for(i=1;i<arr.length;i++)
    {
        if(arr.A[i]<min)
            min=arr.A[i];
    }
    return min;
}

int Sum(struct Array arr)
{
    int s=0;
    int i;
    for(i=0;i<arr.length;i++)
        s+=arr.A[i];

    return s;
```

```
}

float Avg(struct Array arr)
{
    return (float)Sum(arr)/arr.length;
}

void Reverse(struct Array *arr)
{
    int *B;
    int i,j;

    B=(int *)malloc(arr->length*sizeof(int));
    for(i=arr->length-1,j=0;i>=0;i--,j++)
        B[j]=arr->A[i];
    for(i=0;i<arr->length;i++)
        arr->A[i]=B[i];

}

void Reverse2(struct Array *arr)
{
    int i,j;
    for(i=0,j=arr->length-1;i<j;i++,j--)
    {
        swap(&arr->A[i],&arr->A[j]);
    }
}

void InsertSort(struct Array *arr,int x)
{
    int i=arr->length-1;
    if(arr->length==arr->size)
        return;
    while(i>=0 && arr->A[i]>x)
    {
        arr->A[i+1]=arr->A[i];
        i--;
    }
    arr->A[i+1]=x;
```

```
    arr->length++;

}

int isSorted(struct Array arr)
{
    int i;
    for(i=0;i<arr.length-1;i++)
    {
        if(arr.A[i]>arr.A[i+1])
            return 0;
    }
    return 1;
}

void Rearrange(struct Array *arr)
{
    int i,j;
    i=0;
    j=arr->length-1;

    while(i<j)
    {
        while(arr->A[i]<0)i++;
        while(arr->A[j]>=0)j--;
        if(i<j)swap(&arr->A[i],&arr->A[j]);
    }

}

struct Array* Merge(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
```

```
    {
            if(arr1->A[i]<arr2->A[j])
                arr3->A[k++]=arr1->A[i++];
            else
                arr3->A[k++]=arr2->A[j++];
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];
    arr3->length=arr1->length+arr2->length;
    arr3->size=10;

    return arr3;
}


struct Array* Union(struct Array *arr1,struct Array
*arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            arr3->A[k++]=arr2->A[j++];
        else
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];
```

```
    for(;j<arr2->length;j++)
        arr3->A[k++]=arr2->A[j];

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Intersection(struct Array *arr1,struct
Array *arr2)
{
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            i++;
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else if(arr1->A[i]==arr2->A[j])
        {
            arr3->A[k++]=arr1->A[i++];
            j++;
        }
    }

    arr3->length=k;
    arr3->size=10;

    return arr3;
}

struct Array* Difference(struct Array *arr1,struct
Array *arr2)
{
```

```
    int i,j,k;
    i=j=k=0;

    struct Array *arr3=(struct Array
*)malloc(sizeof(struct Array));

    while(i<arr1->length && j<arr2->length)
    {
        if(arr1->A[i]<arr2->A[j])
            arr3->A[k++]=arr1->A[i++];
        else if(arr2->A[j]<arr1->A[i])
            j++;
        else
        {
            i++;
            j++;
        }
    }
    for(;i<arr1->length;i++)
        arr3->A[k++]=arr1->A[i];


    arr3->length=k;
    arr3->size=10;

    return arr3;
}



int main()
{
    struct Array arr1;
    int ch;
    int x,index;

    printf("Enter Size of Array");
    scanf("%d",&arr1.size);
    arr1.A=(int *)malloc(arr1.size*sizeof(int));
    arr1.length=0;
```

```c
    do
    {
    printf("\n\nMenu\n");
    printf("1. Insert\n");
    printf("2. Delete\n");
    printf("3. Search\n");
    printf("4. Sum\n");
    printf("5. Display\n");
    printf("6.Exit\n");

    printf("enter you choice ");
    scanf("%d",&ch);

    switch(ch)
    {
        case 1: printf("Enter an element and index
");
            scanf("%d%d",&x,&index);
            Insert(&arr1,index,x);
            break;
        case 2: printf("Enter index ");
            scanf("%d",&index);
            x=Delete(&arr1,index);
            printf("Deleted Element is %d\n",x);
            break;
        case 3:printf("Enter element to search ");
            scanf("%d",&x);
            index=LinearSearch(&arr1,x);
            printf("Element index %d",index);
            break;
        case 4:printf("Sum is %d\n",Sum(arr1));
            break;
        case 5:Display(arr1);

    }
    }while(ch<6);
    return 0;
}
```

Array C++ class

```
#include <iostream>
using namespace std;
template<class T>
class Array
{
private:
    T *A;
    int size;
    int length;
public:
    Array()
    {
        size=10;
        A=new T[10];
        length=0;
    }
    Array(int sz)
    {
        size=sz;
        length=0;
        A=new T[size];
    }
    ~Array()
    {
        delete []A;
    }
    void Display();
    void Insert(int index,T x);
    T Delete(int index);
};
template<class T>
void Array<T>::Display()
{
    for(int i=0;i<length;i++)
        cout<<A[i]<<" ";
    cout<<endl;
}
template<class T>
void Array<T>::Insert(int index,T x)
{
    if(index>=0 && index<=length)
    {
        for(int i=length-1;i>=index;i--)
```

```
            A[i+1]=A[i];
        A[index]=x;
        length++;
    }
}
template<class T>
T Array<T>::Delete(int index)
{
    T x=0;
    if(index>=0 && index<length)
    {
        x=A[index];
        for(int i=index;i<length-1;i++)
            A[i]=A[i+1];
        length--;
    }
    return x;
}
int main()
{
    Array<char> arr(10);

    arr.Insert(0,'a');
    arr.Insert(1,'c');
    arr.Insert(2,'d');
    arr.Display();
    cout<<arr.Delete(0)<<endl;
    arr.Display();
    return 0;
}
```

Array using C++ modified

```
#include <iostream>

using namespace std;
class Array
{
private:
    int *A;
    int size;
    int length;
    void swap(int *x,int *y);

public:
    Array()
    {
        size=10;
        length=0;
        A=new int[size];
    }
    Array(int sz)
    {
        size=sz;
        length=0;
        A=new int[size];
    }
    ~Array()
    {
        delete []A;
    }
    void Display();
    void Append(int x);
    void Insert(int index,int x);
    int Delete(int index);
    int LinearSearch(int key);
    int BinarySearch(int key);
   int Get(int index);
    void Set(int index,int x);
```

```
    int Max();
    int Min();
    int Sum();
    float Avg();
    void Reverse();
    void Reverse2();
    void InsertSort(int x);
    int isSorted();
    void Rearrange();
    Array* Merge(Array arr2);
    Array* Union(Array arr2);
    Array* Diff(Array arr2);
    Array* Inter(Array arr2);
};

void Array::Display()
{
    int i;
    cout<<"\nElements are\n";
    for(i=0;i<length;i++)
        cout<<A[i]<<" ";
}

void Array::Append(int x)
{
    if(length<size)
        A[length++]=x;

}

void Array::Insert(int index,int x)
{
    int i;
    if(index>=0 && index <=length)
    {
        for(i=length;i>index;i--)
            A[i]=A[i-1];
        A[index]=x;
        length++;
```

```
    }
}

int Array::Delete(int index)
{
    int x=0;
    int i;

    if(index>=0 && index<length)
    {
        x=A[index];
        for(i=index;i<length-1;i++)
            A[i]=A[i+1];
        length--;
        return x;
    }

    return 0;
}

void Array::swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}

int Array::LinearSearch(int key)
{
    int i;
    for(i=0;i<length;i++)
    {
        if(key==A[i])
        {
            swap(&A[i],&A[0]);
            return i;
        }
    }
    return -1;
```

```
}
int Array::BinarySearch(int key)
{
    int l,mid,h;
    l=0;
    h=length-1;

    while(l<=h)
    {
        mid=(l+h)/2;
        if(key==A[mid])
            return mid;
        else if(key<A[mid])
            h=mid-1;
        else
            l=mid+1;
    }
    return -1;
}

int Array::Get(int index)
{
    if(index>=0 && index<length)
        return A[index];
    return -1;
}
void Array::Set(int index,int x)
{
    if(index>=0 && index< length)
        A[index]=x;
}
int Array::Max()
{
    int max=A[0];
    int i;
    for(i=1;i<length;i++)
    {
```

```
        if(A[i]>max)
            max=A[i];
    }
    return max;
}
int Array::Min()
{
    int min=A[0];
    int i;
    for(i=1;i<length;i++)
    {
        if(A[i]<min)
            min=A[i];
    }
    return min;
}

int Array::Sum()
{
    int s=0;
    int i;
    for(i=0;i<length;i++)
        s+=A[i];

    return s;
}

float Array::Avg()
{
    return (float)Sum()/length;
}

void Array::Reverse()
{
    int *B;
    int i,j;

    B=(int *)malloc(length*sizeof(int));
    for(i=length-1,j=0;i>=0;i--,j++)
        B[j]=A[i];
```

```
    for(i=0;i<length;i++)
        A[i]=B[i];

}
void Array::Reverse2()
{
    int i,j;
    for(i=0,j= length-1;i<j;i++,j--)
    {
        swap(& A[i],& A[j]);
    }
}
void Array::InsertSort(int x)
{
    int i= length-1;
    if( length== size)
        return;
    while(i>=0 &&  A[i]>x)
    {
        A[i+1]= A[i];
        i--;
    }
    A[i+1]=x;
    length++;

}
int Array::isSorted()
{
    int i;
    for(i=0;i<length-1;i++)
    {
        if(A[i]>A[i+1])
            return 0;
    }
    return 1;
}
```

```
void Array::Rearrange()
{
    int i,j;
    i=0;
    j= length-1;

    while(i<j)
    {
        while( A[i]<0)i++;
        while( A[j]>=0)j--;
        if(i<j)swap(& A[i],& A[j]);
    }

}

Array* Array::Merge(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else
            arr3->A[k++]=arr2.A[j++];
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
    for(;j<arr2.length;j++)
        arr3->A[k++]=arr2.A[j];
    arr3->length=length+arr2.length;

    return arr3;
}

Array* Array::Union(Array arr2)
```

```
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else if(arr2.A[j]<A[i])
            arr3->A[k++]=arr2.A[j++];
        else
        {
            arr3->A[k++]=A[i++];
            j++;
        }
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
    for(;j<arr2.length;j++)
        arr3->A[k++]=arr2.A[j];

    arr3->length=k;

    return arr3;
}
Array* Array::Inter(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
```

```
            i++;
        else if(arr2.A[j]<A[i])
            j++;
        else if(A[i]==arr2.A[j])
        {
            arr3->A[k++]=A[i++];
            j++;
        }
    }

    arr3->length=k;

    return arr3;
}
Array* Array::Diff(Array arr2)
{
    int i,j,k;
    i=j=k=0;

    Array *arr3=new Array(length+arr2.length);

    while(i<length && j<arr2.length)
    {
        if(A[i]<arr2.A[j])
            arr3->A[k++]=A[i++];
        else if(arr2.A[j]<A[i])
            j++;
        else
        {
            i++;
            j++;
        }
    }
    for(;i<length;i++)
        arr3->A[k++]=A[i];
```

```
    arr3->length=k;

    return arr3;
}

int main()
{
    Array *arr1;
    int ch,sz;
    int x,index;

    cout<<"Enter Size of Array";
    scanf("%d",&sz);
    arr1=new Array(sz);

    do
    {
        cout<<"\n\nMenu\n";
        cout<<"1. Insert\n";
        cout<<"2. Delete\n";
        cout<<"3. Search\n";
        cout<<"4. Sum\n";
        cout<<"5. Display\n";
        cout<<"6.Exit\n";

        cout<<"enter you choice ";
        cin>>ch;

        switch(ch)
        {
            case 1: cout<<"Enter an element and
index ";
                cin>>x>>index;
                arr1->Insert(index,x);
                break;
            case 2: cout<<"Enter index ";
                cin>>index;
                x=arr1->Delete(index);
```

```
                cout<<"Deleted Element is"<<x;
                break;
            case 3:cout<<"Enter element to search
";
                cin>>x;
                index=arr1->LinearSearch(x);
                cout<<"Element index "<<index;
                break;
            case 4:cout<<"Sum is "<<arr1->Sum();
                break;
            case 5:arr1->Display();

        }
    }while(ch<6);
    return 0;
}
```

Diagonal Matrix C

```c
#include <stdio.h>

struct Matrix
{
    int A[10];
    int n;
};

void Set(struct Matrix *m,int i,int j,int x)
{
    if(i==j)
        m->A[i-1]=x;

}
int Get(struct Matrix m,int i,int j)
{
    if(i==j)
        return m.A[i-1];
    else
        return 0;
}
void Display(struct Matrix m)
{
    int i,j;
    for(i=0;i<m.n;i++)
    {
        for(j=0;j<m.n;j++)
        {
            if(i==j)
                printf("%d ",m.A[i]);
            else
                printf("0 ");
        }
        printf("\n");
    }
}
```

```c
int main()
{
    struct Matrix m;
    m.n=4;

    Set(&m,1,1,5);Set(&m,2,2,8);Set(&m,3,3,9);Set(&m,
4,4,12);
    printf("%d \n",Get(m,2,2));
    Display(m);

    return 0;
}
```

Diagonal Matrix CPP

```cpp
#include <iostream>

using namespace std;

class Diagonal
{
private:
    int *A;
    int n;
public:
    Diagonal()
    {
        n=2;
        A=new int[2];
    }
    Diagonal(int n)
    {
        this->n=n;
        A=new int[n];
    }
    ~Diagonal()
    {
        delete []A;
    }
    void Set(int i,int j,int x);
    int Get(int i,int j);
    void Display();
    int GetDimension(){return n;}
};

void Diagonal::Set(int i,int j,int x)
{
    if(i==j)
        A[i-1]=x;
}

int Diagonal::Get(int i,int j)
{
    if(i==j)
        return A[i-1];
    return 0;
}
void Diagonal::Display()
{
```

```
    for(int i=1;i<=n;i++)
    {
        for(int j=1;j<=n;j++)
        {
            if(i==j)
                cout<<A[i-1]<<" ";
            else
                cout<<"0 ";
        }
        cout<<endl;
    }
}

int main()
{
    int d;
    cout<<"Enter Dimensions";
    cin>>d;

    Diagonal dm(d);

    int x;
    cout<<"Enter All Elements";
    for(int i=1;i<=d;i++)
    {
        for(int j=1;j<=d;j++)
        {
            cin>>x;
            dm.Set(i,j,x);
        }
    }

    dm.Display();

    return 0;
}
```

Lower Triangular C

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

struct Matrix
{
    int *A;
    int n;
};


void Set(struct Matrix *m,int i,int j,int x)
{
    if(i>=j)
        m->A[m->n*(j-1)+(j-2)*(j-1)/2+i-j]=x;
}

int Get(struct Matrix m,int i,int j)
{
    if(i>=j)
        return m.A[m.n*(j-1)+(j-2)*(j-1)/2+i-j];
    else
        return 0;
}


void Display(struct Matrix m)
{
    int i,j;
    for(i=1;i<=m.n;i++)
    {
        for(j=1;j<=m.n;j++)
        {
            if(i>=j)
                printf("%d ",m.A[m.n*(j-1)+
(j-2)*(j-1)/2+i-j]);
            else
                printf("0 ");
```

```
        }
        printf("\n");
    }
}

int main()
{
    struct Matrix m;
    int i,j,x;

    printf("Enter Dimension");
    scanf("%d",&m.n);
    m.A=(int *)malloc(m.n*(m.n+1)/2*sizeof(int));
    printf("enter all elements");
    for(i=1;i<=m.n;i++)
    {
        for(j=1;j<=m.n;j++)
        {
            scanf("%d",&x);
            Set(&m,i,j,x);
        }
    }
    printf("\n\n");
    Display(m);


    return 0;
}
```

Lower Triangular CPP

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

struct Matrix
{
    int *A;
    int n;
};


void Set(struct Matrix *m,int i,int j,int x)
{
    if(i>=j)
        m->A[m->n*(j-1)+(j-2)*(j-1)/2+i-j]=x;
}

int Get(struct Matrix m,int i,int j)
{
    if(i>=j)
        return m.A[m.n*(j-1)+(j-2)*(j-1)/2+i-j];
    else
        return 0;
}


void Display(struct Matrix m)
{
    int i,j;
    for(i=1;i<=m.n;i++)
    {
        for(j=1;j<=m.n;j++)
        {
            if(i>=j)
                printf("%d ",m.A[m.n*(j-1)+
(j-2)*(j-1)/2+i-j]);
            else
                printf("0 ");
```

```
        }
        printf("\n");
    }
}

int main()
{
    struct Matrix m;
    int i,j,x;

    printf("Enter Dimension");
    scanf("%d",&m.n);
    m.A=(int *)malloc(m.n*(m.n+1)/2*sizeof(int));
    printf("enter all elements");
    for(i=1;i<=m.n;i++)
    {
        for(j=1;j<=m.n;j++)
        {
            scanf("%d",&x);
            Set(&m,i,j,x);
        }
    }
    printf("\n\n");
    Display(m);


    return 0;
}
```

Sparse Matrix using C

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

struct Element
{
    int i;
    int j;
    int x;
};

struct Sparse
{
    int m;
    int n;
    int num;
    struct Element *ele;
};

void create(struct Sparse *s)
{
    int i;

    printf("Eneter Dimensions");
    scanf("%d%d",&s->m,&s->n);
    printf("Number of non-zero");
    scanf("%d",&s->num);

    s->ele=(struct Element *)malloc(s->num*sizeof(struct
Element));
    printf("Eneter non-zero Elements");
    for(i=0;i<s->num;i++)
        scanf("%d%d%d",&s->ele[i].i,&s->ele[i].j,&s-
>ele[i].x);

}

void display(struct Sparse s)
{
    int i,j,k=0;

    for(i=0;i<s.m;i++)
    {
        for(j=0;j<s.n;j++)
```

```
        {
            if(i==s.ele[k].i && j==s.ele[k].j)
                printf("%d ",s.ele[k++].x);
            else
                printf("0 ");
        }
        printf("\n");
    }
}

struct Sparse * add(struct Sparse *s1,struct Sparse *s2)
{
    struct Sparse *sum;
    int i,j,k;
    i=j=k=0;

    if(s1->n != s2->n && s1->m != s2->m)
        return NULL;
    sum=(struct Sparse *)malloc(sizeof(struct Sparse));
sum->ele=(struct Element *)malloc((s1->num+s2-
>num)*sizeof(struct Element));

    while(i<s1->num && j<s2->num)
    {
        if(s1->ele[i].i<s2->ele[j].i)
            sum->ele[k++]=s1->ele[i++];
        else if(s1->ele[i].i>s2->ele[j].i)
            sum->ele[k++]=s2->ele[j++];
        else
        {
            if(s1->ele[i].j<s2->ele[j].j)
                sum->ele[k++]=s1->ele[i++];
            else if(s1->ele[i].j>s2->ele[j].j)
                sum->ele[k++]=s2->ele[j++];
            else
            {
                sum->ele[k]=s1->ele[i];
                sum->ele[k++].x=s1->ele[i++].x+s2->ele[j+
+].x;
            }

        }
    }
    for(;i<s1->num;i++)sum->ele[k++]=s1->ele[i];
    for(;j<s2->num;j++)sum->ele[k++]=s2->ele[j];
    sum->m=s1->m;
    sum->n=s1->n;
    sum->num=k;
```

```
    return sum;

}


int main()
{
    struct Sparse s1,s2,*s3;

    create(&s1);
    create(&s2);

    s3=add(&s1,&s2);

    printf("First Matrix\n");
    display(s1);
    printf("Second Matrix\n");
    display(s2);
    printf("Sum Matrix\n");
    display(*s3);


    return 0;
}
```

Sparse Matrix using C

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

struct Element
{
    int i;
    int j;
    int x;
};

struct Sparse
{
    int m;
    int n;
    int num;
    struct Element *ele;
};

void create(struct Sparse *s)
{
    int i;

    printf("Eneter Dimensions");
    scanf("%d%d",&s->m,&s->n);
    printf("Number of non-zero");
    scanf("%d",&s->num);

    s->ele=(struct Element *)malloc(s->num*sizeof(struct
Element));
    printf("Eneter non-zero Elements");
    for(i=0;i<s->num;i++)
        scanf("%d%d%d",&s->ele[i].i,&s->ele[i].j,&s-
>ele[i].x);

}

void display(struct Sparse s)
{
    int i,j,k=0;

    for(i=0;i<s.m;i++)
    {
        for(j=0;j<s.n;j++)
```

```
        {
            if(i==s.ele[k].i && j==s.ele[k].j)
                printf("%d ",s.ele[k++].x);
            else
                printf("0 ");
        }
        printf("\n");
    }
}

struct Sparse * add(struct Sparse *s1,struct Sparse *s2)
{
    struct Sparse *sum;
    int i,j,k;
    i=j=k=0;

    if(s1->n != s2->n && s1->m != s2->m)
        return NULL;
    sum=(struct Sparse *)malloc(sizeof(struct Sparse));
sum->ele=(struct Element *)malloc((s1->num+s2-
>num)*sizeof(struct Element));

    while(i<s1->num && j<s2->num)
    {
        if(s1->ele[i].i<s2->ele[j].i)
            sum->ele[k++]=s1->ele[i++];
        else if(s1->ele[i].i>s2->ele[j].i)
            sum->ele[k++]=s2->ele[j++];
        else
        {
            if(s1->ele[i].j<s2->ele[j].j)
                sum->ele[k++]=s1->ele[i++];
            else if(s1->ele[i].j>s2->ele[j].j)
                sum->ele[k++]=s2->ele[j++];
            else
            {
                sum->ele[k]=s1->ele[i];
                sum->ele[k++].x=s1->ele[i++].x+s2->ele[j+
+].x;
            }

        }
    }
    for(;i<s1->num;i++)sum->ele[k++]=s1->ele[i];
    for(;j<s2->num;j++)sum->ele[k++]=s2->ele[j];
    sum->m=s1->m;
    sum->n=s1->n;
    sum->num=k;
```

```
    return sum;

}


int main()
{
    struct Sparse s1,s2,*s3;

    create(&s1);
    create(&s2);

    s3=add(&s1,&s2);

    printf("First Matrix\n");
    display(s1);
    printf("Second Matrix\n");
    display(s2);
    printf("Sum Matrix\n");
    display(*s3);


    return 0;
}
```

Sparse Matrix using C++

```
#include <iostream>

using namespace std;

class Element
{
public:
    int i;
    int j;
    int x;
};

class Sparse
{
private:
    int m;
    int n;
    int num;
    Element *ele;
public:
    Sparse(int m,int n,int num)
    {
        this->m=m;
        this->n=n;
        this->num=num;
        ele=new Element[this->num];
    }
    ~Sparse()
    {
        delete [] ele;
    }

    Sparse operator+(Sparse &s);

    friend istream & operator>>(istream &is,Sparse &s);
    friend ostream & operator<<(ostream &os,Sparse &s);


};

Sparse Sparse::operator+(Sparse &s)
{
    int i,j,k;
```

```
    if(m!=s.m || n!=s.n)
        return Sparse(0,0,0);
    Sparse *sum=new Sparse(m,n,num+s.num);

    i=j=k=0;
    while(i<num && j<s.num)
    {
        if(ele[i].i<s.ele[j].i)
            sum->ele[k++]=ele[i++];
        else if(ele[i].i > s.ele[j].i)
            sum->ele[k++]=s.ele[j++];
        else
        {
            if(ele[i].j<s.ele[j].j)
                sum->ele[k++]=ele[i++];
            else if(ele[i].j > s.ele[j].j)
                sum->ele[k++]=s.ele[j++];
            else
            {
                sum->ele[k]=ele[i];
                sum->ele[k++].x=ele[i++].x+s.ele[j++].x;
            }

        }
    }
    for(;i<num;i++)sum->ele[k++]=ele[i];
    for(;j<s.num;j++)sum->ele[k++]=s.ele[j];
    sum->num=k;

    return *sum;

}









    istream & operator>>(istream &is,Sparse &s)
    {
        cout<<"Enter non-zero elements";
        for(int i=0;i<s.num;i++)
            cin>>s.ele[i].i>>s.ele[i].j>>s.ele[i].x;
        return is;
    }

    ostream & operator<<(ostream &os,Sparse &s)
    {
```

```
        int k=0;
        for(int i=0;i<s.m;i++)
        {
            for(int j=0;j<s.n;j++)
            {
                if(s.ele[k].i==i && s.ele[k].j==j)
                    cout<<s.ele[k++].x<<" ";
                else
                    cout<<"0 ";
            }
            cout<<endl;
        }
        return os;
    }



int main()
{
    Sparse s1(5,5,5);
    Sparse s2(5,5,5);

    cin>>s1;
    cin>>s2;

    Sparse sum=s1+s2;

    cout<<"First Matrix"<<endl<<s1;
    cout<<"Second MAtrix"<<endl<<s2;
    cout<<"Sum Matrix"<<endl<<sum;

    return 0;
}
```

Polynomial Representation

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

struct Term
{
    int coeff;
    int exp;
};
struct Poly
{
    int n;
    struct Term *terms;
};

void create(struct Poly *p)
{
    int i;
    printf("Number of terms?");
    scanf("%d",&p->n);
    p->terms=(struct Term*)malloc(p->n*sizeof(struct
Term));

    printf("Enter terms\n");
    for(i=0;i<p->n;i++)
        scanf("%d%d",&p->terms[i].coeff,&p-
>terms[i].exp);

}

void display(struct Poly p)
{
    int i;
    for(i=0;i<p.n;i++)

printf("%dx%d+",p.terms[i].coeff,p.terms[i].exp);
    printf("\n");
}
```

```
struct Poly *add(struct Poly *p1,struct Poly *p2)
{
    int i,j,k;
    struct Poly *sum;

    sum=(struct Poly*)malloc(sizeof(struct Poly));
    sum->terms=(struct Term *)malloc((p1->n+p2-
>n)*sizeof(struct Term));
    i=j=k=0;

    while(i<p1->n && j<p2->n)
    {
        if(p1->terms[i].exp>p2->terms[j].exp)
            sum->terms[k++]=p1->terms[i++];
        else if(p1->terms[i].exp < p2->terms[j].exp)
            sum->terms[k++]=p2->terms[j++];
        else
        {
            sum->terms[k].exp=p1->terms[i].exp;
            sum->terms[k++].coeff=p1->terms[i+
+].coeff+p2->terms[j++].coeff;
        }
    }
    for(;i<p1->n;i++)sum->terms[k++]=p1->terms[i];
    for(;j<p2->n;j++)sum->terms[k++]=p2->terms[j];

    sum->n=k;
    return sum;


}

int main()
{
    struct Poly p1,p2,*p3;

    create(&p1);
    create(&p2);

    p3=add(&p1,&p2);
```

```
    printf("\n");
    display(p1);
    printf("\n");
    display(p2);
    printf("\n");
    display(*p3);


    return 0;
}
```