Let us now see a program that shows how to perform these operations on an array.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

#define MAX 5

void insert ( int *, int pos, int num ) ;
void del ( int *, int pos ) ;
void reverse ( int * ) ;
void display ( int * ) ;
void search ( int *, int num ) ;

int main( )
{
    int arr[5] ;

    system ( "cls" ) ;

    insert ( arr, 1, 11 ) ;
    insert ( arr, 2, 12 ) ;
    insert ( arr, 3, 13 ) ;
    insert ( arr, 4, 14 ) ;
    insert ( arr, 5, 15 ) ;

    printf ( "Elements of Array:\n" ) ;
    display ( arr ) ;

    del ( arr, 5 ) ;
    del ( arr, 2 ) ;
    printf ( "After deletion:\n" ) ;
    display ( arr ) ;

    insert ( arr, 2, 222 ) ;
    insert ( arr, 5, 555 ) ;
```

```
    printf ( "After insertion:\n" ) ;
    display ( arr ) ;
    reverse ( arr ) ;
    printf ( "After reversing:\n" ) ;
    display ( arr ) ;
    search ( arr, 222 ) ;
    search ( arr, 666 ) ;

    return 0 ;
}

/* inserts an element num at given position pos */
void insert ( int *arr, int pos, int num )
{
    /* shift elements to right */
    int i ;
    for ( i = MAX - 1 ; i >= pos ; i-- )
        arr[i] = arr[i - 1] ;
    arr[i] = num ;
}

/* deletes an element from the given position pos */
void del ( int *arr, int pos )
{
    /* skip to the desired position */
    int i ;
    for ( i = pos ; i < MAX ; i++ )
        arr[i - 1] = arr[i] ;
    arr[i - 1] = 0 ;
}

/* reverses the entire array */
void reverse ( int *arr )
{
    int i ;
    for ( i = 0 ; i < MAX / 2 ; i++ )
    {
        int temp = arr[i] ;
```

```
        arr[i] = arr[MAX - 1 - i] ;
        arr[MAX - 1 - i] = temp ;
    }
}

/* searches array for a given element num */
void search ( int *arr, int num )
{
    /* Traverse the array */
    int i ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        if ( arr[i] == num )
        {
            printf ( "The element %d is present at %dth position.\n\n", num,
                    i + 1 ) ;
            return ;
        }
    }

    if ( i == MAX )
        printf ( "The element %d is not present in the array.\n\n", num ) ;
}

/* displays the contents of a array */
void display ( int *arr )
{
    /* traverse the entire array */
    int i ;
    for ( i = 0 ; i < MAX ; i++ )
        printf ( "%d\t", arr[i] ) ;
    printf ( "\n" ) ;
}
```

Merging Of Two Arrays

Merging of arrays involves two steps–sorting the arrays that are to be merged, and adding the sorted elements of both the arrays to new array in a sorted order. Let us now see a program th demonstrates merging of two arrays into a third array.

```
#include <stdio.h>
#include <conio.h>
#include <malloc.h>
#include <windows.h>

#define MAX1 5
#define MAX2 7

int *arr ;

int* create ( int ) ;
void sort ( int *, int ) ;
void display ( int *, int ) ;
```

```
int* merge ( int *, int * ) ;
int main( )
{
    int *a, *b, *c ;

    system ( "cls" ) ;

    printf (  "Enter elements for first array:\n\n" ) ;
    a = create ( MAX1 ) ;

    printf ( "Enter elements for second array:\n\n" ) ;
    b = create ( MAX2 ) ;

    sort ( a, MAX1 ) ;
    sort ( b, MAX2 ) ;

    printf ( "First array:\n" ) ;
    display ( a, MAX1 ) ;
    printf ( "Second array:\n" ) ;
    display ( b, MAX2 ) ;
    printf ( "After Merging:\n" ) ;

    c = merge ( a, b ) ;
    display ( c, MAX1 + MAX2 ) ;

    return 0 ;
}

/* creates array of given size, dynamically */
int* create ( int size )
{
    int *arr, i ;
    arr = ( int * ) malloc ( sizeof ( int ) * size ) ;

    for ( i = 0 ; i < size ; i++ )
    {
        printf ( "Enter the element no. %d: ", i + 1 ) ;
        scanf ( "%d", &arr[i] ) ;
```

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

/* sorts array in ascending order */
void sort ( int *arr, int size )
{
    int i, temp, j ;
    for ( i = 0 ; i < size ; i++ )
    {
        for ( j = i + 1 ; j < size ; j++ )
        {
            if ( arr[i] > arr[j] )
            {
                temp = arr[i] ;
                arr[i] = arr[j] ;
                arr[j] = temp ;
            }
        }
    }
}

/* displays the contents of array */
void display ( int *arr, int size )
{
    int i ;
    for ( i = 0 ; i < size ; i++)
        printf (  "%d\t", arr[i] ) ;
    printf ( "\n" ) ;
}

/* merges two arrays of different size */
int* merge ( int *a, int *b )
{
    int *arr ;
    int i, k, j ;
    int size = MAX1 + MAX2 ;
```

```
    arr = ( int * ) malloc ( sizeof ( int ) * ( size ) ) ;

    for ( k = 0, j = 0, i = 0 ; i <= size ; i++ )
    {
        if ( a[k] < b[j] )
        {
            arr[i] = a[k] ;
            k++ ;
            if ( k >= MAX1 )
            {
                for ( i++ ; j < MAX2 ; j++, i++ )
                    arr[i] = b[j] ;
            }
        }
        else
        {
            arr[i] = b[j] ;
            j++ ;
            if ( j >= MAX2 )
            {
                for ( i++ ; k < MAX1 ; k++, i++ )
                    arr[i] = a[k] ;
            }
        }
    }

    return arr ;
}
```

Common Matrix Operations

Common matrix operations are addition, multiplication, transposition, evaluation of the determinant of a square matrix, etc. The following program demonstrates these different matrix operations.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

#define MAX 3

void create ( int [3][3] ) ;
```

```
void display ( int [3][3] ) ;
void matadd ( int [3][3], int [3][3], int [3][3] ) ;
void matmul ( int [3][3], int [3][3], int [3][3] ) ;
void transpose ( int [3][3], int [3][3] ) ;

int main( )
{
    int mat1[3][3], mat2[3][3], mat3[3][3], mat4[3][3], mat5[3][3] ;

    system ( "cls" ) ;

    printf ( "Enter elements for first array:\n\n" ) ;
    create ( mat1 ) ;

    printf ( "Enter elements for second array:\n\n" ) ;
    create ( mat2 ) ;

    printf ( "First Array:\n" ) ;
    display ( mat1 ) ;
    printf ( "Second Array:\n" ) ;
    display ( mat2 ) ;

    matadd ( mat1, mat2, mat3 ) ;
    printf ( "After Addition:\n" ) ;
    display ( mat3 ) ;

    matmul ( mat1, mat2, mat4 ) ;
    printf ( "After Multiplication:\n" ) ;
    display ( mat4 ) ;

    transpose ( mat1, mat5 ) ;
    printf ( "Transpose of first matrix:\n" ) ;
    display ( mat5 ) ;

    return 0 ;
}

/* creates matrix mat */
```

```
void create ( int mat[3][3] )
{
    int i, j ;

    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
        {
            printf ( "Enter the element: " ) ;
            scanf ( "%d", &mat[i][j] ) ;
        }
    }
    printf ( "\n" ) ;
}

/* displays the contents of matrix */
void display ( int mat[3][3] )
{
    int i, j ;

    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            printf ( "%d\t", mat[i][j] ) ;
        printf ( "\n" ) ;
    }
}

/* adds two matrices m1 and m2 */
void matadd ( int m1[3][3], int m2[3][3], int m3[3][3] )
{
    int i, j ;

    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            m3[i][j] = m1[i][j] + m2[i][j] ;
    }
```

```
}

/* multiplies two matrices m1 and m2 */
void matmul ( int m1[3][3], int m2[3][3], int m3[3][3] )
{
    int i, j, k ;
    for ( k = 0 ; k < MAX ; k++ )
    {
        for ( i = 0 ; i < MAX ; i++ )
        {
            m3[k][i] = 0 ;
            for ( j = 0 ; j < MAX ; j++ )
                m3[k][i] += m1[k][j] * m2[j][i] ;
        }
    }
}

/* obtains transpose of matrix m1 */
void transpose ( int m1[3][3], int m2[3][3] )
{
    int i, j ;

    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            m2[i][j] = m1[j][i] ;
    }
}
```

More Matrix Operations

In addition to matrix operations that we discussed in previous program there are some more operations that can be performed on a matrix. These operations include calculating the determinant value for a given matrix, checking whether a matrix is a singular matrix or an orthogonal matrix, etc. Let us see a program in which we have implemented these operations, for a 3 x 3 matrix.

```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <windows.h>

#define MAX 3

void matrix ( int [3][3] ) ;
void create ( int [3][3] ) ;
void display ( int [3][3] ) ;
```

```
void matmul ( int [3][3], int [3][3], int [3][3] ) ;
void transpose ( int [3][3], int [3][3] ) ;
double determinant ( int [3][3] ) ;
int isortho ( int [3][3] ) ;

int main( )
{
    int mat [3][3] ;
    double d ;

    system ( "cls" ) ;

    printf ( "Enter elements for array:\n\n" ) ;
    create ( mat ) ;

    printf ( "The Matrix:\n" ) ;
    display ( mat ) ;

    d = determinant ( mat ) ;
    printf ( "The determinant for given matrix: %d.\n\n", d ) ;

    if ( d == 0 )
        printf ( "Matrix is singular.\n\n" ) ;
    else
        printf ( "Matrix  is not singular.\n\n" ) ;

    d = isortho ( mat ) ;

    if ( d != 0 )
        printf ( "Matrix is orthogonal.\n\n" ) ;
    else
        printf ( "Matrix is not orthogonal.\n\n" ) ;

    return 0 ;
}

/* initializes the matrix mat with 0 */
void matrix ( int mat[3][3] )
```

```
{
    int i, j ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            mat[i][j] = 0 ;
    }
}

/* creates matrix mat */
void create ( int mat[3][3] )
{
    int n, i, j ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
        {
            printf ( "Enter the element: " ) ;
            scanf ( "%d", &n ) ;
            mat[i][j] = n ;
        }
    }
    printf ( "\n" ) ;
}

/* displays the contents of matrix */
void display ( int mat[3][3] )
{
    int i, j ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            printf ( "%d\t", mat[i][j] ) ;
        printf ( "\n" ) ;
    }
    printf ( "\n" ) ;
}
```

```
/* multiplies two matrices */
void matmul ( int mat1[3][3], int mat2[3][3], int mat3[3][3] )
{
    int i, j, k ;
    for ( k = 0 ; k < MAX ; k++ )
    {
        for ( i = 0 ; i < MAX ; i++ )
        {
            mat3[k][i] = 0 ;
            for ( j = 0 ; j < MAX ; j++ )
                mat3[k][i] += mat1[k][j] * mat2[j][i] ;
        }
    }
}

/* obtains transpose of matrix m1 */
void transpose ( int mat[3][3], int m[3][3] )
{
    int i, j ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        for ( j = 0 ; j < MAX ; j++ )
            m[i][j] = mat[j][i] ;
    }
}

/* finds the determinant value for given matrix */
double determinant ( int mat[3][3] )
{
    int i, j, k ;
    double sum, p ;
    sum = 0.0 ; j = 1 ; k = MAX - 1 ;

    for ( i = 0 ; i < MAX ; i++ )
    {
        p = pow ( -1.0, i ) ;

        if ( i == MAX - 1 )
```

```
            k = 1 ;
        sum = sum + ( mat[0][i] * ( mat[1][j] *
                                    mat[2][k] - mat[2][j] *
                                    mat[1][k] ) ) * p ;
        j = 0 ;
    }

    return sum ;
}

/* checks if given matrix is an orthogonal matrix */
int isortho ( int mat[3][3] )
{
    /* transpose the matrix */
    int m1[3][3], m2[3][3], i ;
    transpose ( mat, m1 ) ;

    /* multiply the matrix with its transpose */
    matmul ( mat, m1, m2 ) ;

    /* check for the identity matrix */
    for ( i = 0 ; i < MAX ; i++ )
    {
        if ( m2[i][i] == 1 )
            continue ;
        else
            break ;
    }
    if ( i == MAX )
        return 1 ;
    else
        return 0 ;
}
```

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

#define MAX 10

struct term
{
        int coeff ;
        int exp ;
} ;

struct poly
{
        struct term t [10] ;
        int noofterms ;
} ;
```

```
void initpoly ( struct poly * ) ;
void polyappend ( struct poly *, int c, int e ) ;
struct poly polyadd ( struct poly, struct poly ) ;
void display ( struct poly ) ;

int main( )
{
    struct poly p1, p2, p3 ;

    system ( "cls" ) ;

    initpoly ( &p1 ) ;
    initpoly ( &p2 ) ;
    initpoly ( &p3 ) ;

    polyappend ( &p1, 1, 7 ) ;
    polyappend ( &p1, 2, 6 ) ;
    polyappend ( &p1, 3, 5 ) ;
    polyappend ( &p1, 4, 4 ) ;
    polyappend ( &p1, 5, 2 ) ;

    polyappend ( &p2, 1, 4 ) ;
    polyappend ( &p2, 1, 3 ) ;
    polyappend ( &p2, 1, 2 ) ;
    polyappend ( &p2, 1, 1 ) ;
    polyappend ( &p2, 2, 0 ) ;

    p3 = polyadd ( p1, p2 ) ;

    printf ( "First polynomial:\n" ) ;
    display ( p1 ) ;

    printf ( "Second polynomial:\n" ) ;
    display ( p2 ) ;

    printf ( "Resultant polynomial:\n" ) ;
    display ( p3 ) ;
```

```
    return 0 ;
}

/* initializes elements of struct poly */
void initpoly ( struct poly *p )
{
    int i ;
    p -> noofterms = 0 ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        p -> t[i].coeff = 0 ;
        p -> t[i].exp = 0 ;
    }
}

/* adds the term of polynomial to the array t */
void polyappend ( struct poly *p, int c, int e )
{
    p -> t[p -> noofterms].coeff = c ;
    p -> t[p -> noofterms].exp =  e ;
    ( p -> noofterms ) ++ ;
}

/* displays the polynomial equation */
void display ( struct poly p )
{
    int flag = 0, i ;
    for ( i = 0 ; i < p.noofterms ; i++ )
    {
        if ( p.t[i].exp != 0 )
            printf ( "%d x^%d + ", p.t[i].coeff, p.t[i].exp ) ;
        else
        {
            printf ( "%d", p.t[i].coeff ) ;
            flag = 1 ;
        }
    }
```

```
    if ( !flag )
        printf ( "\b\b  " ) ;
    printf ( "\n\n" ) ;
}

/* adds two polynomials p1 and p2 */
struct poly polyadd ( struct poly p1, struct poly p2 )
{
    int i, j, c ;
    struct poly p3 ;
    initpoly ( &p3 ) ;

    if ( p1.noofterms > p2.noofterms )
        c = p1.noofterms ;
    else
        c = p2.noofterms ;

    for ( i = 0, j = 0 ; i <= c ; p3.noofterms++ )
    {
        if ( p1.t[i].coeff == 0 && p2.t[j].coeff == 0 )
            break ;
        if ( p1.t[i].exp >= p2.t[j].exp )
        {
            if ( p1.t[i].exp == p2.t[j].exp )
            {
                p3.t[p3.noofterms].coeff = p1.t[i].coeff + p2.t[j].coeff ;
                p3.t[p3.noofterms].exp = p1.t[i].exp ;
                i++ ;
                j++ ;
            }
            else
            {
                p3.t[p3.noofterms].coeff = p1.t[i].coeff ;
                p3.t[p3.noofterms].exp = p1.t[i].exp ;
                i++ ;
            }
        }
        else
```

```
        {
            p3.t[p3.noofterms].coeff = p2.t[j].coeff ;
            p3.t[p3.noofterms].exp = p2.t[j].exp ;
            j++ ;
        }
    }
    return p3 ;
}
```

Multiplication Of Polynomials

Let us now see a program that carries out multiplication o
polynomials.

```
#include <stdio.h>
#include <conio.h>
#include <windows.h>

#define MAX 10

struct term
{
    int coeff ;
    int exp ;
} ;

struct poly
{
    struct term t [10] ;
    int noofterms ;
} ;

void initpoly ( struct poly * ) ;
void polyappend ( struct poly *, int, int ) ;
struct poly polyadd ( struct poly, struct poly ) ;
```

```
struct poly polymul ( struct poly, struct poly ) ;
void display ( struct poly ) ;

int main( )
{
    struct poly p1, p2, p3 ;

    system ( "cls" ) ;

    initpoly ( &p1 ) ;
    initpoly ( &p2 ) ;
    initpoly ( &p3 ) ;

    polyappend ( &p1, 1, 4 ) ;
    polyappend ( &p1, 2, 3 ) ;
    polyappend ( &p1, 2, 2 ) ;
    polyappend ( &p1, 2, 1 ) ;

    polyappend ( &p2, 2, 3 ) ;
    polyappend ( &p2, 3, 2 ) ;
    polyappend ( &p2, 4, 1 ) ;

    p3 = polymul ( p1, p2 ) ;

    printf ( "First polynomial:\n" ) ;
    display ( p1 ) ;

    printf ( "Second polynomial:\n" ) ;
    display ( p2 ) ;

    printf ( "Resultant polynomial:\n" ) ;
    display ( p3 ) ;

    return 0 ;
}

/* initializes elements of struct poly */
void initpoly ( struct poly *p )
```

```
{
    int i ;
    p -> noofterms = 0 ;
    for ( i = 0 ; i < MAX ; i++ )
    {
        p -> t[i].coeff = 0 ;
        p -> t[i].exp = 0 ;
    }
}

/* adds the term of polynomial to the array t */
void polyappend ( struct poly *p, int c, int e )
{
    p -> t[p -> noofterms].coeff = c ;
    p -> t[p -> noofterms].exp =  e ;
    ( p -> noofterms ) ++ ;
}

/* displays the polynomial equation */
void display ( struct poly p )
{
    int flag = 0, i ;
    for ( i = 0 ; i < p.noofterms ; i++ )
    {
        if ( p.t[i].exp != 0 )
            printf ( "%d x^%d + ", p.t[i].coeff, p.t[i].exp ) ;
        else
        {
            printf ( "%d", p.t[i].coeff ) ;
            flag = 1 ;
        }
    }
    if ( !flag )
        printf ( "\b\b  " ) ;
    printf ( "\n\n" ) ;
}
/* adds two polynomials p1 and p2 */
struct poly polyadd ( struct poly p1, struct poly p2 )
```

```
{
    int i, j, c ;
    struct poly p3 ;
    initpoly ( &p3 ) ;

    if ( p1.noofterms > p2.noofterms )
        c = p1.noofterms ;
    else
        c = p2.noofterms ;

    for ( i = 0, j = 0 ; i <= c ; p3.noofterms++ )
    {
        if ( p1.t[i].coeff == 0 && p2.t[j].coeff == 0 )
            break ;
        if ( p1.t[i].exp >= p2.t[j].exp )
        {
            if ( p1.t[i].exp == p2.t[j].exp )
            {
                p3.t[p3.noofterms].coeff = p1.t[i].coeff + p2.t[j].coeff ;
                p3.t[p3.noofterms].exp = p1.t[i].exp ;
                i++ ;
                j++ ;
            }
            else
            {
                p3.t[p3.noofterms].coeff = p1.t[i].coeff ;
                p3.t[p3.noofterms].exp = p1.t[i].exp ;
                i++ ;
            }
        }
        else
        {
            p3.t[p3.noofterms].coeff = p2.t[j].coeff ;
            p3.t[p3.noofterms].exp = p2.t[j].exp ;
            j++ ;
        }
    }
    return p3 ;
```

```
}

/* multiplies two polynomials p1 and p2 */
struct poly polymul ( struct poly p1, struct poly p2 )
{
    int coeff, exp ;
    struct poly temp, p3 ;

    initpoly ( &temp ) ;
    initpoly ( &p3 ) ;

    if ( p1.noofterms != 0 && p2.noofterms != 0 )
    {
        int i ;
        for ( i = 0 ; i < p1.noofterms ; i++ )
        {
            int j ;

            struct poly p ;
            initpoly ( &p ) ;

            for ( j = 0 ; j < p2.noofterms ; j++ )
            {
                coeff = p1.t[i].coeff * p2.t[j].coeff ;
                exp = p1.t[i].exp + p2.t[j].exp ;
                polyappend ( &p, coeff, exp ) ;
            }

            if ( i != 0 )
            {
                p3 = polyadd ( temp, p ) ;
                temp = p3 ;
            }
            else
                temp = p ;
        }
    }
    return p3 ;
```