Program 1: Design a C program which sorts the strings using array of pointers

```
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
void main()
{
        char * temp;
        int i, j, diff, num_strings;
        char * strArray[10];
        printf("Enter the number of strings: ");
        scanf("%d",&num_strings);
        for (i = 0; i < num_strings ;i++)
        {
                strArray[i] = (char *)malloc(sizeof(char)*20);
                printf("Enter string %d: ", i+1);
                scanf("%s",strArray[i]);
        }
        printf("Before Sorting\n");
        for (i = 0; i < num_strings ;i++)
        {
                printf("%s\n",strArray[i]);
        }
        for (i = 0; i < num_strings - 1; i++)
        {
                for (j = 0; j < num_strings-1-i; j++)
                {
                        diff = strcmp(strArray[j], strArray[j+1]);
                        if (diff > 0)
                        {
                                temp = strArray[j];
                                strArray[j] = strArray[j+1];
                                strArray[j+1] = temp;

                        }
                }
        }
        printf("After Sorting\n");
        for (i = 0; i < num_strings ;i++)
        {
                printf("%s\n",strArray[i]);
        }
}
```

Program 2: Write a C program to Search a Key element using Linear search Technique

```
#include<stdio.h>
void main()
{
	int a[20], i, n, key, flag = 0, pos;
	printf("Enter value of n : ");
	scanf("%d", &n);
	for (i = 0; i < n; i++)
	{
		printf("Enter element for a[%d] : ", i);
		scanf("%d", &a[i]);
	}
	printf("Enter key element : ");
	scanf("%d", &key);
	for (i = 0; i < n; i++)
	{
		if (key == a[i])
		{
			flag = 1;
			pos = i;
			break;
		}
	}
	if (flag == 1)
	{
		printf("The key element %d is found at the position %d\n", key, pos);
	}
	else
	{
		printf("The key element %d is not found in the array\n", key);
	}
}
```

Program 3: Write a C program to Search a Key element using Binary search Technique

```
#include<stdio.h>
void main()
{
	int a[20], i, j, n, key, flag = 0, low, high, mid, temp;
	printf("Enter value of n : ");
	scanf("%d", &n);
	for (i = 0; i < n; i++)
	{
		printf("Enter element for a[%d] : ", i);
		scanf("%d", &a[i]);
```

```
}
printf("Enter key element : ");
scanf("%d", &key);
for (i = 0; i < n - 1; i++)
{
	for (j = 0; j < n - i - 1; j++)
	{
		if (a[j] > a[j+1])
		{
			temp = a[j];
			a[j] = a[j+1];
			a[j+1] = temp;
		}
	}
}
printf("After sorting the elements in the array are\n");
for (i = 0; i < n; i++)
{
	printf("Value of a[%d] = %d\n", i, a[i]);
}
low = 0;
high = n - 1;
while (flag == 0 && low <= high)
{
	mid = (low + high) / 2 ;
	if (a[mid] == key)
	{
		flag = 1;
		break;
	}
	else if (a[mid] < key)
	{
		low = mid + 1;
	}
	else if (a[mid] > key)
	{
		high = mid - 1;
	}
}
if (flag == 1)
{
	printf("The key element %d is found at the position %d\n", key, mid);
}
else
{
```

```
                printf("The Key element %d is not found in the array\n", key);
        }
}
```

Program 4 : Write a C program to implement Fibonacci Search technique

```
#include <stdio.h>
#include <conio.h>
int min(int x, int y)
{
        return (x<=y)? x : y;
}
int fibonaccianSearch(int arr[], int x, int n)
{
        int m2 = 0;
        int m1 = 1;
        int m = m2 + m1;
        while (m < n)
        {
                m2 = m1;
                m1 = m;
                m = m2 + m1;
        }
        int offset = -1;
        while (m > 1)
        {
                int i = min(offset+m2, n-1);
                if (arr[i] < x)
                {
                        m = m1;
                        m1 = m2;
                        m2 = m - m1;
                        offset = i;
                }
                else if (arr[i] > x)
                {
                        m = m2;
                        m1 = m1 - m2;
                        m2 = m - m1;
                }
                else return i;
        }
        if(m1  && arr[offset+1]==x)
                return offset+1;
        return -1;
```

```
}
int main()
{
	int size;
	int *arr, i,x,result=-1;
	printf("Enter the size of an array: ");
	scanf("%d",&size);
	arr = (int*) malloc(size * sizeof(int));
	printf("Enter the %d array elements\n",size);
	for (i = 0; i < size; i++)
	{
		scanf("%d", &arr[i]);
	}
	printf("Enter the element to be searched: ");
	scanf("%d",&x);
	result = fibonaccianSearch(arr,x,size+1);
	if (result != -1 )
		printf("Element found at index: %d.\n",result);
	else
		printf("Element not found.\n");
	return 0;
}
```

Program 5 : Write a C program to Sort the elements using Insertion Sort Technique

```
#include<stdio.h>
void main()
{
	int a[20], i, n, j, temp, pos;
	printf("Enter value of n : ");
	scanf("%d", &n);
	for (i = 0; i < n; i++)
	{
		printf("Enter element for a[%d] : ", i);
		scanf("%d", &a[i]);
	}
	printf("Before sorting the elements in the array are\n");
	for (i = 0; i < n; i++)
	{
		printf("Value of a[%d] = %d\n", i, a[i]);
	}
	for (pos = 1; pos < n; pos++)
	{
		temp = a[pos];
		for (j = pos; j > 0; j--)
```

```
                {
                        if(a[j-1] > temp)
                {
                                a[j] = a[j-1];
                                a[j-1] = temp;
                        }
                }
        }
        printf("After sorting the elements in the array are\n");
        for (i = 0; i < n; i++)
        {
                printf("Value of a[%d] = %d\n", i, a[i]);
        }
}
```

Program 6 : Write a C program to Sort the elements using Selection Sort - Smallest element method Technique

```
#include<stdio.h>
void main()
 {
        int a[20], i, n, j, small, index;
        printf("Enter value of n : ");
        scanf("%d", &n);
        for (i = 0; i < n; i++)
        {
                printf("Enter element for a[%d] : ", i);
                scanf("%d", &a[i]);
        }
        printf("Before sorting the elements in the array are\n");
        for (i = 0; i < n; i++)
        {
                printf("Value of a[%d] = %d\n", i, a[i]);
        }
        for (i = 0; i < n; i++)
        {
                small = a[i] ;
                index = i;
                for (j = i + 1; j < n; j++)
                {
                        if (a[j] < small)
                        {
                                small = a[j];
                                index = j;
                        }
```

```
                }
                a[index] = a[i];
                a[i] = small;
        }
        printf("After sorting the elements in the array are\n");
        for (i = 0; i < n; i++)
        {
                printf("Value of a[%d] = %d\n", i, a[i]);
        }
}
```

Program 7 : Write a C program to sort given elements using shell sort technique.

```
#include <stdio.h>
#include <conio.h>
int shellSort(int arr[], int n)
{
        for (int gap = n/2; gap > 0; gap /= 2)
        {
                for (int i = gap; i < n; i += 1)
                {
                        int temp = arr[i];
                        int j;
                        for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                                arr[j] = arr[j - gap];
                        arr[j] = temp;
                }
        }
        return 0;
}
void printArray(int arr[], int n)
{
        for (int i=0; i<n; i++)
                printf("%d ",arr[i]);
        printf("\n");
}
int main()  {
        int size;
        int *arr, i;
        printf("Enter array size : ");
        scanf("%d",&size);
        arr = (int*) malloc(size * sizeof(int));
        printf("Enter %d elements : ",size);
        for (i = 0; i < size; i++)
         {
```

```
            scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        printArray(arr,size);

        shellSort(arr,size);

        printf("After sorting the elements are : ");
        printArray(arr,size);
        return 0;
}
```

Program 8 : Write a C program to Sort the elements using Bubble Sort Technique

```
#include<stdio.h>
void main()
{
        int a[20], i, n, j, temp;
        printf("Enter value of n : ");
        scanf("%d", &n);
        for (i = 0; i < n; i++)
        {
                printf("Enter element for a[%d] : ", i);
                scanf("%d", &a[i]);
        }
        printf("Before sorting the elements in the array are\n");
        for (i = 0; i < n; i++)
        {
                printf("Value of a[%d] = %d\n", i, a[i]);
        }
        for (i = 0; i < n - 1; i++)
        {
                for (j = 0; j < n - i - 1; j++)
                {
                        if (a[j] > a[j+1])
                        {
                                temp = a[j];
                                a[j] = a[j+1];
                                a[j+1] = temp;
                        }
                }
        }
        printf("After sorting the elements in the array are\n");
        for (i = 0; i < n; i++)
        {
```

```
                printf("Value of a[%d] = %d\n", i, a[i]);
        }
}
```

Program 9 : Write a program to sort Ascending order the given elements using quick sort technique.

```
#include <stdio.h>
void main()
{
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++)
        {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        quickSort(arr, 0, n - 1);
        printf("After sorting the elements are : ");
        display(arr, n);
}
void display(int arr[15], int n)
{
        int i;
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");
}
int partition(int arr[15], int lb, int ub)
{
        int pivot, down = lb, up = ub, temp;
        pivot = arr[lb];
        while (down < up)
        {
                while (arr[down] <= pivot && down < up)
                {
                        down++;
                }
                while (arr[up] > pivot)
                {
                        up--;
                }
```

```
                if (down < up)
                {
                        temp = arr[up];
                        arr[up] = arr[down];
                        arr[down] = temp;
                }
        }
        arr[lb] = arr[up];
        arr[up] = pivot;
        return up;
}
void quickSort(int arr[15], int low, int high)
{
        int j;
        if (low < high)
        {
                j = partition(arr, low, high);
                quickSort(arr, low, j - 1);
                quickSort(arr, j + 1, high);
        }
}
```

Program 10 : Write a C program to sort the given elements using Heap sort

```
#include <stdio.h>
void main()
{
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
        printf("Enter %d elements : ", n);
        for (i = 0; i < n; i++)
        {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        display(arr, n);
        heapsort(arr,n);
        printf("After sorting the elements are : ");
        display(arr, n);
}
void display(int arr[15], int n)
{
        int i;
        for (i = 0; i < n; i++)
```

```
                printf("%d ", arr[i]);
        printf("\n");
}
void heapify(int arr[], int n, int i)
{
        int largest = i;
        int l = 2*i + 1;
        int r = 2*i + 2;
        int temp;
        if (l < n && arr[l] > arr[largest])
                largest = l;
        if (r < n && arr[r] > arr[largest])
                largest = r;
        if (largest != i)
        {
                temp = arr[i];
                arr[i] = arr[largest];
                arr[largest] = temp;
                heapify(arr, n, largest);
        }
}
void heapsort(int arr[], int n)
{
        int i,temp;
        for(i = n/2-1; i >=0 ; i--)
        {
                heapify(arr,n,i);
        }
        for(i = n-1; i >= 0; i--)
        {
                temp = arr[0];
                arr[0] = arr[i];
                arr[i] = temp;
                heapify(arr,i,0);
        }
}
```

Program 11 : Write a C program to Sort given elements using Merge sort.

```
#include <stdio.h>
void main()
{
        int arr[15], i, n;
        printf("Enter array size : ");
        scanf("%d", &n);
```

```
	printf("Enter %d elements : ", n);
	for (i = 0; i < n; i++)
	{
		scanf("%d", &arr[i]);
	}
	printf("Before sorting the elements are : ");
	display(arr, n);
	splitAndMerge(arr, 0, n - 1);
	printf("After sorting the elements are : ");
	display(arr, n);
}
void display(int arr[15], int n)
{
	int i;
	for (i = 0; i < n; i++)
		printf("%d ", arr[i]);
	printf("\n");
}
void merge(int arr[15], int low, int mid, int high)
{
	int i = low, h = low, j = mid + 1, k, temp[15];
	while (h <= mid && j <= high)
	{
		if (arr[h] <= arr[j])
		{
			temp[i] = arr[h];
			h++;
		}
		else
		 {
			temp[i] = arr[j];
			j++;
		}
		i++;
	}
	if (h > mid)
	{
		for (k = j; k <= high; k++)
		{
			temp[i] = arr[k];
			i++;
		}
	}
	else
	 {
```

```
                for (k = h; k <= mid; k++)
                {
                        temp[i] = arr[k];
                        i++;
                }
        }
        for (k = low; k <= high; k++)
        {
                arr[k] = temp[k];
        }
}
void splitAndMerge(int arr[15], int low, int high)
{
        if (low < high)
        {
                int mid = (low + high) / 2;
                splitAndMerge(arr, low, mid);
                splitAndMerge(arr, mid + 1, high);
                merge(arr, low, mid, high);
        }
}
```

Program 12 : Write a C program to sort given elements using Radix sort

```
#include <stdio.h>
#include <conio.h>
int largest(int a[], int n)
{
   int large = a[0], i;
   for(i = 1; i < n; i++)
   {
      if(large < a[i])
         large = a[i];
   }
   return large;
}

void printArray(int arr[], int n)
{
        for (int i=0; i<n; i++)
                printf("%d ",arr[i]);
        printf("\n");
}
```

```
int main()
{
        int size;
        int *arr, i;
        printf("Enter array size : ");
         scanf("%d",&size);
        arr = (int*) malloc(size * sizeof(int));
        printf("Enter %d elements : ",size);
         for (i = 0; i < size; i++)
         {
                scanf("%d", &arr[i]);
        }
        printf("Before sorting the elements are : ");
        printArray(arr,size);
        RadixSort(arr,size);
        printf("After sorting the elements are : ");
        printArray(arr,size);
        return 0;
}

void RadixSort(int a[], int n)
{
   int bucket[10][10], bucket_count[10];
   int i, j, k, remainder, NOP=0, divisor=1, large, pass;
   large = largest(a, n);
   while(large > 0)
   {
      NOP++;
      large/=10;
   }
   for(pass = 0; pass < NOP; pass++)
   {
      for(i = 0; i < 10; i++)
      {
         bucket_count[i] = 0;
      }
      for(i = 0; i < n; i++)
      {
         remainder = (a[i] / divisor) % 10;
         bucket[remainder][bucket_count[remainder]] = a[i];
         bucket_count[remainder] += 1;
      }
      i = 0;
      for(k = 0; k < 10; k++)
     {
```

```
            for(j = 0; j < bucket_count[k]; j++)
            {
               a[i] = bucket[k][j];
               i++;
            }
         }
         divisor *= 10;
      }
}
```

Program 13 : C program to which performs all operations on singly linked list.

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

void menu()
{
    printf("Options\n");
    printf("1 : Insert elements into the linked list\n");
    printf("2 : Delete elements from the linked list\n");
    printf("3 : Display the elements in the linked list\n");
    printf("4 : Count the elements in the linked list\n");
    printf("5 : Exit()\n");
}

struct node
{
    int data;
    struct node *next;
};
typedef struct node node;
struct node *head=NULL;
node* createnode(int data)
{
    node* temp=(node*)malloc(sizeof(node));
    temp->data=data;
    temp->next=NULL;
    return temp;
}
void insert(int data)
{
    node* newnode=createnode(data);
    node* temp;
    if(head==NULL)
    {
        head=createnode(data);
    }
    else
    {
        temp=head;
```

```
        while(temp->next!=NULL)
        {
            temp=temp->next;
        }
        temp->next=newnode;
    }


}
void delete(int position)
{
    int i;
    node* temp;
    if(head==NULL)
    {
        printf("List is empty");

    }
    else
    {
        temp=head;
        for(i=1;i<position-1;i++)
        {
            temp=temp->next;
        }
        temp->next=temp->next->next;
        printf("Deleted successfully\n");
    }
}
void display()
{
    node* temp;
    temp=head;
    if(head==NULL)
    {
        printf("List is empty\n");
    }
    while(temp!=NULL)
    {
        printf("%d ",temp->data);
        temp=temp->next;
    }
    printf("\n");
}
void count()
{
    int c=0;
    node * temp;

    if(head==NULL)
    {
        printf("List is Empty\n");
```

```
    }
    else
    {
        temp=head;
        while(temp!=NULL)
        {
            c++;
            temp=temp->next;
        }
    }
    printf("No of elements in the linked list are : %d\n",c);;
}
void main()
{
    int choice,data,position,c;

    printf("Singly Linked List Example - All Operations\n");
    menu();
    printf("Enter your option : ");
    scanf("%d",&choice);
    while(choice!=5)
    {
        switch(choice)
        {
            case 1:
            {
                printf("Enter elements for inserting into linked list : ");
                scanf("%d",&data);

                insert(data);

                break;
            }
            case 2:
            {
                printf("Enter position of the element for deleteing the element : ");
                scanf("%d",&position);
                delete(position);
                break;
            }
            case 3:
            {
                printf("The elements in the linked list are : ");
                display();
                break;
            }
            case 4:
            {
                count();

                break;
```

```
            }
            case 5:
            {
                exit(0);
            }
            default:
            {
                printf("Enter options from 1 to 5\n");
                exit(0);
            }
        }
        menu();
        printf("Enter your option : ");
        scanf("%d",&choice);
    }
}
```

Program 14 : C program which performs all operations on double linked list.

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

struct dnode
{
   struct dnode *prev;
   int data;
   struct dnode *next;
};

struct dnode *start = NULL;

void insert(int);
void remov(int);
void display();

int main()
{
   int n, ch;
   do
   {
      printf("Operations on doubly linked list");
      printf("\n1. Insert \n2.Remove\n3. Display\n0. Exit");
      printf("\nEnter Choice 0-4? : ");
      scanf("%d", &ch);
      switch (ch)
      {
```

```
            case 1:
               printf("Enter number: ");
               scanf("%d", &n);
               insert(n);
               break;
            case 2:
              printf("Enter number to delete: ");
               scanf("%d", &n);
               remov(n);
               break;
            case 3:
               display();
               break;
         }
      }while (ch != 0);
}

void insert(int num)
{
   struct dnode *nptr,  *temp = start;
   nptr = malloc(sizeof(struct dnode));
   nptr->data = num;
   nptr->next = NULL;
   nptr->prev = NULL;
   if (start == NULL)
   {
      start = nptr;
   }
   else
   {
      while (temp->next != NULL)
         temp = temp->next;
      nptr->prev = temp;
      temp->next = nptr;
   }
}
void remov(int num)
{
   struct dnode *temp = start;
   while (temp != NULL)
   {
      if (temp->data == num)
      {
          if (temp == start)
         {
```

```
            start = start->next;
            start->prev = NULL;
         }
         else
         {

            if (temp->next == NULL)
               temp->prev->next = NULL;
            else
            {
               temp->prev->next = temp->next;
               temp->next->prev = temp->prev;
            }
            free(temp);
         }
         return ;
      }
      temp = temp->next;
   }
   printf("%d not found.\n", num);
}


void display()
{
   struct dnode *temp = start;
   while (temp != NULL)
   {
      printf("%d\t", temp->data);
      temp = temp->next;
   }
   printf("\n");
}
```

Program 15 : C program to which performs all operations on Circular linked list.

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

```
void deletion();
void find();
void print();
struct node *head = NULL;
int main()
{
	int choice;
	printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT\n");
	while(1)
	{
		printf("1.INSERT ");
		printf("2.DELETE ");
		printf("3.FIND ");
		printf("4.PRINT ");
		printf("5.QUIT\n");
		printf("Enter the choice: ");
		scanf("%d", &choice);
		switch(choice)
		{
			case 1:insert();break;
			case 2:deletion();break;
			case 3:find();break;
			case 4:print();break;
			case 5:exit(0);
		}
	}
}
void insert()
{
	int x,n;
	struct node *newnode,*temp = head, *prev;
	newnode = (struct node*)malloc(sizeof(struct node));
	printf("Enter the element to be inserted: ");
	scanf("%d", &x);
	printf("Enter the position of the element: ");
	scanf("%d", &n);
	newnode->data = x;
	newnode->next = NULL;
	if(head == NULL)
	{
	   head = newnode;
	   newnode->next = newnode;
	}
```

```
        else if(n == 1)
        {
                temp = head;
                newnode->next = temp;
                while(temp->next != head)
                        temp = temp->next;
                temp->next = newnode;
                head = newnode;
        }
        else
        {
            for(int i = 1; i < n-1; i++)
           {
                temp = temp->next;
           }
           newnode->next = temp->next;
           temp->next = newnode;
        }
}

void deletion()
{       struct node *temp = head, *prev, *temp1 = head;
        int key, count = 0;
        printf("Enter the element to be deleted: ");
        scanf("%d", &key);
        if(temp->data == key)
        {
           prev = temp -> next;
           while(temp->next != head)
           {
                temp = temp->next;
           }
           temp->next = prev;
           free(head);
           head = prev;
           printf("Element deleted\n");
        }
        else
        {
           while(temp->next != head)
           {
                if(temp->data == key)
                {
                        count += 1;
```

```
                        break;
                }
                prev = temp;
                temp = temp->next;
            }
            if(temp->data == key)
            {
                prev->next = temp->next;
                free(temp);
                printf("Element deleted\n");
            }
            else
            {
                printf("Element does not exist...!\n");
            }
        }
}

void find()
{
        struct node *temp = head;
        int key, count = 0;
        printf("Enter the element to be searched: ");
        scanf("%d", &key);
        while(temp->next != head)
        {
            if(temp->data == key)
            {
                count = 1;
                break;
            }
            temp = temp->next;
        }
        if (count == 1)
                printf("Element exist...!\n");
        else
        {
                if(temp->data == key)
                        printf("Element exist...!\n");
                else
                    printf("Element does not exist...!\n");
        }
}

void print()
```

```
{
        struct node *temp = head;
        printf("The list element are: ");

        while(temp->next != head)
        {
                printf("%d -> ",temp->data);
                temp = temp->next;
        }
        printf("%d -> ", temp->data) ;
        printf("\n");
}
```

Program 16 : Implementation of Circular Queue using Dynamic Array

```
#include <stdio.h>
#include <stdlib.h>
int *cqueue;
int front, rear;
int maxSize;

void initCircularQueue()
{
        cqueue = (int *)malloc(maxSize * sizeof(int));
        front = -1;
        rear = -1;
}

void dequeue()
{
        if (front == -1)
        {
                printf("Circular queue is underflow.\n");
        }
         else
        {
                printf("Deleted element = %d\n", *(cqueue + front));
                if (rear == front)
                {
                        rear = front = -1;
                }
                else if (front == maxSize - 1)
                {
                        front = 0;
                }
```

```
                else
                {
                        front++;
                }
        }
}

void enqueue(int x)
{
        if (((rear == maxSize - 1) && (front == 0)) || (rear + 1 == front))
        {
                printf("Circular queue is overflow.\n");
        }
        else
        {
                if (rear == maxSize - 1)
                {
                        rear = -1;
                }
                else if (front == -1)
                {
                        front = 0;
                }
                rear++;
                cqueue[rear] = x;
                printf("Successfully inserted.\n");
        }
}

void display()
{
        int i;
        if (front == -1 && rear == -1)
        {
                printf("Circular queue is empty.\n");
        }
        else
        {
                printf("Elements in the circular queue : ");
                if (front <= rear)
                {
                        for (i = front; i <= rear; i++)
                        {
                                printf("%d ", *(cqueue + i));
                        }
                }
```

```
                else
                {
                        for (i = front; i <= maxSize - 1; i++)
                        {
                                printf("%d ", *(cqueue + i));
                        }
                        for (i = 0; i <= rear; i++)
                        {
                                printf("%d ", *(cqueue + i));
                        }
                }
                printf("\n");
        }
}

int main()
{
        int op, x;
        printf("Enter the maximum size of the circular queue : ");
        scanf("%d", &maxSize);
        initCircularQueue();
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                exit(0);
                }
        }
}
```

Program 17 : Write a C program to implement different Operations on Stack using Array representation

```
#include <stdio.h>
#include <stdlib.h>
#define STACK_MAX_SIZE 10
int arr[STACK_MAX_SIZE];
int top = -1;

void push(int element)
{
	if(top == STACK_MAX_SIZE - 1)
	{
		printf("Stack is overflow.\n");
	}
	else
	{
		top = top + 1;
		arr[top] = element;
		printf("Successfully pushed.\n");
	}
}

void display()
{
	if (top < 0)
	{
		printf("Stack is empty.\n");
	}
	else
	 {
		printf("Elements of the stack are : " );
		for(int i = top; i >= 0; i--)
		{
			printf("%d ", arr[i]);
		}
		printf("\n");
	}
}
```

```
void pop()
{
	int x;
	if(top < 0)
	{
		printf("Stack is underflow.\n");
	}
	else
	{
		x = arr[top];
		top = top - 1;
		printf("Popped value = %d\n",x);
	}
}

void peek()
{
	int x;
	if(top < 0)
	{
		printf("Stack is underflow.\n");
	}
	 else
	{
		x = arr[top];
		printf("Peek value = %d\n",x);
	}
}

void isEmpty()
{
	if (top < 0)
	{
		printf("Stack is empty.\n");
	}
	 else
	{
		printf("Stack is not empty.\n");
	}
}

int main()
{
	int op, x;
```

```
        while(1)
        {
                printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op)
                {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

Program 18 : Write a C program to implement different Operations on Stack using Linked Lists

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

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

typedef struct stack *stk;
stk top = NULL;
```

```
stk push(int x)
{
	stk temp;
	temp = (stk)malloc(sizeof(struct stack));
	if(temp == NULL)
	{
		printf("Stack is overflow.\n");
	}
	 else
	{
		temp -> data = x;
		temp -> next = top;
		top = temp;
		printf("Successfully pushed.\n");
	}
}

void display()
{
	stk temp = top;
	if(temp == NULL)
	{
		printf("Stack is empty.\n");
	}
	else
	{
		printf("Elements of the stack are : ");
		while(temp != NULL)
		{
			printf("%d ", temp -> data);
			temp = temp -> next;
		}
		printf("\n");
	}
}

stk pop()
{
	stk temp;
	if(top == NULL)
	{
		printf("Stack is underflow.\n");
	}
```

```
	else
	{
		temp = top;
		top = top -> next;
		printf("Popped value = %d\n", temp -> data);
		free(temp);
	}
}

void peek()
{
	stk temp;
	if(top == NULL)
	{
		printf("Stack is underflow.\n");
	}
	else
	{
		temp = top;
		printf("Peek value = %d\n", temp -> data);
	}
}

void isEmpty()
{
	if(top == NULL)
	{
		printf("Stack is empty.\n");
	}
	else
	{
		printf("Stack is not empty.\n");
	}
}

int main()
{
	int op, x;
	while(1)
	{
		printf("1.Push 2.Pop 3.Display 4.Is Empty 5.Peek 6.Exit\n");
		printf("Enter your option : ");
		scanf("%d", &op);
```

```
                switch(op)
                {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d", &x);
                                push(x);
                                break;
                        case 2:
                                pop();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                peek();
                                break;
                        case 6:
                                exit(0);
                }
        }
}
```

Program 19 : Write a C program to implement different Operations on Queue using Array representation

```
#include <conio.h>
#include <stdio.h>
#define MAX 10
int queue[MAX];
int front = -1, rear = -1;
void enqueue(int x)
{
        if (rear == MAX - 1)
        {
                printf("Queue is overflow.\n");
        }
        else
        {
                rear++;
                queue[rear] = x;
                printf("Successfully inserted.\n");
```

```
        }
        if (front == -1)
        {
                front++;
        }
}
void dequeue()
{
        if (front == -1)
        {
                printf("Queue is underflow.\n");
        }
        else
        {
                printf("Deleted element = %d\n",queue[front]);
                if (rear == front)
                {
                        rear = front = -1;
                }
                else
                {
                        front++;
                }
        }
}

void display()
{
        if (front == -1 && rear == -1)
        {
                printf("Queue is empty.\n");
        }
        else
        {
                printf("Elements in the queue : ");
                for (int i = front; i <= rear; i++)
                {
                        printf("%d ",queue[i]);
                }
                printf("\n");
        }
}

void size()
{
```

```
	if(front == -1 && rear == -1)
		printf("Queue size : 0\n");
	else
		printf("Queue size : %d\n",rear-front+1);
}

void isEmpty()
{
	if(front == -1 && rear == -1)
		printf("Queue is empty.\n");
	else
		printf("Queue is not empty.\n");
}

int main()
{
	int op, x;
	while(1)
	{
		printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
		printf("Enter your option : ");
		scanf("%d",&op);
		switch(op)
		{
			case 1:
				printf("Enter element : ");
				scanf("%d",&x);
				enqueue(x);
				break;
			case 2:
				dequeue();
				break;
			case 3:
				display();
				break;
			case 4:
				isEmpty();
				break;
			case 5:
				size();
				break;
			case 6: exit(0);
		}
	}
}
```

Program 20 : Write a C program to implement different Operations on Queue using Dynamic Array

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

int *queue;
int front, rear;
int maxSize;

void initQueue()
 {
        queue = (int *)malloc(maxSize*sizeof(int));
        front = -1;
        rear = -1;
}

void enqueue(int x)
{
        if (rear == maxSize - 1)
        {
                printf("Queue is overflow.\n");
        }
        else
        {
                rear++;
                queue[rear] = x;
                printf("Successfully inserted.\n");
        }
        if (front == -1)
        {
                front++;
        }
}

void dequeue()
{
        if (front == -1)
        {
                printf("Queue is underflow.\n");
        }
        else
        {
                printf("Deleted element = %d\n", *(queue+front));
```

```
                if (rear == front)
                {
                        rear = front = -1;
                }
                else
                {
                        front++;
                }
        }
}

void display()
{
        if (front == -1 && rear == -1)
        {
                printf("Queue is empty.\n");
        }
        else
        {
                printf("Elements in the queue : ");
                for (int i = front; i <= rear; i++)
                {
                        printf("%d ",*(queue+i));
                }
                printf("\n");
        }
}
int main()
{
        int op, x;
        printf("Enter the maximum size of the queue : ");
        scanf("%d", &maxSize);
        initQueue();
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
                switch(op)
                {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
```

```
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                exit(0);
                }
        }
}
```

Program 21 : Write a C program to implement different Operations on Queue using Linked Lists

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

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

typedef struct queue *Q;
Q front = NULL, rear = NULL;

void enqueue(int element)
{
        Q temp = NULL;
        temp = (Q)malloc(sizeof(struct queue));
        if(temp == NULL)
        {
                printf("Queue is overflow.\n");
        }
         else
        {
                temp -> data = element;
                temp -> next = NULL;

                if(front == NULL)
                {
                        front = temp;
                }
```

```
                else
                {
                        rear -> next = temp;
                }
                rear = temp;
                printf("Successfully inserted.\n");
        }
}
void dequeue()
{
        Q temp = NULL;
        if(front == NULL)
        {
                printf("Queue is underflow.\n");
        }
        else
        {
                temp = front;
                if (front == rear)
                {
                        front = rear = NULL;
                }
                else
                {
                        front = front -> next;
                }
                printf("Deleted value = %d\n", temp -> data);
                free(temp);
        }
}
void display()
{
        if(front == NULL)
        {
                printf("Queue is empty.\n");
        }
        else
        {
                Q temp = front;
                printf("Elements in the queue : ");
                while(temp  != NULL)
                {
                        printf("%d ", temp -> data);
                        temp = temp -> next;
                }
```

```
                printf("\n");
        }
}
void size()
{
        int count =0;
        if(front == NULL)
        {
                printf("Queue size : 0\n");
        }
        else
        {
                Q temp = front;
                while(temp  != NULL)
                {
                        temp = temp -> next;
                        count = count + 1;
                }
                printf("Queue size : %d\n",count);
        }
}

void isEmpty()
{
        if(front == NULL )
        {
                printf("Queue is empty.\n");
        }
        else
        {
                printf("Queue is not empty.\n");
        }
}

int main()
{
        int op, x;
        while(1)
        {
                printf("1.Enqueue 2.Dequeue 3.Display 4.Is Empty 5.Size 6.Exit\n");
                printf("Enter your option : ");
                scanf("%d",&op);
```

```
                switch(op)
                {
                        case 1:
                                printf("Enter element : ");
                                scanf("%d",&x);
                                enqueue(x);
                                break;
                        case 2:
                                dequeue();
                                break;
                        case 3:
                                display();
                                break;
                        case 4:
                                isEmpty();
                                break;
                        case 5:
                                size();
                                break;
                        case 6: exit(0);
                }
        }
}
```

Program 22 : Reversing the links of a linked list

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

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

static void reverse(struct Node** head_ref)
{
  struct Node* prev = NULL;
  struct Node* current = *head_ref;
  struct Node* next = NULL;
  while (current != NULL)
  {
        next = current->next;
        current->next = prev;
        prev = current;
```

```
        current = next;
    }
    *head_ref = prev;
}

void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
    new_node->data = new_data;
    new_node->next = (*head_ref);
    (*head_ref) = new_node;
}

void printList(struct Node* head)
{
    struct Node* temp = head;
    while (temp != NULL)
    {
        printf("%d", temp->data);
        if ( temp -> next != NULL)
        {
            printf("->");
        }
        temp = temp->next;
    }
}

int main()
{
    struct Node* head = NULL;
    int i, count = 0, num = 0;
    printf("How many  numbers you want to enter:");
    scanf(" %d", &count);
    for (i = 0; i < count; i++)
    {
        printf("Enter number %d:", i+1);
        scanf(" %d", &num);
        push(&head, num);
    }
    printf("Given linked list:");
    printList(head);
    reverse(&head);
    printf("\nReversed linked list:");
    printList(head);
}
```

Program 23 : Program to insert into BST and traversal using In-order, Pre-order and Post-order

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

struct node
{
	int data;
	struct node *left, *right;
};

typedef struct node *BSTNODE;

BSTNODE newNodeInBST(int item)
{
	BSTNODE temp =  (BSTNODE)malloc(sizeof(struct node));
	temp->data = item;
	temp->left = temp->right = NULL;
	return temp;
}

void inorderInBST(BSTNODE root)
{
	if (root != NULL)
	{
		inorderInBST(root->left);
		printf("%d ", root->data);
		inorderInBST(root->right);
	}
}

void preorderInBST(BSTNODE root)
{
	if (root != NULL)
	{
		printf("%d ", root->data);
		preorderInBST(root->left);
		preorderInBST(root->right);
	}
}

void postorderInBST(BSTNODE root)
{
   if (root != NULL)
  {
```

```
        postorderInBST(root->left);
        postorderInBST(root->right);
        printf("%d ", root->data);
    }
}

BSTNODE insertNodeInBST(BSTNODE node, int ele)
{
        if (node == NULL)
        {
                printf("Successfully inserted.\n");
                return newNodeInBST(ele);
        }
        if (ele < node->data)
                node->left  = insertNodeInBST(node->left,ele);
        else if (ele > node->data)
                node->right = insertNodeInBST(node->right,ele);
        else
                printf("Element already exists in BST.\n");
        return node;
}

void main()
{
        int x, op;
        BSTNODE root = NULL;
        while(1)
        {
        printf("1.Insert 2.Inorder Traversal 3.Preorder Traversal 4.Postorder Traversal 5.Exit\n");
                printf("Enter your option : ");
                scanf("%d", &op);
                switch(op)
                {
                        case 1:
                                        printf("Enter an element to be inserted : ");
                                        scanf("%d", &x);
                                        root = insertNodeInBST(root,x);
                                        break;
                        case 2:
                                        if(root == NULL)
                                        {
                                                printf("Binary Search Tree is empty.\n");
                                        }
                                        else
                                        {
```

```
                                        printf("Elements of the BST (in-order traversal): ");
                                        inorderInBST(root);
                                        printf("\n");
                                }
                                break;
                        case 3:
                                if(root == NULL)
                                {
                                        printf("Binary Search Tree is empty.\n");
                                }
                                else
                                {
                                        printf("Elements of the BST (pre-order traversal): ");
                                        preorderInBST(root);
                                        printf("\n");

                                }
                                break;
                        case 4:
                                if(root == NULL)
                                {
                                        printf("Binary Search Tree is empty.\n");
                                }
                                else
                                {
                                       printf("Elements of the BST (post-order traversal): ");
                                        postorderInBST(root);
                                        printf("\n");
                                }
                                break;
                        case 5:
                                exit(0);
                }
        }
}
```

Program 24 : Write a Program to Search an element using Binary Search and Recursion

```
#include <stdio.h>

void read(int a[20], int n)
{
        int i;
        printf("Enter %d elements : ", n);
```

```
	for (i = 0; i < n; i++)
	{
		scanf("%d", &a[i]);
	}
}
void bubbleSort(int a[20], int n)
{
	int i, j, temp;
	for (i = 0; i < n - 1; i++)
	{
		for (j = 0; j < n - i - 1; j++)
		{
			if (a[j] > a[j+1])
			{
				temp = a[j];
				a[j] = a[j+1];
				a[j+1] = temp;
			}
		}
	}
}
void display(int a[20], int n)
{
	int i;
	for (i = 0; i < n; i++)
	{
		printf("%d ", a[i]);
	}
	printf("\n");
}
int binarySearch(int a[20], int low, int high, int key)
{
	int mid;
	if (low <= high)
	{
		mid = (low + high) / 2;
		if (a[mid] == key)
			return mid;
		else if (key < a[mid])
			binarySearch(a, low, mid - 1, key);
		else if (key > a[mid])
			binarySearch(a, mid + 1, high, key);
	}
	else
	{
```

```
                return -1;
        }
}

void main()
{
        int a[20], n, key, flag;
        printf("Enter value of n : ");
        scanf("%d", &n);
        read(a, n);
        bubbleSort(a, n);
        printf("After sorting the elements are : ");
        display(a, n);
        printf("Enter key element : ");
        scanf("%d", &key);
        flag = binarySearch(a, 0, n - 1, key);
        if (flag == -1)
        {
                printf("The given key element %d is not found\n", key);
        }
        else
        {
                printf("The given key element %d is found at position : %d\n", key, flag);
        }
}
```

Program 25 : Graph traversals implementation - Breadth First Search

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 99
struct node
{
        struct node *next;
        int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int queue[MAX], front = -1,rear = -1;
int n;
```

```
void insertQueue(int vertex)
{
	if(rear == MAX-1)
		printf("Queue Overflow.\n");
	else
	{
		if(front == -1)
			front = 0;
		rear = rear+1;
		queue[rear] = vertex ;
	}
}

int isEmptyQueue()
{
	if(front == -1 || front > rear)
		return 1;
	else
		return 0;
}

int deleteQueue()
{
	int deleteItem;
	if(front == -1 || front > rear)
	{
		printf("Queue Underflow\n");
		exit(1);
	}
	deleteItem = queue[front];
	front = front+1;
	return deleteItem;
}

void BFS(int v)
{
	int w;
	insertQueue(v);
	while(!isEmptyQueue())
	{
		v = deleteQueue( );
		printf("\n%d",v);
		visited[v]=1;
		GNODE g = graph[v];
		for(;g!=NULL;g=g->next)
```

```
                {
                        w=g->vertex;
                        if(visited[w]==0)
                        {
                                insertQueue(w);
                                visited[w]=1;
                        }
                }
        }
}

void main()
{
        int N, E, s, d, i, j, v;
        GNODE p, q;
        printf("Enter the number of vertices : ");
        scanf("%d",&N);
        printf("Enter the number of edges : ");
        scanf("%d",&E);
        for(i=1;i<=E;i++)
        {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                q=(GNODE)malloc(sizeof(struct node));
                q->vertex=d;
                q->next=NULL;
                if(graph[s]==NULL)
                {
                        graph[s]=q;
                }
                else
                {
                        p=graph[s];
                        while(p->next!=NULL)
                                p=p->next;
                        p->next=q;
                }
        }
        for(i=1;i<=n;i++)
                visited[i]=0;
        printf("Enter Start Vertex for BFS : ");
        scanf("%d", &v);
        printf("BFS of graph : ");
```

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

Program 26 : Graph traversals implementation - Depth First Search

```
#include<stdio.h>
#include<stdlib.h>
struct node
{
   struct node *next;
   int vertex;
};
typedef struct node * GNODE;
GNODE graph[20];
int visited[20];
int n;

void DFS(int i)
{
   GNODE p;
   printf("\n%d",i);
   p=graph[i];
   visited[i]=1;
   while(p!=NULL)
  {
      i=p->vertex;
      if(!visited[i])
          DFS(i);
       p=p->next;
   }

}

void main()
{
            int N,E,i,s,d,v;
            GNODE q,p;
            printf("Enter the number of vertices : ");
            scanf("%d",&N);
            printf("Enter the number of edges : ");
            scanf("%d",&E);
            for(i=1;i<=E;i++)
```

```
        {
                printf("Enter source : ");
                scanf("%d",&s);
                printf("Enter destination : ");
                scanf("%d",&d);
                q=(GNODE)malloc(sizeof(struct node));
                q->vertex=d;
                q->next=NULL;
                if(graph[s]==NULL)
                        graph[s]=q;
                else
                {
                        p=graph[s];
                        while(p->next!=NULL)
                        p=p->next;
                        p->next=q;
                }
        }
   for(i=0;i<n;i++)
      visited[i]=0;
   printf("Enter Start Vertex for DFS : ");
   scanf("%d", &v);
   printf("DFS of graph : ");
   DFS(v);
   printf("\n");
}
```

Program 27 : Travelling Sales Person problem using Dynamic programming

```
#include<stdio.h>
int ary[10][10], completed[10], n, cost = 0;
void takeInput()
{
  int i, j;
  printf("Number of villages: ");
  scanf("%d", & n);
  for (i = 0; i < n; i++)
 {
    for (j = 0; j < n; j++)
      scanf("%d", & ary[i][j]);

    completed[i] = 0;
  }

  printf("The cost list is:");
```

```
  for (i = 0; i < n; i++)
  {
    printf("\n");

    for (j = 0; j < n; j++)
      printf("\t%d", ary[i][j]);
  }
}
void mincost(int city)
{
  int i, ncity;

  completed[city] = 1;

  printf("%d-->", city + 1);
  ncity = least(city);

  if (ncity == 999)
  {
    ncity = 0;
    printf("%d", ncity + 1);
    cost += ary[city][ncity];

    return;
  }

  mincost(ncity);
}
int least(int c)
{
  int i, nc = 999;
  int min = 999, kmin;

  for (i = 0; i < n; i++)
  {
    if ((ary[c][i] != 0) && (completed[i] == 0))
      if (ary[c][i] + ary[i][c] < min)
      {
        min = ary[i][0] + ary[c][i];
        kmin = ary[c][i];
        nc = i;
      }
   }
```

```
  if (min != 999)
    cost += kmin;

  return nc;
}
int main()
{
  takeInput();
  printf("\nThe Path is:\n");
  mincost(0);
  printf("\nMinimum cost is %d", cost);
  return 0;
}
```

Program 28 : Write a C program to Open a File and to Print its contents on the screen

```
#include <stdio.h>
void main()
{
        FILE *fp;
        char ch;
        fp = fopen("SampleText1.txt", "w");
        printf("Enter the text with @ at end : ");
        while ((ch = getchar()) != '@')
        {
                putc(ch, fp);
        }
        putc(ch, fp);
        fclose(fp);
        fp = fopen("SampleText1.txt", "r");
        printf("Given message is : ");
        while ((ch = getc(fp)) != '@')
        {
                putchar(ch);
        }
        printf("\n");
        fclose(fp);
}
```

Program 29 : Write a C program to Copy contents of one File into another File

```
#include <stdio.h>
void main()
{
	FILE *fp, *fp1, *fp2;
	char ch;
	fp = fopen("SampleTextFile1.txt", "w");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	putc(ch, fp);
	fclose(fp);
	fp1 = fopen("SampleTextFile1.txt", "r");
	fp2 = fopen("SampleTextFile2.txt", "w");
	while ((ch = getc(fp1)) != '@')
	{
		putc(ch, fp2);
	}
	putc(ch, fp2);
	fclose(fp1);
	fclose(fp2);
	fp2 = fopen("SampleTextFile2.txt", "r");
	printf("Copied text is : ");
	while ((ch = getc(fp2)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp2);
}
```

Program 30 : Write a C program to Merge two Files and stores their contents in another File

```
#include <stdio.h>
void main()
{
	FILE *fp1, *fp2, *fp3;
	char ch;
	fp1 = fopen("SampleDataFile1.txt", "w");
	printf("Enter the text with @ at end for file-1 :\n");
	while ((ch = getchar()) != '@')
	{
```

```
		putc(ch, fp1);
	}
	putc(ch, fp1);
	fclose(fp1);
	fp2 = fopen("SampleDataFile2.txt", "w");
	printf("Enter the text with @ at end for file-2 :\n");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp2);
	}
	putc(ch, fp2);
	fclose(fp2);
	fp1 = fopen("SampleDataFile1.txt", "r");
	fp3 = fopen("SampleDataFile3.txt", "w");
	while ((ch = getc(fp1)) != '@')
	{
		putc(ch, fp3);
	}
	fclose(fp1);
	fp2 = fopen("SampleDataFile2.txt", "r");
	while ((ch = getc(fp2)) != '@')
	{
		putc(ch, fp3);
	}
	putc(ch, fp3);
	fclose(fp2);
	fclose(fp3);
	fp3 = fopen("SampleDataFile3.txt", "r");
	printf("Merged text is : ");
	while ((ch = getc(fp3)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp3);
}
```

Program 31 : Write a C program to Delete a File

```
#include <stdio.h>
void main()
{
	FILE *fp;
	int status;
	char fileName[40], ch;
```

```
	printf("Enter a new file name : ");
	gets(fileName);
	fp = fopen(fileName, "w");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	putc(ch, fp);
	fclose(fp);
	fp = fopen(fileName, "r");
	printf("Given message is : ");
	while ((ch = getc(fp)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp);
	status = remove(fileName);
	if (status == 0)
		printf("%s file is deleted successfully\n", fileName);
	else
	{
		printf("Unable to delete the file -- ");
		perror("Error\n");
	}
}
```

Program 32 : Write a C program to Copy last n characters from one File to another File

```
#include <stdio.h>
void main()
{
	FILE *fp, *fp1, *fp2;
	int num, length;
	char ch;
	fp = fopen("TestDataFile1.txt", "w");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	putc(ch, fp);
	fclose(fp);
```

```
	fp1 = fopen("TestDataFile1.txt", "r");
	fp2 = fopen("TestDataFile2.txt", "w");
	printf("Enter number of characters to copy : ");
	scanf("%d", &num);
	fseek(fp1, 0L, SEEK_END);
	length = ftell(fp1);
	fseek(fp1, (length - num - 1), SEEK_SET);
	while ((ch = getc(fp1)) != '@')
	{
		putc(ch, fp2);
	}
	putc(ch, fp2);
	fclose(fp1);
	fclose(fp2);
	fp2 = fopen("TestDataFile2.txt", "r");
	printf("Copied text is : ");
	while ((ch = getc(fp2)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp2);
}
```

Program 33 : Write a C program to Reverse first n characters in a File

```
#include <stdio.h>
#include <string.h>
void stringReverse(char[]);
void main()
{
	FILE *fp;
	int num, i;
	char ch, data[100];
	fp = fopen("TestDataFile3.txt", "w+");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	putc(ch, fp);
	printf("Enter number of characters to copy : ");
	scanf("%d", &num);
```

```
	i = 0;
	rewind(fp);
	while (i < num)
	{
		data[i] = getc(fp);
		i++;
	}
	data[i] = '\0';
	rewind(fp);
	stringReverse(data);
	fputs(data, fp);
	fclose(fp);
	fp = fopen("TestDataFile3.txt", "r");
	printf("Result is : ");
	while ((ch = getc(fp)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp);
}
void stringReverse(char data[100])
{
	int i, j;
	char temp;
	i = j = 0;
	while (data[j] != '\0')
	{
		j++;
	}
	j--;
	while (i < j)
	{
		temp = data[i];
		data[i] = data[j];
		data[j] = temp;
		i++;
		j--;
	}
}
```

Program 34 : Write a C program to Append data to an existing File

```
#include <stdio.h>
void main()
{
	FILE *fp;
	char ch;
	fp = fopen("DemoTextFile1.txt", "w");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	fclose(fp);
	fp = fopen("DemoTextFile1.txt", "a");
	printf("Enter the text to append to a file with @ at end : ");
	while ((ch = getchar()) != '@')
	{
		putc(ch, fp);
	}
	putc(ch, fp);
	fclose(fp);
	fp = fopen("DemoTextFile1.txt", "r");
	printf("File content after appending : ");
	while ((ch = getc(fp)) != '@')
	{
		putchar(ch);
	}
	printf("\n");
	fclose(fp);
}
```

Program 35 : Write a C program to Count number of Characters, Words and Lines of a given File

```
#include <stdio.h>
void main() {
	FILE *fp;
	char ch;
	int charCount = 0, wordCount = 0, lineCount = 0;
	fp = fopen("DemoTextFile2.txt", "w");
	printf("Enter the text with @ at end : ");
	while ((ch = getchar()) != '@')
	{
```

```
                putc(ch, fp);
        }
        putc(ch, fp);
        fclose(fp);
        fp = fopen("DemoTextFile2.txt", "r");
        do
        {
                if ((ch == ' ') || (ch == '\n') || (ch == '@'))
                        wordCount++;
                else
                        charCount++;
                if (ch == '\n' || ch == '@')
                        lineCount++;
        } while ((ch = getc(fp)) != '@');
        fclose(fp);
        printf("Total characters : %d\n", charCount);
        printf("Total words : %d\n", wordCount);
        printf("Total lines : %d\n", lineCount);
}
```

Program 36 : Linked list Female gender first

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

struct Node
{
   int data;
   char name[20];
   char gender;
   struct Node *next;
};

void segregateEvenOdd(struct Node **head_ref)
{
   struct Node *end = *head_ref;
   struct Node *prev = NULL;
   struct Node *curr = *head_ref;

   while (end->next != NULL)
      end = end->next;
```

```
  struct Node *new_end = end;

  while (curr->data %2 != 0 && curr != end)
 {
     new_end->next = curr;
     curr = curr->next;
     new_end->next->next = NULL;
     new_end = new_end->next;
  }

  if (curr->data%2 == 0)
 {
     *head_ref = curr;

     while (curr != end)
     {
        if ( (curr->data)%2 == 0 )
        {
           prev = curr;
           curr = curr->next;
        }
        else
        {
           prev->next = curr->next;

           curr->next = NULL;

           new_end->next = curr;

           new_end = curr;

           curr = prev->next;
        }
     }
  }
 else
          prev = curr;
 if (new_end!=end && (end->data)%2 != 0)
 {
     prev->next = end->next;
     end->next = NULL;
     new_end->next = end;
  }
  return;
}
```

```
void push(struct Node** head_ref, char new_name[20], char new_gender)
{
   struct Node* new_node = (struct Node*) malloc(sizeof(struct Node));
   strcpy(new_node->name, new_name);
   new_node->gender = new_gender;
   if (new_gender == 'F')
          new_node->data  = 0;
   else if (new_gender == 'M')
          new_node->data  = 1;
    new_node->next = (*head_ref);
    (*head_ref)    = new_node;
}
void printList(struct Node *node)
{
   while (node!=NULL)
   {
      printf("%s (%c)", node->name, node->gender);
      node = node->next;
      if (node!=NULL)
          printf(" --> ");
   }
}

int main()
{
   struct Node* head = NULL;
   char name[20];
   char gender;
   int noOfInputs, i;
   int option;
   printf("Insert Data\n");
   do
   {
      printf("Enter Name: ");
      scanf(" %s", name);
      printf("Enter Gender: ");
      scanf(" %c", &gender);
      push(&head, name, gender);
      printf("1 : Insert into Linked List\n");
      printf("0 : Exit\n");
      printf("Enter your option: ");
      scanf(" %d", &option);

   } while(option == 1);
```

```
    printf("Original Linked list \n");
    printList(head);
    segregateEvenOdd(&head);
    printf("\nModified Linked list \n");
    printList(head);
    printf("\n");
    return 0;
}
```

Program 37 : Indexing of a file

```
#include <stdio.h>
#define MAX 25

struct indexfile
{
    int indexId;
    int kIndex;
};
int main()
{
    int numbers[MAX];
    struct indexfile index[MAX];
    int i, num, low, high, br = 4;
    int noOfStudents;
    printf("How many numbers do you want to enter:");
    scanf(" %d", &noOfStudents);
    printf("Enter %d numbers:", noOfStudents);
    for (i = 0; i < noOfStudents; i++)
    {
        scanf("%d", &numbers[i]);
    }

    for (i = 0; i < (noOfStudents / 5); i++)
   {
        index[i].indexId = numbers[br];
        index[i].kIndex = br;
        br = br + 5;
    }
    printf("Enter a number to search:");
    scanf("%d", &num);
    for (i = 0; (i < noOfStudents / 5) && (index[i].indexId <= num); i++);
    if(i != 0)
            low = index[i - 1].kIndex;
```

```
    else
            low = 0;
    if(index[i].kIndex != 0 && index[i].kIndex <= noOfStudents)
            high = index[i].kIndex;
    else
            high = noOfStudents;
    for (i = low; i <= high; i++)
   {
        if (num == numbers[i])
        {
            printf("Number found at position:%d", i);
            return 0;
        }
    }
    printf("\nNumber not found.");
    return 0;
}
```

Program 38 : Write a C program to Convert an Infix expression into Postfix expression

```
#include<stdlib.h>
#include<string.h>
#include<stdio.h>
#include<ctype.h>
#define STACK_MAX_SIZE 20
char stack [STACK_MAX_SIZE];
int top = -1;

//Return 1 if stack is empty else return 0.
int isEmpty() {
        if(top<0)
                return 1;
        else
                return 0;
}

//Push the character into stack
void push(char x) {
        if(top == STACK_MAX_SIZE - 1) {
                printf("Stack is overflow.\n");
        } else {
                top = top + 1;
                stack[top] = x;
        }
}
```

```
//pop a character from stack
char pop() {
   if(top < 0) {
      printf("Stack is underflow : unbalanced parenthesis\n");
      exit(0);
   }
   else
      return stack[top--];
}

// Return 0 if char is '('
// Return 1 if char is '+' or '-'
// Return 2 if char is '*' or '/' or '%'
int priority(char x) {
   if(x == '(')
      return 0;
   if(x == '+' || x == '-')
      return 1;
   if(x == '*' || x == '/' || x == '%')
      return 2;
}

//Output Format
//if expression is correct then output will be Postfix Expression : <postfix notation>
//If expression contains invalid operators then output will be "Invalid symbols in infix expression.
Only alphanumberic and { '+', '-','*', '%%', '/' } are allowed."
//If the expression contains unbalanced paranthesis the output will be "Invalid infix expression :
unbalanced parenthesis."
void convertInfix(char * e) {
            int x;
            int k=0;
            char * p = (char *)malloc(sizeof(char)*strlen(e));
            while(*e != '\0') {
      if(isalnum(*e))
         p[k++]=*e;
      else if(*e == '(')
         push(*e);
      else if(*e == ')') {
         while(!isEmpty() && (x = pop()) != '('
            p[k++]=x;
      }
      else if (*e == '+' || *e == '-' || *e == '*' || *e == '/' || *e == '%') {
         while(priority(stack[top]) >= priority(*e))
            p[k++]=pop();
```

```
            push(*e);
        }
        else {
            printf("Invalid symbols in infix expression. Only alphanumeric and { '+', '-','*', '%%', '/' } are
allowed.\n");
            exit(0);
        }
        e++;
    }
    while(top != -1) {
            x=pop();
            if(x == '(') {
                    printf("Invalid infix expression : unbalanced parenthesis.\n");
                    exit(0);
            }
        p[k++] = x;
    }
    p[k++]='\0';
    printf("Postfix expression : %s\n",p);
}

int main() {
            char exp[20];
            char *e, x;
            printf("Enter the expression : ");
            scanf("%s",exp);
            e = exp;
            convertInfix(e);
}
```

Program 39 : Infix to Prefix Conversion

```
#define SIZE 50
#include<string.h>
#include <ctype.h>
#include<stdio.h>
char *strrev(char *str)
{
    char c, *front, *back;
    if(!str || !*str)
   {
        return str;
    }
    for(front=str,back=str+strlen(str)-1;front < back;front++,back--)
    {
```

```
        c=*front;
        *front=*back;
        *back=c;
    }
    return str;
}

char s[SIZE];
int top = -1;
void push (char elem)
{
            s[++top] = elem;
}

char pop ()
{
            return (s[top--]);
}

int pr (char elem)
{
  switch (elem)
  {
    case '#':
      return 0;
    case ')':
      return 1;
    case '+':
    case '-':
      return 2;
    case '*':
    case '/':
      return 3;
    }
}

void main ()
{
  char infx[50], prfx[50], ch, elem;
  int i = 0, k = 0;
  printf ("Enter Infix Expression:");
  scanf ("%s", infx);
  push ('#');
  strrev (infx);
  while ((ch = infx[i++]) != '\0')
```

```
 {
    if (ch == ')')
              push (ch);
    else if (isalnum (ch))
              prfx[k++] = ch;
    else if (ch == '(')
    {
              while (s[top] != ')')
              {
                 prfx[k++] = pop ();
              }
              elem = pop ();
     }
    else
    {
             while (pr (s[top]) >= pr (ch))
            {
              prfx[k++] = pop ();
             }
             push (ch);
    }
}
 while (s[top] != '#')
  {
           prfx[k++] = pop ();
  }
 prfx[k] = '\0';
 strrev (prfx);
 strrev (infx);
 printf ("Prefix Expression:%s\n",  prfx);
}
```

Program 40 : Postfix to Infix Conversion

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
# define MAX 20
char str[MAX],stack[MAX];
int top=-1;


void push(char c)
{
```

```
   stack[++top]=c;
}
char pop()
{
   return stack[top--];
}
char *strrev(char *str)
{
   char c, *front, *back;
   if(!str || !*str)
      return str;
   for(front=str,back=str+strlen(str)-1;front < back;front++,back--)
  {
      c=*front;*front=*back;*back=c;
   }
   return str;
}
void postfix()
{
   int n,i,j=0;
   char a,b,op,x[20];
   printf("Enter a Postfix expression:");
   fflush(stdin);
   scanf("%s", str);
   strrev(str);
   n=strlen(str);
   for(i=0;i<MAX;i++)
   {
      stack[i]='\0';
   }
   printf("Infix expression:");
   for(i=0;i<n;i++)
   {
      if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/')
      {
         push(str[i]);
      }
      else
      {
         x[j]=str[i]; j++;
         x[j]=pop(); j++;
      }
   }
   x[j]=str[top--];
   strrev(x);
```

```
   printf("%s\n",x);
}
void main()
{
   postfix();
}
```

Program 41 : Prefix to Infix Conversion

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
# define MAX 20
char str[MAX],stack[MAX];
int top=-1;
void push(char c)
{
   stack[++top]=c;
}
char pop()
{
   return stack[top--];
}
void prefix()
{
   int n,i;
   char a,b,op;
   printf("Enter a Prefix expression:");
   fflush(stdin);
   scanf("%s", str);
   n=strlen(str);
   for(i=0;i<MAX;i++)
   {
      stack[i]='\0';
   }
   printf("Infix expression:");
   for(i=0;i<n;i++)
   {
      if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/')
      {
         push(str[i]);
      }
```

```
      else
      {
         op=pop();
         a=str[i];
         if(op == '\0')
         {
           printf("%c",a);
         }
         else
         {
           printf("%c%c",a,op);
         }
      }
   }

          if(top >= 0)
          {
                    printf("%c\n",str[top--]);
          }
          else
          {
                    printf("\n");
          }
          // printf("%c\n",str[top--]);
}
void main()
{
   prefix();
}
```

Program 42 : Postfix to Prefix Conversion

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
# define MAX 20
char *strrev(char *str)
{
    char c, *front, *back;
    if(!str || !*str)
        return str;

    for(front=str,back=str+strlen(str)-1;front < back;front++,back--)
   {
```

```
        c=*front;
        *front=*back;
        *back=c;
    }
    return str;
}
char str[MAX],stack[MAX];
int top=-1;
void push(char c)
{
    stack[++top]=c;
}
char pop()
{
    return stack[top--];
}
void post_pre()
{
    int n,i,j=0; char c[20];
    char a,b,op;
    printf("Enter the postfix expression:");
    scanf("%s", str);
    n=strlen(str);
    for(i=0;i<MAX;i++)
    stack[i]='\0';
    printf("Prefix expression is:");
    for(i=n-1;i>=0;i--)
    {
        if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/')
        {
            push(str[i]);
        }
        else
        {
            c[j++]=str[i];
            while((top!=-1)&&(stack[top]=='@'))
            {
                a=pop(); c[j++]=pop();
            }
            push('@');
        }
    }
    c[j]='\0';
    strrev(c);
    printf("%s\n",c);
}
```

```
void main()
{
    post_pre();
}
```

Program 43 : Prefix to Postfix Conversion

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
#include<stdlib.h>
# define MAX 20
char str[MAX],stack[MAX];
int top=-1;
void push(char c)
{
    stack[++top]=c;
}
char pop()
{
    return stack[top--];
}
void pre_post()
{
    int n,i,j=0; char c[20];
    char a,b,op;
    printf("Enter a Prefix expression:");
    scanf("%s", str);
    n=strlen(str);
    for(i=0;i<MAX;i++)
    stack[i]='\0';
    printf("Postfix expression is:");
    for(i=0;i<n;i++)
    {
        if(str[i]=='+'||str[i]=='-'||str[i]=='*'||str[i]=='/')
        {
            push(str[i]);
        }
        else
        {
            c[j++]=str[i];
            while((top!=-1)&&(stack[top]=='@'))
            {
                a=pop(); c[j++]=pop();
            }
            push('@');
```

```
        }
    }
    c[j]='\0';
    printf("%s\n",c);
}
void main()
{
    pre_post();
}
```

Program 44 : Create table datatype and support different operations on it.

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
# define MAX_INPUT 20

typedef struct
{
   enum{INT, FLOAT, STRING} valueType;
   union unionStud
   {
      int intValue;
      float floatValue;
      char stringValue[MAX_INPUT];
   } studData;
} structStudent;

structStudent student[MAX_INPUT][MAX_INPUT];
int rowLabel = 0, columnLabel = 0;

int valueType(char *);
void assignInitialValues();
void displayStudentRecords();
void addStudent(char *);
void addSubject(char *);
int findStudentRowNumber(char *);
int findSubjectColumnNumber(char *);
void insertData(int, int, char *);
void addStudentMarks(char *, char *, char *);
int isStudentMarksAdditionPossible(int);
int isSubjectMarksAdditionPossible(int);
```

```
float calculateStudentTotalMarks(char *);
float calculateSubjectTotalMarks(char *);
float calculateStudentAverageMarks(char *);
float calculateSubjectAverageMarks(char *);
void deleteStudentRecords(char *);
void deleteSubjectRecords(char *);
void ftoa(float, char* , int);

int valueType(char *value)
{
   //char value[MAX_INPUT] = "";
   double temp;
   int n;
   char str[MAX_INPUT] = "";
   double val = 1e-12;

   if (sscanf(value, "%lf", &temp) == 1)
   {
      n = (int)temp; // typecast to int.
      if (fabs(temp - n) / temp > val)
         return 2;   //float
      else
         return 1;  //integer
   }
   else if (sscanf(value, "%s", str) == 1)
      return 3;  //string
}
void assignInitialValues()
{
   int i, j, noOfStudents, noOfSubjects, returnValue;
   char studentName[10], subjectName[10], marks[10];
   for (i = 0; i < 10; i++)
   {
      for(j = 0; j < 10; j++)
     {
         student[i][j].studData.intValue = -1;
         student[i][j].valueType = INT;
      }
   }
   printf("Enter number of students:");
   scanf(" %d", &noOfStudents);
   printf("Number of subjects:");
   scanf(" %d", &noOfSubjects);
```

```
  for (i = 0; i < noOfStudents; i++)
  {
     printf("Enter student %d name:", i+1);
     scanf("%s", studentName);
     strcpy(student[i+1][0].studData.stringValue, studentName);
     student[i+1][0].valueType = STRING;
  }
  for (j = 0; j < noOfSubjects; j++)
  {
     printf("Enter subject %d name:", j+1);
     scanf("%s", subjectName);
     strcpy(student[0][j+1].studData.stringValue, subjectName);
     student[0][j+1].valueType = STRING;
   }
  for (i = 1; i <= noOfStudents; i++)
 {
     for(j = 1; j <= noOfSubjects; j++)
     {
        printf("Enter %s %s marks:",
          student[i][0].studData.stringValue,student[0][j].studData.stringValue);
        scanf("%s", marks);
        returnValue = valueType(marks);
        if(returnValue == 1)
       {
           student[i][j].studData.intValue = atoi(marks);
           student[i][j].valueType = INT;
        }
        else if(returnValue == 2)
        {
           student[i][j].studData.floatValue = atof(marks);
           student[i][j].valueType = FLOAT;
        }
        else if(returnValue == 3)
        {
           strcpy(student[i][j].studData.stringValue, marks);
           student[i][j].valueType = STRING;
        }

     }
  }
  rowLabel = noOfStudents + 1;
  columnLabel = noOfSubjects + 1;
}
```

```
void displayStudentRecords()
{
   int i, j;
   for (i = 0; i < rowLabel; i++)
   {
      for (j = 0; j < columnLabel; j++)
     {
         if (i == 0 && j == 0) printf("\t");
         else if (student[i][j].valueType == INT)
         {
            if (student[i][j].studData.intValue == -1)
                  printf("\t");
            else
                  printf("%d\t", student[i][j].studData.intValue);
         }
         else if (student[i][j].valueType == FLOAT)
                  printf("%.2f\t", student[i][j].studData.floatValue);
         else if (student[i][j].valueType == STRING)
                  printf("%s\t", student[i][j].studData.stringValue);
      }
      printf("\n");
   }
}
void addStudent(char *studentName)
{
   int returnValue = 0;
   if (rowLabel == 0) rowLabel = 1;
          returnValue = valueType(studentName);
   if(returnValue == 1)
   {
      student[rowLabel][0].studData.intValue = atoi(studentName);
      student[rowLabel][0].valueType = INT;
    }
    else if(returnValue == 2)
    {
      student[rowLabel][0].studData.floatValue = atof(studentName);
      student[rowLabel][0].valueType = FLOAT;
     }
     else if(returnValue == 3)
     {
      strcpy(student[rowLabel][0].studData.stringValue, studentName);
      student[rowLabel][0].valueType = STRING;
      }
```

```
   rowLabel++;
}
void addSubject(char *subjectName)
{
   int returnValue = 0;
   if (columnLabel == 0) columnLabel = 1;
   returnValue = valueType(subjectName);
   if(returnValue == 1)
   {
      student[0][columnLabel].studData.intValue = atoi(subjectName);
      student[0][columnLabel].valueType = INT;
   }
   else if(returnValue == 2)
  {
      student[0][columnLabel].studData.floatValue = atof(subjectName);
      student[0][columnLabel].valueType = FLOAT;
   }
   else if(returnValue == 3)
   {
      strcpy(student[0][columnLabel].studData.stringValue, subjectName);
      student[0][columnLabel].valueType = STRING;
   }
   columnLabel++;
}
int findStudentRowNumber(char *studentName)
{
   int i, rowNumber, studentNotFound = -2;
   for (i = 0; i < rowLabel; i++)
  {
      if (student[i][0].valueType = STRING)
      {
         if(strcmp(student[i][0].studData.stringValue, studentName) == 0)
         {
            rowNumber = i;
            return rowNumber;
         }
      }
   }
   return studentNotFound;
}

int findSubjectColumnNumber(char *subjectName)
{
```

```
   int j, columnNumber, subjectNotFound = -2;
   for (j = 0; j < rowLabel; j++)
  {
      if (student[0][j].valueType = STRING)
      {
         if(strcmp(student[0][j].studData.stringValue, subjectName) == 0)
         {
            columnNumber = j;
            return columnNumber;
         }
      }
   }
   return subjectNotFound;
}
void insertData(int rowNumber, int columnNumber, char *marks)
{
   int returnValue;
   returnValue = valueType(marks);
   if(returnValue == 1)
   {
      student[rowNumber][columnNumber].studData.intValue = atoi(marks);
      student[rowNumber][columnNumber].valueType = INT;
   }
   else if(returnValue == 2)
   {
      student[rowNumber][columnNumber].studData.floatValue = atof(marks);
      student[rowNumber][columnNumber].valueType = FLOAT;
   }
  else if(returnValue == 3)
   {
      strcpy(student[rowNumber][columnNumber].studData.stringValue, marks);
      student[rowNumber][columnNumber].valueType = STRING;
   }
}
void addStudentMarks(char *studentName, char *subjectName, char *marks)
{
   int rowNumber, columnNumber, returnValue;
   rowNumber = findStudentRowNumber(studentName);
   if (rowNumber == -2)
   {
      printf("Student %s not found.\n", studentName);
      return;
   }
```

```
   columnNumber = findSubjectColumnNumber(subjectName);
   if (columnNumber == -2)
   {
      printf("Subject %s not found.\n", subjectName);
      return;
   }
   insertData(rowNumber, columnNumber, marks);
}
int isStudentMarksAdditionPossible(int rowNumber)
{
            int j, possible = 1;
            for (j = 1; j < columnLabel; j++)
            {
              if (student[rowNumber][j].valueType == STRING)
              {
                    possible = 0;
                    return possible;
               }
            }
   return possible;
}
int isSubjectMarksAdditionPossible(int columnNumber)
{
   int i, possible = 1;
   for (i = 1; i < rowLabel; i++)
  {
      if (student[i][columnNumber].valueType == STRING)
     {
         possible = 0;
         return possible;
      }
   }
   return possible;
}
float calculateStudentTotalMarks(char *studentName)
{
   int j, rowNumber, possible;
   float sum = 0;
   rowNumber = findStudentRowNumber(studentName);
   if (rowNumber == -2) return (-2);  //Student not found;
   possible = isStudentMarksAdditionPossible(rowNumber);
   if (possible == 1)
   {
```

```
      for (j = 1; j < columnLabel; j++)
     {
      if (student[rowNumber][j].valueType == INT && student[rowNumber][j].studData.intValue != -1)
     {
             if (student[rowNumber][j].valueType == INT)
                    sum = sum + student[rowNumber][j].studData.intValue;
            else if (student[rowNumber][j].valueType == FLOAT)
                    sum = sum + student[rowNumber][j].studData.floatValue;
         }
      }
      return sum;
   }
   else
          return (-1);
}
float calculateSubjectTotalMarks(char *subjectName) {
   int i, columnNumber, possible;
   float sum = 0;
   columnNumber = findSubjectColumnNumber(subjectName);
   if (columnNumber == -2) return (-2);  //Subject not found;
   possible = isSubjectMarksAdditionPossible(columnNumber);
   if (possible == 1) {
      for (i = 1; i < rowLabel; i++) {
         if (student[i][columnNumber].valueType == INT &&
student[i][columnNumber].studData.intValue != -1) {
            if (student[i][columnNumber].valueType == INT) sum = sum +
student[i][columnNumber].studData.intValue;
            else if (student[i][columnNumber].valueType == FLOAT) sum = sum +
student[i][columnNumber].studData.floatValue;
         }
      }
      return sum;
   } else return (-1);
}
float calculateStudentAverageMarks(char *studentName)
{
   float sum, average;
   sum = calculateStudentTotalMarks(studentName);
   if ((int)sum == -1)
          return (-1);
   else if ((int)sum == -2)
          return (-2);
   else
```

```
  {
     average = sum/(columnLabel-1);
     return average;
   }
}
float calculateSubjectAverageMarks(char *subjectName)
{
   float sum, average;
   sum = calculateSubjectTotalMarks(subjectName);
   if ((int)sum == -1)
          return (-1);
   else if ((int)sum == -2)
          return (-2);
   else
  {
     average = sum/(rowLabel-1);
     return average;
   }
}
void deleteStudentRecords(char *studentName)
{
   int i, j, rowNumber;
   char toStrValue[10];
   rowNumber = findStudentRowNumber(studentName);
   if (rowNumber == -2)
  {
     printf("Student %s not found.\n", studentName);
     return;
   }
   for (i = rowNumber; i < rowLabel - 1; i++)
   {
     for (j = 0 ; j < columnLabel ; j++)
     {
        if(student[i+1][j].valueType == INT)
        {
           sprintf(toStrValue, "%d", student[i+1][j].studData.intValue);
           insertData(i, j, toStrValue);    //3rd variable string type
        }
        else if (student[i+1][j].valueType == FLOAT)
       {
           ftoa(student[i+1][j].studData.floatValue, toStrValue, 2);
           insertData(i, j, toStrValue);
        }
```

```
        else if (student[i+1][j].valueType == STRING)
        {
           insertData(i, j, student[i+1][j].studData.stringValue);
        }
      }
   }
   for (j = 0; j < columnLabel; j++)
   {
      insertData(rowLabel-1, j, "-1");
   }
   rowLabel--;
}
void deleteSubjectRecords(char *subjectName)
 {
   int i, j, columnNumber;
   char toStrValue[10];
   columnNumber = findSubjectColumnNumber(subjectName);
   if (columnNumber == -2)
   {
      printf("Subject %s not found.\n", subjectName);
      return;
   }
   for (j = columnNumber; j < columnLabel - 1; j++)
   {
      for (i = 0; i < rowLabel; i++)
     {
        if(student[i][j+1].valueType == INT)
        {
           sprintf(toStrValue, "%d", student[i][j+1].studData.intValue);
           insertData(i, j, toStrValue);    //3rd variable string type
        }
        else if (student[i][j+1].valueType == FLOAT)
       {
           ftoa(student[i][j+1].studData.floatValue, toStrValue, 2);
           insertData(i, j, toStrValue);
        }
        else if (student[i][j+1].valueType == STRING)
       {
           insertData(i, j, student[i][j+1].studData.stringValue);
        }
     }
   }
   for (i = 0; i < rowLabel; i++)
```

```
    {
        insertData(i, columnLabel-1, "-1");
    }
    columnLabel--;
}
void reverse(char* str, int len)
{
    int i = 0, j = len - 1, temp;
    while (i < j)
    {
        temp = str[i];
        str[i] = str[j];
        str[j] = temp;
        i++;
        j--;
    }
}
int intToStr(int x, char str[], int d)
{
    int i = 0;
    while (x)
    {
        str[i++] = (x % 10) + '0';
        x = x / 10;
    }
    while (i < d)
        str[i++] = '0';
    reverse(str, i);
    str[i] = '\0';
    return i;
}
 void ftoa(float n, char* res, int afterpoint)
 {
    int ipart = (int)n;
    float fpart = n - (float)ipart;
    int i = intToStr(ipart, res, 0);
    if (afterpoint != 0)
    {
        res[i] = '.'; // add dot
        fpart = fpart * pow(10, afterpoint);
        intToStr((int)fpart, res + i + 1, afterpoint);
    }
}
```

```
int main()
{
   int choice;
   char studentName[MAX_INPUT], subjectName[MAX_INPUT], marks[MAX_INPUT];
   float sum, average;
   assignInitialValues();
   do
   {
      printf("Result menu:\n");
      printf("1 : Display students records.\n");
      printf("2 : Add student\n");
      printf("3 : Add subject\n");
      printf("4 : Add/Update marks\n");
      printf("5 : Calculate total marks of a student\n");
      printf("6 : Calculate total marks of all students in a subject\n");
      printf("7 : Calculate average marks of a student\n");
      printf("8 : Calculate average marks scored by all students in a subject\n");
      printf("9 : Delete student\n");
      printf("10 : Delete subject\n");
      printf("0 : Exit\n");
      printf("Enter choice(0-10):");
      scanf(" %d", &choice);
      switch (choice) {
         case 1:
            displayStudentRecords();
            break;
         case 2:
            printf("Enter student name:");
            scanf("%s", studentName);
            addStudent(studentName);
            break;
         case 3:
            printf("Enter subject name:");
            scanf("%s", subjectName);
            addSubject(subjectName);
            break;
         case 4:
            printf("Enter student name:");
            scanf("%s", studentName);
            printf("Enter subject name:");
            scanf("%s", subjectName);
            printf("Enter marks:");
```

```
    scanf("%s", marks);
    addStudentMarks(studentName, subjectName, marks);
    break;
  case 5:
    printf("Enter student name:");
    scanf("%s", studentName);
    sum = calculateStudentTotalMarks(studentName);
    if((int)sum == -1) printf("Addition is not possible as some values are non numeric.\n");
    else if((int)sum == -2) printf("Student %s not found.\n", studentName);
    else printf("%s total marks:%.2f\n", studentName, sum);
    break;
  case 6:
    printf("Enter subject name:");
    scanf("%s", subjectName);
    sum = calculateSubjectTotalMarks(subjectName);
    if((int)sum == -1) printf("Addition is not possible as some values are non numeric.\n");
    else if((int)sum == -2) printf("Subject %s not found.\n", subjectName);
    else printf("%s total marks:%.2f\n", studentName, sum);
    break;
  case 7:
    printf("Enter student name:");
    scanf("%s", studentName);
    average = calculateStudentAverageMarks(studentName);
    if((int)average == -1) printf("Addition is not possible as some values are non numeric.\n");
    else if((int)average == -2) printf("Student %s not found.\n", studentName);
    else printf("%s average marks per subject:%.2f\n", studentName, average);
    break;
  case 8:
    printf("Enter subject name:");
    scanf("%s", subjectName);
    average = calculateSubjectAverageMarks(subjectName);
    if((int)average == -1) printf("Addition is not possible as some values are non numeric.\n");
    else if((int)average == -2) printf("Subject %s not found.\n", subjectName);
    else printf("%s average marks per student:%.2f\n", subjectName, average);
    break;
  case 9:
    printf("Enter student name:");
    scanf("%s", studentName);
    deleteStudentRecords(studentName);
    break;
  case 10:
    printf("Enter subject name:");
    scanf("%s", subjectName);
```

```
            deleteSubjectRecords(subjectName);
            break;
        }
    }
    while (choice != 0);
    return 0;
}
```