1 : : ----------TO CHECK STRING IS PALINDROME OR NOT----------

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

// A function to check if a string str is palindrome
void isPalindrome(char str[])
{
   // Start from leftmost and rightmost corners of str
   int l = 0;
   int h = strlen(str) - 1;

   // Keep comparing characters while they are same
   while (h > l)
   {
      if (str[l++] != str[h--])
      {
         printf("%s is Not Palindrome \n", str);
         return;
      }
   }
   printf("%s is palindrome \n", str);
}
int main()
{
   isPalindrome("abba");
   isPalindrome("abbccbba");
   isPalindrome("geeks");
   return 0;
}
```

2 : : ----------TO FIND SUBSTRING FROM A STRING----------

```
#include <stdio.h>


int main()
{
  char string[1000], sub[1000];
  int position, length, c = 0;


  printf("Input a string\n");
  gets(string);


  printf("Enter the position and length of substring\n");
  scanf("%d%d", &position, &length);


  while (c < length) {
    sub[c] = string[position+c-1];
    c++;
  }
  sub[c] = '\0';


  printf("Required substring is \"%s\"\n", sub);


  return 0;
}
```

3 : :----------STACK USING ARRAY----------

```
#include<stdio.h>

int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);
int main()
{
   //clrscr();
   top=-1;
   printf("\n Enter the size of STACK[MAX=100]:");
   scanf("%d",&n);
   printf("\n\t STACK OPERATIONS USING ARRAY");
   printf("\n\t--------------------------------");
   printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
   do
   {
      printf("\n Enter the Choice:");
      scanf("%d",&choice);
      switch(choice)
      {
         case 1:
         {
            push();
            break;
         }
         case 2:
         {
            pop();
            break;
```

```
        }
        case 3:
        {
           display();
           break;
        }
        case 4:
        {
           printf("\n\t EXIT POINT ");
           break;
        }
        default:
        {
           printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
        }
      }
   }
   while(choice!=4);
   return 0;
}
void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");

   }
   else
   {
      printf(" Enter a value to be pushed:");
```

```
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
   {
      printf("\n\t The popped elements is %d",stack[top]);
      top--;
   }
}
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK \n");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
      printf("\n Press Next Choice");
   }
   else
   {
      printf("\n The STACK is empty");}
   }
```

4 : : ----------QUEUE  USING ARRAY----------

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define n 5
int main()
{
   int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
   //clrscr();
   printf("Queue using Array");
   printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
   while(ch)
   {
      printf("\nEnter the Choice:");
      scanf("%d",&ch);
      switch(ch)
      {
      case 1:
         if(rear==x)
            printf("\n Queue is Full");
         else
         {
            printf("\n Enter no %d:",j++);
            scanf("%d",&queue[rear++]);
         }
         break;
      case 2:
         if(front==rear)
         {
            printf("\n Queue is empty");
```

```
        }
        else
        {
            printf("\n Deleted Element is %d",queue[front++]);
            x++;
        }
        break;
    case 3:
        printf("\n Queue Elements are:\n ");
        if(front==rear)
            printf("\n Queue is Empty");
        else
        {
            for(i=front; i<rear; i++)
            {
                printf("%d",queue[i]);
                printf("\n");
            }
            break;
        case 4:
            exit(0);
        default:
            printf("Wrong Choice: please see the options");
        }
    }
  }
  return 0;
}
```

5:----------BUBBLE SORT----------

```
#include<stdio.h>
int main()
{
   int a[10],i,j,temp,n;
   //clear();
   printf("\n Enter the max no.of Elements to Sort: \n");
   scanf("%d",&n);
   printf("\n Enter the Elements : \n");
   for(i=0; i<n; i++)
   {
      scanf("%d",&a[i]);
   }
   for(i=0; i<n; i++)
      for(j=i+1; j<n; j++)
      {
         if(a[i]>a[j])
         {
            temp=a[i];
            a[i]=a[j];
            a[j]=temp;
         }
      }
   for(i=0; i<n; i++)
   {
      printf("%d\t",a[i]);
   }
   return 0;
}
```

6:----------INSERTION SORT----------

```
#include<stdio.h>
void InsertionSort(int a[], int n){
   int j, p;
   int tmp;
   for(p = 1; p < n; p++)
   {
      tmp = a[p];
      for(j = p; j > 0 && a[j-1] > tmp; j--)
         a[j] = a[j-1];
      a[j] = tmp;
   }
}
int main()
{
   int i, n, a[10];
   printf("Enter the number of elements :: ");
   scanf("%d",&n);
   printf("Enter the elements :: ");
   for(i = 0; i < n; i++)
   {
      scanf("%d",&a[i]);
   }
   InsertionSort(a,n);
   printf("The sorted elements are ::  ");
   for(i = 0; i < n; i++)
      printf("%d  ",a[i]);
   printf("\n");
   return 0;
}
```

```
7:----------------------------------------------------QUICK SORT----------------------------------------

#include <stdio.h>

#define MAX 10


void swap(int *m,int *n)

{

  int temp;

  temp = *m;

  *m = *n;

  *n = temp;

}

int get_key_position(int x,int y )

{

  return((x+y) /2);

}


// Function for Quick Sort

void quicksort(int list[],int m,int n)

{

  int key,i,j,k;

  if( m < n)

  {

    k = get_key_position(m,n);

    swap(&list[m],&list[k]);

    key = list[m];

    i = m+1;

    j = n;

    while(i <= j)

    {

      while((i <= n) && (list[i] <= key))
```

```
            i++;
        while((j >= m) && (list[j] > key))
                    j--;
            if( i < j)
                    swap(&list[i],&list[j]);
     }
     swap(&list[m],&list[j]);
     quicksort(list,m,j-1);
     quicksort(list,j+1,n);
  }
}

// Function to read the data
void read_data(int list[],int n)
{
  int j;
  printf("\n\nEnter the elements:\n");
  for(j=0;j<n;j++)
     scanf("%d",&list[j]);
}

// Function to print the data
void print_data(int list[],int n)
{
  int j;
  for(j=0;j<n;j++)
     printf("%d\t",list[j]);
}

int main()
```

```
{
  int list[MAX], num;
  //clrscr();
  printf("\n***** Enter the number of elements Maximum [10] *****\n");
  scanf("%d",&num);
  read_data(list,num);
  printf("\n\nElements in the list before sorting are:\n");
  print_data(list,num);
  quicksort(list,0,num-1);
  printf("\n\nElements in the list after sorting are:\n");
  print_data(list,num);
  return 0;
}
```

8:----------SELECTION SORT----------

```
#include <stdio.h>
void selection_sort();
int a[30], n;
int main(){
   int i;
   printf("\nEnter size of an array: ");
   scanf("%d", &n);
   printf("\nEnter elements of an array:\n");
   for(i=0; i<n; i++)
      scanf("%d", &a[i]);
   selection_sort();
   printf("\n\nAfter sorting:\n");
   for(i=0; i<n; i++)
      printf("\n%d", a[i]);
   return 0;
}
void selection_sort(){
   int i, j, min, temp;
   for (i=0; i<n; i++) {
      min = i;
      for (j=i+1; j<n; j++){
         if (a[j] < a[min])
            min = j;
      }
      temp = a[i];
      a[i] = a[min];
      a[min] = temp;
   }
}
```

9:----------LINEAR SEARCH----------

```
#include <stdio.h>

int main()

{

   int a[10], i, item;

   printf("\nEnter SEVEN elements of an array:\n");

   for (i=0; i<=6; i++)

      scanf("%d", &a[i]);

   printf("\nEnter item to search: ");

   scanf("%d", &item);

   for (i=0; i<=9; i++)

      if (item == a[i])

      {

         printf("\nItem found at location %d", i+1);

         break;

      }

   if (i > 9)

      printf("\nItem does not exist.");

   return 0;

}
```

10:----------BINARY SEARCH----------

```
#include <stdio.h>
#define MAX_LEN 10


/* Non-Recursive function*/
void b_search_nonrecursive(int l[],int num,int ele)
{
  int l1,i,j, flag = 0;
  l1 = 0;
  i = num-1;
  while(l1 <= i)
  {
    j = (l1+i)/2;
    if( l[j] == ele)
    {
   printf("\nThe element %d is present at position %d in list\n",ele,j);
       flag =1;
       break;
    }
    else
      if(l[j] < ele)
        l1 = j+1;
      else
        i = j-1;
  }
  if( flag == 0)
  printf("\nThe element %d is not present in the list\n",ele);
}


/* Recursive function*/
```

```
int b_search_recursive(int l[],int arrayStart,int arrayEnd,int a)
{
  int m,pos;
  if (arrayStart<=arrayEnd)
  {
    m=(arrayStart+arrayEnd)/2;
    if (l[m]==a)
      return m;
    else if (a<l[m])
      return b_search_recursive(l,arrayStart,m-1,a);
    else
      return b_search_recursive(l,m+1,arrayEnd,a);
   }
   return -1;
}

void read_list(int l[],int n)
{
   int i;
   printf("\nEnter the elements:\n");
   for(i=0;i<n;i++)
       scanf("%d",&l[i]);
}

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

```
/*main function*/
main()
{
  int l[MAX_LEN], num, ele,f,l1,a;
  int ch,pos;

  //clrscr();

  printf("=======================================================");
  printf("\n\t\t\tMENU");
  printf("\n=======================================================");
  printf("\n[1] Binary Search using Recursion method");
  printf("\n[2] Binary Search using Non-Recursion method");
  printf("\n\nEnter your Choice:");
  scanf("%d",&ch);

  if(ch<=2 & ch>0)
  {
    printf("\nEnter the number of elements : ");
    scanf("%d",&num);
    read_list(l,num);
    printf("\nElements present in the list are:\n\n");
    print_list(l,num);
    printf("\n\nEnter the element you want to search:\n\n");
    scanf("%d",&ele);

  switch(ch)
  {
```

```
    case 1:printf("\nRecursive method:\n");
       pos=b_search_recursive(l,0,num,ele);
       if(pos==-1)
       {
       printf("Element is not found");
       }
       else
       {
       printf("Element is found at %d position",pos);
       }
       //getch();
       break;

    case 2:printf("\nNon-Recursive method:\n");
       b_search_nonrecursive(l,num,ele);
       //getch();
       break;
    }
  }
//getch();
}
```

11:----------LINKED LISTS OPERATIONS----------

```
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
   int e;
   Position next;
};

void Insert(int x, List l, Position p)
{
   Position TmpCell;
   TmpCell = (struct Node*) malloc(sizeof(struct Node));
   if(TmpCell == NULL)
      printf("Memory out of space\n");
   else
   {
      TmpCell->e = x;
      TmpCell->next = p->next;
      p->next = TmpCell;
   }
}

int isLast(Position p)
{
```

```
    return (p->next == NULL);
}

Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != NULL && p->next->e != x)
        p = p->next;
    return p;
}

void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);

    if(!isLast(p))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
    else
        printf("Element does not exist!!!\n");
}

void Display(List l)
{
    printf("The list element are :: ");
    Position p = l->next;
```

```
   while(p != NULL)
   {
      printf("%d -> ", p->e);
      p = p->next;
   }
}

void Merge(List l, List l1)
{
   int i, n, x, j;
   Position p;
   printf("Enter the number of elements to be merged :: ");
   scanf("%d",&n);

   for(i = 1; i <= n; i++)
   {
      p = l1;
      scanf("%d", &x);
      for(j = 1; j < i; j++)
         p = p->next;
      Insert(x, l1, p);
   }
   printf("The new List :: ");
   Display(l1);
   printf("The merged List ::");
   p = l;
   while(p->next != NULL)
   {
      p = p->next;
   }
```

```
    p->next = l1->next;
    Display(l);
}



int main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = NULL;
    List p = l;
    printf("LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. MERGE\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            p = l;
            printf("Enter the element to be inserted :: ");
            scanf("%d",&x);
            printf("Enter the position of the element :: ");
            scanf("%d",&pos);

            for(i = 1; i < pos; i++)
            {
                p = p->next;
            }
```

```
            Insert(x,l,p);
            break;

        case 2:
            p = l;
            printf("Enter the element to be deleted :: ");
            scanf("%d",&x);
            Delete(x,p);
            break;

        case 3:
            l1 = (struct Node *) malloc(sizeof(struct Node));
            l1->next = NULL;
            Merge(l, l1);
            break;

        case 4:
            Display(l);
            break;
        }
    }
    while(ch<5);
    return 0;
}
```

12:----------STACK USING  LINKED LISTS----------

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

struct Node
{
  int data;
  struct Node *next;
}*top = NULL;

void push(int);
void pop();
void display();

int main()
{
  int choice, value;

  printf("\n:: Stack using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Push\n2. Pop\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d",&choice);
    switch(choice){
          case 1: printf("Enter the value to be insert: ");
                  scanf("%d", &value);
                  push(value);
                  break;
```

```
            case 2: pop(); break;
            case 3: display(); break;
            case 4: exit(0);
            default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
  return 0;
}
void push(int value)
{
  struct Node *newNode;
  newNode = (struct Node*)malloc(sizeof(struct Node));
  newNode->data = value;
  if(top == NULL)
    newNode->next = NULL;
  else
    newNode->next = top;
  top = newNode;
  printf("\nInsertion is Success!!!\n");
}
void pop()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    printf("\nDeleted element: %d", temp->data);
    top = temp->next;
    free(temp);
  }
```

```
}
void display()
{
  if(top == NULL)
    printf("\nStack is Empty!!!\n");
  else{
    struct Node *temp = top;
    while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
    }
    printf("%d--->NULL",temp->data);
  }
}
```

```
13:----------------------------------------------------QUEUES USING LINKED LISTS---------------------------------

#include<stdio.h>

#include<conio.h>

#include<stdlib.h>


struct Node

{

  int data;

  struct Node *next;

}*front = NULL,*rear = NULL;




void insert(int value)

{

  struct Node *newNode;

  newNode = (struct Node*)malloc(sizeof(struct Node));

  newNode->data = value;

  newNode -> next = NULL;

  if(front == NULL)

    front = rear = newNode;

  else{

    rear -> next = newNode;

    rear = newNode;

  }

  printf("\nInsertion is Success!!!\n");

}

void deleteo()

{

  if(front == NULL)
```

```
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    front = front -> next;
    printf("\nDeleted element: %d\n", temp->data);
    free(temp);
  }
}
void display()
{
  if(front == NULL)
    printf("\nQueue is Empty!!!\n");
  else{
    struct Node *temp = front;
    while(temp->next != NULL){
        printf("%d--->",temp->data);
        temp = temp -> next;
    }
    printf("%d--->NULL\n",temp->data);
  }
}
int main()
{
  int choice, value;

  printf("\n:: Queue Implementation using Linked List ::\n");
  while(1){
    printf("\n****** MENU ******\n");
    printf("1. Insert\n2. Delete\n3. Display\n4. Exit\n");
    printf("Enter your choice: ");
```

```
    scanf("%d",&choice);
    switch(choice){
        case 1: printf("Enter the value to be insert: ");
                scanf("%d", &value);
                insert(value);
                break;
        case 2: deleteo(); break;
        case 3: display(); break;
        case 4: exit(0);
        default: printf("\nWrong selection!!! Please try again!!!\n");
    }
  }
  return 0;
}
```