BCA4B06- Programming Laboratory II: Data Structures & RDBMS

1. Sort a given list of strings

2. Reverse a string using pointers.

3. Implement Pattern matching algorithm.

4. Search an element in the 2-dimensional array

5. Append 2 arrays

6. Merge two sorted array into one sorted array.

7. Search an element in the array using iterative binary search.

8. Search an element in the array using recursive binary search.

9. Implement sparse matrix

10. Implement polynomial using arrays

11. Implement singly linked list of integers.

12. Delete a given element from a singly linked list

13. Sort a singly linked list.

14. Delete an element from a singly linked list

15. Implement a doubly linked list of integers

16. Implement a circular linked list.

17. Implement polynomial using linked list

18. Addition of 2 polynomials

19. Implement Stack using array

20. Implement Stack using linked list

21. Infix expression into its postfix expression

22. Implement Queue using array

23. Implement Queue using linked list

24. Implement a binary search tree of characters.

25. Traverse a binary search tree non recursively in preorder

26. Traverse a binary search tree non recursively in inorder

27. Traverse a binary search tree non recursively in postorder

28. Traverse a binary search tree recursively in preorder

29. Traverse a binary search tree recursively inorder

30. Traverse a binary search tree recursively postorder.

31. Delete an element from a binary search tree.

32. Search an element in a binary search tree

33. Implement linear sort

34. Implement bubble sort

35. Implement exchange sort

36. Implement selection sort.

37. Implement insertion sort.

38. Implement quick sort.

39. Implement merge sort.

40. Implement heap sort

1. C program to read N names, store them in the form of an array and sort them in alphabetical order.

```
#include <stdio.h>

#include <string.h>

void main()

{

   char name[10][8], tname[10][8], temp[8];

    int i, j, n;

    printf("Enter the value of n \n");

    scanf("%d", &n);

    printf("Enter %d names \n", n);

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

    {

        scanf("%s", name[i]);

        strcpy(tname[i], name[i]);

    }

    for (i = 0; i < n - 1 ; i++)

    {

        for (j = i + 1; j < n; j++)

        {

            if (strcmp(name[i], name[j]) > 0)

            {

                strcpy(temp, name[i]);

                strcpy(name[i], name[j]);

                strcpy(name[j], temp);

            }

        }

    }

    printf("\n----------------------------------------\n");

    printf("Input NamestSorted names\n");

    printf("------------------------------------------\n");
    for (i = 0; i < n; i++)
    {
        printf("%s\t\t%s\n", tname[i], name[i]);
    }
    printf("------------------------------------------\n");
}
```

2. C program to reverse a string using pointers

```
#include<stdio.h>
int string_length(char*);
void reverse(char*);
main()
{
   char s[100];
   printf("Enter a string\n");
   gets(s);
   reverse(s);
   printf("Reverse of the string is \"%s\".\n", s);
   return 0;
}
void reverse(char *s)
{
   int length, c;
   char *begin, *end, temp;
   length = string_length(s);
   begin  = s;
   end    = s;
   for (c = 0; c < length - 1; c++)
      end++;
   for (c = 0; c < length/2; c++)
   {
      temp   = *end;
      *end   = *begin;
      *begin = temp;
      begin++;
      end--;
   }
}
```

```
int string_length(char *pointer)
{
   int c = 0;
   while( *(pointer + c) != '\0' )
      c++;
   return c;
}
```

3.Implement Pattern matching algorithm.

```
#include <stdio.h>
#include <string.h>
int match(char [], char []);
int main()
 {
  char a[100], b[100];
  int position;
  printf("Enter some text\n");
  gets(a);
  printf("Enter a string to find\n");
  gets(b);
  position = match(a, b);
  if (position != -1) {
    printf("Found at location: %d\n", position + 1);
  }
  else
  {
    printf("Not found.\n");
   }
  return 0;
}
int match(char text[], char pattern[])
{
  int c, d, e, text_length, pattern_length, position = -1;
  text_length    = strlen(text);
  pattern_length = strlen(pattern);
  if (pattern_length > text_length)
  {
     return -1;
   }
  for (c = 0; c <= text_length - pattern_length; c++)
{
    position = e = c;
    for (d = 0; d < pattern_length; d++)
 {
      if (pattern[d] == text[e])
     {
        e++;
      }
      else
      {
        break;
      }
    }
    if (d == pattern_length)
    {
      return position;
    }
  }
  return -1;
}
```

4. Search an element in the 2-dimensional array

```
#include <stdio.h>
void main()
{
int i,j,item,loc=0,loc1=0;
int a[2][2];
printf("\n\tThis Program is Used To seaech an element in 2Dimensional Array
using Linear Search\n");
printf("\n\tEneter The Value Of Array:");
for(i=1;i<=2;i++)
{
for(j=1;j<=2;j++)
{
scanf("%d",&a[i][j]);
}
}
printf("\n\tEneter The Value To Be Serched:");
scanf("%d",&item);
for(i=1;i<=2;i++)
{
for(j=1;j<=2;j++)
{
if(item==a[i][j])
{
loc=i;
loc1=j;
break;
}
}
}
printf("\n\tThe Item is at %d Row And %d Coloumn.",loc,loc1);
printf("\n\n\t\tSearch Completed.");

getch();

}
```

5. Append 2 arrays

```
#include<stdio.h>

int main()
{
        int aSize, bSize, mSize, i, j;
        int a[10], b[10], Merged[20];
        printf("\n Please Enter the First Array Size  :  ");
        scanf("%d", &aSize);
        printf("\nPlease Enter the First Array Elements :  ");
        for(i = 0; i < aSize; i++)
        {
        scanf("%d", &a[i]);
        }
        printf("\n Please Enter the Second Array Size  :  ");
        scanf("%d", &bSize);
        printf("\nPlease Enter the Second Array Elements  :  ");
        for(i = 0; i < bSize; i++)
        {
        scanf("%d", &b[i]);
        }

        for(i = 0; i < aSize; i++)
        {
        Merged[i] = a[i];
        }

        mSize = aSize + bSize;

        for(i = 0, j = aSize; j < mSize && i < bSize; i++, j++)
        {
                Merged[j] = b[i];
        }

        printf("\n a[%d] Array Elements After Merging \n", mSize);
        for(i = 0; i < mSize; i++)
        {
        printf(" %d \t ",Merged[i]);
        }

        return 0;
}
```

6. Search an element in the array using binary search.

```
#include <stdio.h>
int main()
{
   int c, first, last, middle, n, search, array[100];

   printf("Enter number of elements\n");
   scanf("%d",&n);

   printf("Enter %d integers\n", n);

   for (c = 0; c < n; c++)
      scanf("%d",&array[c]);

   printf("Enter value to find\n");
   scanf("%d", &search);

   first = 0;
   last = n - 1;
   middle = (first+last)/2;

   while (first <= last) {
      if (array[middle] < search)
         first = middle + 1;
      else if (array[middle] == search) {
         printf("%d found at location %d.\n", search, middle+1);
         break;
      }
      else
         last = middle - 1;
      middle = (first + last)/2;
   }
   if (first > last)
      printf("Not found! %d isn't present in the list.\n", search);

   return 0;
}
```

7. C program to implement recursive Binary Search

```c
#include <stdio.h>
int binarySearch(int arr[], int l, int r, int x)
{
   if (r >= l)
   {
        int mid = l + (r - l)/2;
        if (arr[mid] == x)
            return mid;
        if (arr[mid] > x)
            return binarySearch(arr, l, mid-1, x);
        return binarySearch(arr, mid+1, r, x);
   }
   return -1;
}
int main(void)
{
   int arr[] = {2, 3, 4, 10, 40};
   int n = sizeof(arr)/ sizeof(arr[0]);
   int x = 10;
   int result = binarySearch(arr, 0, n-1, x);
   (result == -1)? printf("Element is not present in array")
                 : printf("Element is present at index %d", result);
   return 0;
}
```

8. Implement sparse matrix

```
#include <stdio.h>

#define MAX 20

void read_matrix(int a[10][10], int row, int column);

void print_sparse(int b[MAX][3]);

void create_sparse(int a[10][10], int row, int column, int b[MAX][3]);

int main()

{

    int a[10][10], b[MAX][3], row, column;

    printf("\nEnter the size of matrix (rows, columns): ");

    scanf("%d%d", &row, &column);

    read_matrix(a, row, column);

    create_sparse(a, row, column, b);

    print_sparse(b);

    return 0;

}

void read_matrix(int a[10][10], int row, int column)

{

    int i, j;

    printf("\nEnter elements of matrix\n");

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

    {

        for (j = 0; j < column; j++)

        {

            printf("[%d][%d]: ", i, j);

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

        }

    }

}

void create_sparse(int a[10][10], int row, int column, int b[MAX][3])
```

```
{
    int i, j, k;
    k = 1;
    b[0][0] = row;
    b[0][1] = column;
    for (i = 0; i < row; i++)
    {
        for (j = 0; j < column; j++)
        {
            if (a[i][j] != 0)
            {
                b[k][0] = i;
                b[k][1] = j;
                b[k][2] = a[i][j];
                k++;
            }
        }
        b[0][2] = k - 1;
    }
}
void print_sparse(int b[MAX][3])
{
    int i, column;
    column = b[0][2];
    printf("\nSparse form - list of 3 triples\n\n");
    for (i = 0; i <= column; i++)
    {
        printf("%d\t%d\t%d\n", b[i][0], b[i][1], b[i][2]);
    }
}
```

9. Implement polynomial using arrays

```
#include <stdio.h>
int main()
{
    int a[27],b[27],c[54],m,n,i,j,z,y=0,t,s=0;
    printf ("How many terms you want to add in the 1st polynomial ?? : ");
    scanf ("%d",&n);
    printf ("Enter 1st polynomial : \n");
    for (i=0;i<n*3;i=i+3)
    {
        printf ("Enter coefficient : ");
        scanf("%d",&a[i]);
        printf ("Enter power of x : ");
        scanf("%d",&a[i+1]);
        printf ("Enter power of y : ");
        scanf("%d",&a[i+2]);
    }
    printf ("1st polynomial is : ");
    for (i=0;i<n*3;i=i+3)
    {
        printf ("(%dx^%dy^%d) + ",a[i],a[i+1],a[i+2]);
    }
    printf (" 0 \n");
    printf ("How many terms you want to add in the 2nd polynomial ?? : ");
    scanf ("%d",&m);

    printf ("Enter 2nd polynomial : \n");
    for (i=0;i<m*3;i=i+3)
    {
        printf ("Enter coefficient : ");
        scanf("%d",&b[i]);

        printf ("Enter power of x : ");

        scanf("%d",&b[i+1]);

        printf ("Enter power of y : ");

        scanf("%d",&b[i+2]);

    }

    printf ("2nd polynomial is : ");

    for (i=0;i<m*3;i=i+3)

    {

        printf ("(%dx^%dy^%d) + ",b[i],b[i+1],b[i+2]);

    }

    printf (" 0\n");

    printf ("Enter 1 to add  : ");

    scanf("%d",&z);

    switch (z)

    {

        case 1:

        for (i=0;i<n*3;i++)

        {c[i]=a[i];}

        for (i=n*3,j=0;i<(n+m)*3,j<m*3;i++,j++)

        {c[i]=b[j];}

        for (i=0;i<(m+n)*3;i=i+3)

    {

        printf ("(%dx^%dy^%d) + ",c[i],c[i+1],c[i+2]);

    }

    printf (" 0\n");

    }

    for (i=1;i<(m+n)*3;i=i+3)

    {

        for (j=4;j<(m+n)*3;j=j+3)
```

```
        {
            if (c[i]==c[j])
            { if(c[i+1]==c[j+1])
            {
                c[i-1]=c[i-1]+c[j-1];
                c[j-1]=0;
            }
            }
        }
    }
    printf ("ADDITION \n");
    for (i=0;i<(m+n)*3;i=i+3)
    if (c[i]!=0)
    {
        printf ("(%dx^%dy^%d) + ",c[i],c[i+1],c[i+2]);
    }
    else
    printf (" ");
    printf (" 0 \n");
}
```

10.Implement singly linked list

```
#include <stdio.h>
#include <malloc.h>
#include <stdlib.h>
struct node
 {
  int value;
  struct node *next;
 };
void insert();
void display();
void delete();
int count();
typedef struct node DATA_NODE;
DATA_NODE *head_node, *first_node, *temp_node = 0, *prev_node, next_node;
int data;
int main()
{
  int option = 0;
  printf("Singly Linked List Example - All Operations\n");
  while (option < 5)
  {
    printf("\nOptions\n");
    printf("1 : Insert into Linked List \n");
    printf("2 : Delete from Linked List \n");
    printf("3 : Display Linked List\n");
    printf("4 : Count Linked List\n");
    printf("Others : Exit()\n");
    printf("Enter your option:");
    scanf("%d", &option);
    switch (option)
   {
      case 1:     insert();
                break;
      case 2:        delete();
                 break;
      case 3:        display();
                   break;
      case 4:        count();
                    break;
      default:        break;
    }
  }
  return 0;
}
void insert()
{
  printf("\nEnter Element for Insert Linked List : \n");
  scanf("%d", &data);
  temp_node = (DATA_NODE *) malloc(sizeof (DATA_NODE));
  temp_node->value = data;

  if (first_node == 0)
  {
    first_node = temp_node;
  }
  else
  {
    head_node->next = temp_node;
  }
  temp_node->next = 0;
  head_node = temp_node;
  fflush(stdin);
}
void delete()
{
  int countvalue, pos, i = 0;
  countvalue = count();
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  printf("\nEnter Position for Delete Element : \n");
  scanf("%d", &pos);
  if (pos > 0 && pos <= countvalue)
  {
    if (pos == 1)
    {
      temp_node = temp_node -> next;
      first_node = temp_node;
      printf("\nDeleted Successfully \n\n");
    }
    else
    {
      while (temp_node != 0)
      {
        if (i == (pos - 1))
        {
            prev_node->next = temp_node->next;
             if(i == (countvalue - 1))
             {
                   head_node = prev_node;
             }
            printf("\nDeleted Successfully \n\n");
             break;
        }
        else
        {
           i++;
           prev_node = temp_node;
           temp_node = temp_node -> next;
        }
      }
    }
  }
  else
    printf("\nInvalid Position \n\n");
}
void display()
{
  int count = 0;
  temp_node = first_node;
  printf("\nDisplay Linked List : \n");
  while (temp_node != 0)
  {
    printf("# %d # ", temp_node->value);
    count++;
    temp_node = temp_node -> next;
   }
  printf("\nNo Of Items In Linked List : %d\n", count);
}
int count()
```

```
{
  int count = 0;
  temp_node = first_node;
  while (temp_node != 0)
  {
    count++;
    temp_node = temp_node -> next;
   }
  printf("\nNo Of Items In Linked List : %d\n", count);
  return count;
}
```

11.Implement a doubly linked list of integers

```
#include <stdio.h>

#include <stdlib.h>

typedef struct Node

{

    int data;

    struct Node *next;

    struct Node *prev;

} Node;

void insert(Node *current, int data);

void delete(Node *current, int data);

void print(Node *current);

int find(Node *current, int data);

void insert(Node *current, int data)

{

    while(current->next != NULL)

    {

        current = current->next;

    }

    current->next = (Node *)malloc(sizeof(Node));

    (current->next)->prev = current;

    current = current->next;

    current->data = data;

    current->next = NULL;

}


void delete(Node *current, int data)

{


    while (current->next != NULL && (current->next)->data != data)
    {
        current = current->next;
    }
    if(current->next == NULL)
    {
        printf("\nElement %d is not present in the list\n", data);
        return;
    }
    Node *tmp = current->next;
    if(tmp->next == NULL)
    {
        current->next = NULL;
    } else
    {
        current->next = tmp->next;
        (current->next)->prev = tmp->prev;
    }
    tmp->prev = current;
    free(tmp);
    return;
}
void print(Node *current)
{
    while(current != NULL)
    {
        printf("%d ", current->data);
        current = current->next;
    }
}
```

```
int find(Node *current, int data)
{
    current = current->next;
    while(current != NULL)
    {
        if(current->data == data)
        {
            return 1;
        }
        current = current->next;
    }
    return 0;
}

int main()
{
    Node *head = (Node *)malloc(sizeof(Node));
    head->next = NULL;
    head->prev = NULL;
    int data = 0;
    int usr_input = 0;
    while(1)
    {
        printf("0. Exit\n");
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Print\n");
        printf("4. Find\n");
        scanf("%d", &usr_input);
        if( usr_input == 0)
        {
            exit(0);

        }
        else if(usr_input == 1)
        {
            printf("\nEnter an element you want to insert: ");
            scanf("%d", &data);
            insert(head, data);

        }
        else if(usr_input == 2)
        {
            printf("\nEnter an element you want to delete: ");
            scanf("%d", &data);
            delete(head, data);
        }
        else if(usr_input == 3)
        {
            printf("The list is ");
            print(head->next);
            printf("\n\n");
        }
        else if(usr_input == 4)
        {
            printf("\nEnter an element you want to find: ");
            scanf("%d", &data);
            int is_found = find(head, data);
            if (is_found)
            {
                printf("\nElement is found\n\n");
            }
            else
            {
                printf("\nElement is NOT found\n\n");
            }
        }
    }
    return 0;
}
```

12.Implement a circular linked list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node
{
    int info;
    struct Node *next;
}node;
node *front=NULL,*rear=NULL,*temp;
void create();
void del();
void display();
int main()
{
    int chc;
    do
    {
     printf("\nMenu\n\t 1 to create the element : ");
    printf("\n\t 2 to delete the element : ");
    printf("\n\t 3 to display the queue : ");
    printf("\n\t 4 to exit from main : ");
    printf("\nEnter your choice : ");
    scanf("%d",&chc);
        switch(chc)
        {
            case 1:             create();
                            break;

             case 2:             del();
                              break;
             case 3:             display();
                            break;

             case 4:             return 1;

            default:
                 printf("\nInvalid choice :");
         }
    }while(1);

    return 0;
}
void create()
{
    node *newnode;
    newnode=(node*)malloc(sizeof(node));
    printf("\nEnter the node value : ");
    scanf("%d",&newnode->info);
    newnode->next=NULL;
    if(rear==NULL)
    front=rear=newnode;
    else
    {
        rear->next=newnode;
        rear=newnode;
    }
    rear->next=front;
}
void del()
```

```
{
    temp=front;
    if(front==NULL)
        printf("\nUnderflow :");
    else
    {
        if(front==rear)
        {
            printf("\n%d",front->info);
            front=rear=NULL;
        }
        else
        {
            printf("\n%d",front->info);
            front=front->next;
            rear->next=front;
        }
    temp->next=NULL;
    free(temp);
    }
}
void display()
{
    temp=front;
    if(front==NULL)
        printf("\nEmpty");
    else
    {
        printf("\n");
        for(;temp!=rear;temp=temp->next)
```

```
            printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next);
            printf("\n%d address=%u next=%u\t",temp->info,temp,temp->next);
    }
}
```

13.Implement polynomial using linked list

```
#include<stdio.h>

#include<malloc.h>

#include<conio.h>

struct link

{

       int coeff;

       int pow;

       struct link *next;

};

struct link *poly1=NULL,*poly2=NULL,*poly=NULL;

void create(struct link *node)

{

 char ch;

 do

 {

  printf("\n enter coeff:");

  scanf("%d",&node->coeff);

  printf("\n enter power:");

  scanf("%d",&node->pow);

  node->next=(struct link*)malloc(sizeof(struct link));

  node=node->next;

  node->next=NULL;

  printf("\n continue(y/n):");

  ch=getch();

 }

 while(ch=='y' || ch=='Y');

}

void show(struct link *node)

{
 while(node->next!=NULL)
 {
  printf("%dx^%d",node->coeff,node->pow);
  node=node->next;
  if(node->next!=NULL)
   printf("+");
 }
}
void polyadd(struct link *poly1,struct link *poly2,struct link *poly)
{
     while(poly1->next &&  poly2->next)
     {
      if(poly1->pow>poly2->pow)
      {
       poly->pow=poly1->pow;
       poly->coeff=poly1->coeff;
       poly1=poly1->next;
       }
      else if(poly1->pow<poly2->pow)
      {
       poly->pow=poly2->pow;
       poly->coeff=poly2->coeff;
       poly2=poly2->next;
       }
      else
      {
       poly->pow=poly1->pow;
       poly->coeff=poly1->coeff+poly2->coeff;
       poly1=poly1->next;
       poly2=poly2->next;
       }
      poly->next=(struct link *)malloc(sizeof(struct link));
      poly=poly->next;
      poly->next=NULL;
     }
     while(poly1->next || poly2->next)
     {
      if(poly1->next)
      {
       poly->pow=poly1->pow;
       poly->coeff=poly1->coeff;
       poly1=poly1->next;
       }
      if(poly2->next)
      {
       poly->pow=poly2->pow;
       poly->coeff=poly2->coeff;
       poly2=poly2->next;
       }
       poly->next=(struct link *)malloc(sizeof(struct link));
       poly=poly->next;
       poly->next=NULL;
       }
}
main()
{
      char ch;
      do{
      poly1=(struct link *)malloc(sizeof(struct link));
      poly2=(struct link *)malloc(sizeof(struct link));
```

```
    poly=(struct link *)malloc(sizeof(struct link));
    printf("\nenter 1st number:");
    create(poly1);
    printf("\nenter 2nd number:");
    create(poly2);
    printf("\n1st Number:");
    show(poly1);
    printf("\n2nd Number:");
    show(poly2);
    polyadd(poly1,poly2,poly);
    printf("\nAdded polynomial:");
    show(poly);
    printf("\n add two more numbers:");
    ch=getch();
    }
    while(ch=='y' || ch=='Y');
}
```

14.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");
    }
}
```

15.Stack using linked list

```
#include <stdio.h>

#include <stdlib.h>

struct node

{    int info;

    struct node *ptr;

}*top,*top1,*temp;

int topelement();

void push(int data);

void pop();

void empty();

void display();

void destroy();

void stack_count();

void create();

int count = 0;

void main()

{

    int no, ch, e;

    printf("\n 1 - Push");

    printf("\n 2 - Pop");

    printf("\n 3 - Top");

    printf("\n 4 - Empty");

    printf("\n 5 - Exit");

    printf("\n 6 - Dipslay");

    printf("\n 7 - Stack Count");

    printf("\n 8 - Destroy stack");

    create();

    while (1)

    {
```

```
printf("\n Enter choice : ");
scanf("%d", &ch);
switch (ch)
{
case 1:   printf("Enter data : ");
            scanf("%d", &no);
            push(no);
            break;
case 2:            pop();
              break;
case 3:    if (top == NULL)
           printf("No elements in stack");
             else
            {
                e = topelement();
                 printf("\n Top element : %d", e);
            }
             break;
case 4:          empty();
            break;
case 5:            exit(0);
case 6:            display();
           break;
case 7:          stack_count();
                      break;
case 8:            destroy();
                   break;
default :
    printf(" Wrong choice, Please enter correct choice  ");
    break;
```

```
        }
    }
}
void create()
{
    top = NULL;
}
void stack_count()
{
    printf("\n No. of elements in stack : %d", count);
}
void push(int data)
{
    if (top == NULL)
    {
        top =(struct node *)malloc(1*sizeof(struct node));
        top->ptr = NULL;
        top->info = data;
    }
    else
    {
        temp =(struct node *)malloc(1*sizeof(struct node));
        temp->ptr = top;
        temp->info = data;
        top = temp;
    }
    count++;
}
void display()
{
    top1 = top;
    if (top1 == NULL)
    {
        printf("Stack is empty");
        return;
    }
    while (top1 != NULL)
    {
        printf("%d ", top1->info);
        top1 = top1->ptr;
    }
 }
void pop()
{
    top1 = top;

    if (top1 == NULL)
    {
        printf("\n Error : Trying to pop from empty stack");
        return;
    }
    else
        top1 = top1->ptr;
    printf("\n Popped value : %d", top->info);
    free(top);
    top = top1;
    count--;
}
int topelement()
{
    return(top->info);
}
void empty()
{
    if (top == NULL)
        printf("\n Stack is empty");
    else
        printf("\n Stack is not empty with %d elements", count);
}
void destroy()
{
    top1 = top;
    while (top1 != NULL)
    {
        top1 = top->ptr;
        free(top);
        top = top1;
        top1 = top1->ptr;
    }
    free(top1);
    top = NULL;
    printf("\n All stack elements destroyed");
    count = 0;
}
```

16. Infix expression into its postfix expression

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define SIZE 100
char stack[SIZE];
int top = -1;
void push(char item)
{
      if(top >= SIZE-1)
      {
            printf("\nStack Overflow.");
      }
      else
      {
            top = top+1;
            stack[top] = item;
      }
}
char pop()
{
      char item ;

      if(top <0)
      {
            printf("stack under flow: invalid infix expression");
            getchar();
            exit(1);
      }
      else
      {
            item = stack[top];
            top = top-1;
            return(item);
      }
}
int is_operator(char symbol)
{
      if(symbol == '^' || symbol == '*' || symbol == '/' || symbol == '+' ||
symbol =='-')
      {
            return 1;
      }
      else
      {
      return 0;
      }
}
int precedence(char symbol)
{
      if(symbol == '^')/* exponent operator, highest precedence*/
      {
            return(3);
      }
      else if(symbol == '*' || symbol == '/')
      {
            return(2);
      }
      else if(symbol == '+' || symbol == '-')          /* lowest precedence
*/
```

```
	{
		return(1);
	}
	else
	{
		return(0);
	}
}
void InfixToPostfix(char infix_exp[], char postfix_exp[])
{
	int i, j;
	char item;
	char x;
	push('(');
	strcat(infix_exp,")");
	i=0;
	j=0;
	item=infix_exp[i];
	while(item != '\0')
	{
		if(item == '(')
		{
			push(item);
		}
		else if( isdigit(item) || isalpha(item))
		{
			postfix_exp[j] = item;
			j++;
		}
		else if(is_operator(item) == 1)
```

```
            {
            x=pop();
                  while(is_operator(x) == 1 && precedence(x)>=
precedence(item))
                  {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                  }
                  push(x);
                  push(item);
            }
            else if(item == ')')
            {
                  x = pop();
                  while(x != '(')
                  {
                        postfix_exp[j] = x;
                        j++;
                        x = pop();
                  }
            }
            else
            {
                  printf("\nInvalid infix Expression.\n");
                  getchar();
                  exit(1);
            }
            i++;
            item = infix_exp[i];
      }
      if(top>0)

      {

            printf("\nInvalid infix Expression.\n");

            getchar();

            exit(1);

      }

      if(top>0)

      {

            printf("\nInvalid infix Expression.\n");

            getchar();

            exit(1);

      }

      postfix_exp[j] = '\0';

}

int main()

{

char infix[SIZE], postfix[SIZE];

printf("ASSUMPTION: The infix expression contains single letter variables and
single digit constants only.\n");

printf("\nEnter Infix expression : ");

gets(infix);

InfixToPostfix(infix,postfix);

printf("Postfix Expression: ");

puts(postfix);

return 0;

}
```

17. Implement Queue using array

```
#include<stdio.h>

#include<conio.h>

#define SIZE 10

void enQueue(int);

void deQueue();

void display();

int queue[SIZE], front = -1, rear = -1;

void main()

{

   int value, choice;

   clrscr();

   while(1){

      printf("\n\n***** MENU *****\n");

      printf("1. Insertion\n2. Deletion\n3. Display\n4. Exit");

      printf("\nEnter your choice: ");

      scanf("%d",&choice);

      switch(choice){

       case 1: printf("Enter the value to be insert: ");

             scanf("%d",&value);

             enQueue(value);

             break;

       case 2: deQueue();

             break;

       case 3: display();

             break;

       case 4: exit(0);

       default: printf("\nWrong selection!!! Try again!!!");

      }

   }
```

```
}
void enQueue(int value){
   if(rear == SIZE-1)
      printf("\nQueue is Full!!! Insertion is not possible!!!");
   else{
      if(front == -1)
       front = 0;
      rear++;
      queue[rear] = value;
      printf("\nInsertion success!!!");
   }
}
void deQueue(){
   if(front == rear)
      printf("\nQueue is Empty!!! Deletion is not possible!!!");
   else{
      printf("\nDeleted : %d", queue[front]);
      front++;
      if(front == rear)
       front = rear = -1;
   }
}
void display()
{
   if(rear == -1)
      printf("\nQueue is Empty!!!");
   else{
      int i;
      printf("\nQueue elements are:\n");
      for(i=front; i<=rear; i++)
```

```
        printf("%d\t",queue[i]);
   }
}
```

18. Queue Datastructure using Linked List

```
#include<stdio.h>

#include<conio.h>

#define SIZE 5

void enQueue(int);

void deQueue();

void display();

int cQueue[SIZE], front = -1, rear = -1;

void main()

{

   int choice, value;

   clrscr();

   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("\nEnter the value to be insert:  ");

             scanf("%d",&value);

             enQueue(value);

             break;

       case 2: deQueue();

             break;

       case 3: display();

             break;

       case 4: exit(0);

       default: printf("\nPlease select the correct choice!!!\n");

      }
   }
}
void enQueue(int value)
{
   if((front == 0 && rear == SIZE - 1) || (front == rear+1))
      printf("\nCircular Queue is Full! Insertion not possible!!!\n");
   else{
      if(rear == SIZE-1 && front != 0)
       rear = -1;
      cQueue[++rear] = value;
      printf("\nInsertion Success!!!\n");
      if(front == -1)
       front = 0;
   }
}
void deQueue()
{
   if(front == -1 && rear == -1)
      printf("\nCircular Queue is Empty! Deletion is not possible!!!\n");
   else{
      printf("\nDeleted element : %d\n",cQueue[front++]);
      if(front == SIZE)
       front = 0;
      if(front-1 == rear)
       front = rear = -1;
   }
}
void display()
{
   if(front == -1)
```

```
      printf("\nCircular Queue is Empty!!!\n");
   else{
      int i = front;
      printf("\nCircular Queue Elements are : \n");
      if(front <= rear){
       while(i <= rear)
          printf("%d\t",cQueue[i++]);
      }
      else{
       while(i <= SIZE - 1)
          printf("%d\t", cQueue[i++]);
       i = 0;
       while(i <= rear)
          printf("%d\t",cQueue[i++]);
      }
   }
}
```

19.Program to Create Binary Tree and display using In-Order Traversal

```
#include<stdio.h>
#include<conio.h>
struct Node
{
   int data;
   struct Node *left;
   struct Node *right;
};

struct Node *root = NULL;
int count = 0;

struct Node* insert(struct Node*, int);
void display(struct Node*);

void main(){
   int choice, value;
   clrscr();
   printf("\n----- Binary Tree -----\n");
   while(1){
      printf("\n***** MENU *****\n");
      printf("1. Insert\n2. Display\n3. Exit");
      printf("\nEnter your choice: ");
      scanf("%d",&choice);
      switch(choice){
         case 1: printf("\nEnter the value to be insert: ");
                 scanf("%d", &value);
                 root = insert(root,value);
                 break;
         case 2: display(root); break;
         case 3: exit(0);
         default: printf("\nPlease select correct operations!!!\n");
      }
   }
}

struct Node* insert(struct Node *root,int value){
   struct Node *newNode;
   newNode = (struct Node*)malloc(sizeof(struct Node));
   newNode->data = value;
   if(root == NULL){
      newNode->left = newNode->right = NULL;
      root = newNode;
      count++;
   }
   else{
      if(count%2 != 0)
         root->left = insert(root->left,value);
      else
         root->right = insert(root->right,value);
   }
   return root;
}
```

```
// display is performed by using Inorder Traversal
void display(struct Node *root)
{
   if(root != NULL){
      display(root->left);
      printf("%d\t",root->data);
      display(root->right);
   }
}
```

20. Implement linear search

```c
#include <stdio.h>
int main()
{
  int array[100], search, c, n;
  printf("Enter number of elements in array\n");
  scanf("%d", &n);
  printf("Enter %d integer(s)\n", n);
  for (c = 0; c < n; c++)
  scanf("%d", &array[c]);
  printf("Enter a number to search\n");
  scanf("%d", &search);
  for (c = 0; c < n; c++)
  {
    if (array[c] == search)    /* If required element is found */
    {
      printf("%d is present at location %d.\n", search, c+1);
      break;
    }
  }
  if (c == n)
    printf("%d isn't present in the array.\n", search);

  return 0;
}
```

21. Implement bubble sort

```
#include <stdio.h>
int main()
{
  int array[100], n, c, d, swap;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 0 ; c < n - 1; c++)
  {
    for (d = 0 ; d < n - c - 1; d++)
    {
      if (array[d] > array[d+1]) /* For decreasing order use < */
      {
        swap       = array[d];
        array[d]   = array[d+1];
        array[d+1] = swap;
      }
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c < n; c++)
     printf("%d\n", array[c]);
  return 0;
}
```

22. Implement exchange sort

```
#include <stdio.h>

void sort( int [], int );

void sort( int a[], int elements )

{

    int i, j, temp;

    i = 0;

        while( i < (elements - 1) )

        {

            j = i + 1;

            while( j < elements )

            {

                if( a[i] > a[j] )

                {

                    temp = a[i];

                    a[i] = a[j];

                    a[j] = temp;

                }

                j++;

            }

            i++;

        }

}

main()

{

    int numbers[] = { 10, 9, 8, 23, 19, 11, 2, 7, 1, 13, 12 };

    int loop;

    printf("Before the sort the array was \n");

    for( loop = 0; loop < 11; loop++ )
```

```
      printf(" %d ", numbers[loop] );

      sort( numbers, 11 );

      printf("\nAfter the sort the array was \n");

      for( loop = 0; loop < 11; loop++ )

                  printf(" %d ", numbers[loop] );

}
```

23. Implement selection sort.

```c
#include<stdio.h>
int main()
{
   int i, j, count, temp, number[25];
   printf("How many numbers u are going to enter?: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);
   for(i=0;i<count;i++){
      for(j=i+1;j<count;j++){
         if(number[i]>number[j])
         {
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }
   }
   printf("Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);
   return 0;
}
```

24. Implement insertion sort.

```c
#include <stdio.h>
int main()
{
  int n, array[1000], c, d, t;
  printf("Enter number of elements\n");
  scanf("%d", &n);
  printf("Enter %d integers\n", n);
  for (c = 0; c < n; c++)
    scanf("%d", &array[c]);
  for (c = 1 ; c <= n - 1; c++)
  {
    d = c;
    while ( d > 0 && array[d-1] > array[d])
    {
      t          = array[d];
      array[d]   = array[d-1];
      array[d-1] = t;

      d--;
    }
  }
  printf("Sorted list in ascending order:\n");
  for (c = 0; c <= n - 1; c++) {
    printf("%d\n", array[c]);
  }

  return 0;
}
```

25. Implement quick sort.

```
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;
      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
            i++;
         while(number[j]>number[pivot])
            j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }
      temp=number[pivot];
      number[pivot]=number[j];
      number[j]=temp;
      quicksort(number,first,j-1);
      quicksort(number,j+1,last);
   }
}
int main()
{
   int i, count, number[25];
   printf("How many elements are u going to enter?: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);
   quicksort(number,0,count-1);
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);
   return 0;
}
```

26. Implement merge sort.

```
#include<stdio.h>

void mergesort(int a[],int i,int j);

void merge(int a[],int i1,int j1,int i2,int j2);

int main()

{

    int a[30],n,i;

    printf("Enter no of elements:");

    scanf("%d",&n);

    printf("Enter array elements:");

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

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

    mergesort(a,0,n-1);

    printf("\nSorted array is :");

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

        printf("%d ",a[i]);

    return 0;

}


void mergesort(int a[],int i,int j)

{

    int mid;

    if(i<j)

    {

        mid=(i+j)/2;

        mergesort(a,i,mid);

        mergesort(a,mid+1,j);

        merge(a,i,mid,mid+1,j);

    }
}

void merge(int a[],int i1,int j1,int i2,int j2)
{
    int temp[50];
    int i,j,k;
    i=i1;
    j=i2;
    k=0;
    while(i<=j1 && j<=j2)
    {
        if(a[i]<a[j])
            temp[k++]=a[i++];
        else
            temp[k++]=a[j++];
    }
    while(i<=j1)
        temp[k++]=a[i++];
    while(j<=j2)
        temp[k++]=a[j++];
    for(i=i1,j=0;i<=j2;i++,j++)
        a[i]=temp[j];
}
```

27. Implement heap sort

```
#include<stdio.h>

#include<conio.h>

#define MAX_SIZE 5

void heap_sort();

void heap_adjust(int, int);

int arr_sort[MAX_SIZE], t, a;

int main()

{

  int i;

  printf("Simple Heap Sort Example - Functions and Array\n");

  printf("\nEnter %d Elements for Sorting\n", MAX_SIZE);

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

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

  printf("\nYour Data   :");

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

  {

    printf("\t%d", arr_sort[i]);

  }

  heap_sort();

  printf("\n\nSorted Data :");

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

  {

    printf("\t%d", arr_sort[i]);

  }

  getch();

}

void heap_sort()

{
  for (int i = MAX_SIZE / 2 - 1; i >= 0; i--)
    heap_adjust(MAX_SIZE, i);
  for (int i = MAX_SIZE - 1; i >= 0; i--)
  {
    t = arr_sort[0];
    arr_sort[0] = arr_sort[i];
    arr_sort[i] = t;
    heap_adjust(i, 0);
    printf("\nHeap Sort Iteration %d : ", i);
    for (a = 0; a < MAX_SIZE; a++) {
      printf("\t%d", arr_sort[a]);
    }
  }
}
void heap_adjust(int n, int i)
{
  int large = i, left = 2 * i + 1, right = 2 * i + 2;
  if (left < n && arr_sort[left] > arr_sort[large])
    large = left;
  if (right < n && arr_sort[right] > arr_sort[large])
    large = right;
  if (large != i)
  {
    t = arr_sort[i];
    arr_sort[i] = arr_sort[large];
    arr_sort[large] = t;
    heap_adjust(n, large);
  }
}
```