**********pointer**********
****

```
#include<stdio.h>

void read_array(int *pa, int size);
void print_array(int *pa, int size);
void reverse_array(int *pa, int size, int *pal);
void *search_array(int *pa, int size, int *key);
int max_lement(int *pa, int size);



   void main()
   {
          int a[20], size, key, *pa, *pal,ch, i, *b,c;
          do
           {

                printf("\n");
                printf("**** MENUE ****\n");
                printf("1. Read array\n2.Print array\n3.Reverse array\n4.Search array n5.Max
Element\n6.close\n");
                printf("Enter your choice\n"); scanf("%d", &ch);

                switch(ch)
                {

   case 1: printf("Enter the Size: ");
          scanf("%d", &size);
          read_array(&a[0],size);

   break;

   case 2: print_array(&a[0],size);

   break;

   case 3: reverse_array(&a[0],size,&a[size-1]);
          printf("Elements of an array are: ");
          for(i=0;i<size; i++)
          printf("%d",a[i]);
```

```
break;

case 4: printf("Enter the key Element");

scanf("%d", &key);

b=search_array(&a[0], size, &key);

printf("The Element found is: %d", *b);

break;

case 5: c= max_element(&a[0], size);

printf("The largest element in array is:%d",c);

break;

case 6: printf("Closed \n");

break;

} }while(ch!=6);

}

void read_array(int *pa, int size)
{

int i;

printf("Enter array elemnts: ");

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

{

scanf("%d",pa);


pa++;
```

```
}

}

void print_array(int *pa, int size)

{

int i;

printf(" Array Elements are: ");

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

{ printf("%d ", *pa);
pa++;

}
}

void reverse_array(int *pa, int size, int *pal)
{

int i,j,temp;

j=size-1;

for(i=0;i<j; i++, j--)

{  temp=*pa;
   *pa= *pal;
   *pal=temp;
    pa++;
    pal--;

  }
}

void *search_array(int *pa, int size, int *key)

{

int i;
```

```
for(i=0;i<size; i++) {

if((*key)=*pa) return pa; pa++;

} }

int max_element(int *pa, int size)
 {

int i,l;

l=*pa;

pa++;

for (i=1;i<size; i++)

{
if(*pa>l)
l=*pa;
pa++;

}
return l;

}
```

**********INFIX TO POSTFIX**********

```
#include <stdio.h>
#include <ctype.h>
#define MAX 20

// Prototypes
void push(char);
char pop();
int priority(char);

// Stack declaration
char stack[MAX];
int top=-1;

int main()
{
        char exp[20], *e, x;
        // reading the infix expression
        printf("\nEnter the exp: ");
        scanf("%s", exp);



        // conversion to equivalent postfix expression
        e=exp;
        printf("\nThe equivalent postfix expression is: ");
        while(*e!='\0')
        {
                        if(isalnum(*e))         // operand
                                printf("%c", *e);
                        else
                                if(*e=='(')
                                        push(*e);
```

```
                        else
                                if(*e==')')
                                {
                                        while((x=pop())!='('
                                                printf("%c", x);
                                }
                        else      // operator
                        {
                                        while(priority(stack[top])>=priority(*e))
                                                printf("%c", pop());
                                        push(*e);
                        }
                e++;
     } // end of while

   // pop the remaining operators and print
   while(top!=-1)
      printf("%c", pop());
   return 0;
}


// Stack Push Operation
void push(char x)
{
          if(top==MAX-1)
          {
                printf("\nStack Overflow");
                return;
          }
          stack[++top]=x;
}


// Stack Pop Operation
char pop()
{
          if(top==-1)
                return -1;
          else
                return stack[top--];
}
```

```
// Returns priority of the operator parameter
int priority(char x)
{
	if(x=='(')
		return 0;
	if(x=='+' || x== '-')
		return 1;
	if(x=='*' || x== '/')
		return 2;
}
```

**********CONSIDER STUDENTS TAKING ADMISSION FOR COLLAGE NAME CET RANK BRANCH
****

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

// Define a structure to represent a student
struct Student {
    char name[50];
    int cetRank;
    char branch[20];
};

// Function to read student data
```

```
void input(struct Student students[], int n)
{
    // Input student details
    for (int i = 0; i < n; ++i) {
        printf("Enter details for student %d:\n", i + 1);
        printf("Name: ");
        scanf("%s", students[i].name);
        printf("CET Rank: ");
        scanf("%d", &students[i].cetRank);
        printf("Branch: ");
        scanf("%s", students[i].branch);
    }
}
// Function to segregate students based on branch
void segregateStudents(struct Student students[], int n, char targetBranch[])
{
    printf("\nStudents opting for %s branch:\n", targetBranch);
    for (int i = 0; i < n; ++i) {
        if (strcasecmp(students[i].branch, targetBranch) == 0) {
            printf("Name: %s, CET Rank: %d\n", students[i].name, students[i].cetRank);
        }
    }
}

int main() {
    // Define the number of students
    int n;
    printf("Enter the number of students: ");
    scanf("%d", &n);

    // Declare an array of structures to store student information
    struct Student students[n];

    // Input student details
     input(students,n);

    // Segregate students based on branch
    segregateStudents(students, n, "CSE");
    segregateStudents(students, n, "EE");
    segregateStudents(students, n, "Mech");

    return 0;
}
```

**********QUEUE**********
****

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

// Structure to create a node with data and the next pointer
struct node {
   int data;
   struct node * next;
};

struct node * front = NULL;
struct node * rear = NULL;

// Enqueue() operation on a queue
void enqueue(int value) {
   struct node * ptr;
   ptr = (struct node * ) malloc(sizeof(struct node));
   ptr -> data = value;
   ptr -> next = NULL;
   if ((front == NULL) && (rear == NULL)) {
      front = rear = ptr;
   } else {
      rear -> next = ptr;
      rear = ptr;
   }
```

```
    printf("Node is Inserted\n\n");
}

// Dequeue() operation on a queue
int dequeue() {
    if (front == NULL) {
        printf("\nUnderflow\n");
        return -1;
    } else {
        struct node * temp = front;
        int temp_data = front -> data;
        front = front -> next;
        free(temp);
        return temp_data;
    }
}

// Display all elements of the queue
void display() {
    struct node * temp;
    if ((front == NULL) && (rear == NULL)) {
        printf("\nQueue is Empty\n");
    } else {
        printf("The queue is \n");
        temp = front;
        while (temp) {
            printf("%d--->", temp -> data);
            temp = temp -> next;
        }
        printf("NULL\n\n");
    }
}


int main() {
    int choice, value;
    printf("\nImplementation of Queue using Linked List\n");
    while (choice != 4) {
        printf("1.Enqueue\n2.Dequeue\n3.Display\n4.Exit\n");
        printf("\nEnter your choice : ");
        scanf("%d", & choice);
        switch (choice) {
            case 1:
```

```
            printf("\nEnter the value to insert: ");
            scanf("%d", & value);
            enqueue(value);
            break;
         case 2:
            printf("Popped element is :%d\n", dequeue());
            break;
         case 3:
            display();
            break;
         case 4:
            exit(0);
            break;
         default:
            printf("\nWrong Choice\n");
      }
   }
   return 0;
}
```

**********// TW 5 Warehouse as an Ordered List**********

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

typedef struct node
{
   int data;
   struct node *next;
```

```
}NODE;

NODE* add(NODE*, int);
void disp(NODE*);
int search(NODE*, int);

int main()
{
   //Create an empty warehouse
   NODE *head=NULL;
   int opt, item;
   while(1)
   {
      printf("\n1: Add item    2: Disp Warehouse    ");
      printf("3: Search item   4: exit");
      printf("\nEnter your option: ");
      scanf("%d", &opt);
      switch(opt)
      {
         case 1: printf("\nEnter item to add to warehouse: ");
                 scanf("%d", &item);
                 head=add(head, item);
                 break;
         case 2: disp(head);   break;
         case 3: printf("\nEnter the item to search: ");
                 scanf("%d", &item);
                 if(search(head, item))
                   printf("\nItem %d is present in the warehouse", item);
                 else
                   printf("\nItem %d is NOT present in the warehouse", item);
                 break;
         case 4: exit(0);
      }
   }
   return 0;
}


NODE* add(NODE* head, int item)
{
   NODE *prev, *curr;
   NODE *newnode=(NODE*)malloc(sizeof(NODE));
   if(newnode==NULL)
```

```
    {
        printf("\nMalloc failure");
        exit(1);
    }
    newnode->data=item;
    newnode->next=NULL;
    // Case i - List is empty
    if(head==NULL)
        head=newnode;
    else // Case ii - adding the smallest item
        if(item < head->data)
        {
             newnode->next = head;
             head = newnode;
        }
    else // Case iii
    {
        prev=head;
        curr=head->next;
        while(curr && item > curr->data)
        {
           prev=prev->next;
           curr=curr->next;
        }//end of while
        newnode->next=curr;
        prev->next=newnode;

    }//end of else
    return head;
}


void disp(NODE *head)
{
    if(head==NULL)
    {
        printf("\nWarehouse is empty");
        return;
    }
    printf("\nThe warehouse items are: ");
    while(head)
    {
        printf("%d    ", head->data);
```

```
        head=head->next;
    }
}


int search(NODE* head, int item)
{
    if(head==NULL)
    {
        printf("\nWarehouse is empty");
        return 0;
    }
    while(head && (item > head->data))
          head=head->next;

    if(head==NULL)
        return 0;

    if(item==head->data)
        return 1;
    else
        return 0;
}
```

****
****