# INDEX

| | | | |
|---|---|---|---|
| 6 (Stacks) | | linked-list | |
| | 20. | Program for reversing elements of the stack | |
| | 21. | Program to convert an infix expression to postfix expression using stacks | |
| 7 (Stacks Contd.) | 22. | Program to implement stacks using two queues | |
| | 23. | Program to use stack for postfix expression evaluation | |
| | 24. | Program to check if two stacks are identical or not | |
| 8 (Queues ) | 25. | Program to implement growable queues using arrays | |
| | 26. | Program to implement queues using linked-list | |
| | 27. | Program to implement queues using two stacks | |
| | 28. | Program to implement double-ended queues with insert and delete functions | |
| 9 (Trees) | 29. | Program to store binary trees in a 1D array | |
| | 30. | Program to store binary trees in linked list | |
| | 31. | Program for finding an element given its key in a 1D array implementation of binary search trees | |
| | 32. | Program for fnding the element given its key in a linked list implementation of BST using recursion | |
| 10 (Trees Contd.) | 33. | Program to find the maximum depth of the BST | |
| | 34. | Program to traverse the BST in Inorder, Preorder, Postorder with and without recursion | |
| | 35. | Program to create its corresponding BST from a given unsorted array and print all the elements in sorted order (i.e. inorder traversal) | |
| 11 (Graphs) | 36. | Program to implement graphs using adjacency matrix, incidence matrix, adjacency list and direct representation | |
| | 37. | Program to traverse the graph in Depth-frst order on an adjacent matrix implementation of graph | |

| | | | |
|---|---|---|---|
| | 38. | Program to traverse the graph in Breadth-frst order on an adjacent matrix implementation of graph | |
| | 39. | Program for detecting a cycle in an undirected graph using DFS | |
| 12 (Sorting) | 40. | Program to implement Merge sort recursively. | |
| | 41. | Program to implement Quick sort recursively. | |
| | 42. | Program to implement Insertion sort. | |
| | 43. | Program to implement Selection sort. | |
| | 44. | Program to implement Radix sort. | |
| 13 (Miscella -neous) | 45. | N people have decided to elect a leader by arranging themselves in a circle and eliminating every M th person around the circle, closing ranks as each person drops out. Find which person will be the last one remaining (with rank 1). [Use Circular Linked List] | |
| | 46. | Please find the value of C, U, B, E, D using the following equation: (C+U+B+E+D)^3=CUBED, where C, U, B, E, D are integers. | |
| 14 (Miscella -neous Contd.) | 47. | Program to implement Stack in a way such that Push, Pop, and GetMinimum operations take constant time. [Use two stacks] | |
| | 48. | We are given a list of prices of a stock for N number of days. We need to fnd the span for each day. Span is defned as number of consecutive days before the given day where the price of stock was less than or equal to price at given day. **For example,** {100, 60,70,65, 80, 85} span will be {1, 1, 2, 1, 4, 5}. For first day span is always 1. In example we can see that for day 2 at 60, there is no day before it where price was less than 60. Hence span is 1 again. For day 3, price at day 2 (60) is less than 70, hence span is 2. Similarly, for day 4 and day 5. Remember days should be consecutive, that why span for day 4 is 1 even though there was a day 2 where price was less than 65. [Stock Span Problem] | |

**Program: 1** Program to find minimum and maximum element in a given array.

```
#include <iostream>
using namespace std;
int main(){
      int max=0, min, n;
      cout<<"Enter the number of elements: ";
      cin>>n;
      int arr[n];
      cout<<"Enter elements of array:";
      for(int i=0; i<n; i++)
            cin>>arr[i];
      max=arr[0];
      for(int j=1; j<n; j++){
            if(arr[j]>max){
                  max= arr[j];
            }
      }
      min= arr[0];
      for(int k=1; k<n; k++){
            if(arr[k]<min)
            min= arr[k];
      }
      cout<<"Maximum element in array is :" << max << endl;
      cout<<"Minimum element in array is :" << min << endl;
      return(0);
}
```

---

**Program: 2** Program to find median in the unsorted array of integers.

```
#include <iostream>
using namespace std;
int main(){
      int n;
      cout<<"Enter number of elements in array: ";
      cin>>n;
      int arr[n],temp;
      cout<<"Enter elements of array:"<<endl;
      for(int i=0; i<n; i++)
            cin>>arr[i];
      for(int j=0; j<n; j++){
            for(int k=j; k<n-1; k++){
                  if(arr[k+1]<arr[k]){
                        temp= arr[k+1];
                        arr[k+1]= arr[k];
                        arr[k]= temp;
                  }
            }
      }
      cout << "\nSORTED ARRAY IS: ";
```

```
21.      for(int l=0; l<n; l++){
22.           cout<<endl<<arr[l]<<" ";
23.      }
24.      if(n%2!=0){
25.           cout<<"\nMEDIAN IS: "<<arr[n/2]<<endl;
26.      }
27.      else{
28.           int b=n/2;
29.           int c=n/2+1;
30.           cout<< "\nMEDIAN IS: "<<(arr[b-1]+arr[c-1])/2<<endl;
31.      }
32.      return 0;
33. }
```

---

**Program: 3** Program to find the second largest element in unsorted array.

```
1.  #include <iostream>
2.  using namespace std;
3.  int main(){
4.        int n;
5.        cout<<"Enter number of elements in array: ";
6.        cin>>n;
7.        int arr[n],temp;
8.        cout<<"Enter elements of array:"<<endl;
9.        for(int i=0; i<n; i++)
10.            cin>>arr[i];
11.       for(int j=0; j<n; j++){
12.            for(int k=j; k<n-1; k++){
13.                 if(arr[k+1]<arr[k]){
14.                      temp= arr[k+1];
15.                      arr[k+1]= arr[k];
16.                      arr[k]= temp;
17.                 }
18.            }
19.       }
20.       cout<<"Second largest element is: "<<arr[n-2]<<endl;
21.       return 0;
22. }
```

---

**Program: 4** Program for finding the union and intersection of two unsorted arrays.

```
1.  #include <stdio.h>
2.  #include <iostream>
3.  #include <algorithm>
4.  using namespace std;
5.  int binarySearch(int arr[], int l, int r, int x);
6.  void printUnion(int arr1[], int arr2[], int m, int n){
7.      if(m>n){
8.          int *tempp=arr1;
```

```
        arr1=arr2;
        arr2=tempp;
        int temp=m;
        m=n;
        n=temp;
    }
    sort(arr1, arr1+m);
    for (int i=0; i<m; i++)
        cout<<arr1[i]<<" ";
    for (int i=0; i<n; i++)
        if (binarySearch(arr1, 0, m-1, arr2[i])==-1)
            cout<<arr2[i]<<" ";
}
void printIntersection(int arr1[], int arr2[], int m, int n){
    if (m>n){
        int *tempp=arr1;
        arr1=arr2;
        arr2=tempp;
        int temp=m;
        m=n;
        n=temp;
    }
    sort(arr1, arr1 + m);
    for (int i=0; i<n; i++)
        if (binarySearch(arr1, 0, m-1, arr2[i])!=-1)
            cout<<arr2[i]<<" ";
}
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(){
    int arr1[]={7, 1, 5, 2, 3, 6};
    int arr2[]={3, 8, 6, 20, 7};
    int m=sizeof(arr1)/sizeof(arr1[0]);
    int n=sizeof(arr2)/sizeof(arr2[0]);
    cout<<"Union of two arrays is :";
    printUnion(arr1, arr2, m, n);
    cout<<"\nIntersection of two arrays is : ";
    printIntersection(arr1, arr2, m, n);
    printf("\n");
    return 0;
}
```

**Program: 5** Program for multiplying 2 2D-matrices

```
#include <stdio.h>
int main(){
      int m, n, p, q, c, d, k, sum=0;
      int first[10][10], second[10][10], multiply[10][10];
      printf("Enter the number of rows and columns of first matrix\n");
      scanf("%d%d", &m, &n);
      printf("Enter the elements of first matrix\n");
      for (c=0; c<m; c++)
            for (d=0; d<n; d++)
                  scanf("%d", &first[c][d]);
      printf("Enter the number of rows and columns of second
matrix\n");
      scanf("%d %d", &p, &q);
      if (n!=p)
            printf("Matrices with entered orders can't be multiplied
with each other.\n")
      else{
            printf("Enter the elements of second matrix\n");
            for (c=0; c<p; c++)
                  for (d=0; d<q; d++)
                        scanf("%d", &second[c][d]);
            for (c=0; c<m; c++){
                  for (d=0; d<q; d++){
                        for (k=0; k<p; k++){
                              sum=sum + first[c][k]*second[k][d];
                              }
                        multiply[c][d]=sum;
                        sum=0;
                  }
            }
            printf("Product of entered matrices:-\n");
            for (c=0; c<m; c++){
                  for (d=0; d<q; d++)
                        printf("%d\t", multiply[c][d]);
                  printf("\n");
            }
      }
      return 0;
}
```

---

**Program: 6** Program to find $N^{th}$ fibonacci number using recursion

```
#include <stdio.h>
int fibo(int);
int main(){
      int num;
      int result;
      printf("Enter the nth number in fibonacci series: ");
```

```
        scanf("%d", &num);
        if (num<0){
                printf("Please Enter Positive number!!!\n");
        }
        else{
                result=fibo(num);
                printf("The %d number in fibonacci series is %d\n", num,
   result);
        }
        return 0;
}
int fibo(int num){
        if (num==0){
                return 0;
        }
        else if(num==1){
                return 1;
        }
        else{
                return(fibo(num - 1)+fibo(num – 2));
        }
}
```

---

**Program: 7** Program to find greatest common divisor of 2 numbers using recursion.

```
#include <stdio.h>
int gcd(int a, int b);
int main(){
        int n1, n2, s;
        printf("Enter two integers: ");
        scanf("%d %d",&n1,&n2);
        s=gcd(n1, n2);
        printf("The gcd of entered two numbers is %d\n",s);
        return 0;
}
int gcd(int a, int b){
        if(a<0)
                a=-a;
        if(b<0)
                b=-b;
        if(a==b)
                return a;
        else
                return (gcd(b,a-b));
}
```

---

**Program: 8** Program to find sum of all elements of an integer array using recursion.

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

```
int findSum(int A[], int n){
    if (n<=0)
        return 0;
    return (findSum(A, n-1)+A[n-1]);
}
int main(){
    int n;
    printf("Enter Number of Elements: ");
    scanf("%d",&n);
    int A[n];
    for(int i=0;i<n;i++)
        scanf("%d",&A[i]);
    printf("The sum of all the elements of array is %d\n", findSum(A, n));
    return 0;
}
```

---

**Program: 9** Program for finding the solution of Towers of Hanoi problem having ‘n’ discs

```
#include <stdio.h>
void towers(int num, char frompeg, char topeg, char auxpeg){
      if(num==1){
            printf("\n Move disk 1 from peg %c to peg %c", frompeg, topeg);
            return;
      }
      towers(num - 1, frompeg, auxpeg, topeg);
      printf("\n Move disk %d from peg %c to peg %c", num, frompeg, topeg);
      towers(num - 1, auxpeg, topeg, frompeg);
}
int main(){
     int num;
     printf("Enter the number of disks : ");
     scanf("%d", &num);
     printf("The sequence of moves involved in the Tower of Hanoi are :\n");
     towers(num, 'A', 'C', 'B');
     printf("\n");
     return 0;
}
```

---

**Program: 10** Program to find the middle element in the linked-list.

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node* next;
};
```

```
void printMiddle(struct Node *head){
    struct Node *slow_ptr=head;
    struct Node *fast_ptr=head;
    if (head!=NULL){
        while (fast_ptr!=NULL && fast_ptr->next!=NULL){
            fast_ptr=fast_ptr->next->next;
            slow_ptr=slow_ptr->next;
        }
        printf("The middle element is [%d]\n\n", slow_ptr->data);
    }
}
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 *ptr){
    while (ptr!=NULL){
        printf("%d->", ptr->data);
        ptr=ptr->next;
    }
    printf("NULL\n");
}
int main(){
    struct Node* head=NULL;
    int i;
    for (i=1; i<10; i++){
        push(&head, i);
        printList(head);
        printMiddle(head);
    }
    return 0;
}
```

---

**Problem: 11** Program for reversing a linked-list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
 int num;
 struct node *next;
};
void create(struct node **head){
 int c, ch;
 struct node *temp, *rear;
 do{
   printf("Enter number: ");
   scanf("%d", &c);
   temp=(struct node *)malloc(sizeof(struct node));
```

```
   temp->num=c;
   temp->next=NULL;
   if (*head==NULL){
    *head=temp;
   }
   else{
    rear->next=temp;
   }
   rear=temp;
   printf("Do you wish to continue [1/0]: ");
   scanf("%d", &ch);
 }while (ch!=0);
 printf("\n");
}
void reverse(struct node **head){
 struct node *p, *q, *r;
 p=q=r=*head;
 p=p->next->next;
 q=q->next;
 r->next=NULL;
 q->next=r;
 while (p!=NULL){
  r=q;
  q=p;
  p=p->next;
  q->next=r;
 }
 *head=q;
}
void release(struct node **head){
 struct node *temp=*head;
 *head=(*head)->next;
 while ((*head)!=NULL){
  free(temp);
  temp=*head;
  (*head)=(*head)->next;
 }
}
void display(struct node *p){
 while (p!=NULL){
  printf("%d\t", p->num);
  p=p->next;
 }
 printf("\n");
}
int main(){
 struct node *p=NULL;
 int n;
 printf("Enter data into the list\n");
 create(&p);
```

```
64. printf("Before reversing the list:\n");
65. display(p);
66. printf("Reversing the list...\n");
67. reverse(&p);
68. printf("After reversing list:\n");
69. display(p);
70. release(&p);
71. return 0;
72.}
```

---

**Problem: 12** Program for segregating even and odd key nodes in a linked-list

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. struct node{
4.  int num;
5.  struct node *next;
6. };
7. void create(struct node **head){
8.   int c, ch;
9.   struct node *temp, *rear;
10.  do{
11.   printf("Enter number: ");
12.   scanf("%d", &c);
13.   temp = (struct node *)malloc(sizeof(struct node));
14.   temp->num = c;
15.   temp->next = NULL;
16.   if (*head == NULL){
17.    *head = temp;
18.   }
19.   else{
20.    rear->next = temp;
21.   }
22.   rear = temp;
23.   printf("Do you wish to continue [1/0]: ");
24.   scanf("%d", &ch);
25.  }while(ch!=0);
26.  printf("\n");
27.}
28.
29.void generate_evenodd(struct node *list, struct node **head){
30. struct node *even = NULL, *odd = NULL, *temp;
31. struct node *reven, *rodd;
32. while (list != NULL){
33.  temp = (struct node *)malloc(sizeof(struct node));
34.  temp->num = list->num;
35.  temp->next = NULL;
36.  if (list->num % 2 == 0){
37.   if (even == NULL){
38.     even = temp;
```

```
   }
   else{
     reven->next = temp;
   }
   reven = temp;
  }
  else{
   if (odd == NULL){
    odd = temp;
   }
   else{
    rodd->next = temp;
   }
   rodd = temp;
  }
  list = list->next;
 }
 reven->next = odd;
 *head = even;
}
void release(struct node **head){
 struct node *temp = *head;
 *head = (*head)->next;
 while ((*head) != NULL){
  free(temp);
  temp = *head;
 (*head) = (*head)->next;
 }
}
void display(struct node *p){
  while (p != NULL){
   printf("%d\t", p->num);
   p = p->next;
  }
  printf("\n");
}
int main(){
 struct node *p = NULL, *q = NULL;
 int key, result;
 printf("Enter data into the list\n");
 create(&p);
 printf("Displaying the nodes in the list:\n");
 display(p);
 generate_evenodd(p, &q);
 printf("Displaying the list with even and then odd:\n");
 display(q);
 release(&p);
 return 0;
}
```

**Problem: 13** Program for detecting a loop in a linked-list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
 int num;
 struct node *next;
};
void create(struct node **head){
 int c, ch;
 struct node *temp, *rear;
 do{
  printf("Enter number: ");
  scanf("%d", &c);
  temp=(struct node *)malloc(sizeof(struct node));
  temp->num=c;
  temp->next=NULL;
  if (*head==NULL){
   *head=temp;
  }
  else{
    rear->next=temp;
  }
  rear=temp;
  printf("Do you wish to continue [1/0]: ");
  scanf("%d", &ch);
  } while (ch!=0);
   printf("\n");
}
void makecycle(struct node **p){
 struct node *rear, *front;
 int n, count=0, i;
 front=rear=*p;
 while (rear->next!=NULL){
  rear=rear->next;
  count++;
 }
 if (count)
  n=rand() % count;
 else
  n=1;
 for (i=0; i<n - 1; i++)
   front=front->next;
 rear->next=front;
}
void release(struct node **head){
 struct node *temp=*head;
 temp=temp->next;
 while ((*head)!=NULL){
  free(temp);
```

```
  temp=*head;
  (*head)=(*head)->next;
 }
}
int detectcycle(struct node *head){
 int flag=1, count=1, i;
 struct node *p, *q;
 p=q=head;
 q=q->next;
 while (1){
  q=q->next;
  if (flag){
   p=p->next;
  }
  if (q==p){
   q=q->next;
   while (q!=p){
    count++;
    q=q->next;
   }
   q=p=head;
   for (i=0; i<count; i++){
    q=q->next;
   }
   while (p!=q){
    p=p->next;
    q=q->next;
   }
   q->next=NULL;
   return 1;
   }
 else if (q->next==NULL){
  return 0;
 }
 flag=!flag;
 }
}
int main(){
 struct node *p=NULL;
 int result;
 printf("Enter data into the list\n");
 create(&p);
 makecycle(&p); //comment it to avoid cycle creation
 printf("Working...\n");
 result=detectcycle(p);
 if (result){
  printf("Cycle detected in the linked list.\n");
 }
 else{
  printf("No cycle detected in the linked list.\n");
```

```
99. }
100.      release (&p);
101.      return 0;
102.     }
```

---

**Problem: 14** Program to check whether a given linked-list is palindrome or not

```
1. #include <stdio.h>
2. #include <stdlib.h>
3. #include <stdbool.h>
4. struct Node{
5.     char data;
6.     struct Node* next;
7. };
8. void reverse(struct Node**);
9. bool compareLists(struct Node*, struct Node *);
10.bool isPalindrome(struct Node *head){
11.    struct Node *slow_ptr=head, *fast_ptr=head;
12.    struct Node *second_half, *prev_of_slow_ptr=head;
13.    struct Node *midnode=NULL;
14.    bool res=true;
15.    if (head!=NULL && head->next!=NULL){
16.        while (fast_ptr!=NULL && fast_ptr->next!=NULL){
17.            fast_ptr=fast_ptr->next->next;
18.            prev_of_slow_ptr=slow_ptr;
19.            slow_ptr=slow_ptr->next;
20.        }
21.        if (fast_ptr!=NULL){
22.            midnode=slow_ptr;
23.            slow_ptr=slow_ptr->next;
24.        }
25.        second_half=slow_ptr;
26.        prev_of_slow_ptr->next=NULL;
27.        reverse(&second_half);
28.        res=compareLists(head, second_half);
29.
30.        reverse(&second_half);
31.        if (midnode!=NULL){
32.            prev_of_slow_ptr->next=midnode;
33.            midnode->next=second_half;
34.        }
35.        else  prev_of_slow_ptr->next=second_half;
36.    }
37.    return res;
38.}
39.void reverse(struct Node** head_ref){
40.    struct Node* prev =NULL;
41.    struct Node* current=*head_ref;
42.    struct Node* next;
43.    while (current!=NULL){
```

```
        next=current->next;
        current->next=prev;
        prev=current;
        current=next;
    }
    *head_ref=prev;
}
bool compareLists(struct Node* head1, struct Node *head2){
    struct Node* temp1=head1;
    struct Node* temp2=head2;

    while (temp1 && temp2){
        if (temp1->data==temp2->data){
            temp1=temp1->next;
            temp2=temp2->next;
        }
        else return 0;
    }
    if (temp1==NULL && temp2==NULL)
        return 1;
    return 0;
}
void push(struct Node** head_ref, char 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 *ptr){
    while (ptr!=NULL){
        printf("%c->", ptr->data);
        ptr=ptr->next;
    }
    printf("NULL\n");
}
int main(){
    struct Node* head=NULL;
    char str[]="JISHAN SHAIKH";
    int i;
    for (i=0; str[i]!='\0'; i++){
       push(&head, str[i]);
       printList(head);
       isPalindrome(head)? printf("Is Palindrome\n\n"):
                           printf("Not Palindrome\n\n");
    }
    return 0;
}
```

---

**Problem: 15** Program to create a circular linked-list and always insert data at tail

```
#include <stdio.h>
#include <stdlib.h>
struct node{
 int data;
 struct node *link;
};
struct node *head=NULL, *x, *y, *z;
void create();
void ins_at_beg();
void ins_at_pos();
void del_at_beg();
void del_at_pos();
void traverse();
int  main(){
 int ch;
 printf("\nEnter your choice (1,2,3,4,5,6,10) : \n\n");
 printf("\n 1.Creation \n 2.Insertion at beginning \n 3.Insertion at
remaining");
 printf("\n 4.Deletion at beginning \n 5.Deletion at remaining \n
6.only traverse");
 printf("\n 10.Exit\n");
 while(1){
  printf("\n Enter your choice:");
  scanf("%d", &ch);
  switch(ch){
   case 1:
         create();
         break;
   case 2:
         ins_at_beg();
         break;
   case 3:
         ins_at_pos();
         break;
   case 4:
         del_at_beg();
         break;
   case 5:
         del_at_pos();
         break;
   case 6:
         traverse();
         break;
   default:
         exit(0);
  }
 }
 return 0;
}
void create(){
```

```
int c;
x=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d", &x->data);
x->link=x;
head=x;
printf("\n If you wish to continue press 1 otherwise 0:");
scanf("%d", &c);
while (c!=0){
 y=(struct node*)malloc(sizeof(struct node));
 printf("\n Enter the data:");
 scanf("%d", &y->data);
 x->link=y;
 y->link=head;
 x=y;
 printf("\n If you wish to continue press 1 otherwise 0:");
 scanf("%d", &c);
 }
}
void ins_at_beg(){
x=head;
y=(struct node*)malloc(sizeof(struct node));
printf("\n Enter the data:");
scanf("%d", &y->data);
while (x->link!=head){
  x=x->link;
 }
 x->link=y;
 y->link=head;
 head=y;
}
void ins_at_pos(){
struct node *ptr;
int c=1, pos, count=1;
y=(struct node*)malloc(sizeof(struct node));
if (head==NULL){
 printf("cannot enter an element at this place");
}
printf("\n Enter the data:");
scanf("%d", &y->data);
printf("\n Enter the position to be inserted:");
scanf("%d", &pos);
x=head;
ptr=head;
while (ptr->link!=head){
 count++;
 ptr=ptr->link;
}
count++;
if (pos>count){
```

```
99.   printf("OUT OF BOUND");
100.        return;
101.      }
102.      while (c<pos){
103.        z=x;
104.        x=x->link;
105.        c++;
106.      }
107.       y->link=x;
108.       z->link=y;
109.     }
110.     void del_at_beg(){
111.       if (head==NULL)
112.        printf("\n List is empty");
113.       else{
114.        x=head;
115.        y=head;
116.        while (x->link!=head){
117.         x=x->link;
118.        }
119.        head=y->link;
120.        x->link=head;
121.        free(y);
122.        }
123.     }
124.     void del_at_pos(){
125.      if (head==NULL)
126.       printf("\n List is empty");
127.      else{
128.       int c=1, pos;
129.       printf("\n Enter the position to be deleted:");
130.       scanf("%d", &pos);
131.       x=head;
132.       while (c<pos){
133.         y=x;
134.         x=x->link;
135.         c++;
136.       }
137.       y->link=x->link;
138.       free(x);
139.       }
140.     }
141.     void traverse(){
142.      if (head==NULL)
143.       printf("\n List is empty");
144.      else{
145.       x=head;
146.       while (x->link!=head){
147.        printf("%d->", x->data);
148.        x=x->link;
```

```
      }
      printf("%d", x->data);
     }
    }
```

---

**Problem: 16** Program to create a doubly linked-list from a given singly linked-list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
 int num;
 struct node *next;
};
void create(struct node **);
void move (struct node *);
void release(struct node **);
void display(struct node *);
int main(){
 struct node *p=NULL, *q=NULL;
 int result, count;
 printf("\nEnter data into the list\n");
 create(&p);
 printf("Displaying list:\n");
 display(p);
 move(p);
 release (&p);
 return 0;
}
void move(struct node *head){
 struct node *p, *q;
 int ch;
 p=q=head;
 printf("\nPointer at %d\n", head->num);
 do{
  printf("Select option:\n1. Move front\n2. Move back\n3. Exit\nYour
choice: ");
  scanf("%d", &ch);
  switch(ch){
    case 1: if(q->next!=NULL){
              q=q->next;
              printf("\nPointer at %d\n", q->num);
            }
            else{
               printf("\nPointer at last node %d. Cannot move
ahead.\n", q->num);
            }
            break;

    case 2: while (p->next!=q)
              p=p->next;
```

```
            if (p==q)
              printf("\nPointer at first node %d. Cannot move
   behind.\n", q->num);
            else{
              q=p;
              p=head;
              printf("\nPointer at %d\n", q->num);
            }
            break;
    case 3:
          return;
    default: printf("\nInvalid choice entered. Try again\n");
  }
 }while (1);
}
void create(struct node **head){
 int c, ch;
 struct node *temp, *rear;
 do{
  printf("Enter number: ");
  scanf("%d", &c);
  temp=(struct node *)malloc(sizeof(struct node));
  temp->num=c;
  temp->next=NULL;
  if (*head==NULL){
    *head=temp;
  }
  else{
   rear->next=temp;
  }
  rear=temp;
  printf("Do you wish to continue [1/0]: ");
  scanf("%d", &ch);
 }while (ch!=0);
 printf("\n");
}
void display(struct node *head){
 while (head!=NULL){
  printf("%d\t", head->num);
  head=head->next;
 }
 printf("\n");
}
void release(struct node **head){
 struct node *temp;
 while ((*head)!=NULL){
  temp=*head;
  (*head)=(*head)->next;
  free(temp);
 }
```

```
}
```

---

**Problem: 17** Program for reversing a doubly linked-list

```
#include<stdio.h>
#include<stdlib.h>
struct Node{
  int data;
  struct Node *next;
  struct Node *prev;
};
void reverse(struct Node **head_ref){
     struct Node *temp=NULL;
     struct Node *current=*head_ref;
     while (current!=NULL){
       temp=current->prev;
       current->prev=current->next;
       current->next=temp;
       current=current->prev;
     }
     if(temp!=NULL)
        *head_ref=temp->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->prev=NULL;

    new_node->next=(*head_ref);

    if((*head_ref)!=NULL)
      (*head_ref)->prev=new_node ;

    (*head_ref)=new_node;
}
void printList(struct Node *node){
  while(node!=NULL){
   printf("%d ", node->data);
   node=node->next;
  }
}
int main(){
 struct Node* head=NULL;
  push(&head, 2);
  push(&head, 4);
  push(&head, 8);
  push(&head, 10);
  printf("\n Original Linked list ");
  printList(head);
  reverse(&head);
```

```
47.  printf("\n Reversed Linked list ");
48.  printList(head);
49.  printf("\n");
50.  return 0;
51.}
```

---

**Problem: 18** Program to implement growable stack using arrays

```
1.  #include <stdio.h>
2.  #include <stdlib.h>
3.  struct node{
4.      int data;
5.      struct node *link;
6.  }*top = NULL;
7.  #define MAX 5
8.  void push();
9.  void pop();
10. void empty();
11. void stack_full();
12. int stack_count();
13. void destroy();
14. void print_top();
15. int main(){
16.     int choice;
17.     while (1){
18.         printf("1. push an element \n");
19.         printf("2. pop an element \n");
20.         printf("3. check if stack is empty \n");
21.         printf("4. check if stack is full \n");
22.         printf("5. count/display elements present in stack \n");
23.         printf("6. empty and destroy stack \n");
24.         printf("7. Print top of the stack \n");
25.         printf("8. exit \n");
26.         printf("Enter your choice \n");
27.         scanf("%d",&choice);
28.         switch (choice){
29.         case 1:
30.             push();
31.             break;
32.         case 2:
33.             pop();
34.             break;
35.         case 3:
36.             empty();
37.             break;
38.         case 4:
39.             stack_full();
40.             break;
41.         case 5:
42.             stack_count();
```

```
            break;
        case 6:
            destroy();
            break;
        case 7:
            print_top();
            break;
        case 8:
            return 0;
        default:
            printf("wrong choice\n");
        }
    }
}
void push(){
    int val,count;
    struct node *temp;
    temp = (struct node*)malloc(sizeof(struct node));
    count = stack_count();
    if (count <= MAX - 1){
        printf("\nEnter value which you want to push into the
stack :\n");
        scanf("%d",&val);
        temp->data = val;
        temp->link = top;
        top = temp;
    }
    else
        printf("WARNING: STACK FULL\n");
}
void pop(){
    struct node *temp;
    if (top==NULL)
        printf("**Stack is empty**\n");
    else{
        temp = top;
        printf("Value popped out is %d \n",temp->data);
        top = top->link;
        free(temp);
    }
}
void empty(){
    if (top == NULL)
        printf("STACK IS EMPTY\n");
    else
        printf("elements are present, stack is not empty \n");
}
void stack_full(){
    int count;
    count = stack_count();
```

```
    if (count==MAX){
        printf("stack is full\n");
    }
    else
        printf("stack is not full \n");
}
int stack_count(){
    int count = 0;
         struct node *temp;
         temp = top;
         while (temp!=NULL){
             printf(" %d\n",temp->data);
             temp = temp->link;
             count++;
         }
         return count;
     }
     int st_count(){
         int count = 0;
         struct node *temp;
         temp = top;
         while (temp!=NULL){
             temp = temp->link;
             count++;
         }
         return count;
     }
     void destroy(){
         struct node *temp;
         temp = top;
         while (temp!=NULL){
             pop();
             temp = temp->link;
         }
         printf("stack destroyed\n");
     }
     void print_top(){
         if (top == NULL)
             printf("\n**Top is not available for an EMPTY
  stack**\n");
         else
             printf("\nTop of the stack is %d \n",top->data);
     }
```

---

**Problem: 19** Program to implement stack with the help of linked-list

```
#include <iostream>
#include <stdlib.h>
#include <stdio.h>
using namespace std;
```

```
struct node{
 int item;
 struct node *next;
};
struct node * Insert(struct node *root,int data){
struct node *n;
n=(struct node*)malloc(sizeof(struct node));
n->next=NULL;
n->item=data;
if(root==NULL){
 root=n;
}
else{
 struct node *par;
 par=root;
  while(par->next!=NULL){
    par=par->next;
  }
  par->next=n;
}
return(root);
}
struct node * Delete(struct node *root){
struct node *p, *q;
p=q=root;
while(p->next!=NULL){
 q=q->next;
 p=q->next;
}
q->next=NULL;
printf("Popped Item is :%d\n",p->item);
free(p);
return(root);
}
struct node* Traverse (struct node *root){
struct node *tar;
tar=root;
while(tar!=NULL){
  printf("%d ",tar->item);
 tar=tar->next;
}
return(root);
}
int main(){
int choice,data;
struct node *root=NULL;
while(true){
printf("\nEnter your choice\n1: PUSH\n2: POP\n3: OVERVIEW\n4:
EXIT\n");
scanf("%d",&choice);
```

```
printf("\n");
switch(choice){
case 1:
      printf("Enter Data: ");
      scanf("%d",&data);
      root=Insert(root,data);
      break;
case 2:
      root=Delete(root);
      break;
case 3:
      root=Traverse(root);
      break;
case 4:
      exit(0);
}
}
return 0;
}
```

---

**Problem: 20** Program for reversing elements of the stack

```
#include <stdio.h>
#include <stdlib.h>
struct node{
  int a;
  struct node *next;
};
void generate(struct node **);
void display(struct node *);
void stack_reverse(struct node **);
void Delete(struct node **);
int main(){
  struct node *head=NULL;
  generate(&head);
  printf("\nThe sequence of contents in stack\n");
  display(head);
  printf("\nInversing the contents of the stack\n");
  stack_reverse(&head);
  printf("\nThe contents in stack after reversal\n");
  display(head);
  delete(&head);
  return 0;
}
void stack_reverse(struct node **head){
  struct node *temp, *prev;
  if (*head==NULL){
    printf("Stack does not exist\n");
  }
  else if ((*head)->next==NULL){
```

```
     printf("Single node stack reversal brings no difference\n");
  }
  else if ((*head)->next->next==NULL){
    (*head)->next->next=*head;
    *head=(*head)->next;
    (*head)->next->next=NULL;
  }
  else{
    prev=*head;
    temp=(*head)->next;
    *head=(*head)->next->next;
    prev->next=NULL;
    while ((*head)->next!=NULL){
      temp->next=prev;
      prev=temp;
      temp=*head;
      *head=(*head)->next;
    }
    temp->next=prev;
    (*head)->next=temp;
  }
}
void display(struct node *head){
  if (head!=NULL){
    printf("%d  ", head->a);
    display(head->next);
  }
}
void generate(struct node **head){
 int num, i;
 struct node *temp;
 printf("Enter length of list: ");
 scanf("%d", &num);
 for (i=num; i>0; i--){
   temp=(struct node *)malloc(sizeof(struct node));
   temp->a=i;
   if (*head==NULL){
       *head=temp;
       (*head)->next=NULL;
   }
   else{
      temp->next=*head;
      *head=temp;
   }
  }
}
void Delete(struct node **head){
 struct node *temp;
 while (*head!=NULL){
  temp=*head;
```

```
  *head=(*head)->next;
  free(temp);
 }
}
```

---

**Problem: 21** Program to convert an infix expression to postfix expression using stacks

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
struct Stack{
      int top;
      unsigned capacity;
      int* array;
};
struct Stack* createStack( unsigned capacity ){
      struct Stack* stack = (struct Stack*) malloc(sizeof(struct
Stack));
      if (!stack)
            return NULL;
      stack->top = -1;
      stack->capacity = capacity;
      stack->array = (int*) malloc(stack->capacity * sizeof(int));
      if (!stack->array)
            return NULL;
      return stack;
}
int isEmpty(struct Stack* stack){
      return stack->top == -1 ;
}
char peek(struct Stack* stack){
      return stack->array[stack->top];
}
char pop(struct Stack* stack){
      if (!isEmpty(stack))
            return stack->array[stack->top--] ;
      return '$';
}
void push(struct Stack* stack, char op){
      stack->array[++stack->top] = op;
}
int isOperand(char ch){
      return (ch >= 'a' && ch <= 'z') || (ch >= 'A' && ch <= 'Z');
}
int Prec(char ch){
      switch (ch){
      case '+':
      case '-':
            return 1;
```

```
      case '*':
      case '/':
            return 2;

      case '^':
            return 3;
      }
      return -1;
}


int infixToPostfix(char* exp){
      int i, k;
      struct Stack* stack = createStack(strlen(exp));
      if(!stack)
            return -1 ;
      for (i = 0, k = -1; exp[i]; ++i){
            if (isOperand(exp[i]))
                  exp[++k] = exp[i];

            else if (exp[i] == '(')
                  push(stack, exp[i]);

            else if (exp[i] == ')'){
                  while (!isEmpty(stack) && peek(stack) != '(')
                        exp[++k] = pop(stack);
                  if (!isEmpty(stack) && peek(stack) != '(')
                        return -1;
                  else
                        pop(stack);
            }
            else{
                  while (!isEmpty(stack) && Prec(exp[i]) <=
Prec(peek(stack)))
                        exp[++k] = pop(stack);
                  push(stack, exp[i]);
            }
      }
      while (!isEmpty(stack))
            exp[++k] = pop(stack );

      exp[++k] = '\0';
      printf( "%sn", exp );
}
int main(){
      char exp[] = "a+b*(c^d-e)^(f+g*h)-i";
      infixToPostfix(exp);
      return 0;
}
```

**Problem: 22** Program to implement stacks using two queues

```
#include <stdio.h>
#include <stdlib.h>
#define QUEUE_EMPTY_MAGIC 0xdeadbeef

typedef struct _queue_t{
    int *arr;
    int rear, front, count, max;
}queue_t;

queue_t *queue_allocate(int n);
void queue_insert(queue_t * q, int v);
int queue_remove(queue_t * q);
int queue_count(queue_t * q);
int queue_is_empty(queue_t * q);

void stack_push(queue_t * q, int v){
  queue_insert(q, v);
}

int stack_pop(queue_t * q){
 int i, n=queue_count(q);
 int removed_element;
 for(i=0; i<(n-1); i++){
   removed_element=queue_remove(q);
   queue_insert(q, removed_element);
 }
 removed_element=queue_remove(q);
 return removed_element;
}

int stack_is_empty(queue_t * q){
  return queue_is_empty(q);
}

int stack_count(queue_t * q){
 return queue_count(q);
}

int queue_count(queue_t * q){
  return q->count;
}

queue_t * queue_allocate(int n) {
 queue_t *queue;
 queue=(queue_t*)malloc(sizeof(queue_t));
 if (queue==NULL)
   return NULL;
 queue->max=n;
 queue->arr=(int*)malloc(sizeof(int) * n);
```

```
 queue->rear=n - 1;
 queue->front=n - 1;
 return queue;
}

void queue_insert(queue_t * q, int v){
 if (q->count==q->max)
   return;
 q->rear=(q->rear + 1) % q->max;
 q->arr[q->rear]=v;
 q->count++;
}

int queue_remove(queue_t * q){
 int retval;
 if (q->count==0)
    return QUEUE_EMPTY_MAGIC;
 q->front=(q->front + 1) % q->max;
 retval=q->arr[q->front];
 q->count--;
 return retval;
}

int queue_is_empty(queue_t * q){
 return (q->count==0);
}

void queue_display(queue_t * q){
 int i=(q->front + 1) % q->max, elements=queue_count(q);
 while (elements--){
   printf("[%d], ", q->arr[i]);
   i=(i>=q->max) ? 0 : (i + 1);
 }
}

int main(){
 queue_t *q;
 int x, select;
 q=queue_allocate(128);
 do
{
  printf("\n[1] Push\n[2] Pop\n[0] Exit");
  printf("\nChoice: ");
  scanf(" %d", &select);
  switch (select) {
  case 1:
         printf("\nEnter value to Push:");
         scanf(" %d", &x);
         stack_push(q, x);
```

```
        printf("\n\n*********************************\nCurrent
Queue:\n");
             queue_display(q);
             printf("\n\nPushed Value: %d", x);
             printf("\n*********************************\n");
            break;
      case 2:
             x=stack_pop(q);
printf("\n\n\n\n*********************************\nCurrent
Queue:\n");
             queue_display(q);
             if (x==QUEUE_EMPTY_MAGIC)
                 printf("\n\nNo values removed");
             else
                 printf("\n\nPopped Value: %d", x);
             printf("\n*********************************\n");
            break;
      case 0:
             printf("\nQutting.\n");
            return 0;
      default:
             printf("\nQutting.\n");
            return 0;
      }
     }while (1);
    return 0;
    }
```

---

**Problem: 23** Program to use stack for postfix expression evaluation

```
#include <stdio.h>
#include <ctype.h>
int stack[100];
int top=-1 ;
void push(int item){
 if(top>=100-1){
   printf("stack overflow");
   return;
 }
 else{
  top++;
  stack[top]=item;
 }
}
int pop(){
 int item;
 if(top==-1){
  printf("stack underflow");
 }
```

```
 else{
  item=stack[top];
  top--;
  return item;
  }
}
void EvalPostfix(char postfix[]){
 int i;
 char ch;
 int val;
 int A, B ;
 for (i=0 ; postfix[i]!=')'; i++){
  ch=postfix[i];
  if (isdigit(ch)){
   push(ch - '0');
  }
  else if (ch=='+' || ch=='-' || ch=='*' || ch=='/'){
   A=pop();
   B=pop();
   switch (ch){
    case '*':
      val=B * A;
      break;
    case '/':
      val=B / A;
      break;
    case '+':
      val=B + A;
        break;
    case '-':
      val=B - A;
      break;
    }
    push(val);
   }
  }
 printf( " \n Result of expression evaluation : %d \n", pop()) ;
}
int main(){
 int i ;
 char postfix[100];
 printf("ASSUMPTION: There are only four operators(*, /, +, -) in an
expression and operand is single digit only.\n");
 printf( " \nEnter postfix expression,\npress right parenthesis ')'
for end expression : ");
 for (i=0 ; i<=99; i++){
  scanf("%c", &postfix[i]);
  if ( postfix[i]==')' ){
   break;
  }
```

```
}
EvalPostfix(postfix);
return 0;
}
```

---

**Problem: 24** Program to check if two stacks are identical or not

```
#include <stdio.h>
#include <iostream>
#include <stdlib.h>
class stack{
      public:
      int arr[100];
      int top;
      int start , full ;
      int valid;
        public:
      stack(){
         top = start = -1;
         full = 100 ;
       valid = 1;
      }
        void push(int a);
      int pop();
};
void stack::push(int a){
 if(top == full-1 ){
  printf("\n the stack is full cannot enter any more values");
 }
 else{
  top++ ;
  arr[top] = a ;
 }
}
int stack::pop(){
 int i;
 if(top == start){
  printf("\n the stack is empty");
  printf("\n A garbage value is printed");
  valid = 0;
 }
 else{
  i = arr[top];
  top--;
  valid = 1;
 }
 if(valid == 1){
  return i;
 }
}
```

```
int main(){
      stack s;
      char c , i ;
      int flag = 1 , count = 0;
      printf("\nEnter the number of elements : ");
      printf("\nEnter the elements : ");
      while(1){
      fflush(stdin);
      printf("\nenter :");
      scanf("%c",&c);
            if( c=='('|| c== '{'|| c== '[')
            {
              s.push(c);
              count++;
            }
                if( c== ')'|| c== '}' || c==']')
            {
             i = s.pop();
             count--;
             switch(c){
                   case ')':
                        if( i == '(')
                              flag = 1;
                        else
                              flag = 0;
                        break;
                  case '}':
                        if( i == '{')
                              flag = 1;
                        else
                              flag = 0;
                        break;
                  case ']':
                        if( i == '[')
                              flag = 1;
                        else
                              flag = 0;
                        break;
                  }
            }
            if( flag == 0 || c == 'y')
             break;
                }
       if( flag == 0 || count != 0)
            printf(" not a valid string");
       else
            printf(" a valid string");
       return 0;
}
```

**Problem: 25** Program to implement growable queues using arrays

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 50
void insert();
void deletee();
void display();
int queue_array[MAX];
int rear = - 1;
int front = - 1;
int main(){
    int choice;
    while (1){
        printf("1.Insert element to queue \n");
        printf("2.Delete element from queue \n");
        printf("3.Display all elements of queue \n");
        printf("4.Quit \n");
        printf("Enter your choice : ");
        scanf("%d", &choice);
        switch (choice){
            case 1:
            insert();
            break;
            case 2:
             deletee();
             break;
            case 3:
            display();
            break;
            case 4:
            exit(1);
            default:
            printf("Wrong choice \n");
        }
    }
}
void insert(){
    int add_item;
    if (rear == MAX - 1)
    printf("Queue Overflow \n");
    else{
        if (front == - 1)
                  front = 0;
        printf("Inset the element in queue : ");
        scanf("%d", &add_item);
        rear = rear + 1;
        queue_array[rear] = add_item;
    }
```

```
}
void deletee(){
    if (front == - 1 || front > rear){
        printf("Queue Underflow \n");
        return ;
    }
    else{
        printf("Element deleted from queue is : %d\n",
  queue_array[front]);
        front = front + 1;
    }
}
void display(){
    int i;
    if (front==-1)
        printf("Queue is empty \n");
    else{
        printf("Queue is : \n");
        for (i = front; i <= rear; i++)
            printf("%d ", queue_array[i]);
        printf("\n");
    }
}
```

---

**Problem: 26** Program to implement queues using linked-list

```
#include <stdio.h>
#include <stdlib.h>
struct Node{
    int data;
    struct Node *next;
}*front = NULL,*rear = NULL;
void insert(int);
void deletee();
void display();
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: deletee(); break;
       case 3: display(); break;
```

```
      case 4: return 0;
      default: printf("\nWrong selection!!! Please try again!!!\n");
     }
  }
}
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 deletee(){
  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);
  }
}
```

---

**Problem: 27** Program to implement queues using two stacks

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

```
int pop(struct node** top);
struct queue{
    struct node *stack1;
    struct node *stack2;
};
void enqueue(struct queue *q, int x){
    push(&q->stack1, x);
}
void dequeue(struct queue *q){
    int x;
    if (q->stack1 == NULL && q->stack2 == NULL) {
        printf("queue is empty");
        return;
    }
    if (q->stack2 == NULL){
        while (q->stack1 != NULL){
        x = pop(&q->stack1);
        push(&q->stack2, x);
        }
    }
    x = pop(&q->stack2);
    printf("%d\n", x);
}
void push(struct node** top, int data){
    struct node* newnode = (struct node*) malloc(sizeof(struct node));
        if (newnode == NULL){
            printf("Stack overflow \n");
            return;
        }
    newnode->data = data;
    newnode->next = (*top);
    (*top) = newnode;
}
int pop(struct node** top){
    int buff;
    struct node *t;
    if((*top)== NULL){
        printf("Stack underflow \n");
        return 0;
    }
    else{
        t = *top;
        buff = t->data;
        *top = t->next;
        free(t);
        return buff;
    }
}
void display(struct node *top1,struct node *top2){
    while (top1 != NULL){
```

```
        printf("%d\n", top1->data);
        top1 = top1->next;
    }
    while (top2 != NULL){
        printf("%d\n", top2->data);
        top2 = top2->next;
    }
}
int main(){
    struct queue *q = (struct queue*)malloc(sizeof(struct queue));
    int f = 0, a;
    char ch = 'y';
    q->stack1 = NULL;
    q->stack2 = NULL;
    while (ch == 'y'||ch == 'Y') {
        printf("Enter your choice\n1.add to queue\n2.remove from
queue\n3.display\n4.exit\n");
        scanf("%d", &f);
        switch(f){
            case 1 : printf("enter the element to be added to
queue\n");
                     scanf("%d", &a);
                     enqueue(q, a);
                     break;
            case 2 : dequeue(q);
                     break;
            case 3 : display(q->stack1, q->stack2);
                     break;
            case 4 : exit(1);
                     break;
            default : printf("invalid\n");
                      break;
        }
    }
}
```

---

**Problem: 28** Program to implement double-ended queues with insert and delete functions

```
#include <stdio.h>
#include <stdlib.h>
typedef struct dequeue{
    int data[30];
    int rear,front;
}dequeue;
void initialize(dequeue *p);
int empty(dequeue *p);
int full(dequeue *p);
void enqueueR(dequeue *p,int x);
void enqueueF(dequeue *p,int x);
int dequeueF(dequeue *p);
```

```
int dequeueR(dequeue *p);
void print(dequeue *p);
int main(){
    int i,x,op,n;
    dequeue q;
    initialize(&q);
    do{
printf("\n1.Create\n2.Insert(rear)\n3.Insert(front)\n4.Delete(rear)\n5
.Delete(front)");
        printf("\n6.Print\n7.Exit\n\nEnter your choice:");
        scanf("%d",&op);
        switch(op){
            case 1: printf("\nEnter number of elements:");
                    scanf("%d",&n);
                    initialize(&q);
                    printf("\nEnter the data:");
                    for(i=0;i<n;i++){
                        scanf("%d",&x);
                        if(full(&q)){
                            printf("\nQueue is full!!");
                            exit(0);
                        }
                        enqueueR(&q,x);
                    }
                    break;
            case 2: printf("\nEnter element to be inserted:");
                    scanf("%d",&x);
                    if(full(&q)){
                        printf("\nQueue is full!!");
                        exit(0);
                    }
                    enqueueR(&q,x);
                    break;
            case 3: printf("\nEnter the element to be inserted:");
                    scanf("%d",&x);
                    if(full(&q)){
                        printf("\nQueue is full!!");
                        exit(0);
                    }
                    enqueueF(&q,x);
                    break;
            case 4: if(empty(&q)){
                        printf("\nQueue is empty!!");
                        exit(0);
                    }
                    x=dequeueR(&q);
                    printf("\nElement deleted is %d\n",x);
                    break;
            case 5: if(empty(&q)){
```

```
                        printf("\nQueue is empty!!");
                        exit(0);
                    }
                    x=dequeueF(&q);
                    printf("\nElement deleted is %d\n",x);
                    break;
            case 6: print(&q);
                    break;
            default: break;
        }
    }while(op!=7);
 return 0;
}
void initialize(dequeue *P){
    P->rear=-1;
    P->front=-1;
}
int empty(dequeue *P){
    if(P->rear==-1)
        return(1);
    return(0);
}
int full(dequeue *P){
    if((P->rear+1)%30==P->front)
        return(1);
    return(0);
}
void enqueueR(dequeue *P,int x){
    if(empty(P)){
        P->rear=0;
        P->front=0;
        P->data[0]=x;
    }
    else{
        P->rear=(P->rear+1)%30;
        P->data[P->rear]=x;
    }
}
void enqueueF(dequeue *P,int x){
        if(empty(P)){
            P->rear=0;
            P->front=0;
            P->data[0]=x;
        }
        else{
            P->front=(P->front-1+30)%30;
            P->data[P->front]=x;
        }
     }
     int dequeueF(dequeue *P){
```

```
        int x;
        x=P->data[P->front];
        if(P->rear==P->front)
            initialize(P);
        else
            P->front=(P->front+1)%30;
        return(x);
    }
    int dequeueR(dequeue *P){
        int x;
        x=P->data[P->rear];
        if(P->rear==P->front)
            initialize(P);
        else
            P->rear=(P->rear-1+30)%30;
        return(x);
    }
    void print(dequeue *P){
        if(empty(P)){
            printf("\nQueue is empty!!");
            exit(0);
        }
        int i;
        i=P->front;
        while(i!=P->rear){
            printf("\n%d",P->data[i]);
            i=(i+1)%30;
        }
        printf("\n%d\n",P->data[P->rear]);
    }
```

---

**Problem: 29** Program to store binary trees in a 1D array

```
#include <stdio.h>
#include <stdlib.h>
int levelorder[50], size;
void display(){
      printf("The binary tree in level order as inserted was :\n");
      for(int i=0; i<size; i++){
            printf("%d\t",levelorder[i]);
      }
      printf("\n");
      return;
}
void insert(int index, int data){
      size++;
      for(int i=size-1; i>=index; i--)
            levelorder[i+1]=levelorder[i];
      levelorder[index]=data;
      return;
```

```
}
void delete(int data){
      int i=0, count=0;
      for(int j=0; j<size; j++){
            if(levelorder[j]==data){
                  count++;
            }
      }
      if(count==0){
            printf("Data is not found in the tree !\n");
            return;
      }
      while(i<size){
            if(levelorder[i]==data){
                  for(int j=i; j<size; j++){
                        levelorder[j]=levelorder[j+1];
                  }
                  size--;
            }
            i++;
      }
      return;
}
int main(){
      printf("NOTE: This array representation of Binary trees expects tree to be complete\nOR\nThe data are inserted in level order from left to right COMPLETE.\n");
      int element, i=0;
      printf("Enter the elements in Level order: <0 to stop>: \n");
      while(1){
            scanf("%d",&element);
            if(element==0)
                  break;
            else
                  levelorder[i++]=element;
      }
      size=i;
      display();
      printf("Inserting value 5 at index 2, the tree will become as: \n");
      insert(2, 5);
      display();
      printf("Deleting element 5.........\n");
      delete(5);
      display();
      return 0;
}
```

---

**Problem: 30** Program to store binary trees in linked list

```
#include <stdio.h>
#include <stdlib.h>
struct node{
	int data;
	struct node *left;
	struct node *right;
};
struct node *insert(int data){
	struct node* temp=(struct node *)malloc(sizeof(struct node));
	temp->data=data;
	temp->left=NULL;
	temp->right=NULL;
	return temp;
}
void inorder(struct node* root){
	if(root==NULL){
		return;
	}
	else{
		inorder(root->left);
		printf("%d\t",root->data);
		inorder(root->right);
	}
}
void preorder(struct node* root){
	if(root==NULL){
		return;
	}
	else{
		printf("%d\t",root->data);
		preorder(root->left);
		preorder(root->right);
	}
}
int main(){
	struct node *root=insert(5);
	root->left=insert(3);
	root->right=insert(6);
	(root->left)->left=insert(2);
	(root->left)->right=insert(3);
	printf("The data of ((root->left)->left) is %d\n",((root->left)->left)->data);
	inorder(root);
	printf("\n");
	preorder(root);
	printf("\n");
	return 0;
}
```

**Problem: 31** Program for fnding an element given its key in a 1D array implementation of binary search trees

```
#include <stdio.h>
#include <stdlib.h>
int levelorder[50], size;
void display(){
      printf("The binary tree in level order as inserted was :\n");
      for(int i=0; i<size; i++){
            printf("%d\t",levelorder[i]);
      }
      printf("\n");
      return;
}
void insert(int index, int data){
      size++;
      for(int i=size-1; i>=index; i--)
            levelorder[i+1]=levelorder[i];
      levelorder[index]=data;
      return;
}
void delete(int data){
      int i=0, count=0;
      for(int j=0; j<size; j++){
            if(levelorder[j]==data){
                  count++;
            }
      }
      if(count==0){
            printf("Data is not found in the tree !\n");
            return;
            }
      while(i<size){
            if(levelorder[i]==data){
                  for(int j=i; j<size; j++){
                        levelorder[j]=levelorder[j+1];
                  }
                  size--;
            }
            i++;
      }
      return;
}
void createtree(){
      int element, i=0;
      printf("Enter the elements in Level order: <0 to stop>: \n");
      while(1){
            scanf("%d",&element);
            if(element==0)
                  break;
            else
```

```
                levelorder[i++]=element;
        }
        size=i;
}
void search(int key){
        int k=0, counter=0;
        while(k<size){
                if(levelorder[k++]==key){
                        printf("Key found at index %d in levelorder traversal
of given binary tree\n",k-1);
                        counter++;
                }
        }
        if(counter==0){
                printf("KEY not found !");
                return;
        }
}
int main(){
        int searchkey=0;
        printf("NOTE: This array representation of Binary trees expects
tree to be complete\nOR\nThe data are inserted in level order from
left to right COMPLETE.\n");
        createtree();
        printf("Enter the key you want to search: \n");
        scanf("%d",&searchkey);
        search(searchkey);
        display();
        return 0;
}
```

---

**Problem: 32** Program for fnding the element given its key in a linked list implementation of BST using recursion

```
#include <stdio.h>
#include <stdlib.h>
struct node{
        int data;
        struct node *left;
        struct node *right;
};
struct node* newnode(int data){
        struct node* temp=(struct node*)malloc(sizeof(struct node));
        temp->data=data;
        temp->left=NULL;
        temp->right=NULL;
        return temp;
}
struct node* insert(struct node* root, int data){
        if(root==NULL){
```

```
            root=newnode(data);
            return root;
      }
      else{
            if(data<=(root->data)){
                  root->left=insert(root->left, data);
            }
            else{
                  root->right=insert(root->right, data);
            }
      }
      return root;
}
struct node *search(struct node* temp, int key){
      if(temp==NULL||temp->data==key){
            return temp;
      }
      else{
            if(temp->data>key)
                  search(temp->left, key);
            else
                  search(temp->right, key);
      }
}
int main(){
      int sea, insertdata, rootdata;
      struct node* temp;
      struct node* root=NULL;
      printf("Enter the root data of BST: ");
      scanf("%d",&rootdata);
      root=insert(root, rootdata);
      do{
            printf("Enter the data to enter in the BST<0 to stop>: ");
            scanf("%d",&insertdata);
            insert(root, insertdata);
      }while(insertdata!=0);
      printf("Enter the element to search in the BST:\n");
      scanf("%d",&sea);
      temp=search(root, sea);
      if(temp==NULL)
            printf("Data not found\n");
      else
            printf("Element found: %d\n",temp->data);
      return 0;
}
```

---

**Problem: 33** Program to fnd the maximum depth of the BST

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

```
struct node{
	int data;
	struct node* left;
	struct node* right;
};
struct node* newnode(int data){
	struct node* Node=(struct node *)malloc(sizeof(struct node));
	Node->data=data;
	Node->left=NULL;
	Node->right=NULL;
	return Node;
}
struct node* insert(struct node* root, int data){
	if(root==NULL){
		root=newnode(data);
		return root;
	}
	else{
		if(data<=(root->data)){
			root->left=insert(root->left, data);
		}
		else{
			root->right=insert(root->right, data);
		}
	}
	return root;
}
int maxdepth(struct node* root){
	int rdepth, ldepth;
	if(root==NULL){
		return 0;
	}
	else{
		rdepth=maxdepth(root->right);
		ldepth=maxdepth(root->left);
		if(rdepth>ldepth){
			return (rdepth+1);
		}
		else{
			return (ldepth+1);
		}
	}
}
int main(){
	int option, data;
	struct node* root=NULL;
	do{
		printf("1. Insert\t2. Find Maxdepth\t3. Exit\n");
		printf("Enter the option: ");
		scanf("%d",&option);
```

```
            switch(option){
                  case 1:
                        printf("\nEnter the data to enter in BST: ");
                        scanf("%d",&data);
                        root=insert(root, data);
                        break;
                  case 2:
                        printf("The maximum depth of the BST is: %d\n",
maxdepth(root));
                        break;
                  case 3:
                        return 0;
                  default:
                        printf("Err...  Please enter correct
option !!!\n");
            }
      }while(option!=3);
      return 0;
}
```

---

**Problem: 34** Program to traverse the BST in Inorder, Preorder, Postorder with and without recursion

```
#include <stdio.h> //Using recursion
#include <stdlib.h>
struct node{
      int data;
      struct node* left;
      struct node* right;
};
struct node* newnode(int data){
      struct node* Node=(struct node *)malloc(sizeof(struct node));
      Node->data=data;
      Node->left=NULL;
      Node->right=NULL;
      return Node;
}
struct node* insert(struct node* root, int data){
      if(root==NULL){
            root=newnode(data);
            return root;
      }
      else{
            if(data<=(root->data)){
                  root->left=insert(root->left, data);
            }
            else{
                  root->right=insert(root->right, data);
            }
      }
      return root;
```

```
}
void inorder(struct node* root){
      if(root==NULL){
            return;
      }
      else{
            inorder(root->left);
            printf("%d\t",root->data);
            inorder(root->right);
      }
}
void preorder(struct node* root){
      if(root==NULL){
            return;
      }
      else{
            printf("%d\t",root->data);
            preorder(root->left);
            preorder(root->right);
      }
}
void postorder(struct node* root){
      if(root==NULL){
            return;
      }
      else{
            preorder(root->left);
            preorder(root->right);
            printf("%d\t",root->data);
      }
}
int main(){
      struct node *root=NULL;
      int rootdata, insertdata;
      printf("Enter the root data for BST: ");
      scanf("%d",&rootdata);
      root=insert(root, rootdata);
      do{
            printf("Enter the data to insert in BST<0 to exit>: ");
            scanf("%d",&insertdata);
            root=insert(root, insertdata);
      }while(insertdata!=0);
      printf("The Inorder traversal of BST is\n");
      inorder(root);
      printf("\nThe Pre-order traversal of BST is\n");
      preorder(root);
      printf("\nThe Post-order traversal of BST is\n");
      postorder(root);
      printf("\n");
      return 0;
```

```
}
```

```
#include <stdio.h> //Not using Recursion
#include <stdlib.h>
struct node{
      int data;
      struct node* left;
      struct node* right;
};
struct node* root=NULL;
struct node* current;
struct node stack[100];
int top=-1;
void push(struct node data){
     if(top==99){
           printf("stack overflow!\n");
           return;
     }
     else{
           top++;
           stack[top]=data;
     }
}
int pop(){
     struct node* key;
     if(top==-1){
           printf("stack underflow\n");
           return 0;
     }
     else{
           key=&stack[top];
           top--;
     }
     return (key->data);
}
struct node* popstack(){
     struct node* key=&stack[top];
     top--;
     return key;
}
int isempty(){
     if(top==-1){
           return 1;
     }
     else{
           return 0;
     }
}
struct node* newnode(int data){
     struct node* temp=(struct node*)malloc(sizeof(struct node));
```

```
	temp->data=data;
	temp->left=NULL;
	temp->right=NULL;
	return temp;
}
struct node* insert(struct node* root, int data){
	if(root==NULL){
		root=newnode(data);
		return root;
	}
	else{
		if(data<=root->data){
			root->left=insert(root->left, data);
		}
		else{
			root->right=insert(root->right, data);
		}
		return root;
	}
}
void inorder(struct node *root){
	struct node *p;
	p=root;
	printf("The In-order traversal of BST is:\n");
	while(1){
		while(p!=NULL){
			top++;
			stack[top]=*p;
			p=p->left;
		}
		if(top==-1)
			break;
		p=&stack[top];
		top--;
		printf("%d\t",p->data);
		p=p->right;
	}
}
void preorder(struct node *root){
	int flag=0;
	struct node *p=root;
	printf("The Pre-order traversal of BST is:\n");
	while(1){
		while(p!=NULL){
			printf("%d\t",p->data);
			top++;
			stack[top]=*p; //push element onto stack
			if(p->left==NULL && p->right==NULL)
				flag=1; //if left child is present for current
node set flag to 1
```

```
                p=p->left;
          }
                if(flag==1) //leaf node has been inserted into stack
   it so pop it out
                {
                      p=&stack[top];
                      top--;
                      flag=0;
                }
                if(top==-1)
                      break;
                p=&stack[top]; //get parent node from stack
                top--;
                p=p->right; //now traverse to the right of parent
          }
    }
    void postorder(struct node* root){
          if(root==NULL)
                return;
          do{
              while(root){
                 if (root->right)
                       push(*root->right);
                 push(*root);
                      root = root->left;
                    }
              root=popstack();
              if(root->right && popstack() == root->right){
                        pop();  // remove right child from stack
                  push(*root);  // push root back to stack
                  root = root->right; // change root so that the
   right
                 }
                    else{
                      printf("%d ", root->data);
                      root = NULL;
                 }
          } while (!isempty());
    }
    int main(){
          int insertdata, choice;
          do{
                printf("\nEnter the data to insert in BST<0 to exit>:
   ");
                scanf("%d",&insertdata);
                root=insert(root, insertdata);
          }while(insertdata!=0);
          do{
                printf("1. In-order\t2. Pre-order\t3. Post-order\t4.
   Exit\n");
                printf("Enter your choice: ");
```

```
                scanf("%d",&choice);
                switch(choice){
                      case 1:
                            inorder(root);
                            printf("\n");
                            break;
                      case 2:
                            preorder(root);
                            printf("\n");
                            break;
                      case 3:
                            postorder(root);
                            printf("\n");
                            break;
                      case 4:
                            return 0;
                      default:
                            printf("Err. Please enter correct
  choice.\n");
                            break;
                }
          }while(choice!=4);
          return 0;
     }
```

---

**Problem: 35** Program to create its corresponding BST from a given unsorted array and print all the elements in sorted order (i.e. inorder traversal)

```
#include <stdio.h>
#include <stdlib.h>
int arr[500];
struct node{
      int data;
      struct node* left;
      struct node* right;
};
struct node* root=NULL;
struct node* newnode(int data){
     struct node* temp=(struct node*)malloc(sizeof(struct node));
     temp->data=data;
     temp->left=NULL;
     temp->right=NULL;
     return temp;
}
struct node* insert(struct node* root, int data){
     if(root==NULL){
           root=newnode(data);
           return root;
     }
     else{
```

```
            if(data<=root->data){
                  root->left=insert(root->left, data);
            }
            else{
                  root->right=insert(root->right, data);
            }
            return root;
      }
}
void inorder(struct node* root){
      if(root==NULL){
            return;
      }
      else{
            inorder(root->left);
            printf("%d\t",root->data);
            inorder(root->right);
      }
}
int main(){
      int i,n;
      printf("Enter the size of array: ");
      scanf("%d",&n);
      printf("\nEnter the elements of unsorted array :\n");
      for(i=0; i<n; i++){
                  scanf("%d",&arr[i]);
                  root=insert(root, arr[i]);
      }
      printf("The inorder traversal of given unsorted array converted
to BST is :\n");
      inorder(root);
      printf("\n");
      return 0;
}
```

---

**Problem: 36** Program to implement graphs using adjacency matrix, incidence matrix, adjacency list and direct representation

```
#include <stdio.h>
#include <stdlib.h>
struct AdjListNode{
    int dest;
    struct AdjListNode* next;
};
struct AdjList{
    struct AdjListNode *head;
};
struct Graph{
    int V;
    struct AdjList* array;
```

```
};
struct AdjListNode* newAdjListNode(int dest){
    struct AdjListNode* newNode =
            (struct AdjListNode*) malloc(sizeof(struct AdjListNode));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}
struct Graph* createGraph(int V){
    struct Graph* graph = (struct Graph*) malloc(sizeof(struct
Graph));
    graph->V = V;
    graph->array = (struct AdjList*) malloc(V * sizeof(struct
AdjList));
    int i;
    for (i = 0; i < V; ++i)
        graph->array[i].head = NULL;
    return graph;
}
void addEdge(struct Graph* graph, int src, int dest){
    struct AdjListNode* newNode = newAdjListNode(dest);
    newNode->next = graph->array[src].head;
    graph->array[src].head = newNode;
    newNode = newAdjListNode(src);
    newNode->next = graph->array[dest].head;
    graph->array[dest].head = newNode;
}
void printGraph(struct Graph* graph){
    int v;
    for (v = 0; v < graph->V; ++v){
        struct AdjListNode* pCrawl = graph->array[v].head;
        printf("\n Adjacency list of vertex %d\n head ", v);
        while (pCrawl){
            printf("-> %d", pCrawl->dest);
            pCrawl = pCrawl->next;
        }
        printf("\n");
    }
}
int main(){
    int V = 5;
    struct Graph* graph = createGraph(V);
    addEdge(graph, 0, 1);
    addEdge(graph, 0, 4);
    addEdge(graph, 1, 2);
    addEdge(graph, 1, 3);
    addEdge(graph, 1, 4);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 4);
    printGraph(graph);
```

```
    return 0;
}
```

---

**Problem: 37** Program to traverse the graph in Depth-first order on an adjacent matrix implementation of graph

```
#include <stdio.h>
void DFS(int);
int G[10][10],visited[10], n;
int main(){
      int i,j;
      printf("Enter number of vertices:");
      scanf("%d",&n);
      printf("\nEnter adjecency matrix of the graph:");
      for(i=0;i<n;i++)
           for(j=0;j<n;j++)
                scanf("%d",&G[i][j]);
      for(i=0;i<n;i++)
             visited[i]=0;
      DFS(0);
}
void DFS(int i){
      int j;
      printf("\n%d",i);
      visited[i]=1;
      for(j=0;j<n;j++)
        if(!visited[j]&&G[i][j]==1)
            DFS(j);
}
```

---

**Problem: 38** Program to traverse the graph in Breadth-first order on an adjacent matrix implementation of graph

```
#include <stdio.h>
#include <stdlib.h>
#define initial 1
#define waiting 2
#define visited 3
int n; int adj[50][50]; int state[50];
void create_graph();
void BF_Traversal();
void BFS(int v);
int queue[50], front = -1,rear = -1;
void insert_queue(int vertex);
int delete_queue();
int isEmpty_queue();
int main(){
    create_graph();
    BF_Traversal();
```

```
    return 0;
}
void BF_Traversal(){
    int v;
    for(v=0; v<n; v++)
        state[v] = initial;
    printf("Enter the starting vertex for BFS: ");
    scanf("%d", &v);
    BFS(v);
}
void BFS(int v){
    int i;
    insert_queue(v);
    state[v]=waiting;
    while(!isEmpty_queue()){
        v = delete_queue( );
        printf("%d ",v);
        state[v] = visited;
        for(i=0; i<n; i++){
            if(adj[v][i] == 1 && state[i] == initial){
                insert_queue(i);
                state[i] = waiting;
            }
        }
    }
    printf("\n");
}
void insert_queue(int vertex){
    if(rear==49)
        printf("Queue overflow!\n");
    else{
        if(front == -1)
            front = 0;
        rear=rear+1;
        queue[rear] = vertex ;
    }
}
int isEmpty_queue(){
    if(front == -1 || front > rear)
        return 1;
    else
        return 0;
}
int delete_queue(){
    int delete_item;
    if(front == -1 || front > rear){
        printf("Queue Underflow\n");
        exit(1);
    }
    delete_item = queue[front];
```

```
    front = front+1;
    return delete_item;
}
void create_graph(){
    int count,max_edge,origin,destin;
    printf("Enter number of vertices : ");
    scanf("%d",&n);
    max_edge = n*(n-1);
    for(count=1; count<=max_edge; count++){
        printf("Enter edge %d( -1 -1 to quit ) : ",count);
        scanf("%d %d",&origin,&destin);
        if((origin == -1) && (destin == -1))
            break;
        if(origin>=n || destin>=n || origin<0 || destin<0){
            printf("Invalid edge!\n");
            count--;
        }
        else{
            adj[origin][destin] = 1;
        }
    }
}
```

---

**Problem: 39** Program for detecting a cycle in an undirected graph using DFS

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct Edge{
    int src, dest;
};
struct Graph{
      int V, E;
      struct Edge* edge;
};
struct Graph* createGraph(int V, int E){
    struct Graph* graph =
          (struct Graph*) malloc( sizeof(struct Graph) );
    graph->V = V;
    graph->E = E;
    graph->edge =
        (struct Edge*) malloc(graph->E*sizeof(struct Edge));
    return graph;
}
int find(int parent[], int i){
    if (parent[i] == -1)
        return i;
    return find(parent, parent[i]);
}
void Union(int parent[], int x, int y){
```

```
    int xset = find(parent, x);
    int yset = find(parent, y);
    parent[xset] = yset;
}
int isCycle( struct Graph* graph ){
    int *parent = (int*) malloc( graph->V * sizeof(int) );
    memset(parent, -1, sizeof(int) * graph->V);

    for(int i = 0; i < graph->E; ++i){
        int x = find(parent, graph->edge[i].src);
        int y = find(parent, graph->edge[i].dest);
        if (x == y)
            return 1;
        Union(parent, x, y);
    }
    return 0;
}
int main(){
    int V = 3, E = 3;
    struct Graph* graph = createGraph(V, E);
    graph->edge[0].src = 0;
    graph->edge[0].dest = 1;
    graph->edge[1].src = 1;
    graph->edge[1].dest = 2;
    graph->edge[2].src = 0;
    graph->edge[2].dest = 2;
    if (isCycle(graph))
        printf( "graph contains cycle" );
    else
        printf( "graph doesn't contain cycle" );
    return 0;
}
```

---

**Problem: 40** Program to implement Merge sort recursively.

```
#include <stdio.h>
void merge(int a[], int, int, int);
void sort(int a[],int, int);

int main(){
      int arr[1000], i, n;
      printf("\n Enter the number of elements in the array : ");
      scanf("%d", &n);
      printf("\n Enter the elements of the array: ");
      for(i=0;i<n;i++){
            scanf("%d", &arr[i]);
      }
      sort(arr, 0, n-1);
      printf("\n The sorted array is: \n");
      for(i=0;i<n;i++)
```

```
	printf(" %d\t", arr[i]);
	return 0;
}
void merge(int arr[], int low, int mid, int high){
	int i=low, j=mid+1, index=low, temp[1000], k;
	while((i<=mid) && (j<=high)){
		if(arr[i] < arr[j]){
			temp[index]=arr[i];
			i++;
		}
		else{
			temp[index]=arr[j];
			j++;
		}
		index++;
	}
	if(i>mid){
		while(j<=high){
			temp[index]=arr[j];
			j++;
			index++;
		}
	}
	else{
		while(i<=mid){
			temp[index]=arr[i];
			i++;
			index++;
		}
	}
	for(k=low;k<index;k++)
		arr[k]=temp[k];
}
void sort(int arr[], int low, int high){
	int mid;
	if(low<high){
		mid=(low+high)/2;
		sort(arr, low, mid);
		sort(arr, mid+1, high);
		merge(arr, low, mid, high);
	}
}
```

---

**Problem: 41** Program to implement Quick sort recursively.

```
#include <stdio.h>
int partition(int a[], int low, int high);
void quick_sort(int a[], int low, int high);
int main(){
	int arr[1000], i, n;
```

```
	printf("Enter the number of elements in the array: ");
	scanf("%d", &n);
	printf("\nEnter the elements of the array: ");
	for(i=0;i<n;i++){
		scanf("%d", &arr[i]);
	}
	quick_sort(arr, 0, n-1);
	printf("\nThe sorted array is:\t");
	for(i=0;i<n;i++)
		printf("%d \t", arr[i]);
	return 0;
}
int partition(int a[], int low, int high){
	int left, right, temp, loc, flag;
	loc=left=low;
	right=high;
	flag=0;
	while(flag != 1){
		while((a[loc] <= a[right]) && (loc!=right))
			right--;
		if(loc==right)
			flag =1;
		else if(a[loc]>a[right]){
			temp=a[loc];
			a[loc]=a[right];
			a[right]=temp;
			loc=right;
		}
		if(flag!=1){
			while((a[loc] >= a[left]) && (loc!=left))
				left++;
			if(loc==left)
				flag =1;
			else if(a[loc] <a[left]){
				temp=a[loc];
				a[loc]=a[left];
				a[left]=temp;
				loc=left;
			}
		}
	}
	return loc;
}
void quick_sort(int a[], int low, int high){
	int loc;
	if(low<high){
		loc=partition(a, low, high);
		quick_sort(a, low, loc-1);
		quick_sort(a, loc+1, high);
	}
```

```
}
```

---

**Problem: 42** Program to implement Insertion sort.

```
#include <stdio.h>
void insertion_sort(int arr[], int n){
      int i, j, temp;
      for(i=1; i<n; i++){
            temp=arr[i];
            j=i-1;
            while((temp < arr[j])&&(j>=0)){
                  arr[j+1]=arr[j];
                  j--;
            }
            arr[j+1]=temp;
      }
}
int main(){
      int arr[500], i, n;
      printf("Enter the number of elements in the array: ");
      scanf("%d", &n);
      printf("\nEnter the elements of the array: ");
      for(i=0;i<n;i++){
            scanf("%d", &arr[i]);
      }
      insertion_sort(arr, n);
      printf("\nThe sorted array is: ");
      for(i=0;i<n;i++)
      printf("\t%d", arr[i]);
      return 0;
}
```

---

**Problem: 43** Program to implement Selection sort.

```
#include <stdio.h>
#include <stdlib.h>
int smallest(int arr[], int k, int n);
void selection_sort(int arr[], int n);
int main(){
      int arr[500], i, n;
      printf("Enter the number of elements in the array: ");
      scanf("%d", &n);
      printf("\nEnter the elements of the array: ");
      for(i=0;i<n;i++){
            scanf("%d", &arr[i]);
      }
      selection_sort(arr, n);
      printf("\nThe sorted array is:");
      for(i=0;i<n;i++)
```

```
            printf("\t %d", arr[i]);
}
int smallest(int arr[], int k, int n){
      int pos=k, small=arr[k], i;
      for(i=k+1;i<n;i++){
            if(arr[i]< small){
                  small=arr[i];
                  pos=i;
            }
      }
      return pos;
}
void selection_sort(int arr[],int n){
      int k, pos, temp;
      for(k=0;k<n;k++){
            pos=smallest(arr, k, n);
            temp=arr[k];
      }
}
```

---

**Problem: 44** Program to implement Radix sort.

```
#include <stdio.h>
int largest(int arr[], int n);
void radix_sort(int arr[], int n);
int main(){
      int arr[1000], i, n;
      printf("\n Enter the number of elements in the array: ");
      scanf("%d", &n);
      printf("\n Enter the elements of the array: ");
      for(i=0;i<n;i++){
            scanf("%d", &arr[i]);
      }
      radix_sort(arr, n);
      printf("\n The sorted array is: \n");
      for(i=0;i<n;i++)
            printf(" %d\t", arr[i]);
      return 0;
}
int largest(int arr[], int n){
      int large=arr[0], i;
      for(i=1;i<n;i++){
            if(arr[i]>large)
                  large=arr[i];
      }
      return large;
}
void radix_sort(int arr[], int n){
      int bucket[1000][1000], bucket_count[1000];
      int i, j, k, remainder, NOP=0, divisor=1, large, pass;
```

```
    large=largest(arr, n);
    while(large>0){
        NOP++;
        large/=1000;
    }
    for(pass=0;pass<NOP;pass++){
        for(i=0;i<1000;i++)
            bucket_count[i]=0;
        for(i=0;i<n;i++){
            remainder=(arr[i]/divisor)%1000;
            bucket[remainder][bucket_count[remainder]]=arr[i];

            bucket_count[remainder] += 1;
        }
        i=0;
        for(k=0;k<1000;k++){
            for(j=0;j<bucket_count[k];j++){
                arr[i]=bucket[k][j];
                i++;
            }
        }
        divisor *= 1000;
    }
}
```

---

**Problem: 45** N people have decided to elect a leader by arranging themselves in a circle and eliminating every M th person around the circle, closing ranks as each person drops out. Find which person will be the last one remaining (with
rank 1). [Use Circular Linked List][Josephus problem]

```
#include <stdio.h>
int josephus(int n, int k){
  if(n==1)
    return 1;
  else
    return (josephus(n-1, k)+k-1)%n+1;
}
int main(){
  int n=20;
  int k=2;
  printf("The chosen place is %d", josephus(n, k));
  return 0;
}
```

---

**Problem: 46** Please fnd the value of C, U, B, E, D using the following equation:
(C+U+B+E+D)^3=CUBED, where C, U, B, E, D are integers.

```
#include <stdio.h>
int main(){
      long long int c, u, b, e, d;
```

```
4.       for(c=0; c<=9; c++){
5.             for(u=0; u<=9; u++){
6.                   for(b=0; b<=9; b++){
7.                         for(e=0; e<=9; e++){
8.                               for(d=0; d<=9; d++){
9.
   if((c+u+b+e+d)*(c+u+b+e+d)*(c+u+b+e+d)==(d+e*10+b*100+u*1000+c*10000))
10.                                          printf("The solution is c=%lld
   u=%lld b=%lld e=%lld d=%lld\n",c,u,b,e,d);
11.                               }
12.                         }
13.                   }
14.            }
15.      }
16.      return 0;
17.}
```

---

**Problem: 47** Program to implement Stack in a way such that Push, Pop, and GetMinimum operations take constant time. [Use two stacks]

```
1.  #include<stdio.h>
2.  int main(){
3.       int normstack[100], minstack[100];
4.       int top=-1; int dop=-1; int count=0; int CurrentMin;
5.       int choice; int data;
6.       while(1){
7.             printf("\n1: Push\t2: Pop\t3: Get-minimum\t4: Exit\n");
8.             scanf("%d",&choice);
9.             switch(choice){
10.                  case 1:
11.                        printf("Enter Data to Push: ");
12.                        scanf("%d", &data);
13.                        normstack[++top]= data;
14.                        if(count==0){
15.                              minstack[++dop]= data;
16.                              CurrentMin=data;
17.                        }
18.                        else if(data<CurrentMin)
19.                              CurrentMin=data;
20.                        count++;
21.                        break;
22.                  case 2:
23.                        data= normstack[top--];
24.                        printf("\nPopped Data is %d\n",data);
25.                        if(data==CurrentMin)
26.                        dop--;
27.                                    CurrentMin= minstack[dop];
28.                        break;
29.                  case 3:
```

```
                        printf("\nCurrent Minimum Element in Stack is
   %d\n",CurrentMin);
                        break;
                        case 4:
                        return 0;
            }
      }
}
```

---

**Problem: 48** We are given a list of prices of a stock for N number of days. We need to fnd the span for each day. Span is defned as number of consecutive days before the given day where the price of stock was less than or equal to price at given day. **For example,** {100, 60,70,65, 80, 85} span will be {1, 1, 2, 1, 4, 5}. For frst day span is always 1. In example we can see that for day 2 at 60, there is no day before it where price was less than 60. Hence span is 1 again. For day 3, price at day 2 (60) is less than 70, hence span is 2. Similarly, for day 4 and day 5. Remember days should be consecutive, that why span for day 4 is 1 even though there was a day 2 where price was less than 65. [Stock Span Problem]

```
#include <stdio.h>
void calculateSpan(int price[], int n, int S[]){
      S[0] = 1;
      for (int i = 1; i < n; i++){
            S[i] = 1;
            for (int j = i-1; (j>=0)&&(price[i]>=price[j]); j--)
                  S[i]++;
      }
}
int main(){
    int price[] = {100, 120, 40, 35, 55, 36};
    int n = sizeof(price)/sizeof(price[0]);
    int S[n];

    calculateSpan(price, n, S);
      for (int i = 0; i < n; i++)
            printf("%d ", S[i]);
    return 0;
}
```

---

**End of the assignment :)**