EX. 01 – INSERT AND DELETE OPERATIONS IN AN ARRAY :

CODING : (INSERTION)

```
#include<stdio.h>
int main()
{
	int a[20], pos, i, n, value;
	printf("\nEnter the Number of Elements in the Array : \n");
	scanf("%d", &n);
	printf("\nEnter %d Elements : \n", n);
	for(i = 0; i < n; i++)
		scanf("%d", &a[i]);
	for(i = 0; i < n; i++)
		printf("%d\t", a[i]);
		printf("\n");
		printf("\nEnter the Location where you want to Insert the Element : \n");
		scanf("%d", &pos);
		printf("\nEnter the Value : \n");
		scanf("%d", &value);
	for(i = n-1; i >= pos-1; i--)
		a[i+1] = a[i];
		a[pos-1] = value;
		printf("\nElement in the Array After Insertion : \n");
	for(i = 0; i <= n; i++)
		printf("%d\t", a[i]);
	return 0;
}
```

OUTPUT :

```
Enter the Number of Elements in the Array : 4

Enter 4 Elements :
5
6
4
3

5	6	4	3

Enter the Location where you want to Insert the Element : 2

Enter the Value : 9

Element in the Array After Insertion :

5	9	6	4	3
```

CODING : (DELETION)

```
#include<stdio.h>
int main()
{
	int a[20], pos, i, n;
	printf("Enter the Number of Elements in the Array : \n");
	scanf("%d", &n);
	printf("\nEnter %d Elements : \n", n);
	for(i = 0; i < n; i++)
		scanf("%d", &a[i]);
	for(i = 0; i < n; i++)
		printf("%d\t", a[i]);
		printf("\n");
		printf("\nEnter the Location you want to Delete Element : \n");
		scanf("%d", &pos);
		if(pos >= n+1)
			printf("\nDeletion Not Possible.\n");
		else
		{
			for(i = pos-1; i < n-1; i++)
				a[i] = a[i+1];
				printf("\nElements in the Array After Deletion : \n");
				for(i = 0; i < n-1; i++)
					printf("%d\t", a[i]);
					printf("\n");
		}
	return 0;
}
```

OUTPUT :

```
Enter the Number of Elements in the Array : 4

Enter 4 Elements :
5
6
8
3

5	6	8	3

Enter the Location you want to Delete Element : 2

Elements in the Array After Deletion :

5	8	3
```

EX. 02 – PUSH AND POP OPERATIONS ON STACK :

CODING :

```
#include<stdio.h>
int stack[10], choice, n, top, x, i;
void push(void);
void pop(void);
void display(void);

int main()
{
	top = -1;
	printf("\nEnter the Size of the Stack[max=100] : ");
	scanf("%d", &n);
	printf("\n\tSTACK OPERATION USING ARRAY.");
	printf("\n\t");
	printf("\n\t1. Push\n\t2. Pop\n\t3. Display\n\t4. Exit");
	printf("\n\t--------------------");

	do
	{
		printf("\nEnter the Choice : ");
		scanf("%d", &choice);
		switch(choice)
		{
			case 1:
			{
				push();
				break;
			}
			case 2:
			{
				pop();
				break;
			}
			case 3:
			{
				display();
				break;
			}
			case 4:
			{
				printf("\n\tEXIT\n");
				break;
			}
			default:
			{
```

```
                        printf("\n\tPlease Enter a Valid Choice(1/2/3/4).");
                }
        }
}
while(choice!=4);
{
        return 0;
}
}
        void push()
        {
                if(top >= n-1)
                {
                        printf("\n\tSTACK IS OVERFLOW.");
                }
                else
                {
                        printf("\nEnter the Value to be Pushed : ");
                        scanf("%d", &x);
                        top++;
                        stack[top] = x;
                }
        }
        void pop()
        {
                if(top <= -1)
                {
                        printf("\n\tSTACK IS UNDERFLOW.");
                }
                else
                {
                        printf("\n\tThe Poped Element is %d", stack[top]);
                        top--;
                }
        }
        void display()
        {
                if(top >= 0)
                {
                        printf("\nThe Element in the Stack are : \n");
                        for(i = top; i >= 0; i--)
                        printf("\n%d", stack[i]);
                }
                else
                {
                        printf("\nSTACK IS EMPTY");
                }
        }
```

OUTPUT :

```
Enter the Size of the Stack[max=100] : 3

        STACK OPERATION USING ARRAY.

        1. Push
        2. Pop
        3. Display
        4. Exit
        --------------------
Enter the Choice : 1

Enter the Value to be Pushed : 12

Enter the Choice : 1

Enter the Value to be Pushed : 23

Enter the Choice : 1

Enter the Value to be Pushed : 34

Enter the Choice : 1

        STACK IS OVERFLOW.
Enter the Choice : 3

The Element in the Stack are :

34
23
12
Enter the Choice : 2

        The Poped Element is 34
Enter the Choice : 3

The Element in the Stack are :

23
12
Enter the Choice : 4

        EXIT
```

EX. 03 – CONVERT INFIX EXPRESSION INTO POSTFIX EXPRESSION USING STACK :

CODING :

```
#include<stdio.h>
#include<ctype.h>
char stack[1000];
int top = -1;
void push(char x)
{
	stack[++top] = x;
}
char pop()
{

	if(top == -1)
		return -1;
	else
		return stack[top--];
}
int priority(char x)
{
	if(x == '(' || x == '^')
		return 0;
	if(x == '+' || x == '-')
		return 1;
	if(x == '*' || x == '/')
		return 2;
	return 0;
}
int main()
{
	char exp[1000];
	char * e, x;
	printf("\nEnter the Expression : ");
	scanf("%s", exp);
	printf("\n");
	e = exp;
	while(*e != '\0')
	{
		if(isalnum(*e))
			printf("%c", *e);
		else if(*e == '(')
			push(*e);
		else if(*e == '(')
		{
			while((x = pop()) != '(')
				printf("%c", x);
```

```
                }
                else
                {
                        while(priority (stack[top]) >= priority(*e))
                        printf("%c", pop());
                        push(*e);
                }
                e++;
        }
        while(top != -1)
        {
                printf("%c", pop());
        }
        return 0;
}
```

OUTPUT :

Enter the Expression : a+b*c

abc*+

EX. 04 – EVALUATE POSTFIX EXPRESSION BY USING STACK :

CODING :

```
#include<stdio.h>
#include<ctype.h>
int stack[20];
int top = -1;
void push(int x)
{
	stack[++top] = x;
}
int pop()
{
	return stack[top--];
}
int main()
{
	char exp[20];
	char * e;
	int n1, n2, n3, num;
	printf("\nEnter the Expression : ");
	scanf("%s", exp);
	e = exp;
	while(*e != '\0')
	{
		if(isdigit(*e))
		{
			num = *e - 48;
			push(num);
		}
		else
		{
			n1 = pop();
			n2 = pop();
			switch(*e)
			{
				case '+':
				{
					n3 = n1 + n2;
					break;
				}
				case '-':
				{
					n3 = n2 - n1;
					break;
				}
				case '*':
```

```
                        {
                                n3 = n1 * n2;
                                break;
                        }
                        case '/':
                        {
                                n3 = n2 / n1;
                                break;
                        }
                }
                push(n3);
        }
        e++;
    }
    printf("\nThe Result of Expression : %s = %d\n\n", exp, pop());
    return 0;
}
```

OUTPUT :

Enter the Expression : 12345*+*+)

The Result of Expression : 12345*+*+) = 47

EX. 05 – INSERT AND DELETE OPERATIONS ON QUEUE :

CODING :

```
#include<stdio.h>
#define n 5

int main()
{
	int queue[n], ch = 1, front = 0, rear = 0, i, j = 1, x = n;
	printf("\nQueue Using Array.\n");
	printf("----------------------------");
	printf("\n1. Insertion.\n2. Deletion.\n3. Display.\n4. Exit.\n");
	printf("----------------------------");
	while(ch != 4)
	{
		printf("\nEnter the Choice : ");
		scanf("%d", & ch);
		switch(ch)
		{
			case 1:
				if(rear == x)
				printf("\nQueue is Full.");
				else
				{
					printf("\nEnter the Inserting Value %d : ", j++);
					scanf("%d", & queue[rear++]);
				}
				break;
			case 2:
				if(front == rear)
				{
					printf("\nQueue is Empty.");
				}
				else
				{
					printf("\nDelete Element is %d ", queue[front++]);
					x++;
				}
				break;
			case 3:
				printf("\nQueue Element are : \n");
				if(front == rear)
				{
					printf("\nQueue is Empty.");
				}
				else
				{
```

```
                                for(i = front; i < rear; i++)
                                {
                                        printf("%d", queue[i]);
                                        printf("\n");
                                }
                        }
                        break;
                case 4:
                {
                        printf("\nExiting....");
                        break;
                }
                default:
                        printf("\nWrong Choice : Please See the Options : ");
        }
    }
    return 0;
}
```

OUTPUT :

```
Queue Using Array.
----------------------------
1. Insertion.
2. Deletion.
3. Display.
4. Exit.
----------------------------
Enter the Choice : 1

Enter the Inserting Value 1 : 12

Enter the Choice : 1

Enter the Inserting Value 2 : 23

Enter the Choice : 1

Enter the Inserting Value 3 : 34

Enter the Choice : 1

Enter the Inserting Value 4 : 45

Enter the Choice : 1

Enter the Inserting Value 5 : 56
```

```
Enter the Choice : 3

Queue Element are :
12
23
34
45
56

Enter the Choice : 2

Delete Element is 12
Enter the Choice : 2

Delete Element is 23
Enter the Choice : 3

Queue Element are :
34
45
56

Enter the Choice : 4

Exiting....
```

EX. 06 – INSERT AND DELETE OPERATIONS ON A LINKED LIST :

CODING :

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

struct node
{
   int data;
   struct node * next;
};
struct node * front = NULL;
struct node * rear = NULL;
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("\nNode is Inserted.\n\n");
}

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;
   }
}
```

```
void display()
{
   struct node * temp;
   if((front == NULL)&&(rear == NULL))
   {
      printf("\nQueue is Empty\n");
   }
   else
   {
      printf("\nThe 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("\n1. Enqueue\n2. Dequeue\n3. Display\n4. Exit\n");
      printf("\n\nEnter your Choice : ");
      scanf("%d", &choice);
      switch(choice)
      {
         case 1:
            printf("\nEnter the Value to be Inserted : ");
            scanf("%d", &value);
            enqueue(value);
            break;

         case 2:
            printf("\nPopped Element is : %d\n", dequeue());
            break;

         case 3:
            display();
            break;

         case 4:
            printf("\nExiting.....");;
            break;
```

```
        default:
           printf("\nYou Entered the Wrong Choice\n");
      }
   }
   return 0;
}
```

OUTPUT :

Implementation of Queue Using Linked List.

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 1

Enter the Value to be Inserted :
12

Node is Inserted.

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 1

Enter the Value to be Inserted : 23

Node is Inserted.

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 1

Enter the Value to be Inserted : 34

Node is Inserted.

1. Enqueue
2. Dequeue
3. Display

```
4. Exit

Enter your Choice : 1

Enter the Value to be Inserted : 45

Node is Inserted.

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 1

Enter the Value to be Inserted : 56

Node is Inserted.

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 3

The Queue is
12---->23---->34---->45---->56---->Null

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 2

Popped Element is : 12

1. Enqueue
2. Dequeue
3. Display
4. Exit

Enter your Choice : 3

The Queue is
23---->34---->45---->56---->Null
```

1. Enqueue
   2. Dequeue
   3. Display
   4. Exit

   Enter your Choice : 4

   Exiting.....

EX. 07 – PREORDEDR, INORDER, POSTORDER TRAVERSAL OF A BINARY TREE :

CODING :

```
#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);
}

void inorder(struct node * node)
{
   if(node == NULL)
   return;
   inorder(node -> left);
   printf("%d", node -> data);
   inorder(node -> right);
}

void preorder(struct node * node)
{
   if(node == NULL)
   return;
   printf("%d", node -> data);
   preorder(node -> left);
   preorder(node -> right);
}

void postorder(struct node * node)
{
   if(node == NULL)
   return;
   postorder(node -> left);
   postorder(node -> right);
   printf("%d", node -> data);
```

```
}

int main()
{
   struct node * root = newNode(1);
   root -> left = newNode(2);
   root -> right = newNode(3);
   root -> left -> left = newNode(4);
   root -> left -> right = newNode(5);

   printf("\nPreOrder Traversal of Binary Tree in : \n");
   preorder(root);

   printf("\n\nInOrder Traversal of Binary Tree in : \n");
   inorder(root);

   printf("\n\nPostOrder Traversal of Binary Tree in : \n");
   postorder(root);

   getchar();
   return 0;
}
```

OUTPUT :

PreOrder Traversal of Binary Tree in : 12453

InOrder Traversal of Binary Tree in : 42513

PostOrder Traversal of Binary Tree in : 45231

EX. 08(i) – LINEAR SEARCH :

CODING :

```
#include <stdio.h>
int linearSearch(int* arr, int size, int key)
{
   // Starting Traversal
   for (int i = 0; i < size; i++)
   {
      // Checking Condition
      if (arr[i] == key)
      {
         return i;
      }
   }
   return -1;
}

// Driver code
int main()
{
   int arr[10] = { 3, 4, 1, 7, 5, 8, 11, 42, 3, 13 };
   int size = sizeof(arr) / sizeof(arr[0]);
   int key = 11;

   // calling linearSearch
   int index = linearSearch(arr, size, key);

   // printing result based on value returned by
   // linearSearch()

   if (index == -1)
   {
      printf("\nThe Given Element is Not Present in the Array : \n");
   }
   else
   {
      printf("\nThe Given Element is Present at %d Position.\n", index);
   }
   return 0;
}
```

OUTPUT :

The Given Element is Present at 6 Position.

EX. 08(ii) – BINARY SEARCH :

CODING :

```
#include<stdio.h>
int binarySearch(int a[], int beg, int end, int val)
{
   int mid;
   if(end >= beg)
   {
      mid = (beg + end) / 2;
      if(a[mid] == val)
      {
         return mid + 1;
      }
      else if(a[mid] < val)
      {
         return binarySearch(a, mid + 1, end, val);
      }
      else
      {
         return binarySearch(a, beg, mid - 1, val);
      }
   }
   return -1;
}

int main()
{
   int a[] = {11, 14, 25, 30, 40, 41, 52, 57, 70};
   int val = 52;
   int n = sizeof(a) / sizeof(a[0]);
   int res = binarySearch(a, 0, n-1, val);

   printf("\nThe Original Elements in the Array are : \n\n");

   for(int i = 0; i < n; i++)
      printf("%d\t", a[i]);
      printf("\n\nElement to be Searched is : %d ", val);

      if(res == -1)
         printf("\n\nThe Given Element is Not Present in the Array.");
      else
         printf("\n\nThe Given Element is Present at %d Position of Array.\n", res);

   printf("\n");
   return 0;
}
```

OUTPUT :

The Original Elements in the Array are :

11 14 25 30 40 41 52 57 70

Element to be Searched is : 52

The Given Element is Present at 7 Position of Array.

EX. 09(i) – BUBBLE SORT :

CODING :

```
#include<stdio.h>
int main()
{
   int a[10], n, i, j, t;
   printf("\nEnter the Number of Elements : \n\n");
   scanf("%d", &n);

   printf("\nEnter the Elements : \n\n");

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

   for(i = 0; i < n-1; i++)
   {
      for(j = 0; j < n-i-1; j++)
      {
         if(a[j] > a[j+1])
         {
            t = a[j];
            a[j] = a[j+1];
            a[j+1] = t;
         }
      }
   }
   printf("\nElements Sorted by Bubble Sort : \n\n");

   for(i = 0; i < n; i++)
      printf("%d\t", a[i]);
      printf("\n");
   return 0;
}
```

OUTPUT :

```
Enter the Number of Elements : 5
Enter the Elements :
4
2
7
3
6

Elements Sorted by Bubble Sort :
2   3   4   6   7
```

EX. 09(ii) – INSERTION SORT :

CODING :

```
#include<stdio.h>
int main()
{
	int a[25], i, j, n, t;
	printf("\nEnter the Number of Elements : \n");
	scanf("%d", &n);
	printf("\nEnter the Elements : \n");
	for(i = 0; i < n; i++)
	{
		scanf("%d", &a[i]);
	}
	for(i = 1; i < n; i++)
	{
		for(j = i; j>0&&a[j-1]>a[j]; j--)
		{
			t = a[j];
			a[j] = a[j-1];
			a[j-1] = t;
		}
	}
	printf("\nSorted Elements using Insertion Sort : \n");
	for(i = 0; i < n; i++)
	printf("%d\t", a[i]);
	printf("\n");
	return 0;
}
```

OUTPUT :

```
Enter the Number of Elements :
5

Enter the Elements :
34
67
15
82
41

Sorted Elements using Insertion Sort :

15	34	41	67	82
```

EX. 10(i) – SELECTION SORT :

CODING :

```
#include<stdio.h>
int main()
{
	int a[10] = {100, 90, 80, 70, 60, 50, 40, 30, 20, 10};
	int n = 10;
	int i, j, pos, t;
	for(i = 0; i < (n-1); i++)
	{
		pos = i;
		for(j = i+1; j < n; j++)
		{
			if(a[pos] > a[j])
				pos = j;
		}
		if(pos !=i)
		{
			t = a[i];
			a[i] = a[pos];
			a[pos] = t;
		}
	}
	printf("\nSorted Elements Using Selection Sort : \n\n");
	for(i = 0; i < n; i++)
		printf("%d\t", a[i]);
		printf("\n\n");
	return 0;
}
```

OUTPUT :

Sorted Elements Using Selection Sort :

10 20 30 40 50 60 70 80 90 100

EX. 10(ii) – QUICK SORT :

CODING :

```
#include<stdio.h>

int partition(int a[], int low, int high)
{
	int i = low, j = high;
	int pivot = a[low], temp;

	while(i < j)
	{
		while(a[i] <= pivot && i <= high)
			i = i + 1;

		while(a[j] > pivot)
			j = j - 1;

			if(i < j)
			{
				temp = a[i];
				a[i] = a[j];
				a[j] = temp;
			}
	}
	a[low] = a[j];
	a[j] = pivot;
	return j;
}

void quicksort(int a[], int low, int high)
{
	int j, l= low, h = high;

	if(low < high)
	{
		j = partition(a, l, h);
		quicksort(a, l, j - 1);
		quicksort(a, j + 1, h);
	}
}

int main()
{
	int low = 0, high = 10, i;
	int a[10] = {10, 20, 80, 11, 75, 25, 62, 3, 4, 5};
	printf("\nThe Given Elements are : \n\n");
```

```
        for(i = 0; i < high; i++)
                printf("%d\t", a[i]);

        quicksort(a, low, high - 1);
        printf("\n\nSorted Elements By Quick Sort : \n\n");

        for(i = 0; i < high; i++)
                printf("%d\t",a[i]);
        printf("\n");
        return 0;
}
```

OUTPUT :

The Given Elements are :

10 20 80 11 75 25 62 3 4 5

Sorted Elements By Quick Sort :

3 4 5 10 11 20 25 62 75 80

EX. 11 – MERGE SORT :

CODING :

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

int a[11]={10, 14, 19, 26, 27, 31, 3, 35, 42, 44, 0};
int b[10];
void merging(int low, int mid, int high)
{
    int l1, l2, i;
   for(l1 = low, l2 = mid + 1, i = low; l1 <= mid && l2 <= high; i++)
   {
         if(a[l1] <= a[l2])
          b[i] =a [l1++];

        else
          b[i] = a[l2++];
     }
    while(l1 <= mid)
      b[i++] = a[l1++];

    while(l2 <= high)
      b[i++] = a[l2++];

   for(i = low; i <= high; i++)
   a[i] = b[i];
}

void sort(int low, int high)
{
   int mid;
   if(low < high)
   {
      mid = (low + high) / 2;
        sort(low, mid);
        sort(mid + 1, high);
        merging(low, mid, high);
    }
     else
     {
       return;
      }
}
```

```
int main()

{
    int i;
    printf("\nList Before Sorting : \n\n");

    for(i = 0; i <= max; i++)
        printf("%d\t", a[i]);
       sort(0, max);
       printf("\n\nList After Sorting : \n\n");

    for(i = 0; i <= max; i++)
      printf("%d\t", a[i]);
      printf("\n");
}
```

OUTPUT :

List Before Sorting :

10 14 19 26 27 31 3 35 42 44 0

List After Sorting :

0 3 10 14 19 26 27 31 35 42 44