Ex. No. 1 ARRAY IMPLEMENTATION OF STACK

Aim: To implement stack operations using array.

```
/*****************************************************************
***************
#include <stdio.h>
#include<conio.h>
#define max 5
static int stack[max];
int top=-1;
void push(int x){
   stack[++top]=x;


}
int pop(){
   return(stack[top--]);
}
void view(){
   int i=4;
   if(top<0){
   printf("\n stack is empty");

   }
   else{
      for(i=top;i>=0;i--)
      {
         printf("%4d",stack[i]);
      }
      printf("\n");
   }
}

int main()
{
  int ch=0;
  int val;
  while(ch!=4){
     printf("\n stack operation\n");
     printf("\n 1.push");
     printf("\n 2.pop");
     printf("\n 3.view");
```

```
    printf("\n 4.quit\n");
    printf("\nenter your choice\n");
    scanf("%d",&ch);
    switch(ch)
    {
       case 1:
       if(top<max-1)
       {
          printf("\n enter stack elements");
          scanf("%d",&val);
          push(val);

       }
       else{
          printf("\n stack overflow");

       }
       break;
       case 2:
       if(top<0){
          printf("\n stack overflow");

       }
       else{
          val=pop();
          printf("\n popped elements is %d",val);

       }
       break;
       case 3:
       view();
       break;
       case 4:
       exit(0);
       default:
       printf("\n invalid choice\n");

    }

  }

   return 0;
}
```

Program2:
Infix expression to postfix expression code

```
#include <stdio.h>
#include<ctype.h>
char stack[100];
int top=-1;
void push(char x)
{
stack[++top]=x;

}
char pop(){
   return(stack[top--]);
}
int priority(char x)
{
   if (x=='('){
   return 0;
   }

if(x=='+'||x=='-')
{
   return 1;
}
if(x=='*'|| x=='/')
{
   return 2;
}
return 0;
}
int main()
{
  char exp[100];
  char*e,x;
```

```
printf("enter 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;
}
```

3.Implement and create and Evaluate Postfix Expression using Stack ADT.

```
#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("\n !!!!!!!!__+_+_+_enter expression:!!!!!!!\n");
   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=n1-n2;
        break;
       }
       case '*':
       {
          n3=n1*n2;
          break;

       }
       case '/':
       {
          n3=n1/n2;
          break;
       }

    }
    push(n3);
```

```
        }
        e++;
    }
printf("\n the result of expression %s=%d\n\n",exp,pop());
    return 0;
}
```

Experiment No 4

Aim: Implement Iterative Tower of Hanoi

S/W / editor used: https://www.onlinegdb.com/

Code:

/* Write C programs that use both recursive and non-recursive functions
To solve Towers of Hanoi problem.*/

```
#include <stdio.h>
#include<stdlib.h>
void tower_re(int n,char source,char auxiliary,char target)
{
    if(n==1)
    {
        printf("mobve disk 1 from %c to %c",source,target);
        return;
    }
    tower_re(n-1,source,target,auxiliary);
    printf("move disk %d from %c to %c",n,source,target);
    tower_re(n-1,auxiliary,source,target);

}
void tower_nonre(int n){
    int disk;
    int total_moves=(1<<n)-1;
    char source='A',auxiliary='B',target='C';
    if(n%2==0){
        char temp=auxiliary;
        auxiliary=target;
        target=temp;
    }
    for (int move=1;move<=total_moves;move++){
        if(move%3==1){
            printf("\n moves disk %d from %c to %c\n",disk,source,target);

        }
        else if(move%3==2){
```

```
        printf("\n move disk %d from %c to %c ",disk,source,auxiliary);

      }
      else{
        printf("\n move disk %d from %c to %c ",disk,auxiliary,target);

      }
   }
}

int main()
{
  int n;
  printf("\n enter the numbers of disk");
  scanf("%d",&n);
  printf("\n recurisve \n");
  tower_re(n,'A','B','C');

  printf("\n  non recurisve \n");
  tower_nonre(n);
   return 0;
}
```

Experiment No 5
Aim: To Implement Linear Queue ADT using array

```
**********************************************
**********************************************/
#include <stdio.h>
#include<stdlib.h>
void insert();
void dequeue();
void display();
int front=-1;
int rear=-1;
int maxsize;
int queue[100];

int main()
```

```
{
  int ch;
  printf("\n enter the size of queue");
  scanf("%d",&maxsize);
  printf("\n1.insert\n2.delete\n3.display\4.exit");
  while(ch!=4){
     printf("\nenter yopur choice\n");
     scanf("%d",&ch);
     switch(ch){
        case 1:
        insert();
        break;
        case 2:
        dequeue();
        break;
        case 3:
        display();
        case 4:
        exit(0);
        break;
       default:
       printf("error");




     }
  }

   return 0;
}
```

```
void insert(){
   int x;
   printf("\n enter elememts:");
   scanf("%d",&x);
   if(front==-1 && rear==-1){
      front=0;
      rear=0;

   }
   else{
      rear=rear+1;
   }
   queue[rear]=x;
   printf("\n value inserted");

}
void dequeue(){
   int x;
   if(front==-1||front>rear)
   {
      printf("\n underflow");
      return;

   }
   else{
      x=queue[front];
      if(front==rear){
         front=-1;
         rear=-1;
      }
      else{
```

```
        front=front+1;
      }
      printf("\n value delete");
   }
}
void display(){
   int i;


      printf("\n elemeents in queue are");
      for(i=front;i<=rear;i++){
         printf("%d",queue[i]);
      }

}
```

Experiment No 6

Aim: To implement Circular Queue ADT using array.

S/W / editor used: https://www.onlinegdb.com/

Code:

```
// Circular Queue implementation in C

#include <stdio.h>

#define SIZE 5

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

// Check if the queue is full
int isFull() {
  if ((front == rear + 1) || (front == 0 && rear == SIZE - 1)) return 1;
  return 0;
}

// Check if the queue is empty
int isEmpty() {
```

```
  if (front == -1) return 1;
  return 0;
}

// Adding an element
void enQueue(int element) {
  if (isFull())
    printf("\n Queue is full!! \n");
  else {
    if (front == -1) front = 0;
    rear = (rear + 1) % SIZE;
    items[rear] = element;
    printf("\n Inserted -> %d", element);
  }
}

// Removing an element
int deQueue() {
  int element;
  if (isEmpty()) {
    printf("\n Queue is empty !! \n");
    return (-1);
  } else {
    element = items[front];
    if (front == rear) {
      front = -1;
      rear = -1;
    }
    // Q has only one element, so we reset the
    // queue after dequeing it. ?
    else {
      front = (front + 1) % SIZE;
    }
    printf("\n Deleted element -> %d \n", element);
    return (element);
  }
}

// Display the queue
void display() {
  int i;
  if (isEmpty())
    printf(" \n Empty Queue\n");
  else {
    printf("\n Front -> %d ", front);
    printf("\n Items -> ");
    for (i = front; i != rear; i = (i + 1) % SIZE) {
      printf("%d ", items[i]);
    }
    printf("%d ", items[i]);
```

```
    printf("\n Rear -> %d \n", rear);
  }
}

int main() {
  // Fails because front = -1
  deQueue();

  enQueue(1);
  enQueue(2);
  enQueue(3);
  enQueue(4);
  enQueue(5);

  // Fails to enqueue because front == 0 && rear == SIZE - 1
  enQueue(6);

  display();
  deQueue();

  display();

  enQueue(7);
  display();

  // Fails to enqueue because front == rear + 1
  enQueue(8);

  return 0;
}
```

Experiment No 7

Aim: To Implement Priority Queue ADT using array

S/W / editor used: https://www.onlinegdb.com/

Code:

The function **check_priority()** is used to check the priority and place element.

```
#include <stdio.h> #include <stdlib.h> #define MAX 10

void create_queue();

void insert_element(int);

void delete_element(int);

void check_priority(int);

void display_priorityqueue();

int pqueue[MAX];

int front, rear;
```

```
void main() {
    int n, choice;
    printf("\nEnter 1 to insert element by priority ");
    printf("\nEnter 2 to delete element by priority ");
    printf("\nEnter 3 to display priority queue ");
    printf("\nEnter 4 to exit");
    create_queue();
    while (1)
    {
        printf("\nEnter your choice : ");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1:
            printf("\nEnter element to insert : ");
            scanf("%d",&n);
            insert_element(n);
            break;
        case 2:
            printf("\nEnter element to delete : ");
            scanf("%d",&n);
            delete_element(n);
            break;
        case 3:
            display_priorityqueue();
            break;
        case 4:
            exit(0);
```

```
        default:
            printf("\n Please enter valid choice");
        }
    }
}  void create_queue() {
    front = rear = -1;
}  void insert_element(int data) {
    if (rear >= MAX - 1)
    {
        printf("\nQUEUE OVERFLOW");
        return;
    }
    if ((front == -1) && (rear == -1))
    {
        front++;
        rear++;
        pqueue[rear] = data;
        return;
    }
    else
        check_priority(data);
    rear++;
}  void check_priority(int data) {
    int i,j;
    for (i = 0; i <= rear; i++)
    {
        if (data >= pqueue[i])
        {
            for (j = rear + 1; j > i; j--)
```

```
            {
                pqueue[j] = pqueue[j - 1];
            }
            pqueue[i] = data;
            return;
        }
    }
    pqueue[i] = data;
}  void delete_element(int data)  {
    int i;
    if ((front==-1) && (rear==-1))
    {
        printf("\nEmpty Queue");
        return;
    }
    for (i = 0; i <= rear; i++)
    {
        if (data == pqueue[i])
        {
            for (; i < rear; i++)
            {
                pqueue[i] = pqueue[i + 1];
            }
            pqueue[i] = -99;
            rear--;
            if (rear == -1)
               front = -1;
            return;
        }
```

```
    }
    printf("\n%d element not found in queue", data);
} void display_priorityqueue() {
    if ((front == -1) && (rear == -1))
    {
        printf("\nEmpty Queue ");
        return;
    }
    for (; front <= rear; front++)
    {
        printf(" %d ", pqueue[front]);
    }
     front = 0;
}
```

Output of the above code:

```
Enter 1 to insert element by priority
Enter 2 to delete element by priority
Enter 3 to display priority queue
Enter 4 to exit
Enter your choice : 1

Enter element to insert : 22

Enter your choice : 1

Enter element to insert : 90
```

```
Enter your choice : 1


Enter element to insert : 87


Enter your choice : 3
 90  87  22
Enter your choice : 2


Enter element to delete : 87


Enter your choice : 3
 90  22
Enter your choice : 4



...Program finished with exit code 0
```

//New code// Priority Queue implementation in C

```
#include <stdio.h>
int size = 0;
void swap(int *a, int *b) {
  int temp = *b;
  *b = *a;
  *a = temp;
}

// Function to heapify the tree
void heapify(int array[], int size, int i) {
  if (size == 1) {
    printf("Single element in the heap");
  } else {
    // Find the largest among root, left child and right child
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < size && array[l] > array[largest])
      largest = l;
    if (r < size && array[r] > array[largest])
```

```
      largest = r;

    // Swap and continue heapifying if root is not largest
    if (largest != i) {
      swap(&array[i], &array[largest]);
      heapify(array, size, largest);
    }
  }
}

// Function to insert an element into the tree
void insert(int array[], int newNum) {
  if (size == 0) {
    array[0] = newNum;
    size += 1;
  } else {
    array[size] = newNum;
    size += 1;
    for (int i = size / 2 - 1; i >= 0; i--) {
      heapify(array, size, i);
    }
  }
}

// Function to delete an element from the tree
void deleteRoot(int array[], int num) {
  int i;
  for (i = 0; i < size; i++) {
    if (num == array[i])
      break;
  }

  swap(&array[i], &array[size - 1]);
  size -= 1;
  for (int i = size / 2 - 1; i >= 0; i--) {
    heapify(array, size, i);
  }
}

// Print the array
void printArray(int array[], int size) {
  for (int i = 0; i < size; ++i)
    printf("%d ", array[i]);
  printf("\n");
}

// Driver code
int main() {
  int array[10];
```

```
  insert(array, 3);
  insert(array, 4);
  insert(array, 9);
  insert(array, 5);
  insert(array, 2);

  printf("Max-Heap array: ");
  printArray(array, size);

  deleteRoot(array, 4);

  printf("After deleting an element: ");

  printArray(array, size);
}
```

****

Experiment No 8
Aim: To Implement Singly Linked List ADT.
S/W / editor used: https://www.onlinegdb.com/

Code:

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

struct Node
{
  int data;
  struct Node *next;
};

void delete (struct Node **head)
{
  struct Node *temp = *head;
 *head = (*head)->next;

  printf ("\n%d deleted\n", temp->data);
  free (temp);
}

void insertStart (struct Node **head, int data)
{

  struct Node *newNode = (struct Node *) malloc (sizeof (struct Node));
  newNode->data = data;
  newNode->next = *head;
  *head = newNode;
  printf ("\n%d Inserted\n", newNode->data);
```

```
}

void display (struct Node *node)
{
  printf ("\nLinked List: ");


  while (node != NULL)
    {
      printf ("%d ", node->data);
      node = node->next;
    }
  printf ("\n");
}

int main ()
{
  struct Node *head = NULL;

  // Need '&' i.e. address as we need to change head
  insertStart (&head, 100);
  insertStart (&head, 80);
  insertStart (&head, 60);
  insertStart (&head, 40);
  insertStart (&head, 20);

  // No Need for '&' as not changing head in display operation
  display (head);

  delete (&head);
  delete (&head);
  display (head);

  return 0;
}
```

Output:

Output

```
100 Inserted

80 Inserted

60 Inserted

40 Inserted

20 Inserted

Linked List: 20 40 60 80 100

20 deleted

40 deleted

Linked List: 60 80 100
```

****

Experiment No 10

Aim: To Implement Binary Search Tree ADT using Linked List.

S/W / editor used: https://www.onlinegdb.com/

Code:

```
#include <stdio.h>


/*  Function for binary search  */
void binary_search(int array[], int size, int n)
{
    int i, first, last, middle;
    first = 0;
    last = size - 1;
    middle = (first+last) / 2;

    while (first <= last) {
        if (array[middle] < n)
            first = middle + 1;
        else if (array[middle] == n) {
            printf("%d found at location %d.\n", n, middle+1);
           break;
        }
        else
            last = middle - 1;

        middle = (first + last) / 2;
    }
    if ( first > last )
        printf("Not found! %d is not present in the list.\n", n);
}
/*  End of binary_search()  */
```

```
int main()
{
    int a[200], i, j, n, size;
    printf("Enter the size of the list:");
    scanf("%d", &size);
    printf("Enter %d Integers in ascending order\n", size);
    for (i = 0; i < size; i++)
        scanf("%d", &a[i]);
    printf("Enter value to find\n");
    scanf("%d", &n);
    printf("Binary search\n");
    binary_search(a, size, n);
    return 0;
}
```

****

Experiment No 11
Aim: To implement Graph Traversal techniques: a) Depth First Search b) Breadth First Search.
S/W / editor used: https://www.onlinegdb.com/
Code:

For DFS

```
#include <stdio.h>
void DFS(int);
int g[10][10],visited[10],n;


int main()
{
   int i,j;
   printf("\n Enter number of vertices");
   scanf("%d",&n);
   printf("\n enter adjecency matrix of the graph");
   for(i=0;i<n;i++)
   {
      for(j=0;j<n;j++)
      {
         scanf("%d",&g[i][j]);
      }

      visited[i]=0;
      DFS(0);

   }


   return 0;
}
void DFS(int i)
```

```
{
   int j;
   printf("\n %d",i);
   for(j=0;j<n;j++){
      if(!visited[j]&&g[i][j]==1){
      DFS(j);
      }
   }
}
```

For BFS

```
#include<stdio.h>
#include<conio.h>
int a[20][20],q[20],visited[20],n,i,j,f=0,r=-1;
void bfs(int v) {
      for (i=1;i<=n;i++)
       if(a[v][i] && !visited[i])
       q[++r]=i;
      if(f<=r) {
            visited[q[f]]=1;
            bfs(q[f++]);
      }
}
void main() {
      int v;
      clrscr();
      printf("\n Enter the number of vertices:");
      scanf("%d",&n);
      for (i=1;i<=n;i++) {
            q[i]=0;
            visited[i]=0;
      }
      printf("\n Enter graph data in matrix form:\n");
      for (i=1;i<=n;i++)
       for (j=1;j<=n;j++)
       scanf("%d",&a[i][j]);
      printf("\n Enter the starting vertex:");
      scanf("%d",&v);
      bfs(v);
      printf("\n The node which are reachable are:\n");
      for (i=1;i<=n;i++)
       if(visited[i])
       printf("%d\t",i); else
       printf("\n Bfs is not possible");
      getch();
}
```