Lab program 1

write a program to stimulate the working of stack using an array including push, pop, display program should display appropriate

```
#include <stdio.h>
#define STACK_SIZE 5
int top = -1;
int s[10];
int item;
void push()
{
    if (top == STACK_SIZE - 1)
    {
     printf("stack overflow \n");
     return;
    }else{
   top = top + 1;
   s[top] = item;
}}
int pop()
{
```

Lab program 1

write a program to stimulate the working of stack using an array including push, pop, display program should display appropriate.

```
#include <stdio.h>
#define STACK_SIZE 5
int top = -1;
int s[10];
int item;
void push()
{
    if (top == stack_SIZE - 1)
    {
     printf("stack overflow \n");
     return;
    }else{
   top = top + 1;
   s[top] = item;
}}
int pop()
{
```

```
if (top == -1)
{
return -1;
}
else
{
return s[top--];
}
}
void display()
{
int i;
if (top == -1)
{
printf ("stack empty");
return;
}
else
{
printf ("contents of stack \n");
for (i=top; i>=0; i--)
{printf ("%d \n", s[i]);
}
}
}
```

```
int item_deleted;
int choice;
for(;;)
{
printf("\n1: push 2.Pop: 3. display
        4. exit\n");
printf("enter choic\n");
scanf("%d",&choice);
switch(choice)
{
case 1: print("Enter Item: ");
        scanf("%d", &item);
        push();
        break;
case 2: item_deleted = pop()
        if (item_deleted == -1)
        {
        printf("stack. empty\n");
    }
    else
    {
    print("item deleted is %d\n",
            item_deleted);
    }
```

```
        break;
    case 3: display();
            break;
    default : exit()
    }
}
getch()
}
```

program2

```
#include <stdio.h>
#include <string.h>
int F(char symol)
{
switch (symbol)
{
case '+':
case '-': return 2;
case '*':
case '/': return 4;
case '^':
case '$': return 5;
case '(': return 0;
case '#': return -1;
default : return 0;
}
}
int G(char symbol)
{
switch (symbol)
{
case '+':
case '-': return 1;
case '*':
case '/': return 3;
```

```
case '^':
case '$': return 6;
case '(': return 9;
case ')': return 0;
default : return 7;
}
}
void infix_postfix (char infix [], char postfix [])
{
int top, i, j;
char s[30], symbol;
top = -1;
s [++top] = '#';
j = 0;
for (i=0; i < strlen (infix ; i++)
 {
 symbol = infix [i];
while ( F (s [top]) > G (symbol) )
{
 postfix [j] = s [top--];
 j++
}
if ( F (s [top]) != G (symbol) )
   s [++top] = symbol;
  else
  top--;
 }
```

```
while (s[top] != '#')
{
postfix[i] = '\0';
}
}
int main()
{
char infix[20];
char postfix[20];
printf("Enter the valid parenthesized infix
        expression\n");
scanf("%s", infix);
infix_postfix(infix, postfix);
printf("The postfix expression is\n");
printf("%s\n", postfix);
return 0;
}
```

Linear queue.

```
#include <stdio.h>
#define MAX 5.
int queue[MAX];
int rear = -1 , froont = -1;
void insert()
{
  int add_item;
  if ( rear == MAX -1 )
  printf ("Queue Overflow \n");
  else
    { if (front == -1 )
          front == 0;
    printf ("enter the dement to be inserted : ");
    scanf ("%d", &add_item);
    rear = rear + 1;
    queue[rear] = add_item;
    }
}
void delete()
{
  if (front == -1 || front > rear )
    { printf ("Queue underflow \n");
      return;
    }
```

```
else
{ printf ("Element deleted from queue is : %d\n",
          queue[front]);
  front = front + 1;
}

void display()
{
 int i;
 if (front == -1)
 printf ("queue is empty \n");
 else
 {
 printf ("queue is : \n");
 from for (i = front; i <= rear; i++)
          printf("%d\n", queue[i]);
          print("\n");
 }
 }

 main()
 {
  int choice;
  while (1)
  {
   printf ("1: INSERT \n");
   printf ("2: DELETE \n");
   printf ("3 : DISPLAY \n");
   printf ("4 : EXIT");
```

```
printf ("Enter your choice : ");
scanf ("%d", &choice);
switch (choice)
{
case 1 :
 insert();
 break;
case 2:
delete();
break;
 case 3:
display();
 break;
case 4:
exit(1);
default:
printf("wrong choice\n");
}
}
}
```

Program 4

Circular Queue

```
#include <stdio.h>
#define q_size 5
int item, front=0, rear=-1, q[q_size], count=0;
void insert_rear()
{ if(count==q_size)
  {
   printf("Queue Overflow\n");
   return;
  }
 rear=(rear+1)%q_size;
 q[rear]=item;
 count++;
 return;
}
int delete_front()
{ if(count==0)
   return -1;
  else{
   item=q[front];
   front=(front+1)%q_size;
  count=count-1;
  return item;
}
```

```
}
void display()
   [ int i,f;
    if(counter==0)
     { printf("Queue is empty\n");
      return;
    }
    f=front;
    printf("contents of Queue:\n");
    for(i=0;i<=count;i++)
       {
        printf("%d\n",q[f]);
        f=(f+1)%q_size;
 }
}
void main()
{
int choice;
for(;;)
{
 printf(" 1:Insert \n 2: DELETE \n 3: DISPLAY
 printf("enter the choice \n");
 scanf("%d", &choice);
switch(choice) {
```

```
case 1 : printf ("Enter an element to be Inserted \n");
         scanf ("%d", &item);
         insert_read ();
         break;
case 2 : item=deleted_front. ();
         if (item == -1)
          printf ("queue is empty \n");
          printf ("item deleted is %d \n", item);
         break;
case 3 : displays ();
         break;
default : exit (0);
}
}
}.
```

program 5

```
#include <stdio.h>
#include <stdlib.h>
typedef struct node{
   int data.
   stud node *link;
}node;
node *root = NULL;
void add_at_end()
{ node *temp;
 temp=(node *) malloc (sizeof (node));
 printf ("Enter the node element \n");
 scanf ("%d ",&temp->data);
 temp -> link = NULL.
 if (root == NULL)
  { root = temp;
 }
 else
  { node *p = root;
   while (p->link != NULL)
    { p = p -> link;
    }
   p -> link = temp;
  }
 }
```

```
void add_at_begin ()
{
  node *temp;
temp = (node*) malloc(sizeof(node));
printf("Enter node element \n");
scanf("%d", &temp->data);
temp->link = null;
if (root == NULL)
  {
   root = temp;
}
else
  {
  temp->link = root;
  root = temp;
}
}

int length()
{
node *p;
  p=root;
  int i=0;
  while (p!=NULL)
  {
   i++;
   p=p->link;
}
```

```
return L;
}
void add_after (){
 node *p, *temp;
 int loc, i=1;
 printf ("Enter the location ");
 scanf ("%d", &loc);
 if (loc > length(L))
 {
 print ("Invalid location, the list has %d nodes", length(L));
}
else
 {
 p =root;
 while (i< loc)
 {
  p=p->link;
  i++;
 }
temp = (node *)malloc (sizeof (node));
printf ("Enter the node element \n");
scan f ("%d", &temp->data);
 temp->link=NULL;
 temp->link = p->link;
 p->link = temp;
  }
 }
```

```
void delete()
{
 int loc ;
 node *temp;
 printf("Enter the location of node to be deleted\n");
 scanf("%d",&loc);
  if (loc > length())
 {printf("There is no such node\n");
 }
 else if (loc == 1)
 {  temp = root ;
    root = temp ->link ;
    temp -> link = null ;
    free(temp);
    }
    else
    {
    node *p=root, *q;
    int i=1 ;
    while (i<loc-1)
     {
       p = p-> link ;
      i++ ;
  }
    q =p->link ;
    p -> link = q-> link ;
   q- Link = NULL ;
   free (q) ;
   }
  }
```

```
void display()()
{
node *temp=root;
if (temp==NULL)
{printf("NO nodes in the list \n");
}
else
{while(temp != NULL)
  {
   printf("%d \n", temp -> data );
   temp=temp ->link;
  }
}
}
int main()
{
int op, len;
while(1)
{printf(" Enter the Operation \n1 -Add in begin \n2 add
        at end\n");
printf("3.Add after a node\n 4. Delete node\n 5 -Display
 \n 6. length of list \n7.Exit \n");
 scanf("%d",&op);
 swith(op)
{
case 1 : add_at_begin();
        break;
case 2 : add_at_end();
        break;
```

```
case 3 : add_after();
         break;
case 4 : delete();
         break;
case 5 : display();
         break;
case 6 : len = length();
         printf("the length is %d \n", len);
         break;
case 7 : exit(0)
         break;
default : printf("No such Operation\n");
}
}
return 0;
}
```

Program 6

```
#include <stdio.h>
#include <stdlib.h>
struct node{
  int info;
  struct node * link;
 };
typedef struct node* NODE;
NODE getnode()
 {
  NODE x;
 x = (NODE) malloc(sizeof(struct node));
 if (x == NULL)
  {
  printf(" memory is full ");
  exit(0);
 }
 return x;
}
NODE delete_front (NODE first)
 {if (first==NULL){printf("List is empty "); return;}
  NODE temp;
  temp = first->link;
  free(first);
  return temp;
}
```

```
NODE delete_rear (NODE first)
{
  if (first == NULL)
  {
    printf (" List is empty ");
    return ;
  }
NODE temp, prev ;
temp = first
while (temp != NULL)
  { prev = temp ;
    temp = temp -> link ;
  }
  prev -> link = NULL ;
  free (temp) ;
  return first;

}
NODE insert (NODE first)
  { int info;
    NODE temp ; temp = getnode( );
    printf (" Enter the item: ");
    scanf (" %d ", &info );
    temp -> info = info ;
    temp -> link = first ;
    return temp ;

  }.
```

```
NODE delete_at_pos ( NODE first )
{
 int pos;
 NODE temp, prev
 printf ("enter the position: ");
 scanf ("%d", &pos);
 temp = first
  for (int i=0 ; i <=pos ; i++ )
     { prev = temp;
       temp = temp -> link;
     } if (temp==NULL)
     { printf ("deletion is not possible"); return first; }
   prev -> link = temp -> link;
    free (temp);
  return first;
 }
```

```
NODE delete_item (NODE
{
if (first == NULL)
{ printf("list is empty" );
return first;
} NODE temp = first; int flag = 0; NOD prev = NULL;
  NODE ptr;
while (temp != NULL )
  {
    if ( temp ->info == key )
      {
        ptr = temp ->link;
        prev ->link = ptr;
        free (temp);
        flag = 1
      }
    prev = temp;
    temp = temp ->link;
  }
if (flag == 0)
{ printf( " Element is not found in list");
 return first;
}
```

```
int main()
{
  int choice, item, pos;
  NODE first = NULL;
  for(;;)
  printf("1: Insert \n 2: Delete front \n 3:
         Delete Rear \n 4: Delet at position \n 5:
         Delete specified item \n");
  printf("Enter the choice:");
  scanf("%d", &choice);
  switch(choice)
   {
  case 1 : insert(first);
           break;
  case 2 : delete_front(first);
           break;
  case 3 : delete_rear(first);
           break;
  case 4: delete_at_pos(first);
          break;
  case 5 : printf("enter the item :");
           scanf("%d", &item);
           first = delete_item(first, item);
           break;
  case 6 : exit(0);
  default : break;
   }
}
```

Program:

```
#std
#include <stdio.h>
#include <stdlib.h>
struct node{
  int info;
 struct node* link;
};
typedef struct node* NODE;
NODE getnode() {
  NODE x;
  x = (NODE) malloc (sizeof (struct node));
 if (x == NULL)
  { printf ("memory is full");
    exit(0);
  }
  return x;
}
NODE insert (NODE first)
{
  int item;
   NODE temp;
   temp = getnode();
   printf ("enter the item to insert : );
    scanf ("%d", &item);
   temp -> info = item;
```

```
void reverse(int NODE first)
{
 NODE cur. = first;
 NODE prev = NULL.;
 NODE NEXT = NULL;
while ( cur != NULL) {
    next = cur->link ;
    cur->link = prev;
     prev = cur ;
   cur = next ;
}
 first = prev ;

}
void sort (NODE first )
 {
   NODE cur , indent,
    int temp ;
   if (first = = NULL) {
      return;
    }.
```

```
while (cur != NULL)
{
  indent = cur->link.
  if (cu->info > inden -> info )
  {
    temp = cur->info;
    cur -> info = inden -> info;
    inden -> info = temp;
  }
  inden = indent -> link;
}
cur = cur -> link;
}
}
}
display (NODE first)
{NODE temp = first
 while( temp != NULL)
{ printf ("%d\t", temp -> data);
 temp = temp -> link;
}
}
```

```
void concant (NODE first, NODE second)
{
if (first == NULL)
  return second;
 if (second == NULL)
   return first;
 NODE temp = first;
  while (temp != NULL)
    temp = temp -> link;
 temp ->link =second;
 return first;
}
void main () {
int choice; NODE first = NULL ; NODE second=NULL;
for (;;)
{
printf ( 1:Insert first \n 2:Insert second \n
 3:Deletefirst \n 4:Delete second \n 5:reverse first
\n6: reverse second \n 7:sort first \n 8:sort
 second \n 9:concat\n10: display );
scanf ("%d", &choice);
```

```
Swithe
 ooh.
switch (choice)
 {
case 1 : Insert (first);
          break;
case 2 : insert (second);
          break;
case 3 : delefe.node (first);
          break;
case 4 : delefe-node (second);
          break;
case 5 : reverse (first);
          break;
case 6 : reverse (second);
          break;
case 7 : ~~reve~~ sort (first);
          break;
case 8 : sort(second);
          break;
case 9 : concat (first, second);
          break;
case 10 : display (first)
          break;
```

program 8

implementation of stack;

```
#include <stdio.h>
#include <stdlib.h>
struct nodef
 int info;
 struct node* link;
 };
typedef structnode* NODE;
NODE getnode()
 {
  NODE x;
  x = (NODE) malloc(size of(struct node));
  if (x==NULL)
   { printf ("memory is full");
     exit(0);
   }
  return x;
 }
void~~inser~~ push (NODE first)
 {
  NODE temp, cur; int item;
  cur = getnode();
  printf("enter the item:");
  scanf("%d", &item);
```

```
if (first == NULL) return cur;
   temp = first;
while (temp != NULL)
   temp = temp -> link;
temp -> link = cur;
return first;
}

NODE pop (NODE first)
{ if (first == NULL) { printf("List is empty"); return; }
  NODE temp, prev = NULL;
  temp = first;
  while (temp != NULL) { prev = temp;
  temp = temp -> link; }
  prev -> link = NULL;
  free (temp);
  return first;
}
```

```
NODE display1(NODE first)
{
 NODE temp;
if(first==NULL) {printf("Empty list"); return first;
 temp = first;
 while (temp != NULL),
  {
   printf ("%d \t", temp->info);
   temp = temp->link;

   } return first;
  }

void main()
 {
   int choice;
   NODE first=NULL;
   for(;;)
   {
   printf("1: Push \n 2: Pop \n 3: DISPLAY");
   scanf("
 printf("Enter your choice:");
scanf("%d", &choice);
switch(choice)
  {
```

```
case 1: push(first);
        break;
case 2: first = pop(first);
        break;
case 3: first = display(first);
        break;
default: break;
```

}

Implementation Queue :-

```
#include <stdio.h>
#include <stdlib.h>
struct node
{
  int info;
  struct node * link;
};
type def struct node * NODE;
NODE getnode ()
{
 NODE x;
 x = (NODE) malloc (sizeof(struct node));
 if (x == NULL)
 {
  printf ("memory is full");
  exit (0);
 }
 return x;
}
```

```
NODE insert (NODE first, int item).
{
  NODE temp, cur;
  cur = getnode;
  cur -> info = item;
  cur -> link = NULL;
  if (first == NULL)
    { return first;
    }
  temp = first;
  while (temp != NULL)
  { temp = temp -> link;
  }
  temp -> link = cur;
  return first;
}
```

```
NODE delete_ (NODE first)
  if(first==NULL){printf("list is empty"); return ;}
{ NODE temp = first -> link;
    free(first)
    return temp;
  }

NOT void display (NODE first)
{
  NODE temp = first;
    if (temp == NULL)
      return { printf ("empty list");
        return;
      }
  while (temp != NULL)
  {
    printf ("%d \t", temp->info);
   temp = temp -> link;

}
  ret
}
```

```
int main()
{
  int choice, item;
  NODE first = NULL;
for(;;)
 {
 printf ("1 : Insert \n 2 : Delete \n 3 : Display");
 printf ("enter the item : ");
 scanf ("%d", &choice);
 switch (choice)
      {
      case 1 : printf ("enter the item: ");
               scanf ("%d", &item);
               first = insert (first, item);
               break;
      case 2 : Delete (first); break;
      case 3 : Display (first); break;
      default : break;
      }
 }
}
```

program 9

LAB 9
1BM19CS162

```
#include < stdio.h>
#include < stdlib.h>
struct node {
        int info;
        struct node * rlink;
        struct node * llink;
    };
typedef struct node * NODE;
NODE getnode()
  {
    NODE x;
   x = (NODE)malloc (sizeof(struct node));
   if (x==NULL)
      {
        printf ("memory full \n");
        exit(0);
      }
    return x;
  }
void freenode (NODE x)
    {
     free(x);
    }
NODE dinsert_front (int item, NODE head)
  {
    NODE temp, cur;
    temp = getnode();
    temp -> info = item;
    cur = head -> rlink;
    head -> rlink = temp;
    temp -> llink = head;
    temp -> rlink = cur;
    cur -> llink = temp;
    return head;
}
```

```
NODE insert_rear (int item, NODE head)
{
    NODE temp, cur;
    temp = getnode();
    temp->info = item;
    cur = head->llink;
    head->llink = temp;
    temp->rlink = head;
    temp->llink = cur;
    cur->rlink = temp;
    return head;
}
NODE delete_front (NODE head)
{
    NODE cur, next;
    if (head->rlink == head)
    {
        printf ("dq empty\n");
        return head;
    }
    cur = head->rlink;
    next = cur->rlink;
    head->rlink = next;
    next->llink = head;
    printf ("node deleted is %d", cur->info);
    freenode (cur);
    return head;
}
```

```
NODE ddelete_rear(NODE head)
{
  NODE cur, prev;
  if (head->rlink == head)
    {
      printf("dq empty\n");
      return head;
    }
  cur = head->llink;
  prev = cur->llink;
  head->llink = prev;
  prev->rlink = head
  printf("the node deleted is %d", cur->info);
  free(cur);
  return head;
}
void display(node head)
{
  NODE temp;
  if (head->rlink == head)
    {
      printf("dq empty\n");
      return;
    }
  printf("contents of dq \n");
  temp = head->rlink;
```

```
while (temp->rlink
while (temp != head)
        printf ("%d \t", temp->info);
        temp = temp->rlink;
        }
        printf ("\n");
    }
int length (NODE first).
  {
    NODE temp = first->rlink;
     int ct = 0;
    while (temp != first)
        {
          ct++;
          temp = temp->rlink;
    }
    printf ("length of the list is %d :", ct);
      return ct;
  }
NODE search (NODE first)
  {
    NODE temp = first->rlink;
   int count = 0, key, flag=0;
   printf ("Enter the key: ");
   scanf ("%d", &key);
while (first != first)
   {
```

```
        count++;
        if (temp->infor == key)
        {
            flag = 1;
            printf("key %d found in position %d", key, count);
        }
        temp = temp->rlink;
    }
    if (flag == 0)
    {
        printf("key is not found in list");
    }
    return first;
}
NODE insert_after(NODE first)
{
    int key, item, flag=0;
    printf("Enter the element: ");
    scanf("%d", &key);
    NODE temp = first->rlink;
    NODE ptr = getnode();
    while (temp != first)
    {
        if (temp->info == key)
        {
            printf("Enter the item need to be inserted: ");
            scanf("%d", &item);
            ptr->info = item;
            ptr->rlink = temp->rlink;
            ptr->llink = temp;
            temp->rlink = ptr;
```

```
        flag = 1;
        return first;
    } temp = temp->rlink;
}
if (flag == 0)
    printf ("There is no such element");
    return first;
}
NODE insert_after (NODE first)
{
    int key, item, flag = 0;
    printf ("Enter the element : ");
    scanf ("%d", &key);
    NODE temp = first->rlink;
    NODE ptr = getnode();
    while (temp != first)
    { if (temp->info == key)
        {
        printf ("Enter the item need to be inserted");
        scanf ("%d", &item);
        ptr->info = item;
        ptr->rlink = temp;
        ptr->llink = temp->llink;
        ptr->llink = ptr;
        flag = 1;
        return first;
    }
    temp = temp->rlink;
}
```

```
NODE head, last;
int item, choice, len;
head = getnode();
   head->rlink = head;
   head->llink = head;;
  for(;;)
{
printf("\n1: insert front \n2: insert rear \n3
   3: delete front \n4: delete rear \n5: display
   \n6: exit \n7: delete duplicate items \n8:
    insert before \n11: insert after \n");
printf("enter the choice \n");
scanf("%d", &choice);
switch(choice)
{
case 1: printf("enter the item at front end \n");
        scanf("%d", &item);
        last = dinsert_front(item, head);
        break;
case 2: printf("enter the item at rear end \n");
        scanf("%d", &item);
        last = dinsert_rear(item, head);
        break;
case 3: last = delete_front(head);
        break;
case 4: last = delete_rear(head);
        break;
```

```
case 5 : display(head);
         break;
case 7 : delete_dup(head);
         break;
case 8 : len = length(head);
         break;
case 9 : head = search(head);
         break;
case 10 : head = insert_after(head);
          break;
case 11 : head = insert_before(head);
          break;
default : break
  }
 }
}
```

program to

Binary Search Tree

```
#include <stdio.h>
#include <stdlib.h>
struct node {
   int key;
   struct node * left , *right ;
  };
struct node * newNode (int item)
 {
  struct node * temp =(struct node *)malloc(sizeof
                                  ( struct node));
  temp ->key =item ;
 temp ->left = temp->right= NULL;
  return temp;
}
void inorder ( struct node* root)
{ if (root != NULL) {
      inorder (root->left);
      printf("%d \t", root ->key);
      inorder (root->right);

   }
  if (root == NULL)
   {
    printf (" Tree is empty\n");
    return ;
} }
```

```
void preorder (struct node * root)
{ if (root != NULL) {
    printf ("%d \t", root->key);
    preorder (root->left);
    preorder( root->right);
  }
  if (root == NULL)
  {
    printf ("Tree is empty \n");
    return;
    }
}
void postorder ( struct node* root )
{
    if (root != NULL) {
      return ;
    post order - (root->left);
   postorder (root -> right);
   printf ("%d \t", root->key);
}
}
```

```
struct node * insert (struct node * node, int key)
{ if (node == NULL)
      return newNode(key);
  if (key < node -> key)
      node -> left = insert(node -> left, key);
  else if (key > node -> key)
      node -> right = insert (node -> right, key);
  return node;
}
int main()
{
  int ch;
  int item;
  struct node * root = NULL;
  for (;;)
  {
  printf ("1: Create tree \n2: Insert \n3: InOrder \n4:
          preorder \n5: Postorder \n");
  printf("Enter your choice:");
  scanf("%d", &ch);
  switch (ch)
  {
  case 1: printf ("Enter the name :");
          search ("%d", &item);
      root = insert (root, item);
      break;
```

```
struct node * insert (struct node * node, int key)
{ if (node == NULL)
      return newNode(key);
  if (key < node -> key)
      node -> left = insert(node -> left, key);
  else if (key > node -> key)
      node -> right = insert (node -> right, key);
  return node;
}
int main()
{
int ch;
int item;
struct node * root = NULL;
for (;;)
{
printf ("1: Create tree \n2: Insert \n3: InOrder \n4:
        preorder \n5: Postorder \n");
printf("Enter your choice:");
scanf ("%d", &ch);
switch (ch)
{
case 1 : printf ("Enter the name :");
         scanf ("%d", &item);
      root = insert (root, item);
      break;
```

```
        case 2 : printf ("Enter the item: ");
                 scanf ("%d", &item);
                 insert(root, item);
                 break;
        case 3 : inorder (root);
                 break;
        case 4 : preorder(root);
                 break;
        case 5 : postorder(root);
                 break;
        default : break;
        }
    }
    return 0;
}
```