Lab program - 1

1) Write a program to stimulate the working of stack using an array with the following:

a) Push
b) Pop
c) Display

The program should print appropriate messages for Stack overflow, stack underflow.

```
#include <stdio.h>
#include <conio.h>
#define MAX 5

int top = -1, stack [MAX]
void push ();
void pop ();
void display ();
void main ()
{
    int ch;
    while (1)
    {
        printf ("\n*** Stack Menu ***");
        printf ("\n\n1. Push \n2. Pop \n3. Display
                \n4. Exit");
        printf ("\n\n Enter any number between
                1 to 4 :");
        scanf ("%d" &ch);
```

```
switch (ch)
 {
   case 1 : push ();
            break;
   Case 2 : pop ();
            break;
   case 3 : display ();
            break;
   case 4 : Exit (0);
   default : printf ("\n Invalid input
 }
}

id push ()
  int value;

if (top == MAX-1)
{
   printf ("\n The stack is full ");
}
 else
 {
   printf ("\n Enter an element to push:
   Scanf ("%d", & value);
   top = top+1;
   Stack [top] = val;
 }
}

void pop ()
{
   if (top == -1)
```

```
{
  printf("\n The stack is empty !!!");
}
else
{
  printf("\n Deleted element is %d",
          stack[top]);
  top = top - 1
}
}
void display()
{
  int i;
  if (top == -1)
  {
  printf("\n stack is empty");
  }
  else
  {
    printf("\n stack is ....\n");
    for (i = top; i >= 0; --i)
    printf("%d\n", stack[i]);
}
}
```

Date / /
Page
SPLASH

Lab Program - 2

2) WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

Soln.

```
#include <stdio.h>
#define N 100
int stack[N];
int top = -1;

void push(int item) {
    if (top == N-1)
        printf("Stack overflow!\n");
    else
        stack[++top] = item;
}
int pop() {
    if (top == -1)
        printf("Stack underflow!\n");
    else
        return stack[top--];
}
int priority(char op) {
    switch(op) {
    case '*': return 2;
              break;
    case '/': return 2;
              break;
```

```
        case '+' : return 1;
                   break;
        case '-' : return 1;
                   break;
        case '(' : return 0;
                   break;
    }
}
int main()
{
    char s[50];
    char t[50];
    int i;
    int choice = 1;
    do {
        i = 0;
        printf("Enter your infix expression:
               ");
        scanf("%s", s);
        for (int i=0; s[i] != '\0'; i++) {
          switch (s[i]) {
            case '(' : push('(');
                       break;
            case ')' : while (stack[top] != '(')
              {
                t[i++] = pop();
              }
              pop();
              break;
            case '*';
            case '/' :
```

```
        case '+':
        case '-': while (top != -1 &&
                  priority (stack[top]) >=
                  priority (s[i])) {
                    t[j++] = pop ();
                  }
                  push (s[i]);
                  break;
        default : t[j++] = s[i];
        }
    }
    while (top != -1) {
        t[j++] = pop ();
    }
    t[j] = '\0';
    printf ("Postfix expression for %s
            is %s . \n", s, t);
    printf ("\n :: Menu :: \n1 try
            another infix expression. \n2.
            Exit \n Enter your choice : ");
    scanf ("%d", &choice);
} while (choice != 2);
return 0;
}
```

Date __/__/__
Page _____
SPLASH

Circular queue.

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>
#define que_size 3
int item, front = 0, rear = -1, q[que_size],
count = 0;
void insertrear()
{
    if (count == que_size)
    {
        printf("Queue overflow");
        return;
    }
    rear = (rear + 1) % que_size;
    q[rear] = item;
    count++;
}
int deletefront()
{
    if (count == 0) return -1;
    item = q[front];
    front = (front + 1) % que_size;
    count = count - 1;
    return item;
}
void displayq()
{
    int i, f;
    if (count == 0)
    {
```

```
        printf("Queue is empty");
        return;
    }
    f = front;
    printf("Contents of queue \n");
    for( i=0; i<=count; i++)
    {
        printf("%d \n", q[f]);
        f = (f+1) % que_size;
    }
}
void main()
{
    int choice;
    for(;;)
    {
        printf("\n1.Insert rear \n2.Delete from
                \n3.display \n4.Exit \n");
        printf("Enter the choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case1: printf("Enter the item to
                        be inserted:");
                scanf("%d", &item);
                insertrear();
                break;
            case2: item=deletefront();
                if(item==-1)
                printf("Queue is empty\n");
                else
                printf("Item deleted is :%d
```

```
        break;
    case 3: displayq();
            break;
    default: exit(0);
  }
}
}
```

Date / / /
Page
SPLASH

Queue

```
#include <stdio.h>
#include <stdlib.h>
#define Que_size 3.
int item, front=0, rear=-1, q[Que_size];
void insertrear()
{ if( rear= Que_SIZE -1)
  {
     printf("Queue Querflow\n");
     return;
  }
rear = rear+1;
q[rear] = item;
} int deletefront()
{ if ( front>rear)
{ front=0;
rear = -1
return -1;
} return q[ front++];
} void display Q()
{ int i;
if ( front > rear)
{
     printf ("Queue is empty\n");
     return;
}
  printf ("Contents of queue\n");
  for( i= front; i<rear; i++)
  {
       printf ("%d\n", q[i]);
} }
```

```
int main()
{
    int choice;
    for(;;)
    {
        printf("1) Insert rear\n 2) Delete front
                \n 3) Display \n 4) Exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice)
        {
            case 1: printf("Enter the item
                        to be inserted\n");
            scanf("%d", &item);
            insert rear();
            break;
            case 2: item = deletefront();
            if(item == -1)
            printf("Queue is empty\n");
            else
            printf("Item deleted = %d\n",
                                item);
            break;
            case 3: display Q();
            break;
            default: exit(0);
        }
    }
}
```

Date / /
Page
SPLASH

Singly linked list

```
#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("mem full\n");
  exit(0);
 }
 return x;
}
void freenode(NODE x)
{
  free(x);
}
NODE insert-front(NODE first, int item)
 {
  NODE temp;
  temp = getnode();
  temp -> info = item;
  temp -> link = NULL;
```

```
if (first == NULL)
return temp;
temp -> link = first;
first = temp;
return first;
}
NODE If (NODE second, int item)
{
NODE temp;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (second = NULL)
return temp;
temp -> link = second;
second = temp;
return second;
}
NODE temp;
if (first == NULL)
{
printf("list is empty cannot delete\n");
return first;
}
temp = first;
temp = temp -> link;
printf("item deleted at front-end
        is %d \n", first -> info);
free (first)
return temp;
}
NODE insert_rear(NODE first, int item)
```

Date / /
Page
SPLASH

```
{
NODE temp, cur;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL)
  return temp;
cur = first;
while (cur -> link != NULL)
      cur = cur -> link;
      cur -> link = temp;
      return first;
}
NODE IR (NODE second, int item)
 {
NODE temp, cur;
temp = get node();
temp -> info = item;
temp -> link = NULL;
if (second = NULL)
   return temp;
  cur = second;
while (cur -> link != NULL)
      cor -> link cur = cur -> link;
      cur -> link = temp;
      return second;
}
NODE delete_rear (NODE first)
 {
NODE cur, prev;
if (first == NULL)
{
```

```
printf("list is empty cannot delete\n");
return first;
}
if (first -> link == NULL)
{
printf("item deleted is %d\n",
                    first -> info);
free (first);
return NULL;
}
prev = NULL;
cur = first;
while (cur -> link != NULL)
{
prev = cur;
cur = cur -> link;
}
printf("item deleted at rear-end
        is %d", cur -> info);
free (cur);
prev -> link = NULL;
return first;
}
NODE insert_pos(int item, int pos,
                        NODE first)
{
NODE temp;
NODE prev, cur;
int count;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
```

```
if (first == NULL && pos == 1)
  return temp;
if (first == NULL)
{
printf ("\n invalid pos \n");
   return first;
}
if ( pos == 1)
{
temp -> link = first;
return temp;
}
count = 1;
prev = NULL;
cur = first;
while (cur != NULL && count != pos)
{
 prev = cur;
  cur = cur -> link;
   count ++;
}
if ( count == pos)
{
prev -> link = temp;
temp -> link = cur;
return first;
}
printf ("\n Invalid position \n");
return first;
}
NODE delete_pos(int pos, NODE first)
{
```

```
NODE cur;
NODE prev;
int count;
if ( first == NULL || pos <= 0)
{
printf ("Invalid position\n");
return NULL;
}
if (pos == 1)
{
cur = first;
first = first -> link;
freenode (cur);
return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur != NULL)
{
if ( count == pos)
break;
prev = cur;
cur = cur -> link;
count++;
}
if (count != pos)
{
  printf ("Invalid position\n");
  return first;
}
if ( count != pos)
```

```
{
printf("Invalid position specified\n");
  return first;
}
prev -> link = cur -> link;
   freenode(cur);
return first;
}
NODE reverse(NODE first)
     {
        NODE cur, temp;
        cur = NULL;
        while(first != NULL)
         {
           temp = first;
           first = first -> link;
           temp -> link = cur;
           cur = temp;
     }
    return cur;
}
NODE asc(NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
            int temp;
if(first == NULL) {
          return 0;
        }
else {
     while(prev != NULL) {
      cur = prev -> link;
```

```
while(cur1 = NULL) {
if (prev -> info > cur -> info) {
    temp = prev -> info;
    prev -> info = cur -> info;
    cur -> info = temp;
    }

    cur = cur -> link;
    }

    prev = prev -> link;
    }

    }
    return first;
}
NODE des (NODE first)
{
  NODE prev = first;
  NODE cur = NULL;
        int temp;
if (first == NULL) {
        return 0;
    }
    else {
    while (prev1 = NULL) {
    cur = prev -> link;
    while (cur1 = NULL) {
  if (prev -> info < cur->info) {
    temp = prev -> info;
  prev -> info = cur -> info;
  cur -> info = temp;
    }

    cur = cur -> link;
    }
```

```
        prev = prev -> link;
      }
    }
    return first;
  }
NODE concat(NODE first, NODE second)
{
  NODE cur;
  if (first == NULL)
  return second;
  if (second == NULL)
  return first;
  cur = first;
  while (cur -> link != NULL)
  {
    cur = cur -> link;
  }
  cur -> link = second;
  return first;
}

void display (NODE first)
{
  NODE temp;
  if (first == NULL)
  printf ("list empty cannot display
                          item\n");
  for (temp = first ; temp != NULL; temp =
                          temp->link)
  {
    printf ("%d\n", temp -> info);
  }
```

```
};
int main()
{
int item, choice, pos, element, option,
        choice2, item1, num;
NODE first = NULL;
NODE second = NULL;

for (;;)
{
 printf("\n1: Insert - front
          \n2: Delete front
          \n3: Insert front
          \n4: Delete front
          \n5: random position
          \n6: reverse
          \n7: sort
          \n8: concate
          \n9. display - list
          \n10. Exit \n");
printf("Enter the choice\n");
scanf("%d", &choice\n");
scanf Switch (choice)
    {
     case 1: printf("Enter the item at
                    front-end\n");
             scanf("%d", &item);
             first = insert_front(first,
                                  item);
             break;
     case 2: first = delete_front(first);
             break;
```

Date / /
Page
SPLASH

```
case 3: printf("Enter the item at rear-
                                    end \n");
        scanf("%d", &item);
        first = insert_rear(first, item);
        break;
case 4: first = delete_rear(first);
        break;
case 5:
printf("press 1 to insert or 2 to delete at
        any desired position \n");
scanf("%d", &element);
if(element == 1){
        printf("Enter the position to
                        insert \n");
        scanf("%d", &pos);
        printf("Enter the item to insert
                        \n");
        scanf("%d", &item);
        first = insert_pos(item, pos, first);
}
if(element == 2){
        printf("Enter the position to delete
                                    \n");
        scanf("%d", &pos);
        first = delete_pos(pos, first);
}
        break;
case 6:
        first = reverse(first);
        break;
case 7:
        printf("press 1 for ascending sort
```

```
and 2 for descending sort :\n");
scanf("%d", &option);
if(option == 1)
first = asc(first);
if(option == 2)
first = des(first);
break;
case 8:
        printf("Create a second list \n");
        printf("Enter the number of
                elements in second
                list \n");
        scanf("%d", &num);
        for(int i=1; i<=num; i++){
        printf("\n press 1 to insert
                front and 2 to insert
                rear \n");
        scanf("%d", &choice2);

        if(choice2==1){
        printf("Enter the item at
                front-end \n");
        scanf("%d", &item1);
        Second = IF(Second, item1);
        }
if(choice2==2){
printf("Enter the item at rear end\n");
    scanf("%d", &item1);
    Second = IR(Second, item1);
}
}
```

```
first = concate (first, second);
 break;
case 9: display (first);
        break;
default : exit (0);
        break;
 }
}
return 0;
 }
```

Date / /
Page
SPLASH

Double queue

```
#include <stdio.h>
#include <stdlib.h>
#define qsize 5
int f=0, r=-1, ch;
int item, q[10];

int isfull()
  {
    return (r==qsize-1)?1:0;
  }

int isempty()
  {
    return (f>r)?1:0;
  }

void insert_rear().
  {
    if (isfull())
    {
      printf("que overflow\n");
      return;
    }
r = r+1;
q[r] = item;
}

void delete_front()
 {
   if (isempty())
        {
          printf("queue empty\n");
          return;
```

```
}
printf("item deleted is %d\n", q[(f)++]);
if (f > r)
{
    f = 0
    r = -1;
}
}
void insert_front()
{
if (f != 0)
{
    f = f - 1
    q[f] = item;
    return;
}
q[++(r)] = item;
return;
}
else
printf("insertion not possible\n");
}
void delete_rear()
{
if (isempty())
{
    printf("queue is empty\n");
    return;
}
printf("item delete is %d\n",
                    q[(r)--]);
```

```
if (f>r)
    {
      f=0;
      r=-1;
    }
}
void display()
  {
    int i;
    if (isempty())
      {
        printf ("queue empty\n");
        return;
      }
  for(i=f; i<=r; i++)
      printf("%d\n", q[i]);
  }
int main()
  {
    for(;;)
{
printf("1. insert-rear\n 2. insert_front
      \n 3. delete_front \n 5. display
      \n 6. exit\n");
printf("Enter choice\n");
Scanf("%d", &ch);
Switch(ch)
 {
  case1: printf("Enter the item\n");
         Scanf("%d", &item);
         insert_rear();
         break;
```

```
case 2: printf("Enter the item\n");
        scanf("%d", &item);
        insert_front();
        break;
case 3: delete_rear();
        break;
case 4: delete_front();
        break;
case 5: display();
        break;
default: exit(0);
}
}
return 0;
}
```

Date / /
Page
SPLASH

Multiqueue

```
#include <stdio.h>
#include <stdlib.h>
#define N 3
int queue[3][N];
int front[3] = {0,0,0};
int rear[3] = {-1,-1,-1};
int item, pr;
void pq_insert(int);
void pq_delete();
void display();
int main()
{
  int ch;
  for(;;)
  {
    printf("\n PRIORITY QUEUE\n");
    printf(" ********** ");
    printf("\n\t1: Pq - insert\n");
    printf("\n\t2: Pq - delete\n");
    printf("\n\t3: Pq - display\n");
    printf("\n\t4 Exit\n");
    printf("\n Enter Choice:\n");
    Scanf("%d", &ch);
    switch(ch)
    {
      case 1: printf("\nEnter priority
                      number\n");
              Scanf("%d", &pr);
              if(pr>0 && pr<4)
              pq_insert(pr-1);
```

```
else
printf("Only three priority exits
        1 2 3\n");
 break;
case 2:
pqdelete();
 break;
case 3:
display();
 break;
case 4: exit(0);
}
}
return 0;
}
void pqinsert (int pr)
{
 if (rear[pr] == N-1)
 printf("\n Queue overflow");
  else
 {
  printf("\n Enter the item\n");
  scanf("%d", &item);
  rear[pr]++;
  queue[pr][rear[pr]] = item;
 }
}
void pqdelete()
 {
  int i;
  for (i=0; i<3; i++)
   {
```

```
        if (rear[i] == front[i]-1)
        printf("\n Queue empty\n");
        else
        {
          printf("Deleted item is %d
                  of queue[i] [front
                                [i]],i+1);
          front[i]++;
        }
    }
}

void display()
{
    int i,j;
    for(i=0; i<3; i++)
    {
      if (rear[i] == front[i]-1)
      printf("\n Queue empty %d\n",i+1);
      else
      {
        printf("\n Queue %d",i+1);
        for(j=front[i]; j<=rear[i];j++)
        printf("%d\t", queue[i][j]);
      }
    }
}
```

Date __/__/__
Page _____
SPLASH

Lab-8

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <malloc.h>
#include <stdlib.h>
struct node
{
    int item;
    struct node *next;
};
typedef struct node * Node;

Node getNode()
{
    Node x;
    x = (Node) malloc (sizeof( struct node));
    return x;
}
Node insert_front (Node first, int data)
{
    Node new_node;
    new_node = getNode();
    new_node -> item = data;
    new_node -> next = NULL;
    if (first == NULL)
    {
        return new_node;
    }
```

```
    new_node -> next = first;
    first = new_node;
    return first;
}
Node delete_end(Node first)
{
    Node prev, cur;
    if (first == NULL)
    {
        printf("List is Empty and
                cannot be deleted\n");
        return first;
    }
    cur = first;
    while (cur -> next != NULL)
    {
        prev = cur;
        cur = cur -> next;
    }
    prev -> next = NULL;
    free(cur);
    return first;
}
void display(Node first)
{
    int count=0;
    Node temp;
    if (first == NULL)
    {
        printf("\n List is Empty\n");
    }
    for (temp = first; temp != NULL; temp
                                = temp -> next)
```

```
{
    count++;
    printf("%d\n", temp->item);
}
printf("\n Number of Nodes
        in the list are
        %d\n", count);
}
void search(Node first, int data)
{
    int pos=0
    Node temp;
    int i;
    if(first == NULL)
    {
    printf("List is empty\n");
    return;
    }
for(temp=first, i=0; temp!=NULL;
    temp=temp->next; i++)
{
    if(temp->item == data)
    {
    pos=i+1;
    printf("\n Search Successfull\n");
    printf("Element is found at
            position %d\n", pos);
    break;
    }
else
{
    pos=0;
```

```
        }
    }
    if(pos==0)
    {
        printf("\n Search Unsuccessfull \n");
    }
}
void sort(Node first)
{
    int t;
    Node temp;
    if(first==NULL)
    {
        printf("List is Empty \n");
        return;
    }
    for(Node i=first; i!=NULL; i=i->Next)
    {
        for(Node j=i->next; j!=NULL; j=j->Next)
        {
            if((i->item)>(j->item))
            {
                t=i->item;
                i->item=j->item;
                j->item=t;
            }
        }
    }
    printf("\n List is ascending order \n");
```

```
}
int main()
{
    Node first = NULL;
    Node a = NULL;
    Node b = NULL;
    Node ans = NULL;
    int choice, val, pos, n;
    do
    {
        printf("**** Actions ****\n");
        printf("1. Insert front\n");
        printf("2. Delete end\n");
        printf("3: Sort the list\n");
        printf("4. Search for an element\n");
        printf("5. Display the count
                and list\n");
        printf("\n Which action do yo
                wanna perform ?\n");
        scanf("%d", &choice);
        switch(choice)
        {
        case 1:
            printf("Enter the value to be
                    inserted\n");
            scanf("%d", &val);
            first = insert_front(first, val);
            break;
        case 2:
            first = delete_end(first);
            break;
```

```
case 3:
    sort (first)
    display (first);
    break;
case 4:
    printf ("Enter the number to
             Searched \n");
    scanf ("%d", &n);
    Search (first, n);
    break;
case 5:
    printf ("\n The elements are. \n");
    display (first);
    break;
}
while (choice != 6);
}
```

Date / /
Page
SPLASH

Binary tree

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

Struct node
{
  int info;
  Struct node * r link;
  Struct node * l link;
};
typedef struct node * NODE;
NODE getnode()
{
  NODE x;
  x = (NODE) malloc( Sizeof (Struct node));
  if (x == NULL)
  {
   printf ("mem full \n");
    exit (0);
  }
 return x;
}
Void freenode (NODE x)
{
   free(x);
}
NODE insert(NODE root, int item)
{
  NODE temp, cur, prew;
  temp = getnode ();
  temp -> lr link = NULL;
```

```
temp -> llink = NULL;
temp -> info = item;
if (root == NULL)
return temp;
prev = NULL;
cur = root;
while (cur != NULL)
{
    prev = cur;
    cur = (item < cur -> info) ? cur
          -> llink : cur -> rlink;
}
if (item < prev -> info)
prev -> llink = temp;
else
prev -> rlink = temp;
return root;
}
void display (NODE root, int i)
{
int j;
if (root == NULL)
{
display (root -> rlink, i+1);
for (j=0; j<i; j++)
printf ("  ");
printf ("%d \n", root -> info);
display (root -> llink, i+1);
}
NODE delete (NODE root, int item)
{
NODE cur, parent, q, suc;
```

```
if (root == NULL)
{
    printf ("Empty\n");
    return root;
}
parent = NULL;
cur root;
while (cur != NULL && item != cur
                -> info)
{
    parent = cur;
    cur = ( item < cur -> info ) ? cur
            -> llink : cur -> rlink;
}
if (cur == NULL)
{
    printf ("not found\n");
    return root;
}
if (cur -> llink == NULL)
    q = cur -> rlink;
else if ( cur -> rlink == NULL)
    q = cur -> llink;
else
{
    Suc = cur -> rlink;
    while (Suc -> llink != NULL)
    Suc = Suc -> llink;
    Suc -> llink = cur -> llink;
    q = cur -> rlink;
}
```

```
if( parent == NULL)
    return q;
if (cur == parent -> llink)
    parent -> llink = q;
else
    parent -> rlink = q;
    freenode (cur);
    return root;
}
void preorder (NODE root)
{
    if ( root != NULL)
    {
        printf ("%d\n", root -> info);
        preorder ( root -> llink );
        preorder ( root -> rlink);
    }
}
void postorder (NODE root)
{
    if (root != NULL)
    {
        postorder ( root -> llink)
        postorder (root -> rlink)
        printf ("%d\n", root -> info);
    }
}
void inorder (NODE root)
{
    if (root != NULL
```

```
        printf("%d\n", root->info);
        inorder(root->rlink);
    }
}
void main()
{
    int item, choice;
    NODE root = NULL;
    for(;;)
    {
        printf("\n ---MENU---");
        printf("\n1. Insert 4. Post 7.\n2.
                Display 5. In\n 3. Pre.
                6. Delete\n 7. Exit\n");
        printf("Enter your choice");
        switch(choice)
        {
        case 1: printf("Enter the item\n");
                scanf("%d", &item);
                root = insert(root, item);
                break;
        case 2: display(root, 0);
                break;
        case 3: preorder(root);
                break;
        case 4: postorder(root);
                break;
        case 5: inorder(root);
                break;
        case 6: printf("Enter the item\n");
                scanf("%d", &item);
                root = delete(root, item);
```

Date __/__/__
Page _____
SPLASH

```
        break;
    default : exit(0);
              break;
    }
  }
}
```