Lab.-1

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

int main()
{
    top = -1;
    Printf("\n Enter the Size of STACK:");
    scanf("%d", &n);
    Printf("\n STACK IMPLEMENTATION USING ARRAYS\n");
    do
    {
        Printf("\n1. PUSH\n2. POP\n3. DISPLAY\n4. EXIT\n");
        Printf("\n Enter the choice :");
        Scanf("%d", & choice);
        Switch (choice)
        {
        Case 1:
            {
                Push();
                break;
            }
        Case 2:
            {
                POP();
                break;
            }
        Case 3:
            {
                display();
                break;
            }
        Case 4:
            {
                break;
            }
```

```
    default:
        {
            Printf("\n Invalid choice\n");
        }
    }
}
While (choice!=4);
return 0;
}

Void push()
{
    if (top >= n-1)
    {
        Printf("\n STACK OVERFLOW\n");
    }
    else
    {
        Printf("Enter a ELEMENT to be pushed:");
        Scanf("%d", &x);
    top++;
    Stack[top] = x;
    }
}
Void pop()
{
    if (top <= -1)
    {
        Printf("\n STACK UNDERFLOW\n");
    }
    else
    {
        Printf("\n The popped ELEMENT is %d", stack[top]);
    top--;
    }
}
Void display()
{
    if (top >= 0)
    {
```

```
        // Print the stack
        Printf("\n ELEMENTS IN THE STACK\n\n");
        for(i = top ; i >= 0 ; i--)
        Printf("%d\t", stack[i]);
    }
     else
    {
        Printf("\n EMPTY STACK\n");
    }
}
```

LAB.- 2

```
# include <stdio.h>
# include <string.h>
 int F (char Symbol)
{
    switch (Symbol)
   {
       case '+' :
      case '-' : return 2;
      case '*' :
      case '/' : return 4;
      case '^' :
      case '$' : return 5;
      case '(' : return 0;
      case '#' : return -1;
      default :  return 8;
    }
 }
 int G (char Symbol)
 {
     Switch (Symbol)
     {
         case '+' :
         case '-' : return 1;
         case '*' :
         case '/' : return 3;
         case '^' :
         case '$' : 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[j++] = s[top--];
    }
    postfix[j] = '\0';
}
void main()
{
    char infix[20];
    char postfix[20];
    printf("Enter the valid infix expression\n");
    scanf("%s", infix);
    infix - postfix(infix, postfix);
    printf("The postfix exp is\n");
    printf("%s\n", postfix);

}
```

LAB. - 3

```
# include <studio.h>
# include <Stdlib.h)
# define QUE_SIZE 3
int item, front = 0, rear = -1, q[10];
Void insertrear()
{
    if (rear == QUE_SIZE-1)
    {
        Printf("queue Overflow\n");
        return;
    }
    rear = rear+1;
    q[rear] = item;
}
int deletefront()
{
    if(front > rear)
    {
        front = 0;
        rear = -1;
        return -1;
    }
    return q[front++];
}
Void displayQ()
{
    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]);
    }
}
```

```
Void main( )
{
 int choice;

 for( ; ;)
 {
 Printf ("\n1:insertrear\n2:deletefront \n3:display\n4:exit\n");
 Printf("Enter the choice\n");
 Scanf ("%d", &choice);
 Switch (choice)
 {
      case 1: printf ("Enter the item to be inserted \n");
              scanf ("%d", &item);
              insertrear( );
              break;
      Case 2: item = deletefront( );
              if (item == -1)
              Printf("queue is empty\n");
              else
              printf ("item deleted = %d\n", item);
              break;
      Case 3: display( );
              break;
      default: exit(0);
    }
  }
}
```

~~CODE~~

LAB - 5

```
# include <studio.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 delete_front(NODE first)
{
NODE temp;
```

```
if (first == NULL)
{
Printf("List is empty cannot delete\n");
return first;
}
temp = frist
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)
{
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 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;
}
Void display (NODE first)
{
NODE temp;
if (first == NULL)
printf ("List empty cannot display items\n");
Printf ("contents of the list: \n");
for (temp = first; temp != NULL; temp = temp -> link)
{
Printf ("%d \n", temp -> info);
}
}
Void main ()
{
int item, choice, pos;
NODE first = NULL;
for (;;)
{
Printf ("\n 1: Insert_front \n 2: Delete_front \n 3: Insert_rear \
                                                n 4: Delete_rear \n
5: Display_list \n 6: Exit \n");
printf ("Enter the choice \n");
scanf ("%d", &choice);
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
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 ; display (first);
        break;
case 6 : exit (0);
```

```
default: printf("Invalid choice!\n");
            break;
    }
}

}
```

LAB.- 6

```
#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 delete_front(NODE first)
 {
 NODE temp;
 if(first==NULL)
 {
  Printf("List is empty cannot delete\n");
 return first;
 }
 temp=first;
 temp=temp->link;
```

```
Void display (NODE first)
{
 NODE temp;
 if (first == NULL)
printf ("List empty cannot display items\n");
printf ("contents of the list :\n");
for (temp = first; temp != NULL; temp = temp -> link)
{
printf ("%d \n", temp -> info);
}
}
Void main()
{
 int item, choice, pos;
NODE first = NULL;
 for(;;)
{
printf ("\n 1: Insert_front\n 2: Delete_front\n 3: Insert_rear\n
4 : Delete_rear\n 5: Display_list\n 6 : Exit\n");
Printf("Enter the choice\n");
scanf ("%d", &choice);
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;
case 3 : Print-f("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 : display (first);
         break;
         case 6 : exit(0);
default : printf ("Invalid choice!\n");
         break;
 }
}

}
```

```
Printf ("Item deleted at front-end is = %d \n", first -> info);
free (first);
return temp;
}
NODE insert_rear (NODE first, int item)
{
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 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;
}
```

LAB - 7

```
# include<stdio.h>
# include<conio.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 delete_front(NODE First)
 {
 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)
{
NODE temp, cur;
temp = get node
NODE temp, prev, 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 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 order_list (int item, NODE first)
{
NODE temp, prev, cur;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL) return temp;
if (item < first -> info)
{
temp -> link = first;
return temp
}
prev = NULL;
cur = first;
while (cur != NULL && item > cur -> info)
{
prev = cur;
cur = cur -> link;
}
prev -> link = temp;
temp -> link = cur;
return first;
}
/* NODE delete_info (int key, NODE first)
{
NODE prev, cur;
if (first == NULL)
{
printf ("List is empty\n");
return NULL;
}
if (key == first -> info)
{
cur = first;
first = first -> link;
freenode (cur);
return first;
```

```
    }
    Prev = NULL;
    cur = first;
    while (cur != NULL)
    {
    if (key == cur -> info) break;
    Prev = cur;
    cur = cur -> link;
    }
    if (cur == NULL)
    {
    Printf ("search is Unsuccessfull\n");
    return first;
    }
    Prev -> link = cur -> link;
    Printf ("key deleted is %d, cur -> info);
    free node (cur);
    return first;
    }*/
    Void display (NODE first)
    {
    NODE temp;
    if (first == NULL)
    Printf ("list empty cannot display items\n");
    Printf ("contents of the list:\n");
   for (temp = first; temp != NULL; temp = temp -> link)
   {
   printf ("%d\n", temp -> info);
   }
}
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;
}
```

```
NODE reverse (NODE first)
{
NODE cur, temp;
cur = NULL;
while (first != NULL)
{
temp = first;
first = first -> link;
temp -> link = cur;
cur = temp;
}
return cur;
}
void main()
{
int item, choice, key, n, i;
NODE first = NULL, a, b;
for (;;)
{
Printf ("\n1: Insert_front\n 2: Delete_front\n 3: Insert_rear
        \n4: Delete_rear\n");
Printf ("5: order_list\n6: Display_list\n7: concat\n8: Reverse
        \n9: Exit\n");
Printf ("Enter the choice\n");
Scanf ("%d", &choice);
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;
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 ("Enter the item to be inserted in Ordered_list\n");
        Scanf ("%d", &item);
        first = order_list (item, first);
        break;
/*case 6: Printf ("Enter the key to be deleted \n");
        Scanf ("%d" &key);
```

```
        First = delete_info (key, first);
                break; */
    case 6 : display (first);
                break;
    case 7 ; Printf ("Enter the no of nodes in 1\n");
                scanf ("%d", &n);
                a = NULL;
                for (i=0; i<n; i++)
                {
                Printf ("Enter the item \n");
                scanf ("%d", &item);
                a = insert_rear (a, item);
                }
                Printf ("Enter the no of nodes in 2 \n");
                scanf ("%d", &n);
                b = NULL;
                for (i=0; i<n; i++)
                {
                Printf ("Enter the item\n");
                scanf ("%d", &item);
                b = insert_rear (b, item);
                }
                a = concat(a,b);
                display (a)
                break;
    case 8 : first = reverse (first);
                        display (first);
                        break;
    case 9 : exit (0);
                break;
                default: printf ("Invalid choice\n");
        }
    }
    }
```

LAB. - 8

```
#include <stdio.h>
#include <stdlib.h>
#include <process.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\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 delete_front (NODE first)
{
```

```
    NODE temp;
    if (first == NULL)
    {
        printf("List is empty, cannot Delete item\n");
        return first;
    }
    temp = first;
    temp = temp -> link;
    printf("Item Deleted at the front-end is: %d\n", first->info);
    free(first);
    return temp;
}

/*NODE insert_rear(NODE first, int item)
{
    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 delete_rear(NODE first)
{
    NODE cur, Prev;
    if (first == NULL)
    {
        Printf("The List is Empty, cannot Delete Item\n");
        return first;
    }
    if (first -> link == NULL)
    {
        printf("Item Deleted is: %d", first->info);
        free(first);
        return NULL;
    }
    Prev = NULL;
```

```
    cur = first;
    while (cur -> link != NULL)
    {
       prev = cur;
       cur = cur -> link;
    }
    Printf ("Item Deleted at the 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("invalid position\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 ("IP\n");
    return first;
}

Void display (NODE first)
{
    NODE temp;
    if (first == NULL)
        Printf("List is EMPTY, cannot Display Items\n");
    Printf ("\n*********\n");
    for (temp = first ; temp != NULL ; temp = temp -> link)
    {
        Printf("%d\n", temp -> info);
    }
    Printf ("\n*********\n");

}
  Void main()
 {
     int item, choice, pos ;
  NODE first = NULL
 for(;;)
 {
case 1 : printf("Enter the item at front-end: ");
         scanf ("%d", &item);
         first = insert_front (first, item);
         break;
case 2 : first = delete_front (first);
         break;
/* case 1 : Printf ("Enter the item at rear-end: ");
          Scanf ("%d", &item);
          break;
    case 2 : first = delete_rear (first);
          break; */
```

LAB :- 10

CODE:-

```
#include <stdio.h>
#include <process.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 insert (NODE root, int item)
{
  NODE temp, cur, prev;
  temp = getnode();
  temp -> rlink = 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 ("Tree empty\n");
   return root;
}
```

```
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)
    {
        inorder (root -> llink);
        printf ("%d \n", root -> info);
        inorder (root -> rlink);
    }
}
void main()
{
    int item, choice;
    NODE root = NULL;
```

```
for (;;)
{
    printf("\n1. Insert \n2. Display \n3. Pre-order \n4. Post-order \n5. Inorder
        \n6. Delete \n7. Exit \n");
    printf("Enter the choice \n");
    scanf("%d", &choice);
    switch(choice)
    {
        case 1: printf("Enter the item \n");
                scanf("%d", &item);
                root = insert(root, item);
                break;
        case 2: printf("Contents of binary search tree: \n");
                display(root, 0);
                break;
        case 3: printf("Pre-order: \n");
                preorder(root);
                break;
        case 4: printf("Post-order: \n");
                postorder(root);
                break;
        case 5: printf("In-order: \n");
                inorder(root);
                break;
        case 6: printf("Enter the item \n");
                scanf("%d", &item);
                root = delete(root, item);
                break;
        case 7: exit(0);
        default: printf("Invalid choice \n");
    }
}
```