LABSET 1 (Emplyoee-Structure)

```
#include<stdio.h>
void read(char*file);
void write(char*file,int);
void searchEmpId(char*file,unsigned);
void searchEmpSal(char*file,unsigned);
void searchEmpAge(char*file,unsigned);
void searchEmpDep(char*file,char*);
struct Employee{
   unsigned empid;
   char name[25];
   char dep[25];
   unsigned sal;
   unsigned age;
};
int main(){
   char file[20];
   unsigned unsignedKey;
   char keyDep[25];
   int n,ch;
   printf("Enter the file name\n");
   scanf("%s",file);
   printf("Enter the no. of employees to be entered\n");
   scanf("%i",&n);
   write(file,n);
   printf("The details of Employees are as follows\n\n");
   printf("Emp-ID   Emp-NAME   Emp-Department   Emp-Salary   Emp-Age\n\n");
```

```
    read(file);
   label: printf("\n\nEnter the choice to Search record by :\n1- Emp_ID\t2- Salary\t3- Department\t4-
Age\t5- Exit\n\n");
    scanf("%d",&ch);
    switch(ch){
       case 1:printf("Enter the Emp-id to be traced\n\n");
    scanf("%u",&unsignedKey);
    searchEmpId(file,unsignedKey);
       break;
       case 2:printf("Enter the salary to be traced\n\n");
       scanf("%u",&unsignedKey);
    searchEmpSal(file,unsignedKey);
    break;
       case 4:printf("Enter the Age to be traced\n\n");
       scanf("%u",&unsignedKey);
    searchEmpAge(file,unsignedKey);
    break;
       case 3:printf("Enter the Department to be traced\n\n");
       scanf("%s",keyDep);
    searchEmpDep(file,keyDep);
    break;
       case 5:printf("\n\n***Have a Good Day***\n\n");exit(0);
       break;
    default:printf("Entered wrong choice!!\nPlease Enter again!\n");
    }goto label;
    return 0;
}
void write(char*file,int n)
```

```
{
   FILE *fp=fopen(file,"w");
   int i=1;
   struct Employee E;
   while(n){
      printf("Enter the Emp-id Name Department Salary and Age of Employee %d\n\n",i++);
      scanf("%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age);
      fprintf(fp,"%u %s %s %u %u\n",E.empid,E.name,E.dep,E.sal,E.age);
      n--;
   }
   fclose(fp);
}
void read(char*file)
{
   FILE *fp=fopen(file,"r");
  struct Employee E;
   while( fscanf(fp,"%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age)!=EOF){
      fprintf(stdout,"%u  %-14s  %-14s  %-7u  %-3u\n",E.empid,E.name,E.dep,E.sal,E.age);
   }
   fclose(fp);
}
void searchEmpId(char*file,unsigned key)
{
   FILE *fp=fopen(file,"r");
  struct Employee E;
  while( fscanf(fp,"%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age)!=EOF){
      if(key!=E.empid);
```

```
        else{
          fprintf(stdout,"%u  %-14s  %-14s  %-7u  %-3u\n",E.empid,E.name,E.dep,E.sal,E.age);
          return;
        }
    }
    printf("Such Employee details not present\n");
    fclose(fp);
}
void searchEmpSal(char*file,unsigned key)
{
    FILE *fp=fopen(file,"r");
   struct Employee E;
   while( fscanf(fp,"%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age)!=EOF){
        if(key!=E.sal);
        else{
          fprintf(stdout,"%u  %-14s  %-14s  %-7u  %-3u\n",E.empid,E.name,E.dep,E.sal,E.age);
          return;
        }
    }
    printf("Such Employee details not present\n");
    fclose(fp);
}
void searchEmpAge(char*file,unsigned key)
{
    FILE *fp=fopen(file,"r");
   struct Employee E;
   while( fscanf(fp,"%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age)!=EOF){
```

```
      if(key!=E.age);
 else{
        fprintf(stdout,"%u  %-14s  %-14s  %-7u  %-3u\n",E.empid,E.name,E.dep,E.sal,E.age);
        return;
      }
   }
   printf("Such Employee details not present\n");
   fclose(fp);
}
void searchEmpDep(char*file,char*key)
{
   FILE *fp=fopen(file,"r");
  struct Employee E;
  while( fscanf(fp,"%u %s %s %u %u",&E.empid,E.name,E.dep,&E.sal,&E.age)!=EOF){
      if(strcmp(E.dep,key));
      else{
        fprintf(stdout,"%u  %-14s  %-14s  %-7u  %-3u\n",E.empid,E.name,E.dep,E.sal,E.age);
        return;
      }
   }
   printf("Such Employee details not present\n");
   fclose(fp);
}
```

LABSET 2 (STACK IMPLEMENTATION)

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

```
#define MAX 5
typedef struct{
   int top;
   int arr[MAX];
}Stack;
void push(int,Stack*);
int pop(Stack*);
void display(Stack*);
bool isEmpty(Stack*);
int peek(Stack*);


int main(){
   Stack s;
   int ch,n;
   s.top=-1;
   for(;;)
   {
   printf("Enter the choice to operate on stack\n1- PUSH\n2- POP\n3- PEEk\n4- DISPLAY\n5-
EXIT\n");
   scanf("%d",&ch);
   switch(ch){
   case 1: printf("Enter the number to be pushed\n");scanf("%d",&n);
         if(s.top==MAX-1)
            printf("Stack Overflow\n");
         else
            push(n,&s);
         break;
   case 2: if(isEmpty(&s))
```

```
            printf("Stack Underflow\n");
        else
        printf("Element popped is = %d\n",pop(&s));
        break;
    case 3: if(isEmpty(&s))
                printf("Stack is Empty\n");
        else
            printf("Element at the top is = %d\n",peek(&s));
        break;
    case 4: printf("The elements of the stack are\n\n");
        if(isEmpty(&s))
                printf("Stack is Empty\n");


            display(&s);
    break;
    case 5: printf("Have a Good Day\n\n");
        exit(0);
    default:printf("Entered Wrong choice!!\nPlease Enter the correct choice!\n\n");
    }
    }
}
void push(int n,Stack* st){
    st->arr[++st->top]=n;


}
int pop(Stack* st){
return st->arr[st->top--];
```

```
}
int peek(Stack* st){
return st->arr[st->top];
}
void display(Stack* st){
    int i;
    for(i=st->top;i>=0;i--)
   printf("%d\t",st->arr[i]);
}
bool isEmpty(Stack* st){
    if(st->top==-1)
        return true;
    else
        return false;
}
```

LABSET 3 (INFIX TO POSTFIX)

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include<ctype.h>
typedef struct{
int top;
```

```
char arr[30];
}Stack;
void push(Stack*,char);
char peek(Stack*);
char pop(Stack*);
bool isEmpty(Stack*);
int prec(char);
bool isoperand(char);
void InfixToPost(char*iexp,char*exp);
void display(Stack*);
int main()
{
   char iexp[30];
   char exp[30];
   printf("Enter the string of expression\n");
   scanf("%s",iexp);
   InfixToPost(iexp,exp);
   return 0;
}
void InfixToPost(char*iexp,char*exp){
    Stack stk;
     stk.top=-1;
   int i=0,k=0,temp=0;
   char ch;
   for(i=0;iexp[i]!='\0';i++)
   {
      if(isalnum(iexp[i]))
```

```
    exp[k++]=iexp[i];
  else if(iexp[i]=='(')
     push(&stk,iexp[i]);
  else if(iexp[i]==')')
  {
     while((ch=pop(&stk))!='('){
           exp[k++]=ch;
         }
  if(!isEmpty(&stk)&&peek(&stk)!='(')
    {
       printf("Invalid Expression\n\n");
       exit (0);
    }

  }
  else {
     while(!isEmpty(&stk)&&prec(iexp[i])<=prec(peek(&stk))){
           if(iexp[i]==peek(&stk)&&prec(&stk)==3);
              break;
        exp[k++]=pop(&stk);
     }

     push(&stk,iexp[i]);
     printf("operator pushed\n");
     }
}
while(!isEmpty(&stk)){
```

```
        exp[k++]=pop(&stk);
    }
    exp[k++]='\0';
    printf("\n%s",exp);
}
bool isEmpty(Stack*st)
{
    if(st->top==-1)
        return true;
    else
        return false;
}
void push(Stack*st,char value){
    st->arr[++st->top]=value;
    }
char pop(Stack*st){
    return (st->arr[st->top--]);
}
char peek(Stack*st){
    return (st->arr[st->top]);
}
bool isoperand(char c){
    if(c>='a'&&c<='z')
        return true;
    else
        return false;
}
```

```
int prec(char c){
   switch(c){
       case '(':return 0;
       break;
      case '+':
      case '-':return 1;
       break;
     case '*':
     case '/':return 2;
       break;
     case '^':
     case '$':return 3;
       break;
          }
}
void display(Stack*st){
   int i;
   for(i=st->top;i>=0;i--)
            printf("%c\t",st->arr[i]); }
```

LABSET 4(Evaluation of Prefix Expression)

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <math.h>
typedef struct{
   int top;
```

```
   float arr[20];
}Stack;
void push(Stack* stk,float value);
float pop(Stack*stk);
float peek(Stack*stk);
float eval(float op1,float op2,char sym);
bool isOperator(char opr);
int main(){
Stack s;
s.top=-1;
int i,l=0;
float op1,op2;
char sym,prefix[20];
printf("Enter the prefix expression\n\n");
scanf("%s",prefix);

for(i=0;prefix[i]!='\0';i++)//calculating length of string,as strlen not working
   ++l;
for(i=l-1;i>=0;i--){
   sym=prefix[i];
   if(isdigit(sym))
      push(&s,sym-'0');
   else if(isOperator(sym))
      {
         op1=pop(&s);
         op2=pop(&s);
         push(&s,eval(op1,op2,sym));
```

```
        }
    else{
        printf("Invalid Expression\n");return 1;
        }
}
printf("The result of the expression = %g\n\n",pop(&s));
return 0;
}

void push(Stack* stk,float value){
    stk->arr[++stk->top]=value;
}
float pop(Stack*stk){
return (stk->arr[stk->top--]);
}

float peek(Stack*stk){
return (stk->arr[stk->top]);
}
bool isOperator(char opr){
    if(opr=='+'||opr=='-'||opr=='*'||opr=='/'||opr=='^'||opr=='$')
    return true;
return false;
}
float eval(float op1,float op2,char sym){
    switch(sym){
        case '+':return op1+op2;
```

```
        break;
        case '-':return op1-op2;
        break;
        case '*':return op1*op2;
        break;
        case '/':return op1/op2;
        break;
        case '^':
        case '$':return pow(op1,op2);
           }

    return -1;
}
```

LABSET 5 (Ordinary Queue Implementation)

```
#include <stdio.h>
#include <stdlib.h>
#define max 5
typedef struct{
    int f;
    int r;
```

```
    int arr[max];


}Queue;
void enqueue(Queue*q,int val);
void display(Queue*q);
int dequeue(Queue*q);
int isFull(Queue*q);
int isEmpty(Queue*q);
int main()
{
    int ch,val;
    Queue q;
    q.f=-1;
    q.r=-1;
    printf("Queue operations \n");


    for(;;){


enterCh:printf("1.Enqueue\n\n2.Dequeue\n3.Display\n\n");
    printf("Enter your choice\n");
    scanf("%d",&ch);
    switch (ch){
        case 1: if(isFull(&q)){
                    printf("Queue is full\n");
                    printf("Enter any other choice\n");goto enterCh;
                    }
        printf("Enter the element (integer) to be enqueued\n");
```

```
        scanf("%d",&val);
        enqueue(&q,val);
        break;
        case 2: if(isEmpty(&q)){
                printf("Queue underflow!!!\n");
                printf("Enter any other choice\n");goto enterCh;
                }
        printf("The element dequeued is %d \n",dequeue(&q));

        break;
        case 3: if(isEmpty(&q)){
                printf("Queue underflow!!!\n");
                printf("Enter any other choice\n");goto enterCh;
                }
        display(&q);

        break;
        case 4: printf("Thank You. \n ");
              exit (0);
              break;
        default:printf("Wrong choice entered\n");
        printf("Enter any other valid choice\n");goto enterCh;
    }
    }
    return 0;
}
```

```
void enqueue(Queue*q,int val){
if(q->f==-1)
   q->f=0;
q->arr[++q->r]=val;
}
int dequeue(Queue*q){
   int val=q->arr[q->f];
if(q->r==q->f){
   q->r=-1;
   q->f=-1;
}
else
   q->f++;

return val;
}
void display(Queue*q){
int i;
printf("\n\n*** Elements of the queue ***\n\n");
printf("Front-> %d\n",q->f);
for (i=q->f;i<=q->r;i++)
   printf("%d\n",q->arr[i]);
printf("Rear-> %d\n",q->r);
}
int isEmpty(Queue*q){
if(q->f==-1)
return 1;
```

```
return 0;
}
int isFull(Queue*q){
if(q->r==max-1)
   return 1;
return 0;
}
```

LABSET 6 (Circular Queue Implementation)

```
#include <stdio.h>
#include <stdlib.h>
#define max 5

typedef struct{
   int f;
   int r;
   int arr[max];

}Queue;
void enqueue(Queue*q,int val);
void display(Queue*q);
int dequeue(Queue*q);
int isFull(Queue*q);
int isEmpty(Queue*q);
int main()
{
   int ch,val;
```

```
    Queue q;
    q.f=-1;
    q.r=-1;
    printf("Circular Queue operations \n");

    for(;;){

enterCh:printf("1.Enqueue\n\n2.Dequeue\n3.Display\n\n");
    printf("Enter your choice\n");
    scanf("%d",&ch);
    switch (ch){
       case 1: if(isFull(&q)){
                printf("Queue is full\n");
                printf("Enter any other choice\n");goto enterCh;
                }
       printf("Enter the element (integer) to be enqueued\n");
       scanf("%d",&val);
       enqueue(&q,val);
       break;
       case 2: if(isEmpty(&q)){
                printf("Queue underflow!!!\n");
                printf("Enter any other choice\n");goto enterCh;
                }
       printf("The element dequeued is %d \n",dequeue(&q));

       break;
       case 3: if(isEmpty(&q)){
```

```
            printf("Queue underflow!!!\n");
            printf("Enter any other choice\n");goto enterCh;
            }
      display(&q);

      break;
      case 4: printf("Thank You. \n ");
            exit (0);
            break;
      default:printf("Wrong choice entered\n");
      printf("Enter any other valid choice\n");goto enterCh;

   }
   }

   return 0;
}

void enqueue(Queue*q,int val){
if(q->f==-1)
   q->f=0;
q->r=(q->r+1)%max;
q->arr[q->r]=val;
}
int dequeue(Queue*q){
int val=q->arr[q->f];
if(q->r==q->f){
```

```
    q->r=-1;
    q->f=-1;
}
else
    q->f=(q->f+1)%max;

return val;
}
void display(Queue*q){
int i;
printf("\n\n*** Elements of the queue ***\n\n");
printf("Front-> %d\n",q->f);
for (i=q->f; ;i=(i+1)%max){
    printf("%d\n",q->arr[i]);
    if(i==q->r)
    break;
    }
printf("Rear-> %d\n",q->r);
}
int isEmpty(Queue*q){
if(q->f==-1)
return 1;
return 0;
}
int isFull(Queue*q){
if((q->r+1)%max==q->f)
    return 1;
```

```
return 0;
}
```

LABSET 7 (Linked List Operations)

```
#include <stdio.h>
#include <stdlib.h>
struct node{
int data;
struct node*next;
};
typedef struct node* Nodeptr;
Nodeptr getnode(void){
   Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
   return newNode;
}
void insertRear(Nodeptr*first,int val){
Nodeptr last=*first;
Nodeptr newNode=getnode();
newNode->data=val;
newNode->next=NULL;
```

```
if(*first==NULL){
   *first=newNode;
   return;
}
while(last->next!=NULL)
last=last->next;
last->next=newNode;
return;
}
int pop(Nodeptr *first){
   Nodeptr temp=*first;
   if(temp!=NULL){
   int val=temp->data;
   *first=temp->next;
   free(temp);
   return val;
   }
   else
   return -999;
}
int deleteAtpos(Nodeptr *first,int pos){
Nodeptr temp=*first;
Nodeptr curr=NULL;
int val;
if(pos==1&&(temp==NULL||temp!=NULL)){
val=temp->data;
*first=temp->next;
```

```
free(temp);
return val;
}
if(pos>lengthList(temp)||pos<1){
printf("Invalid Position\n");
return -999;
}
int i;
for(i=1;i<pos&&temp!=NULL;i++){
curr=temp;
temp=temp->next;
}
val=temp->data;
curr->next=temp->next;
free(temp);
return val;
}
void insertAtPos(Nodeptr *first,int val,int pos){
Nodeptr temp=*first;
Nodeptr prev=NULL;
int i;
Nodeptr newnode=getnode();
newnode->data=val;
if(temp==NULL&&pos==1){
*first=newnode;
newnode->next=NULL;
return;
```

```
}
if(pos>lengthList(temp)||pos<1){
printf("Invalid position");
return;
}
if(temp!=NULL&&pos==1){
newnode->next=temp;
*first=newnode;
return;
}
for(i=1;i<pos&&temp!=NULL;i++){
   prev=temp;
   temp=temp->next;
}
prev->next=newnode;
newnode->next=temp;
}
void reverseList(Nodeptr *first){
Nodeptr rev=NULL;
Nodeptr temp=NULL;
while(*first!=NULL){
temp=*first;
*first=(*first)->next;
temp->next=rev;
rev=temp;
}
*first=rev;
```

```
}
void display(Nodeptr *first){
Nodeptr temp=*first;
if(temp==NULL){
printf("List is Empty\n");
return;
}
while(temp!=NULL){
printf("%d->",temp->data);
temp=temp->next;
}
printf("\n");
return;
}
int lengthList(Nodeptr temp){
int count=0;
while(temp!=NULL){
count++;
temp=temp->next;
}
return count;
}
int main()
{
   int ch,val,pos;
   Nodeptr head=NULL;
   printf("Linked list operations\n");
```

```
   printf("Enter choice to perform the following operations\n");
   for(;;){
   printf("\n1.Insert at End\n2. Delete Front\n3. Insert at a position\n4. Display\n5. Reverse List\n6. DeleteAtpos");
   scanf("%d",&ch);
   switch(ch){
      case 1: printf("\nEnter an integer to add to the end\n");
            scanf("%d",&val);
            insertRear(&head,val);
            break;
      case 2:if((val=pop(&head))==-999)
            printf("List is Empty\n");
            else
            printf("The element deleted is %d ",val);
            break;
      case 3: printf("Enter the position of the node\n");
            scanf("%d",&pos);
            printf("Enter an integer to add to the position\n");
            scanf("%d",&val);
            insertAtPos(&head,val,pos);
            break;
      case 4: printf("The list is displayed below\n");
            display(&head);
            break;
      case 5:reverseList(&head);
            printf("The list is reversed\n");
            break;
      case 6:printf("Enter the position of the node to delete\n");
```

```
            scanf("%d",&pos);
            if((val=deleteAtpos(&head,pos))!=-999)
            printf("%d is deleted from position %d \n",val,pos);
            break;
    }
    }
    return 0;
}
```

LABSET 8 (Union Intersection)

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node* next;
};
typedef struct node* Nodeptr;
Nodeptr getNode(void){
    Nodeptr new_node=(Nodeptr)malloc(sizeof(struct node));
    return new_node;
}
void insertOrdered(Nodeptr *list,int val){
Nodeptr newNode=getNode();
Nodeptr temp=*list;
Nodeptr prev =NULL;
newNode->data=val;
if(*list==NULL){
```

```
    newNode->next=NULL;
    *list=newNode;
    return;
}
if(newNode->data<=temp->data){
        newNode->next=temp;
        *list=newNode;
        return;
}
while(temp!=NULL){
       if(newNode->data>temp->data){
    prev=temp;
    temp=temp->next;
       }
       else
        break;
}
prev->next=newNode;
newNode->next=temp;
return;
}
Nodeptr mergeList(Nodeptr list1,Nodeptr list2,Nodeptr list3){
if(list1==NULL&&list2==NULL)
    return;
else if(list1!=NULL&&list2==NULL){
    list3=list1;
    return list3;
```

```
}
else if(list2!=NULL&&list1==NULL){
   list3=list2;
   return list3;
}
Nodeptr temp1=list1;
while(temp1!=NULL){
insertOrdered(&list3,temp1->data);
temp1=temp1->next;
}
temp1=list2;
while(temp1!=NULL){
insertOrdered(&list3,temp1->data);
temp1=temp1->next;
}
return list3;
}
Nodeptr remDuplicate(Nodeptr list){
if(list==NULL)
  return;
Nodeptr prev=list;
Nodeptr temp=prev->next;
while(temp!=NULL){
          if(prev->data==temp->data){
              prev->next=temp->next;
              free(temp);
              temp=prev->next;
```

```
            continue;
        }
    prev=temp;
    temp=temp->next;
}
return list;
}
Nodeptr getUnion(Nodeptr list1,Nodeptr list2,Nodeptr list3){
list3=mergeList(list1,list2,list3);
list3=remDuplicate(list3);
return list3;
}
Nodeptr getIntersection(Nodeptr list1,Nodeptr list2,Nodeptr list3){
list1=remDuplicate(list1);
list2=remDuplicate(list2);
Nodeptr temp1=list1;
Nodeptr temp2=list2;
while(temp1!=NULL&&temp2!=NULL ){
    if(temp1->data==temp2->data){
        insertOrdered(&list3,temp1->data);
        temp1=temp1->next;
        temp2=temp2->next;
    }
    else if(temp1->data<temp2->data)
        temp1=temp1->next;
    else
        temp2=temp2->next;
```

```
}


return list3;
}
void display(Nodeptr first){
   Nodeptr temp=first;
   while(temp!=NULL){
      printf("%d -> ",temp->data);
      temp=temp->next;
   }
   printf("\n");
}
int main()
{
   int val;
   Nodeptr list1=NULL;
   Nodeptr list2=NULL;
   printf("Enter the values for list1 and enter ctrl+Z to exit\n");
   while(scanf("%d",&val)!=EOF){
      insertOrdered(&list1,val);  }
   printf("Your list1 is\n");
   display(list1);
   printf("\nEnter the values for list2 and enter ctrl+Z to exit\n");
   while(scanf("%d",&val)!=EOF){
      insertOrdered(&list2,val);
   }
   printf("\nYour list2 is\n");
```

```
    display(list2);
    Nodeptr list3=NULL;
    list3=getUnion(list1,list2,list3);
    printf("Union LIST\n\n");
    display(list3);
    list3=NULL;
    list3=getIntersection(list1,list2,list3);
    printf("Intersection LIST\n\n");
    display(list3);
    return 0;
}
```

LABSET 9

a)Stack using Singly Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int data;
struct node *next;
};
typedef struct node* Nodeptr;
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
return newNode;
}
void push(Nodeptr *first,int val){
    Nodeptr newNode=getNode();
```

```
    newNode->data=val;
    newNode->next=*first;
    *first=newNode;
}
int pop(Nodeptr*first){
    Nodeptr temp=*first;
    int val=temp->data;
    *first=temp->next;
    free(temp);
    return val;
}
void display(Nodeptr*first){
Nodeptr last=*first;
while(last!=NULL){
    printf("%d->",last->data);
    last=last->next;
}
printf("\n");
}
int main(){
    Nodeptr head=NULL;
    int ch;
    int val;
     printf("Stack operations through LINKED LIST\n\n");
    for(;;){
        enterCh:printf("Enter the choice\n");
        printf("1.PUSH\n2.POP\n3.PEEK\n4.DISPLAY\n5.EXIT\n");
```

```
scanf("%d",&ch);
switch (ch){
   case 1: printf("Enter the integer to push\n\n");
           scanf("%d",&val);
           push(&head,val);
           break;
   case 2: if(head==NULL){
              printf("STACK IS EMPTY\n");
              goto enterCh;
           }
           printf("The element popped is %d \n",pop(&head));
           break;
   case 3:if(head==NULL){
              printf("STACK IS EMPTY\n");
              goto enterCh;
           }
           printf("The element at the top is %d \n",head->data);
           break;
   case 4:if(head==NULL){
              printf("STACK IS EMPTY\n");
              goto enterCh;
           }
           display(&head);
           break;
   case 5: exit(0);
   break;
   default:printf("Wrong Choice\n");
```

```
    }
  }
}
```

b)Ordinary Queue using Linked List

```
#include<stdio.h>
#include<stdlib.h>
struct node{
int data;
struct node *next;
};
typedef struct node* Nodeptr;

struct Queue{
  Nodeptr f,r;
};
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
return newNode;
}
void enqueue(struct Queue*q,int val){
  Nodeptr newNode=getNode();
  newNode->data=val;
  newNode->next=NULL;
  if(q->f==NULL){
    q->f=newNode;
```

```
        q->r=newNode;
    }
    q->r->next=newNode;
    q->r=newNode;
}

int dequeue(struct Queue*q){
    Nodeptr temp=q->f;
    q->f=q->f->next;
    int val=temp->data;
    free(temp);
    if(q->f==NULL)
        q->r=NULL;
    return val;
}
void display(struct Queue*q){
printf("front--");
Nodeptr temp=q->f;
while(temp!=q->r){
    printf("%d->",temp->data);
    temp=temp->next;
}
printf("%d->",temp->data);
printf("--rear\n");
}

int main(){
```

```
int ch;
int val;
struct Queue*q=(struct Queue*)malloc(sizeof(struct Queue));
q->r=q->f=NULL;
 printf("Queue operations through LINKED LIST\n\n");
for(;;){

   enterCh:printf("Enter the choice\n");
   printf("1.Enqueue\n2.Dequeue\n3.DISPLAY\n4.EXIT\n");
   scanf("%d",&ch);
   switch (ch){
      case 1: printf("Enter the integer to enqueue\n\n");
            scanf("%d",&val);
            enqueue(q,val);
            break;
      case 2: if(q->f==NULL){
               printf("Queue IS EMPTY\n");
               goto enterCh;
            }
            printf("The element dequeued is %d \n",dequeue(q));
            break;

      case 3:if(q->f==NULL){
               printf("QUEUE IS EMPTY\n");
               goto enterCh;
            }
```

```
                display(q);
                break;
          case 4: exit(0);
          break;
          default:printf("Wrong Choice\n");
       }
    }
}
```

c)Adding two given Polynomials

```
#include<stdio.h>
struct node{
float powx;
float powy;
int flag;
float coeff;
struct node*next;
};
typedef struct  node* Nodeptr;
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
return newNode;
}
void insertFront(Nodeptr *first,float cf,float px,float py){
   Nodeptr newNode=getNode();
   newNode->coeff=cf;
   newNode->powx=px;
```

```
    newNode->powy=py;
    newNode->flag=0;
    newNode->next=*first;
    *first=newNode;
}
void display(Nodeptr*first){
Nodeptr last=*first;
while(last!=NULL){
    printf("+ %.2g x^%.2g y^%.2g",last->coeff,last->powx,last->powy);
    last=last->next;
}
printf("\n");
}
float evaluatePolynomial(Nodeptr list){
    float x,y;
    float sum=0;
    printf("Enter the value of x\n");
    scanf("%f",&x);
    printf("Enter the value of y\n");
    scanf("%f",&y);
    while(list!=NULL){
        sum=sum+list->coeff*pow(x,list->powx)*pow(y,list->powy);
        list=list->next;
    }
return sum;
}
```

```
Nodeptr addPolynomials(Nodeptr list1,Nodeptr list2){
Nodeptr list3=NULL;
Nodeptr temp1=list1,temp2=list2;
while(temp1!=NULL){
   while(temp2!=NULL){
      if((temp1->powx==temp2->powx)&&(temp1->powy==temp2->powy)){
         insertFront(&list3,temp1->coeff+temp2->coeff,temp1->powx,temp1->powy);
         temp2->flag=1;
         temp1->flag=1;
      }
      temp2=temp2->next;
   }
   temp1=temp1->next;
   temp2=list2;
}
temp1=list1;
while(temp1!=NULL){
   if(temp1->flag==0)
      insertFront(&list3,temp1->coeff,temp1->powx,temp1->powy);
   temp1=temp1->next;
}
while(temp2!=NULL){
   if(temp2->flag==0)
      insertFront(&list3,temp2->coeff,temp2->powx,temp2->powy);
   temp2=temp2->next;
}
return list3;
```

```
}

int main(){
Nodeptr list1=NULL,list2=NULL,list3;int i;
float cf,px,py;char exitFlag;
printf("Enter the terms of the first polynomial\n");
for(i=0;;i++){
   printf("Press enter to add next term or Q to quit\n");
   exitFlag=getchar();
   if(exitFlag=='Q'||exitFlag=='q')
      break;
   printf("Enter for term %d\n",i+1);
   printf("Enter the coeff: ");scanf("%f",&cf);
   printf("Enter the power of x: ");scanf("%f",&px);
   printf("Enter the power of y: ");scanf("%f%*c",&py);
   insertFront(&list1,cf,px,py);

   }
printf("Enter the terms of the second polynomial\n");
for(i=0;;i++){
   printf("Press enter to add next term or Q to quit\n");
   exitFlag=getchar();
   if(exitFlag=='Q'||exitFlag=='q')
      break;
   printf("Enter for term %d\n",i+1);
   printf("Enter the coeff: ");scanf("%f",&cf);
   printf("Enter the power of x: ");scanf("%f",&px);
```

```
    printf("Enter the power of y: ");scanf("%f%*c",&py);
    insertFront(&list2,cf,px,py);
    }
list3=addPolynomials(list1,list2);
printf("The resultant of addition of the polynomials is\n");
display(&list3);
printf("The result of evaluation is %g\n",evaluatePolynomial(list3));
return 0;
}
```

LABSET 10 (Doubly Linked List using Header Nodes)

```
#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node*prev;
    struct node*next;
};
typedef struct node*Nodeptr;
Nodeptr getnode(void){
    Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
    return newNode;
}

void insertAfterInfo(Nodeptr*head,int info,int val){
Nodeptr temp=(*head)->next;
Nodeptr newNode=getnode();
```

```
newNode->data=val;
newNode->next=newNode->prev=NULL;
if(temp==NULL&&(*head)->data==info){
(*head)->next=newNode;
newNode->prev=*head;
(*head)->data++;
return;
}
while(temp!=NULL){
   if(temp->data==info)
      break;
   temp=temp->next;
}
if(temp==NULL){
   printf("Info Field does not exist\n");
   return ;
}
newNode->next=temp->next;
if(temp->next!=NULL)
   temp->next->prev=newNode;
newNode->prev=temp;
temp->next=newNode;
(*head)->data++;
}

void display(Nodeptr head){
   Nodeptr temp=head->next;
```

```
    printf("\n\n");
    if(temp==NULL)
        printf("List is Empty\n");
    while(temp!=NULL){
        printf("%d->",temp->data);
        temp=temp->next;
        }
printf("\n\n");
}

int deleteFirst(Nodeptr*last){
    if(*last==NULL)
        return -999;
    Nodeptr temp=*last;
    while(temp->prev!=NULL)
        temp=temp->prev;
    Nodeptr cur=temp->next;
    temp->next=cur->next;
    if(cur->next==NULL)
        *last=NULL;
    else
        cur->next->prev=temp;
    int val=cur->data;
    free (cur);
    return val;
}
```

```
int deleteAtPos(Nodeptr*head,int pos){
Nodeptr temp=(*head)->next,prevPtr;
int val,i;
Nodeptr cur=NULL;
if(pos==1)
{
   cur=temp;
   (*head)->next=temp->next;
   val=cur->data;
   if(cur->next!=NULL)
   cur->next->prev=*head;
   free(cur);
   return val;
}
for(i=1;i<pos;i++){
   prevPtr=temp;
   temp=temp->next;
}
prevPtr->next=temp->next;
if(temp->next!=NULL)
temp->next->prev=prevPtr;
val=temp->data;
free(temp);
return val;
}
void swap(Nodeptr*head,int m,int n)
```

```
{
   int count=1;
   Nodeptr curm=(*head)->next,curn=(*head)->next;
   if(m==n)
      return;
   while(count!=m){
      curm=curm->next;
      count++;
   }
   count=1;
   while(count!=n){
      curn=curn->next;
      count++;
   }
   int temp=curn->data;
   curn->data=curm->data;
   curm->data=temp;

}
int main()
{
   Nodeptr head=getnode();
   head->data=0;
   head->prev=head->next=NULL;
   Nodeptr last=NULL;
   int ch,x,val;
   for(;;){
```

```
   enterAgn:printf("1.Insert next to info given\n2.Delete 1st node if pointer to last node is
given\n3.Delete at Pos\n4.Display\n5.Swap two nodes\n6.Exit");
   printf("Enter choice\n");
   scanf("%d",&ch);
   switch(ch){
      case 1:if(head->next==NULL){
             printf("List is Empty.Enter the info field of header node i.e. 0 \n");
             }
             printf("Enter the info of the node \n");
             scanf("%d",&x);
             printf("Enter the value to insert\n");
             scanf("%d",&val);
             insertAfterInfo(&head,x,val);
             break;
    case 2:  if(head->data==0){
             printf("List is EMPTY\n\n");
             break;
             }
           last=head->next;
           while(last->next!=NULL)
            last=last->next;
           val=deleteFirst(&last);
           if(val!=-999){
              printf("The element deleted is %d\n",val);
              head->data--;
           }
           else
              printf("List is Empty\n");
```

```
            break;
    case 3:if(head->data==0){
             printf("List is Empty\n");
             goto enterAgn;}
          printf("Enter the pos to delete a node\n");
            scanf("%d",&x);
           if(x<1||x>head->data){
               printf("Invalid Position\n");
               goto enterAgn;
              }
          val=deleteAtPos(&head,x);
            if(val!=-999){
               printf("The element deleted is %d\n",val);
               head->data--;
            }
            else
               printf("List is Empty\n");
            break;
case 4:display(head);
            break;
case 5:printf("Enter the position of the mth node \n");
             scanf("%d",&x);
             printf("Enter the position of the nth node \n");
             scanf("%d",&val);
             if(x<1||x>head->data||val<1||val>head->data){
                printf("Invalid Position\n");
                goto enterAgn;
```

```
            }
            swap(&head,x,val);
            break;
          }
    }
    return 0;
}
```

LABSET 11 (DEQUE)

```
#include <stdio.h>
#include <stdlib.h>
struct node{
int data;
struct node*rlink;
struct node*llink;
};
typedef struct node*Nodeptr;
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
newNode->llink=newNode->rlink=NULL;
return newNode;
}
void insertFront(Nodeptr *first,int val){
Nodeptr newNode=getNode();
newNode->data=val;
if(*first==NULL){
```

```
  *first=newNode;
  return;
}
Nodeptr temp=*first;
newNode->rlink=temp;
temp->llink=newNode;
*first=newNode;
}
void insertRear(Nodeptr *first,int val){
Nodeptr newNode=getNode();
newNode->data=val;
if(*first==NULL){
  *first=newNode;
  return;
}
Nodeptr temp=*first;
while(temp->rlink!=NULL)
   temp=temp->rlink;
newNode->llink=temp;
temp->rlink=newNode;
}

void deleteFront(Nodeptr*first){
if((*first)->rlink==NULL){
   printf("%d is deleted\n",(*first)->data);
   free(*first);
   *first=NULL;
```

```
      return;
}
Nodeptr temp=*first;
*first=temp->rlink;
if((*first)->rlink!=NULL)
   (*first)->rlink->llink=*first;
printf("%d is deleted\n",temp->data);
free(temp);
}
void deleteRear(Nodeptr*first){
if((*first)->rlink==NULL){
   free(*first);
   *first=NULL;
        return;
}
Nodeptr last=*first;

while(last->rlink!=NULL)
   last=last->rlink;
printf("%d is deleted\n",last->data);
last->llink->rlink=NULL;
free(last);
}
void display(Nodeptr first){

   while(first!=NULL)
   {
```

```
        printf("%d-> ",first->data);
        first=first->rlink;
    }
    printf("\n\n");

}
int main(){

    int ch;
    int val;
    Nodeptr head=NULL;
     printf("Double Queue operations through LINKED LIST\n\n");
    for(;;){

        enterCh:printf("Enter the choice\n");
        printf("1.Insert Rear\n2.Insert Front\n3.Delete Front\n4.Delete Rear\n5.DISPLAY\n6.EXIT\n");
        scanf("%d",&ch);
        switch (ch){
            case 1: printf("Enter the integer to insert\n\n");
                    scanf("%d",&val);
                    insertRear(&head,val);
                    break;
            case 2: printf("Enter the integer to insert\n\n");
                    scanf("%d",&val);
                    insertFront(&head,val);
                    break;
            case 3: if(head==NULL){
```

```
                printf("Queue IS EMPTY\n");
                goto enterCh;
            }
            deleteFront(&head);
            break;

        case 4:if(head==NULL){
                printf("QUEUE IS EMPTY\n");
                goto enterCh;
            }
            deleteRear(&head);
            break;
        case 5: if(head==NULL){
                printf("QUEUE IS EMPTY\n");
                goto enterCh;
            }
            display(head);
            break;
        case 6: exit(0);
        default:printf("Wrong Choice\n");
      }
   }}
```

LABSET 12 (BST)

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

struct node{
```

```
int data;
struct node*rlink;
struct node*llink;
};
typedef struct node*Nodeptr;
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
newNode->llink=newNode->rlink=NULL;
return newNode;
}
void insert(Nodeptr *root,int val){
Nodeptr temp=*root;
if(*root==NULL){
   *root=getNode();
   (*root)->data=val;
   return;
}
if(val<temp->data)
   insert(&(temp->llink),val);
else if(val>temp->data)
   insert(&(temp->rlink),val);
else
   printf("Data already present\n");
return;
}

void inorder(Nodeptr root){
```

```
if(root!=NULL){
inorder(root->llink);
printf("%d ",root->data);
inorder(root->rlink);
}
}
void preorder(Nodeptr root){
if(root!=NULL){
printf("%d ",root->data);
preorder(root->llink);
preorder(root->rlink);
}
}
void postorder(Nodeptr root){
if(root!=NULL){
postorder(root->rlink);
postorder(root->llink);
printf("%d ",root->data);
}
}
Nodeptr deleteKey(Nodeptr root,int key){

if(root==NULL){
   printf("Data not found\n");
   return;
}
if(key<root->data)
```

```
    root->llink=deleteKey(root->llink,key);
else if(key>root->data)
    root->rlink=deleteKey(root->rlink,key);
else{
    if(root->llink==NULL)
    {
        Nodeptr q=root->rlink;
        printf("%d deleted successfully\n",root->data);
        free(root);
        return q;
    }
    else if(root->rlink==NULL)
    {
        Nodeptr q=root->llink;
        printf("%d deleted successfully\n",root->data);
        free(root);
        return q;
    }
    else{
    Nodeptr succ=root->rlink;
    while(succ->llink!=NULL)
        succ=succ->llink;
    printf("%d deleted successfully\n",root->data);
    root->data=succ->data;

    free(succ);
    }
```

```
    return root;
}



}
int main()
{
    Nodeptr root=NULL;
    int ch,val;
    for(;;){
        printf("\n\n1.Insert\n2.In order Traversal\n3.Pre order Traversal\n4.Post order
Traversal\n5.Delete\n\n");
        printf("Enter the choice: ");
        scanf("%d",&ch);
        switch(ch){
            case 1: printf("\nEnter the value to insert into the tree: ");scanf("%d",&val);
                    insert(&root,val);
                    break;
            case 2: printf("\nInorder Traversal\n\n");
                    inorder(root);
                    break;
            case 3: printf("\nPreorder Traversal\n\n");
                    preorder(root);
                    break;
            case 4: printf("\nPostorder Traversal\n\n");
                    postorder(root);
                    break;
            case 5: printf("\nEnter the key element to delete from tree: ");scanf("%d",&val);
```

```
                root=deleteKey(root,val);
                break;
          default:printf("Invalid case\n\n");
       }
       }

   return 0;
}
```

LABSET 13 (Expression Tree)

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

struct node{
char data;
struct node*rlink;
struct node*llink;
};
typedef struct node*Nodeptr;
Nodeptr getNode(void){
Nodeptr newNode=(Nodeptr)malloc(sizeof(struct node));
newNode->llink=newNode->rlink=NULL;
return newNode;
}
void createTree(Nodeptr*root,char*exp){
```

```
int i,top=-1;
Nodeptr s[10];
Nodeptr newNode;
char c;
for(i=0;exp[i]!='\0';i++)
{   c=exp[i];
    newNode=getNode();
    newNode->data=c;
    if(isalnum(c))
        s[++top]=newNode;

    else{
       newNode->rlink=s[top--];
       newNode->llink=s[top--];
       s[++top]=newNode;
        }

}

*root=s[top--];
}
void inorder(Nodeptr root){
if(root!=NULL){
inorder(root->llink);
printf("%c ",root->data);
inorder(root->rlink);
}
```

```
}
float eval(Nodeptr root){
float n;
switch(root->data)
{
   case '+':return eval(root->llink)+eval(root->rlink);
   case '-':return eval(root->llink)+eval(root->rlink);
   case '/':return eval(root->llink)/eval(root->rlink);
   case '*':return eval(root->llink)*eval(root->rlink);
   case '^':
   case '$':return pow(eval(root->llink),eval(root->rlink));
   default:if(isalpha(root->data))
          {
             printf("\nEnter the value of %c  :",root->data);
             scanf("%f",&n);
             return n;
          }
          else
             return root->data-'0';
}
}

int main(){
char postfix[20];
Nodeptr root=NULL;
printf("Enter the post-fix expression\n");
scanf("%s",postfix);
```

```
createTree(&root,postfix);

inorder(root);

printf("The resultant value of the expression is %g\n",eval(root));

return 0;

}
```