SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Data Structures and Applications Laboratory [3RCSL01]

Program1: Develop a C program to create a sequential file for storing employee records with each record having following information:

Employee_Id	Name	Department	Salary	Age
Non-Zero Positive integer	25 Characters	25 Characters	Positive Integer	Positive integer

Write necessary functions to perform the following operations:

a) Read the details of a record.
b) Display all the records in the file.
c) Search for a specific records based on Department. In case if the required record is not found, suitable message should be displayed.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
struct EMPLOYEE
{
        int empid;
        char name[20];
        char dept[20];
        int salary;
        int age;
}e;

 void add_record(FILE *fp)
 {
         printf("\nEnter the details of the employee.............\n");
         printf("ID: ");
         scanf("%d",&e.empid);
         printf("Name: ");
         scanf("%s",e.name);
         printf("Department: ");
         scanf("%s",e.dept);
         printf("Salary: ");
         scanf("%d",&e.salary);
         printf("Age: ");
         scanf("%d",&e.age);
         fprintf(fp,"%d\t%s\t%s\t%d\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);
         printf("\nRecord saved successfully");
}
```

```
 void display_records(FILE *fp)
 {
     printf("ID\t\tNAME\t\tDEPT\t\tSalary\t\tAGE\n");
     printf("------------------------------------------------------------------\n");

     while((fscanf(fp,"%d%s%s%d%d",&e.empid,e.name,e.dept,&e.salary,&e.age))!=EOF)
         printf("%d\t\t%s\t\t%s\t\t%d\t\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);

}

 void search_record(FILE *fp)
 {
          int flag=0;
          char dept[20];

          printf("\nEnter the dept to search: ");
          scanf("%s",dept);

      while((fscanf(fp,"%d%s%s%d%d",&e.empid,e.name,e.dept,&e.salary,&e.age))!=EOF)
    {
          if(strcmp(e.dept,dept)==0)
          {
                   if(flag==0)
                  {
                       printf("\nSearch Successful !!!");
                       printf("\nID\t\tNAME\t\tDEPT\t\tSalary\t\tAGE\n");
                       printf("------------------------------------------------------------------\n");
                       flag=1;
                   }
          printf("%d\t\t%s\t\t%s\t\t%d\t\t%d\n",e.empid,e.name,e.dept,e.salary,e.age);
          }
 }
 if(flag==0)
      printf("\nFailure, no such record found !!!");
 }

int main()
{
    FILE *fp;
    int choice;

    while(1)
    {

      printf("\n\n1:Add_Record\n2:Search_Record\n3:Display_Records\n4:Exit");
      printf("\nEnter your choice: ");
      scanf("%d",&choice);
```

```
   switch(choice)
  {
       case 1:   fp=fopen("empfile1","a");
                 if(fp==NULL)
                      printf("\nError in opening file");
                 else
                 {
                       add_record(fp);
                       fclose(fp);
                 }
                 break;

       case 2:   fp=fopen("empfile1","r");
                 if(fp==NULL)
                     printf("\nError in opening file");
                 else
                 {
                      search_record(fp);
                      fclose(fp);
                 }
                 break;

       case 3:   fp=fopen("empfile1","r");
                 if(fp==NULL)
                      printf("\nNo records to display !!!");
                  else
                  {
                     display_records(fp);
                     fclose(fp);
                  }
                  break;

       case 4: exit(0);

       default: printf("\nInvalid choice !!!");
     }
  }
 return 0;
}
```

Program2: Develop a C program to implement Stack of names to perform the push, pop and display operations.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAXSIZE 3
typedef struct
{
    char items[MAXSIZE][25];
    int top;
}STACK;

int isfull(STACK s)
{
    if(s.top==MAXSIZE-1)
        return 1;
    return 0;
}

int isempty(STACK s)
{
    if(s.top==-1)
        return 1;
    return 0;

}

void PUSH(STACK *s,char name[])
{
    strcpy(s->items[++s->top],name);
    printf("\nName %s is pushed on to the stack",name);
}

char* POP(STACK *s)
{
    return(s->items[s->top--]);
}

void DISPLAY(STACK s)
{
    int i;
    printf("\nSTACK CONTENTS:\nBOS->");
    for(i=0;i<=s.top;i++)
        printf("%s->",s.items[i]);
     printf("TOS");
}
```

```
int main()
{
  STACK s;
  int choice;
  char name[20];
  s.top=-1;

  while(1)
  {
    printf("\n\n1:PUSH\n2:POP\n3:DISPLAY\n4:EXIT");
    printf("\nEnter your choice: ");
    scanf("%d",&choice);
    switch(choice)
    {
      case 1: if(isfull(s))
                   printf("\nStack Overflow !!!");
            else
           {
                   printf("\nEnter the name to be pushed: ");
                   scanf("%s",name);
                   PUSH(&s,name);
            }
            break;

      case 2: if(isempty(s))
                  printf("\nStack Underflow !!!");
            else
                  printf("\nName %s is popped from the Stack",POP(&s));
            break;

      case 3: if(isempty(s))
                  printf("\nStack is Empty !!!");
            else
                  DISPLAY(s);
            break;

      case 4:exit(0);

      default: printf("\nInvalid choice");
    }
  }
  return 0;
}
```

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Data Structures and Applications Laboratory [3RCSL01]

Program3: Develop a C program to convert a valid infix expression to postfix.

Solution:

```
#include<stdio.h>
#include<ctype.h>
#define MAXSIZE 25

typedef struct
{
        char items[MAXSIZE];
        int top;
}STACK;

void PUSH(STACK *s,char data)
{
        s->items[++s->top]=data;
 }

char POP(STACK *s)
{
        return(s->items[s->top--]);
}

char PEEK(STACK s)
{
        return(s.items[s.top]);
}

int preced(char symb)
{
        switch(symb)
       {
              case '#':
              case '(': return 0;

              case '+':
              case '-': return 1;

              case '*':
              case '/':
              case '%': return 2;
       case '$':
       case '^': return 3;
   }
}

int main()
{
    STACK s;
    char infix[30],postfix[30],symb,ch;
    int i,j=0;

    s.top = -1;

    printf("\nEnter a valid infix expression:\n");
    scanf("%s",infix);

    PUSH(&s,'#');
    for(i=0;infix[i]!='\0';i++)
   {
        symb=infix[i];

        if(isalnum(symb))
             postfix[j++]=symb;
       else
       {
             switch(symb)
            {
                  case '(': PUSH(&s,'(');
                          break;

                  case ')': while((ch=POP(&s))!='(')
                               postfix[j++]=ch;
                          break;

                  default: while(preced(symb)<=preced(PEEK(s)))
                          {
                              if(symb==PEEK(s) && preced(symb)==3)
                                  break;

                                postfix[j++]  = POP(&s);
                          }
                          PUSH(&s,symb);
               }
          }

       }
```

```
        while(PEEK(s)!='#')
                postfix[j++]=POP(&s);
        postfix[j]='\0';

        printf("\nResultant Postfix Expression:\n");
        printf("%s",postfix);
        return 0;
}
```

Program4: Develop a C program to evaluate the given postfix expression.

Solution:

```
#include<stdio.h>
#include<math.h>
#include<ctype.h>
#define MAXSIZE 25

typedef struct
{
      float items[MAXSIZE];
      int top;
}STACK;

void PUSH(STACK *s,float data)
{
       s->items[++s->top]=data;
}

float POP(STACK *s)
{
       return(s->items[s->top--]);
}

float compute(float op1,char symb,float op2)
{
       switch(symb)
      {
            case '+': return op1+op2;

            case '-':  return op1-op2;

            case '*': return op1*op2;

            case '/': return op1/op2;

            case '$':
            case '^': return pow(op1,op2);
        }
  }

  int main()
  {
         STACK s;
         char postfix[30],symb;
         float n1,n2,res,data;
         int i;

         s.top=-1;
        printf("\nEnter a valid postfix expression:\n");
        scanf("%s",postfix);

        for(i=0;postfix[i]!='\0';i++)
        {
              symb=postfix[i];

              if(isdigit(symb))
                   PUSH(&s,symb-'0');
              else if(isalpha(symb))
             {
                   printf("\n%c = ",symb);
                   scanf("%f",&data);
                   PUSH(&s,data);
              }
             else
             {
                  n2=POP(&s);
                  n1=POP(&s);
                  res=compute(n1,symb,n2);
                  PUSH(&s,res);
              }
          }

          printf("\nResult of evaluation: %f",POP(&s));
          return 0;
}
```

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Data Structures and Applications Laboratory [3RCSL01]

Program5: *Develop a C program to implement Linear Queue of characters to perform the insertion, deletion and display operations.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE  3
typedef struct
{
        char items[MAXSIZE];
        int f,r;
}QUEUE;
int isfull(QUEUE q)
{
        if(q.r == MAXSIZE-1)
             return 1;
        return 0;
}

int isempty(QUEUE q)
{
       if(q.f == -1)
            return 1;
       return 0;
}

void INSERT(QUEUE *q,char data)
{
        q->items[++q->r]=data;
        printf("\nCharacter \'%c\' is inserted into queue",data);
        if(q->f==-1)
            q->f=0;
}

char DELETE(QUEUE *q)
{
         char data;
         data = q->items[q->f];
         if(q->f == q->r)
              q->f = q->r = -1;
         else
              q->f++;
         return(data);
}
```

```
void DISPLAY(QUEUE q)
{
         int i;
         printf("\nQUEUE CONTENTS:\nFRONT->");
         for(i=q.f;i<=q.r;i++)
              printf("%c->",q.items[i]);
         printf("REAR");
}

int main()
{
       QUEUE q;
       int choice;
       char data;
       q.f=q.r=-1;
       while(1)
      {
            printf("\n\n1:Insert\n2:Delete\n3:Display\n4:Exit");
            printf("\nEnter your choice: ");
            scanf("%d",&choice);
            switch(choice)
           {
                    case 1: if(isfull(q))
                                      printf("\nQueue Overflow !!!");
                                 else
                               {
                                      printf("\nEnter the character to be inserted: ");
                                      getchar();
                                      scanf("%c",&data);
                                      INSERT(&q,data);
                               }
                               break;

                     case 2: if(isempty(q))
                                      printf("\nQueue Underflow !!!");
                                else
                                      printf("\nCharacter \’%c\’ is deleted from queue",DELETE(&q));
                                break;

                     case 3: if(isempty(q))
                                      printf("\nQueue is Empty !!!");
                                else
                                      DISPLAY(q);
                                break;
                     case 4: exit(0);
                    default: printf("\nInvalid choice");
             }
        }
       return 0;
}
```

Program6: *Develop a C program to implement Circular Queue of integers to perform the insertion, deletion and display operations.*

Solution:

```
#include<stdio.h>
#include<stdlib.h>
#define MAXSIZE  3
int count;
typedef struct
{
        int items[MAXSIZE];
        int f,r;
}QUEUE;

int isfull(QUEUE q)
{
        if(q.f == (q.r+1)%MAXSIZE)
              return 1;
        return 0;
}

int isempty(QUEUE q)
{
       if(q.f == -1)
              return 1;
       return 0;
}

void INSERT(QUEUE *q,int data)
{
        q->r=(q->r+1)%MAXSIZE;
        q->items[q->r]=data;
        printf("\n%d is inserted into circular queue",data);
        count++;
        if(q->f==-1)
               q->f=0;
}

int DELETE(QUEUE *q)
{
         int data;
         data = q->items[q->f];
         count--;
         if(q->f == q->r)
               q->f = q->r = -1;
         else
               q->f = (q->f +1)%MAXSIZE;
         return(data);
}
```

```
void DISPLAY(QUEUE q)
{
        int i;
        printf("\nQUEUE CONTENTS:\nFRONT->");
        for(i=1;i<=count;i++)
       {
               printf("%d->",q.items[q.f]);
               q.f=(q.f+1)%MAXSIZE;
       }
        printf("REAR");
}
int main()
{
      QUEUE q;
      int choice;
      int data;
      q.f = q.r = -1;
      while(1)
     {
           printf("\n\n1:Insert\n2:Delete\n3:Display\n4:Exit");
           printf("\nEnter your choice: ");
           scanf("%d",&choice);
           switch(choice)
          {
                   case 1: if(isfull(q))
                                   printf("\nCircular Queue Overflow !!!");
                           else
                           {
                                   printf("\nEnter the data to be inserted: ");
                                   scanf("%d",&data);
                                   INSERT(&q,data);
                           }
                           break;
                   case 2:  if(isempty(q))
                                   printf("\nCircular Queue Underflow !!!");
                            else
                                   printf("\n%d is deleted from queue",DELETE(&q));
                            break;
                   case 3:  if(isempty(q))
                                   printf("\nCircular Queue is Empty !!!");
                            else
                                   DISPLAY(q);
                            break;
                   case 4:  exit(0);
                   default: printf("\nInvalid choice");
            }
    }
    return 0;
}
```

SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Data Structures and Applications Laboratory [3RCSL01]

Program7: *Define a structure to represent a node in a Singly Linked List. Each node must contain following information: player name, team name and batting average. Develop a C program using functions to perform the following operations on a list of cricket players:*

a) Add a player at the end of the list.

b) Search for a specific player and update his/her batting average if the player exists.

c) Display the details of all the players.

Solution:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
        char player[20];
        char team[20];
        float bavg;
        struct node *next;
}NODE;

NODE * addPlayer(NODE *first)
{
        NODE *newnode,*temp;
        newnode=(NODE*)malloc(sizeof(NODE));
        newnode->next=NULL;
        printf("\nEnter the player details.....\n");
        printf("Name: ");scanf("%s",newnode->player);
        printf("Team: ");scanf("%s",newnode->team);
        printf("Batting Average: ");scanf("%f",&newnode->bavg);

        if(first==NULL)
             first=newnode;
         else
         {
             temp=first;
             while(temp->next!=NULL)
                    temp=temp->next;
              temp->next=newnode;
         }
         printf("\nPlayer %s is added at the end of the list",newnode->player);
         return first;
}
```

```
void display(NODE *first)
{
        if(first==NULL)
        {
               printf("\nEmpty list");
               return;
         }
         printf("\nPlayer Details........\n");
         printf("\nNAME\tTEAM\tBATTING AVERAGE\n");
         while(first!=NULL)
         {
               printf("%s\t%s\t%f\n",first->player,first->team,first->bavg);
               first=first->next;
          }
}

NODE *searchPlayer(NODE *first)
{
         NODE *temp;
         char player[20];

         if(first==NULL)
               printf("\nEmpty list");
         else
         {
               printf("\nEnter the player name to search: ");
               scanf("%s",player);

               temp=first;
               while(temp!=NULL && strcmp(temp->player,player)!=0)
                     temp=temp->next;

               if(temp==NULL)
                    printf("\nPlayer %s not existing in the list",player);
               else
               {
                   printf("\nPlayer %s is existing in the list",player);
                   printf("\nCurrent batting average: %f",temp->bavg);
                   printf("\nEnter new value for batting average: ");
                   scanf("%f",&temp->bavg);
                   printf("\nBatting average of player %s is updated successfully ", player);
               }
           }
           return first;
}
```

```
int main()
{
        NODE *first=NULL;
        int choice;

        while(1)
        {
                printf("\n1:ADD PLAYER\n2:SEARCH PLAYER\n3:DISPLAY PLAYER\n4:EXIT");
                printf("\nEnter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: first=addPlayer(first);
                                break;

                        case 2: first=searchPlayer(first);
                                break;

                        case 3: display(first);
                                break;

                        case 4: exit(0);
                       default: printf("\nInvalid choice");
                 }
          }

          return 0;
}
```

Program8: *Develop a C program to add two two-variable polynomials using Singly Linked list.*

Solution:

```
#include <stdio.h>
#include<stdlib.h>
typedef struct node
{
        float coeff;
        float powx;
        float powy;
        int flag;
        struct node *next;
}NODE;

NODE * ins_last(NODE *first,float cf,float px,float py)
{
        NODE *newnode,*temp;
        newnode=(NODE*)malloc(sizeof(NODE));
        newnode->coeff=cf;
        newnode->powx=px;
        newnode->powy=py;
        newnode->flag=0;
        newnode->next=NULL;
        if(first==NULL)
                first=newnode;
         else
         {
                temp=first;
                while(temp->next!=NULL)
                        temp=temp->next;
                temp->next=newnode;
          }
          return first;
}

NODE * read_P(NODE *first)
{
        float cf,px,py;

        printf("\nEnter the coefficient: ");
        scanf("%f",&cf);

        while(cf!=999)
        {
                printf("\nEnter power of x: ");
                scanf("%f",&px);
                printf("\nEnter power of y: ");
        scanf("%f",&py);
        first=ins_last(first,cf,px,py);

        printf("\nEnter the coefficient: ");
        scanf("%f",&cf);
    }
    return first;
}

void display(NODE *first)
{
    if(first==NULL)
    {
        printf("\nEmpty list");
        return;
    }

    while(first->next!=NULL)
    {
        printf("%.0f x^%0.f y^%0.f + ",first->coeff,first->powx,first->powy);
        first=first->next;
    }
    printf("%.0f x^%0.f y^%0.f ",first->coeff,first->powx,first->powy);
}

NODE *add_p(NODE *p1,NODE *p2,NODE *p3)
{
    NODE *temp;
    float cf;
    temp=p2;
    while(p1!=NULL)
    {
        while(p2!=NULL)
        {
            if((p1->powx==p2->powx) &&(p1->powy==p2->powy))
                break;
            p2=p2->next;
        }
        if(p2==NULL)
            p3=ins_last(p3,p1->coeff,p1->powx,p1->powy);
        else
        {
            cf=p1->coeff + p2->coeff;
            if(cf!=0)
            {
                p3=ins_last(p3,cf,p1->powx,p1->powy);
                p2->flag=1;
            }
        }
```

```
                p2=temp;
                p1=p1->next;
        }

        p2=temp;
        while(p2!=NULL)
        {
              if(p2->flag==0)
                    p3=ins_last(p3,p2->coeff,p2->powx,p2->powy);
              p2=p2->next;
       }
       return p3;
}

int main()
{
         NODE *p1=NULL,*p2=NULL,*p3=NULL;

         printf("\nEnter the first polynomial:\n");
         p1=read_P(p1);

         printf("\nEnter the second polynomial:\n");
         p2=read_P(p2);

         p3=add_p(p1,p2,p3);

         printf("\n\nFirst polynomial:\n");
         display(p1);

         printf("\n\nSecond polynomial:\n");
         display(p2);

         printf("\n\nResultant polynomial:\n");
         display(p3);


         return 0;
}
```

Siddaganga Institute of Technology, Tumkur-572103

Department of Computer Science and Engineering

Data Structures and Applications Laboratory [3RCSL01]

LAB PROGRAM9: *Develop a C program to construct two ordered singly linked lists using functions to perform following operations:*

- *Insert an element into a list.*
- *Merge the two lists.*
- *Display the contents of the list.*

Display the two input lists and the resultant list with suitable messages.

Solution:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
        int  info;
        struct node *next;
}NODE;

NODE *insert(NODE *first,int data)
{
        NODE *newnode,*temp,*prev;
        newnode=(NODE*)malloc(sizeof(NODE));
        newnode->info=data;
        if(first==NULL || data<first->info)
        {
              newnode->next=first;
              first=newnode;
        }
        else
        {
               temp=first;
               while(temp!=NULL && data>temp->info)
               {
                      prev=temp;
                      temp=temp->next;
                }
                if(temp==NULL || data!=temp->info)
                {
                       prev->next=newnode;
                        newnode->next=temp;
                }
           }
           return first;
  }
NODE * ins_last(NODE *first,int data)
```

```
 {
     NODE *newnode,*temp;
     newnode = (NODE*)malloc(sizeof(NODE));

     newnode->info = data;
     newnode->next = NULL

     if(first == NULL)
           first = newnode;
     else
    {
         temp = first;
         while(temp->next!=NULL)
                temp = temp->next;
          temp->next = newnode;
     }
    return(first);
 }

void display(NODE *first)
{
        if(first==NULL)
        {
              printf("Empty");
              return;
         }
        printf("Contents:\nBegin->");
        while(first!=NULL)
        {
              printf("%d->",first->info);
              first=first->next;
         }
        printf("End");
}

NODE *merge(NODE *L1,NODE *L2)
{
        NODE *L3=NULL;
        if(L1==NULL && L2==NULL)
        {
              printf("\nList1 and List2 are Empty");
              return NULL;
         }
         while(L1!=NULL && L2!=NULL)
         {
               if(L1->info<L2->info)
               {
                      L3=ins_last(L3,L1->info);
                      L1=L1->next;
                }
                else if(L2->info<L1->info)
                {
                        L3=ins_last(L3,L2->info);
                        L2=L2->next;
                    }
                    else
                    {
                        L3=ins_last(L3,L1->info);
                        L1=L1->next;
                        L2=L2->next;
                    }
                }
                while(L1!=NULL)
                {
                    L3=ins_last(L3,L1->info);
                    L1=L1->next;
                }
                while(L2!=NULL)
                {
                    L3=ins_last(L3,L2->info);
                    L2=L2->next;
                }
                printf("\nLists are merged successfully");
                return L3;
        }

    int main()
    {
            NODE *L1=NULL,*L2=NULL,*L3=NULL;
            int data,choice;

            while(1)
            {
                    printf("\n\n1:INS_LIST1\n2:INS_LIST2\n3:MERGE\n4:DISPLAY\n5:EXIT");
                    printf("\nEnter your choice: ");
                    scanf("%d",&choice);
                    switch(choice)
                    {
                            case 1: printf("\nEnter the data: ");
                                    scanf("%d",&data);
                                    L1=insert(L1,data);
                                    break;

                            case 2: printf("\nEnter the data: ");
                                    scanf("%d",&data);
                                    L2=insert(L2,data);
                                    break;

                            case 3: L3=merge(L1,L2);
                                    printf("\nList3 ");
                                    display(L3);
                                    break;

                            case 4: printf("\nList1 ");
                    display(L1);
                    printf("\nList2 ");
                    display(L2);
                    break;

            case 5: exit(0);
            default: printf("\nInvalid choice");
        }
    }
    return 0;
}
```

LAB PROGRAM10: *Define a structure to represent a node in a Linear Doubly Linked List with header node. Each node must contain following information: Student name, USN, branch and year of admission. Header node should maintain the count of number of students in the list. Develop a C program using functions to perform the following operations on a list of students:*

a) Add a student at the beginning of the list.
b) Display the details of the students of a specified branch.
Display the details of all the students.

Solution:

```
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
        char  name[20];
        char usn[20];
        char branch[20];
        int year;
        struct node *lptr,*rptr;
}NODE;

void ins_first(NODE *head)
{
        NODE *newnode;
        newnode=(NODE*)malloc(sizeof(NODE));

        printf("\nEnter the details of the student...\n");
        printf("Name: ");scanf("%s",newnode->name);
        printf("USN: ");scanf("%s",newnode->usn);
        printf("Branch: ");scanf("%s",newnode->branch);
        printf("Year of admission: ");scanf("%d",&newnode->year);

        newnode->lptr=head;

        newnode->rptr=head->rptr;

        if(head->rptr!=NULL)
               head->rptr->lptr=newnode;

        head->rptr=newnode;

        printf("\nStudent is added successfully to the list");
        head->year++;
}
```

```
void display1(NODE *head)
{
        NODE *first;
        char branch[20];
        int flag=0;

        if(head->rptr==NULL)
        {
               printf("\nEmpty list");
               return;
        }

        printf("\nEnter the branch: ");
        scanf("%s",branch);

        first=head->rptr;
        while(first!=NULL)
        {
               if(strcmp(first->branch,branch)==0)
               {
                    if(flag==0)
                   {
                       printf("\nList of students belonging to branch %s\n",branch);
                       printf("\n\nName\tUSN\tYear of admission\n");
                       flag=1;
                   }
                   printf("%s\t%s\t%d\n",first->name,first->usn,first->year);
                }
                first=first->rptr;
         }
         if(flag==0)
              printf("\nFailure, no student from branch %s",branch);
        }


void display2(NODE *head)
{
        NODE *first;

        if(head->rptr==NULL)
        {
               printf("\nEmpty list");
               return;
        }

        printf("\nName\tUSN\tBranch\tYear of admission\n");
        first=head->rptr;
```

```
    while(first!=NULL)
    {
        printf("%s\t%s\t%s\t%d\n",first->name,first->usn,first->branch,first->year);
        first=first->rptr;
     }
     printf("\nTotal number of students = %d",head->year);
}

int main()
{
    NODE *head;
    int choice;

    head=(NODE*)malloc(sizeof(NODE));
    head->lptr=head->rptr=NULL;
    head->year=0;

    while(1)
    {
        printf("\n1:Add student\n2:Display based on branch\n3:Display all\n4:exit");
        printf("\nEnter your choice: ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1: ins_first(head);
                    break;

            case 2: display1(head);
                     break;

            case 3: display2(head);
                    break;

            case 4: exit(0);

            default: printf("\nInvalid choice");
         }
    }
    return 0;
}
```

LAB PROGRAM11: *Develop a C program to implement Josephus problem using Circular Singly Linked List. Write necessary functions to perform the following operations:*

a) Add a soldier to the list.

b) Delete a soldier from the list.

Solution:

```
/*C program to implement Josephus Problem*/
#include <stdio.h>
#include<stdlib.h>
#include<string.h>
typedef struct node
{
        char  name[20];
        struct node *next;
}NODE;

/*C function to insert a node at the end of the CSLL*/
NODE *ins_last(NODE *last,char name[])
{
        NODE *newnode;

        newnode=(NODE*)malloc(sizeof(NODE));
        strcpy(newnode->name,name);

        if(last==NULL)
                last=newnode;
        else
                newnode->next=last->next;

        last->next=newnode;
        return(newnode);
}



/*C function to delete a node from the CSLL*/
NODE *del_node(NODE *last)
{

        NODE *temp;
        temp=last->next;
        printf("%s ",temp->name);
        last->next=temp->next;
        free(temp);
        return(last);
}
```

```
int main()
{
        NODE *last=NULL;
        char name[20];
        int i,n;

        printf("\nEnter the value of n: ");
        scanf("%d",&n);

        printf("\nEnter the names of the soldiers, type end to terminate:\n");
        scanf("%s",name);
        while(strcmp(name,"end")!=0)
        {
            last=ins_last(last,name);
            scanf("%s",name);
        }

        if(last==NULL)
            printf("\nEmpty list");
        else
        {
            printf("\n\nThe order in which soldiers are eliminated: ");
            while(last->next!=last)
            {
                for(i=1;i<n;i++)
                   last=last->next;

                last=del_node(last);
            }
            printf("\n\nThe soldier who escapes: %s\n",last->name);
         }
         return 0;
}
```

LAB PROGRAM12: *Develop a C program to perform the following operations:*

a) Construct a binary search tree of integers.
b) Traverse the tree in Inorder.
c) Delete a given node from the BST.

Solution:

```
#include<stdio.h>
#include<stdlib.h>
typedef struct node
{
    int info;
    struct node *lchild,*rchild;
}NODE;

NODE * insert(NODE *root,int data)
{
    NODE *newnode,*temp,*parent;
    newnode=(NODE*)malloc(sizeof(NODE));
    newnode->lchild=newnode->rchild=NULL;
    newnode->info=data;

    if(root==NULL)
        root=newnode;
    else
    {
        temp=root;
        while(temp!=NULL)
       {
            parent=temp;
            if(data > temp->info)
                temp=temp->rchild;
            else if(data < temp->info)
                temp=temp->lchild;
            else
            {
                printf("\nData %d is already existing in the BST",data);
                return(root);
            }
        }
        if(data > parent->info)
            parent->rchild=newnode;
        else
            parent->lchild=newnode;
      }
      printf(“\n%d is inserted into BST”,data);
      return(root);
}
```

```
void inorder(NODE *root)
{
        if(root==NULL)
             return;
        inorder(root->lchild);
        printf("%d ",root->info);
        inorder(root->rchild);
 }

NODE *del_key(NODE *root,int key)
{
        NODE *cur,*q,*parent,*successor;
        parent=NULL,cur=root;
        while(cur!=NULL)
        {
                if(cur->info==key)
                     break;
                parent=cur;
                cur= (key<cur->info)?cur->lchild:cur->rchild;
        }

        if(cur==NULL)
        {
                printf("\nKey %d is not found",key);
                return root;
        }
        if(cur->lchild==NULL)
               q=cur->rchild;
        else if(cur->rchild==NULL)
               q=cur->lchild;
         else
         {
               successor = cur->rchild;
               while(successor->lchild != NULL)
                     successor = successor->lchild;

               successor->lchild = cur->lchild;
               q = cur->rchild;
          }
          if (parent == NULL)
          {
               printf("\n%d is deleted from BST",key);
               free(cur);
               return q;
           }

          if(cur == parent->lchild)
                   parent->lchild = q;
          else
```

```
                parent->rchild = q;
            printf("\n%d is deleted from BST",key);
            free(cur);

            return  root;
}

int main()
{
        int choice,data,key;
        NODE *root=NULL;
        while(1)
        {
                printf("\n1:Insert 2:Inorder 3:Delete 4:Exit");
                printf("\nEnter your choice: ");
                scanf("%d",&choice);
                switch(choice)
                {
                        case 1: printf("\nEnter data to be inserted: ");
                                scanf("%d",&data);
                                root=insert(root,data);
                                break;

                        case 2: if(root==NULL)
                                    printf("\nEmpty Tree");
                                else
                                {
                                    printf("\nInorder Traversal: ");
                                    inorder(root);
                                }
                                break;

                        case 3: if(root==NULL)
                                      printf("\nEmpty Tree");
                                else
                                {
                                      printf("\nEnter the key to delete: ");
                                      scanf("%d",&key);

                                      root=del_key(root,key);
                                }
                                break;

                       case 4: exit(0);
                       default: printf("\nInvalid choice");
                }
        }
        return 0;
}
```

LAB PROGRAM13: *Develop a C program to construct an expression tree for a given postfix expression and evaluate the expression tree.*

Solution:

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

typedef struct node
{
        char info;
        struct node *lchild,*rchild;
}NODE;

NODE * create_tree(char postfix[])
{
        NODE *newnode, *stack[20];
        int i=0, top = -1;
        char ch;

         while((ch=postfix[i++])!='\0')
        {
                newnode = (NODE*)malloc(sizeof(NODE));
                newnode->info = ch;
                newnode->lchild = newnode->rchild = NULL;

                if(isalnum(ch))
                        stack[++top]=newnode;
                else
                {
                        newnode->rchild = stack[top--];
                       newnode->lchild = stack[top--];
                       stack[++top]=newnode;
                }
          }
          return(stack[top--]);
}


float  eval(NODE *root)
{
         float num;
         switch(root->info)
         {
                 case  '+' : return (eval(root->lchild) + eval(root->rchild));
                 case  '-' : return (eval(root->lchild) - eval(root->rchild));
                 case  '*' : return (eval(root->lchild) * eval(root->rchild));
                 case  '/' : return (eval(root->lchild) / eval(root->rchild));
                case  '^' : return (pow(eval(root->lchild), eval(root->rchild)));

                default:    if(isalpha(root->info))
                            {
                                printf("\n%c = ",root->info);
                                scanf("%f",&num);
                                return(num);
                            }
                            else
                                return(root->info - '0');
        }
}


int main()
{
        char postfix[30];
        float res;
        NODE * root = NULL;

        printf("\nEnter a valid Postfix expression\n");
        scanf("%s",postfix);

        root = create_tree(postfix);

        res = eval (root);

        printf("\nResult = %f",res);
        return 0;
}
```