R V College of Engineering
(Autonomous institute affiliated to VTU, Belgaum)
Department of Computer Science and Engineering
Mysore Road, Bangalore

B.E - Computer Science & Engineering

LABORATORY MANUAL

DATA STRUCTURES IN C LABORATORY

(12CS33)

PREPARED BY:

Prof. Girish Rao Salanke N S

1. Use Stack operations to do the following:

i) Assign to a variable name Y the value of the third element from the top of the stack and keep the stack undisturbed.

ii) Given an arbitrary integer n pop out the top n elements. A message should be displayed if an unusual condition is encountered.

iii) Assign to a variable name Y the value of the third element from the bottom of the stack and keep the stack undisturbed.
(Hint: you may use a temporary stack)

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 20

void push(int s[],int item,int *top)
{
  if(*top == SIZE -1)
    printf("\n STACK OVERFLOW");
  else
    s[++(*top)]=item;
}

int pop(int s[],int *top)
{
  if(*top == -1)
   {
     printf("\n STACK UNDERFLOW");
     return -1;
   }
  else
  {
    return s[(*top)--];
  }
}

void display(int s[],int top)
{
 int i;
 if(top == -1)
   printf("\n STACK IS EMPTY");
 else
 {
  printf("\n the content of stack is \n");
  for(i=top;i>=0;i--)
  printf("%d\n",s[i]);
 }
}
void pop_top(int s[],int *top)
{
```

```
  int i,y,top1=-1;
  int ts[SIZE];
  if(*top < 2)
    printf("\n stack contains less elements");
  else
   {
      for(i=0;i<2;i++)
    push(ts,pop(s,top),&top1);
      y=pop(s,top);
      printf("\n The top third element is %d\n",y);
      push(s,y,top);
      for(i=0;i<2;i++)
    push(s,pop(ts,&top1),top);
    }
}
void pop_bottom(int s[],int *top)
{
  int i,y,top1=-1;
  int ts[SIZE];
  if(*top < 2)
    printf("\n stack contains less elements");
  else
   {
      for(i=*top;i>=3;i--)
    push(ts,pop(s,top),&top1);
      y=pop(s,top);
      printf("\n The top third element is %d\n",y);
      push(s,y,top);
      while(top1!=-1)
    push(s,pop(ts,&top1),top);
    }
}

void pop_n(int s[],int *top,int n)
{
 int i;
 if(*top < n)
  printf("\n less elements cant pop");
 else
  {
    printf("\n The %d elements popeed are\n",n);
    for(i=0;i<n;i++)
      printf("%d\n",pop(s,top));
  }
}

int main()
{
  int top=-1,item,ch,n;
  int s[SIZE];
  for(;;)
  {
   printf("\n 1.PUSH");
   printf("\n 2.POP");
```

```
    printf("\n 3.DISPLAY");
    printf("\n 4.THIRD ELEMENT FROM TOP");
    printf("\n 5.THIRD ELEMENT FROM BOTTOM");
    printf("\n 6.POP N ELEMENTS");
    printf("\n 7.EXIT");
    printf("\n Read choice :");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:printf("\n enter the lement to be pushed :");
       scanf("%d",&item);
       push(s,item,&top);
       break;
      case 2:item=pop(s,&top);
       if(item !=-1)
       printf("\n The element poped is %d\n",item);
       break;
      case 3:display(s,top);
       break;
      case 4:pop_top(s,&top);
       break;
      case 5:pop_bottom(s,&top);
       break;
      case 6:printf("\n enter the number of lements to be popped :");
       scanf("%d",&n);
       pop_n(s,&top,n);
       break;
      default :exit(0);
    }
   }
return 0;
}
```

2. **Write a program to determine if an input character string is of the form XaY. X is a string of arbitrary length using only the characters from A and B. For example, X may be ABBAB. Y is a string which is the reverse of X. Thus for the string X given above, Y is BABBA. A is any arbitrary character which is not A or B. Given ABAAcABAB the program should display a message that this string is invalid. For the string ABBABCBABBA the program should write a message that it is valid. Use appropriate data structure.**

```
#include<stdio.h>
#include<stdlib.h>
#define SIZE 20

void push(char s[],char item,int *top)
{
  if(*top == SIZE -1)
    printf("\n STACK OVERFLOW");
  else
    s[++(*top)]=item;
}

char pop(char s[],int *top)
{
   return s[(*top)--];
}
void check(char s[],int *top)
{
  char s1[SIZE];
  int flag=0;
  int top1=-1,i,j,m,n;
  while(s[*top] == 'A' || s[*top] == 'B')
       push(s1,pop(s,top),&top1);
  pop(s,top);
  m=*top;
  n=top1;
  if(m==n)
  {
  for(i=*top,j=top1; i>=0,j>=0; i--,j--)
   {
       if (s[i]==s1[j])
      flag=1;
       else
      flag=0;
      break;
   }
  }
 if(flag==1)
    printf("\n valid string");
  else
    printf("\n Invalid string");
```

```
}
int main()
{
  int i,top=-1;
  char s[SIZE];
  char input[20];
  printf("\n enter the string to be verfied\n");
  scanf("%s",input);
  for(i=0;input[i]!='\0';i++)
        push(s,input[i],&top);
  check(s,&top);
  return 0;
}
```

3. **Write a C program that parses Infix arithmetic expressions to Postfix arithmetic expressions using a Stack.**

```
#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#define SIZE 10

void push(char s[],char symbol,int *top)
{
  s[++(*top)]=symbol;
}
char pop(char s[],int *top)
{
  return s[(*top)--];
}
int preced(char symbol)
{
  switch(symbol)
   {
     case '$':return 5;
     case '/':
     case '*':return 4;
     case '+':
     case '-':return 1;
   }
}

int infixtopostfix(char infix[10],char postfix[10])
{
  int i,j=0,top=-1;
  char s[SIZE];
  char symbol,temp;
  for(i=0;infix[i]!='\0';i++)
  {
    symbol=infix[i];
    if(isalnum(symbol))
      postfix[j++]=symbol;
    else
    {
      switch(symbol)
      {
      case '(': push(s,symbol,&top);
              break;
      case ')': temp=pop(s,&top);
              while(temp!='(')
              {
                postfix[j++]=temp;
                temp=pop(s,&top);
              }
              break;
      case '$':
      case '+':
```

```
        case '-':
        case '*':
        case '/': if(top==-1)
                    push(s,symbol,&top);
                  else
                  {
                   while(preced(s[top])>=preced(symbol))
                   postfix[j++]=pop(s,&top);
                   push(s,symbol,&top);
                  }
                  break;
        default: printf("invalid expression");
                 return 0;
         }
       }
    }
    while(top!=-1)
     postfix[j++]=pop(s,&top);
     postfix[j]='\0';
     return 1;
}

int main()
{
    int flag=0;
    char infix[15],postfix[15];
    printf("\n Enter the infix expression : ");
    scanf("%s",infix);
    flag=infixtopostfix(infix,postfix);
    if(flag==1)
      printf("\n postfix expression is : %s",postfix);
    else
      printf("\n error");
      return 0;
 }
```

4. Write a C program to Build Binary Heap to simulate Priority queue.

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

void heap(int a[], int n)
{
       int key, k, pos, j;
       for( k=0; k<n; k++)
        {
             key = a[k];
             pos = k;
             j = (pos-1)/2;
             while( pos > 0 && key>a[j])
              {
                   a[pos] = a[j];
                   pos = j;
                   j = (pos-1)/2;
              }
             a[pos] = key;
         }
}

int main()
  {
      int i, a[10], n;
      printf("Enter n\n");
      scanf("%d", &n);
      printf("Enter Data\n");
      for( i=0; i<n; i++ )
       scanf("%d", &a[i]);
      heap(a,n);
      printf("\n Max Heap : \n");
      for( i=0; i<n; i++ )
       printf("%d\t", a[i]);
      return 0;
   }
```

5. Write a C program to simulate the working of Messaging System in which a message is placed in a Queue by a Message Sender, a message is removed from the queue by a Message Receiver, which can also display the contents of the Queue.

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define QSIZE 10
void insert(char q[QSIZE][10],char *msg,int *f,int *r)
{
  if(*f==(*r+1)%QSIZE)
    printf("\n queue is full");
  else
    {
      *r=(*r+1)%QSIZE;
      strcpy(q[*r],msg);
      if(*f == -1)
      *f = *f + 1;
     printf("\n The message is sent...");
    }
}
char *delet(char q[QSIZE][10],int *f,int *r)
{
  char *del;
  if(*f == -1)
  {
    printf("\n queue is empty");
  }
  else
  {
     del=q[*f];
     if(*f == *r)
     {
       *f = -1;
       *r = -1;
     }
     else
     {
     *f=(*f+1)%QSIZE;
     }
     return del;
  }
}
void display(char q[QSIZE][10],int f,int r)
{
  int i;
```

```
  if(f == -1)
  printf("\n queue is empty");
  else if(f <= r)
   {
    printf("\n the content of queue\n");
    for(i=f;i<=r;i++)
    printf("%s\n",q[i]);
  }
  else
   {
   printf("\n the queue content is \n");
   for(i=r;i<QSIZE;i++)
   printf("%s\n",q[i]);
   for(i=0;i<=f;i++)
   printf("%s\n",q[i]);
  }
}
int main()
{
  int f=-1,r=-1;
  char *msg,*msgdel;
  char queue[QSIZE][10];
  int ch;
  for(;;)
  {
    printf("\n 1.insert");
    printf("\n 2.delete");
    printf("\n 3.display");
    printf("\n 4.exit");
    printf("\n enter ur choice :");
    scanf("%d",&ch);
    switch(ch)
    {
            case 1:printf("\n enter the msg to be inserted \n");
            scanf("%s",msg);
            insert(queue,msg,&f,&r);
            break;
      case 2:msgdel=delet(queue,&f,&r);
             printf("\n the msg retrived is %s",msgdel);
             break;
      case 3:display(queue,f,r);
             break;
       default : exit(0);
    }
  }
  return 0;
 }
```

6. **Write a C program to accept 2 singly linked lists & print the elements which are common in both the lists.**

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

struct node
{
int info;
struct node *link;
};

typedef struct node *NODE;

NODE insert(int item,NODE first)
{
       NODE temp,cur;
       temp=(NODE)malloc(sizeof(struct node));
       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;
}
void display(NODE first)
{
       NODE temp;
       if(first==NULL)
         printf("\n list is empty");
       else
       {
       temp=first;
       while(temp!=NULL)
        {
             printf("%d ",temp->info);
             temp=temp->link;
        }
        }
}


void split(NODE first)
{
       NODE temp;
       NODE one=NULL;
       NODE two=NULL;
       NODE three=NULL;
       NODE four=NULL;
       temp=first;
```

```
        while(temp!=NULL)
        {
         switch((temp->info) % 4)
           {
              case 1: one=insert(temp->info, one);
                      break;

              case 2: two=insert(temp->info, two);
                      break;

              case 3: three=insert(temp->info, three);
                      break;

              case 0: four=insert(temp->info, four);
                      break;
           }
        temp=temp->link;
        }

        printf("\nFirst list : ");

        display(one);

        printf("\nSecond list : ");

        display(two);

        printf("\nThird list : ");

        display(three);

        printf("\nFourth list : ");

        display(four);

    }
int main()
{
        int i,n;
        NODE first;
        first=NULL;
        printf("enter the number of elements : ");
        scanf("%d",&n);
        for(i=1;i<n+1;i++)
              first=insert(i,first);
        printf("\nEnterd list : ");
        display(first);
        split(first);
        return 0;
 }
```

7. Implement working of lift using appropriate data structure.

```
//Implementation of Lift using DOUBLY LINKED LIST

/*Working procedure,
1. Lift always goes from GROUND_FLOOR floor to TOP_FLOOR and comes back
to GROUND_FLOOR and so on.
2. At any floor, people can enter and give the details of desired floor
3. When the floor they want to go, comes, all those who had selected
the floor leave the lift
4. You cannot call the lift from outside.
5. This lift attends each floor, even if no person requests from
outside*/

#include<stdio.h>
#include<malloc.h>
#define TOTAL_FLOOR 5
#define UPWARD 1
#define DOWNWARD 0
#define GROUND_FLOOR 0
#define TOP_FLOOR TOTAL_FLOOR-1
#define LIFT_CAPACITY 15
struct lift
{
int floor_person, floor_info;
struct lift *down, *up; //down is left-link,  and up is right-link
};
typedef struct lift *LIFT;

LIFT getfloor(int floor_number)
{
      LIFT temp = (LIFT)malloc(sizeof(struct lift));
      temp->floor_person = 0, temp->floor_info = floor_number;
      return temp;
}
void main()
{
      int  i;
      int people, lift_people = 0, temp_floor, direction = 1, error;
//direction, 0-down, 1-up
      int floor_data[LIFT_CAPACITY];
      LIFT floor[TOTAL_FLOOR], lift_curr_floor;

      //Generating Floors
      for (i = GROUND_FLOOR; i <= TOP_FLOOR; i++)
            floor[i] = getfloor(i);
```

```
	//Calibrating Floors and lift
	floor[GROUND_FLOOR]->down = NULL;
	floor[GROUND_FLOOR]->up = floor[1];
	floor[TOP_FLOOR]->up = NULL;
	floor[TOP_FLOOR]->down = floor[TOP_FLOOR - 1];

	for (i = 1; i < TOP_FLOOR; i++)
	{
		floor[i]->down = floor[i - 1];
		floor[i]->up = floor[i + 1];
	}

	//Setting lift at GROUND_FLOOR floor
	lift_curr_floor = floor[GROUND_FLOOR];

	//Starting our lift
	while (1)
	{
		//Displaying The direction of Lift movement
		if (direction)
			puts("\nGoing UP");
		else
			puts("\nGoing DOWN");
		//Printing Current lift-floor details
		printf("\nFloor- %d", lift_curr_floor->floor_info);

/*If any person wants to leave the lift at the present floor (desired
floor), he/she leaves */
	if (lift_curr_floor->floor_person)
	{
		if (lift_curr_floor->floor_person == 1)
			puts("\n1 Person Exits");
		else
		printf("\n%d People Exit", lift_curr_floor->floor_person);

	}

		lift_people -= lift_curr_floor->floor_person;
//Updating the "number of people inside lift" after people exit lift
		lift_curr_floor->floor_person = 0;
//Resetting the "current floor Call"
		printf("\nInside Lift - %d", lift_people);
//Printing updated "number of people inside lift"
		people = 0;
```

```
/* Lift accepts people from floor, if and only if the "number of people
inside lift" < "Lift capacity" */
            if (lift_people < LIFT_CAPACITY)
            {
                  do
                  {
                        do
                        {
                              puts("\nEnter Lift ");
                              scanf("%d", &people);
                        } while (people<0);
//Lift overloading situation
                        if (lift_people + people > LIFT_CAPACITY)
                              printf("\nLift   Overloaded,   can   accept
only %d more!", LIFT_CAPACITY - lift_people);
                  } while (lift_people + people > LIFT_CAPACITY);
                  lift_people += people;
//Updating the "number of people inside lift" after people enter
            }


      //If someone enters lift, details of "desired floor" is collected

            if (people)
            {
                  do
                  {
                        printf("\nPlease  Enter  Floor  Details  for  %d
person(s)  ", people);
                        for (i = 0; i < people; i++)
                        {
                              floor_data[i] = 0;
                              scanf("%d", &temp_floor);
                              if (temp_floor > TOP_FLOOR || temp_floor
< GROUND_FLOOR)
                              {
                                    puts("\nFloor Doesnt Exist");
                                    error = 1;
//setting the error tag high to raise error
                                    break;
                              }
```

```
                                    else
                                    {
                                          floor_data[i] = temp_floor;
                                          error = 0;
      // keeping error tag low to say normal execution, no error
                                    }
                              }
                        } while (error);
                        for (i = 0; i<people; i++)
                              (floor[floor_data[i]]->floor_person)++;
                  }

//Changing the direction of movement of lift at TOP FLOOR and GROUND FLOOR

                  if (lift_curr_floor->up == NULL)
                  {
                        direction = DOWNWARD;
                        lift_curr_floor = lift_curr_floor->down;
                  }
                  else if (lift_curr_floor->down == NULL)
                  {
                        direction = UPWARD;
                        lift_curr_floor = lift_curr_floor->up;
                  }

/*Normal movement of lift in the same previous direction, for intermediate
floors */

                  else
                  {
                        if (direction)
                              lift_curr_floor = lift_curr_floor->up;
                        else
                              lift_curr_floor = lift_curr_floor->down;
                  }
            }
      }
```

8. Consider a linked list with n integers. Each node of the list is numbered from 1 to n. Develop a program using 'C' language to split this list into 4 lists so that:

a. First list contains nodes numbered 1,5,9,13,_,_,_
b. Second list contains nodes numbered 2,6,10,14,_,_
c. Third list contains nodes numbered 3,7,11,15,_,_
d. Fourth list contains nodes numbered 4,8,12,16,_

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

struct node
{
   int info;
  struct node* link;
};
typedef struct node* NODE;

NODE insert(int item,NODE first)
{
      NODE temp,cur;
      temp=(NODE)malloc(sizeof(struct node));
      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;
}

void display(NODE first)
{
      NODE temp;
      temp=first;
      while(temp!=NULL)
      {
            printf("%d ",temp->info);
            temp=temp->link;
      }
}

void check(NODE first,NODE second)
{
  NODE temp1,temp2;
  for(temp1=first;temp1!=NULL;temp1=temp1->link)
    for(temp2=second;temp2!=NULL;temp2=temp2->link)
      if(temp1->info==temp2->info)
         printf("%d\t",temp1->info);
```

```
}

int main()
{
      int i,item,m,n;
      NODE first,second;
      first=NULL;
      second=NULL;
      printf("enter the number of elements in first list : ");
        scanf("%d",&m);
      for(i=1;i<=m;i++)
      {
         scanf("%d",&item);
         first=insert(item,first);
      }
      printf("enter the number of elements in second list : ");
        scanf("%d",&n);
      for(i=1;i<=n;i++)
      {
         scanf("%d",&item);
         second=insert(item,second);
      }
      printf("\nEnterd first list : \n");
      display(first);
      printf("\nEntered second list : \n");
      display(second);
      printf("\n the comman elements are\n");
      check(first,second);
      return 0;
}
```

9. Implement a program to multiply two polynomials using circular linked list.

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

struct node
{
  int coeff;
  int power;
  struct node *link;
};
typedef struct node *NODE;

NODE insertr(NODE poly,int co,int po)
{
  NODE temp,cur;
  temp=(NODE)malloc(sizeof(struct node));
  temp->coeff=co;
  temp->power=po;
  if(poly->link == poly)
   {
    temp->link=poly;
    poly->link=temp;
    return poly;
   }
 cur=poly->link;
 while(cur->link != poly)
    cur=cur->link;
 temp->link=poly;
 cur->link=temp;
 return poly;
}

void display(NODE poly)
{
  NODE temp;
  temp=poly->link;
  while(temp->link != poly)
  {
    if(temp->coeff !=0 )
       printf("%d*(x)^%d +",temp->coeff,temp->power);
     temp=temp->link;
   }
   printf(" %d*(x)^%d",temp->coeff,temp->power);
}

NODE AddNewTerm(NODE res, int co, int po)
{
  int flag=0;
  NODE temp,t,cur;
  if(res->link==res)
  {
```

```
    temp=(NODE)malloc(sizeof(struct node));
    temp->coeff=co;
    temp->power=po;
    temp->link=res;
    res->link=temp;
    return res;
  }
  else
  {
    for(temp=res->link;temp!=res;temp=temp->link)
    {
       if(temp->power == po)
      {
         temp->coeff=temp->coeff + co;
         flag=1;
         return res;
      }
    }
    if(flag==0)
     {
      t=(NODE)malloc(sizeof(struct node));
      t->coeff=co;
      t->power=po;
      cur=res->link;
      while(cur->link != res)
         cur=cur->link;
      cur->link=t;
      t->link=res;
      return res;
     }
   }
}
NODE multiplypoly(NODE firsth,NODE secondh)
{
  NODE res,f,s;
  res=(NODE)malloc(sizeof(struct node));
  res->link=res;
  for(f=firsth->link;f!=firsth;f=f->link)
     for(s=secondh->link;s!=secondh;s=s->link)
      res=AddNewTerm(res,f->coeff * s->coeff,f->power + s->power);
  return res;
}

int main()
{
  NODE firsth=NULL,secondh=NULL,resulth=NULL;
  int co,po,i,term1,term2;
  firsth=(NODE)malloc(sizeof(struct node));
  secondh=(NODE)malloc(sizeof(struct node));
  resulth=(NODE)malloc(sizeof(struct node));
  firsth->link=firsth;
  secondh->link=secondh;
  resulth->link=resulth;
  clrscr();
```

```
  printf("\n Read first polynomial\n");
  printf("\n enter the no of terms of first poly :");
  scanf("%d",&term1);
  for(i=1;i<=term1;i++)
  {
  printf("\n Enter the co and po of %d term :",i);
  scanf("%d%d",&co,&po);
  firsth=insertr(firsth,co,po);
  }
  printf("\n The fisrt polynomail is\n");
  display(firsth);
  printf("\n Read second polynomial\n");
  printf("\n enter the no of terms of second poly :");
  scanf("%d",&term2);
  for(i=1;i<=term2;i++)
  {
  printf("\n Enter the co and po of %d term :",i);
  scanf("%d%d",&co,&po);
  secondh=insertr(secondh,co,po);
  }
  printf("\n The second polynomail is\n");
  display(secondh);
  result=multiplypoly(firsth,secondh);
  printf("\n The Result after multiplication is \n");
  display(result);
  return 0;
}
```

10. Design a doubly linked list to represent sparse matrix. Each node in the list can have the row and column index of the matrix element and the value of the element

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

struct node
{
  int row,col,info;
  struct node *llink;
  struct node *rlink;
};
typedef struct node *NODE;

NODE insert(NODE first,int item,int row,int col)
{
  NODE temp,cur;
  temp=(NODE)malloc(sizeof(struct node));
  temp->row=row;
  temp->col=col;
  temp->info=item;
  temp->llink=NULL;
  temp->rlink=NULL;
  if (first == NULL)
     return temp;
  cur=first;
  while(cur->rlink != NULL)
     cur = cur->rlink;
  cur->rlink=temp;
  temp->llink=cur;
  return first;
}


void displaylist(NODE first)
{
  NODE temp;
  if(first == NULL)
    printf("\n list is empty");
  else
  {
    temp=first;
    printf("\n row   col   value\n");
    while(temp!=NULL)
     {
       printf("%d\t%d\t%d\n",temp->row,temp->col,temp->info);
       temp=temp->rlink;
     }
   }
}
```

```
void displaymatrix(NODE first,int row,int col)
{
  int i,j;
  NODE temp;
  temp=first;
  printf("\n the sparse matrix is\n");
  while(temp!=NULL)
  {
   for(i=1;i<=row;i++)
   {
    for(j=1;j<=col;j++)
      {
  if((temp->row == i) && (temp->col == j))
    {
     printf("%d\t",temp->info);
     temp=temp->rlink;
    }
  else
   printf("0\t");
      }
     printf("\n");
   }
 }
}

int main()
{
  NODE first=NULL;
  int row=0,col=0,i,j,item,ch;
  printf("\n enter the order of the martix : ");
  scanf("%d%d",&row,&col);
  for(;;)
  {
    printf("\n 1. Read Matrix");
    printf("\n 2. Display list");
    printf("\m 3. Dispaly matrix");
    printf("\n 4. exit");
    printf("\n Read ur choice :");
    scanf("%d",&ch);
    switch(ch)
    {
       case 1: for(i=1;i<=row;i++)
         {
        for(j=1;j<=col;j++)
         {
           scanf("%d",&item);
           if(item!=0)
             first=insert(first,item,i,j);
         }
         }
         break;
       case 2: displaylist(first);
         break;
```

```
        case 3:displaymatrix(first,row,col);
         break;
        default:exit(0);
      }
   }
   return 0;
}
```

11. Write a C program to implement Hashing using Open Addressing.

```
#include<stdio.h>
#include<conio.h>
#define HASH(x) x%10
#define MAX 10
int values[MAX];
int LinearProb(int number)
{
  int hashvalue,n=0;
  hashvalue=HASH(number);
  if(values[hashvalue]==0)
       return hashvalue;
  else
  {
     do
      {
      hashvalue++;
      if (hashvalue == MAX) break;
      if(values[hashvalue] == 0)
         return hashvalue;
      else
         continue;
      } while(hashvalue!=MAX-1);
      if(hashvalue == MAX || hashvalue == MAX-1)
      {
       hashvalue=-1;
       do
        {
          n++;
          hashvalue++;
        } while(values[hashvalue] != 0);
       return hashvalue;
      }
   }
}
int main()
{
  int n=0,number,i;
  while(n!=MAX)
  {
    number=0;
    printf("\n Read Number :");
    scanf("%d",&number);
    values[LinearProb(number)]=number;
    n++;
  }
  printf("\n The hashed table is\n");
  for(i=0;i<MAX;i++)
     printf("%d->%d\n",i,values[i]);
  return 0;
}
```

12. Write a C program to create Binary Tree and provide insertion and deletion operations and to traverse the tree using In-order, Preorder and Post order (recursively)

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

struct node
{
  int info;
  struct node *llink;
  struct node *rlink;
};
typedef struct node *NODE;

NODE createBST(NODE root,int item)
{
 NODE temp,prev,cur;
 temp=(NODE)malloc(sizeof(struct node));
 temp->info=item;
 temp->llink=NULL;
 temp->rlink=NULL;
 if (root == NULL)
    return temp;
 prev=NULL;
 cur=root;
 while(cur != NULL)
 {
   prev=cur;
   if(item < cur->info)
     cur = cur->llink;
   else
     cur = cur->rlink;
 }
 if(item < prev->info)
  prev->llink=temp;
 else
  prev->rlink=temp;
  return root;
}

NODE minValueNode(NODE root)
{
    NODE cur = root;

    /* loop down to find the leftmost leaf */
    while (cur->llink != NULL)
        cur = cur->llink;
     return cur;
}
```

```
NODE delete(NODE root, int key)
{
   NODE temp;
    // base case
    if (root == NULL) return root;

    // If the key to be deleted is smaller than the root's key,
    // then it lies in left subtree
    if (key < root->info)
      root->llink = delete(root->llink, key);

    // If the key to be deleted is greater than the root's key,
    // then it lies in right subtree
    else if (key > root->info)
      root->rlink = delete(root->rlink, key);

    // if key is same as root's key, then This is the node
    // to be deleted
    else
    {
      // node with only one child or no child
      if (root->llink == NULL)
      {
          temp = root->rlink;
          printf("\n Node is deleted");
          free(root);
          return temp;
      }
      else if (root->rlink == NULL)
      {
          temp = root->llink;
          printf("\n Node is deleted");
          free(root);
          return temp;
      }

      // node with two children: Get the inorder successor (smallest
      // in the right subtree)
      temp = minValueNode(root->rlink);

      // Copy the inorder successor's content to this node
      root->info = temp->info;

      // Delete the inorder successor
      root->rlink = delete(root->rlink, temp->info);
    }
    return root;
}

void preorder(NODE root)
{
 if(root != NULL)
 {
   printf("%d ",root->info);
```

```
   preorder(root->llink);
   preorder(root->rlink);
 }
}

void inorder(NODE root)
{
 if(root != NULL)
 {
   inorder(root->llink);
   printf("%d ",root->info);
   inorder(root->rlink);
 }
}

void postorder(NODE root)
{
 if(root != NULL)
 {
   postorder(root->llink);
   postorder(root->rlink);
   printf("%d ",root->info);
 }
}

int main()
{
  int ch,item;
  NODE root=NULL;
  for(;;)
  {
    printf("\n 1. Insert");
    printf("\n 2. Preorder");
    printf("\n 3. Inorder");
    printf("\n 4. Postorder");
    printf("\n 5. Delete");
    printf("\n Read choice :");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:printf("\n enter the element to be inserted :");
             scanf("%d",&item);
             root=createBST(root,item);
             break;
      case 2:printf("\n The Preorder traversal is\n");
             preorder(root);
             break;
      case 3:printf("\n The Inorder traversal is\n");
             inorder(root);
             break;
      case 4:printf("\n The Postorder traversal is\n");
             postorder(root);
             break;
      case 5:printf("\n enter the element to be deleted :");
```

```
            scanf("%d",&item);
            root=delete(root,item);
            break;
       default:exit(0);
     }
    }
  return 0;
}
```

13. **Given a String representing a parentheses-free infix arithmetic expression, implement a program to place it in a tree in the infix form. Assume that a variable name is a single letter. Traverse the tree to produce an equivalent postfix and prefix expression string.**

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

struct node
 {
   char info;
   struct node *llink;
   struct node *rlink;
 };
typedef struct node *NODE;

NODE createnode(char item)
  {
      NODE temp;
      temp =(NODE)malloc(sizeof(struct node));
      temp->info = item;
      temp->llink = NULL;
      temp->rlink = NULL;
      return temp;
  }

int preced(char x)
  {
      switch(x)
       {
            case '^': return 3;
            case '/':
            case '*': return 2;
            case '+':
            case '-': return 1;
       }
  }

NODE create_exp_tree(char infix[15])
  {
    int i;
    char symbol;
    NODE temp,t1,t2;
    NODE operst[10],treest[10];
    int top1=-1,top2=-1;
    for(i=0;infix[i]!='\0';i++)
    {
      symbol=infix[i];
      if(isalnum(symbol))
       {
```

```
        temp=createnode(symbol);
        treest[++top2]=temp;
         }
        else
         {
         temp=createnode(symbol);
         if(top1 == -1 || preced(operst[top1]->info) < preced(symbol))
           operst[++top1]=temp;
         else
           {
             t1=treest[top2--];
             t2=treest[top2--];
             temp->rlink=t1;
             temp->llink=t2;
             treest[++top2]=temp;
           }
         }
       }
       while(top1!=-1)
       {
             t1=treest[top2--];
             t2=treest[top2--];
             temp=operst[top1--];
             temp->rlink=t1;
             temp->llink=t2;
             treest[++top2]=temp;
       }
       return treest[top2];
}

void preorder(NODE root)
  {
       if(root != NULL)
        {
             printf("%c ",root->info);
             preorder(root->llink);
             preorder(root->rlink);
        }
  }

void inorder(NODE root)
  {
       if(root != NULL)
        {
             inorder(root->llink);
             printf("%c ", root->info);
             inorder(root->rlink);
        }
  }

void postorder(NODE root)
  {
       if(root != NULL)
        {
```

```
            postorder(root->llink);
            postorder(root->rlink);
            printf("%c ", root->info);
        }
    }

int main()
    {
        NODE root = NULL;
        char infix[15];
        printf("\nEnter infix expression\n");
        scanf("%s",infix);
        root = create_exp_tree(infix);
        printf("\nInfix\n");
        inorder(root);
        printf("\npostfix\n");
        postorder(root);
        printf("\nprefix\n");
        preorder(root);
        return 0;
    }
```

14. Implement a simple Dictionary using Trees and Hashing.

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

struct node
{
  char word[10];
  char meaning[25];
  struct node *llink;
  struct node *rlink;
};
typedef struct node *NODE;

NODE createBST(NODE root,char *word, char *meaning)
{
 NODE temp,prev,cur;
 temp=(NODE)malloc(sizeof(struct node));
 strcpy(temp->word,word);
 strcpy(temp->meaning,meaning);
 temp->llink=NULL;
 temp->rlink=NULL;
 if (root == NULL)
    return temp;
 prev=NULL;
 cur=root;
 while(cur != NULL)
 {
   prev=cur;
   if(strcmp(word,cur->word)< 0)
     cur = cur->llink;
   else
     cur = cur->rlink;
 }
 if(strcmp(word,prev->word) < 0)
  prev->llink=temp;
 else
  prev->rlink=temp;
  return root;
}

NODE minValueNode(NODE root)
{
    NODE cur = root;

    /* loop down to find the leftmost leaf */
    while (cur->llink != NULL)
      cur = cur->llink;
     return cur;
}
```

```
NODE delete(NODE root, char *word)
{
   NODE temp;
    // base case
    if (root == NULL) return root;

    // If the key to be deleted is smaller than the root's key,
    // then it lies in left subtree
    if (strcmp(word,root->word) < 0)
      root->llink = delete(root->llink, word);

    // If the key to be deleted is greater than the root's key,
    // then it lies in right subtree
    else if (strcmp(word,root->word) > 0)
      root->rlink = delete(root->rlink, word);

    // if key is same as root's key, then This is the node
    // to be deleted
    else
    {
      // node with only one child or no child
      if (root->llink == NULL)
      {
          temp = root->rlink;
          printf("\n Word is deleted");
          free(root);
          return temp;
      }
      else if (root->rlink == NULL)
      {
          temp = root->llink;
          printf("\n word is deleted");
          free(root);
          return temp;
      }

      // node with two children: Get the inorder successor (smallest
      // in the right subtree)
      temp = minValueNode(root->rlink);

      // Copy the inorder successor's content to this node
      strcpy(root->word,temp->word);

      // Delete the inorder successor
      root->rlink = delete(root->rlink, temp->word);
    }
    return root;
}

void preorder(NODE root)
{
 if(root != NULL)
 {
   printf("%s->%s\n",root->word,root->meaning);
```

```
   preorder(root->llink);
   preorder(root->rlink);
 }
}

void inorder(NODE root)
{
 if(root != NULL)
 {
   inorder(root->llink);
   printf("%s->%s\n",root->word,root->meaning);
   inorder(root->rlink);
 }
}

void postorder(NODE root)
{
 if(root != NULL)
 {
   postorder(root->llink);
   postorder(root->rlink);
   printf("%s->%s\n",root->word,root->meaning);
 }
}

int main()
{
  int ch;
  char word[10],meaning[25];
  NODE root=NULL;
  for(;;)
  {
    printf("\n 1. Insert");
    printf("\n 2. Preorder");
    printf("\n 3. Inorder");
    printf("\n 4. Postorder");
    printf("\n 5. Delete");
    printf("\n 6.Quit");
    printf("\n Read choice :");
    scanf("%d",&ch);
    switch(ch)
    {
      case 1:printf("\n enter the word to be inserted :");
           scanf("%s",word);
           getchar();
           printf("\n enter the meaning of the word\n");
           gets(meaning);
           root=createBST(root,word,meaning);
           break;
      case 2:printf("\n The Preorder traversal is\n");
           preorder(root);
           break;
      case 3:printf("\n The Inorder traversal is\n");
           inorder(root);
```

```
            break;
       case 4:printf("\n The Postorder traversal is\n");
            postorder(root);
            break;
       case 5:printf("\n enter the element to be deleted :");
            scanf("%s",word);
            root=delete(root,word);
            break;
       default:exit(0);
     }
    }
  return 0;
}
```