Advance Problem Solving
C Programming Problems

Submitted By-
Name – Kalpish Singhal
Roll No – 201505513
MTECH PG1 - CSE

Program

Write a Program to find 1/x using recurrence relation.

```
#include <stdio.h>
/*
1/x - USING RECURRENCE RELATION

Recurrence Relation:-
a[0]=1
c[0]=1-x
a[n]=a[n-1]*(1+c[n-1])
c[n]=c[n-1]*c[n-1]
*/
int main()
{
        float e=0.00001,d,x,a=1,c=0;
        scanf("%f",&x);
        c=1-x;
        d=c*-1;
        while(c>e || d>e)
        {
                d=1+c;
                a=a*d;
                c=c*c;
                d=c*-1;
        }
printf("%f",a);
return 0;
}
```

Input .36
Output 2.777776

Program

Write a Program to find the gcd of the given numbers

```
/*Gcd of 2 numbers*/
#include<stdio.h>
int main()
{
	int n1,n2;
	scanf("%d %d",&n1,&n2);
	if (n1<0 || n2 <0)
		{
		printf("invalid input");
		return 0;
		}
	if(n1==0 && n2>0)
	{
	printf("gcd is %d",n2);
	return 0;
	}
	if(n2==0 && n1>0)
	{
	printf("gcd is %d",n1);
	return 0;
	}
	else{

		while (n1!=n2)
		{
			if(n1>n2)
			{
			n1=n1-n2;
			}
			else
			n2=n2-n1;
		}
		printf("gcd is %d",n1);
	}
return 0;
}
```

Output

```
13 17
gcd is 1
```

Program

Write a Program to find the modulus of two numbers without using mod operator.

```
/*Mod of 2 numbers */
#include<stdio.h>
int main()
{
        int num1=0,num2=0,cal,mod;
        scanf("%d %d",&num1,&num2);
        if(num1< 0 || num2<= 0)
                {
                printf("invalid");
                return 0;
                }
        else
        {
        cal=num1/num2;
        mod=num1-num2*cal;
        printf("the value is %d",mod);
        return 0;
        }

}
```

Output

```
13
5
the value is 3
```

Program

Write a Program to perform division using shift operators.

```
/*Division using Shift without / */
#include<stdio.h>

//This function performs division opration usinf bit shift
int divide(int num1, int num2) {

   int temp = 1;
   int quotient = 0;

   while (num1 <= num2) {
      num1 <<= 1;
      temp <<= 1;
   }


      while (temp > 1)
                {
           num1 >>= 1;
                 temp >>= 1;

                                  if (num2 >= num1)
                                  {
                                          num2 -= num1;
                                  quotient += temp;
                                  }
              }
                                  return quotient;

    }
int main()
{
int num1,num2,answer,num_flag=0,num_flag2=0;
scanf("%d %d",&num1,&num2);
if(num2==0)
{
printf("devide by zero error");
return 0;
}
if(num1<0)
{
```

```
num_flag=1;
num1=-1*num1;
}
if(num2<0)
{
num_flag2=1;
num2=-1*num2;
}
answer=divide(num2,num1);
if(num_flag==1)
{
answer=-1*answer;
}
if(num_flag2==1)
{
answer=-1*answer;
}
printf("division is %d",answer);
return 0;
}
```

Output

```
10
5
division is 2
```

Program

Write a Program that prints itself.

```
/* SELF REPRODUCING CODE*/
#include <stdio.h>
main() { char *s="main() { char *s=%c%s%c; printf(s,9,s,10); }"; printf(s,9,s,10); }
```

Output

```
main() { char *s=  main() { char *s=%c%s%c; printf(s,9,s,10); }
; printf(s,9,s,10); }
```

Program

Write a Program to find square root of x.

```
/*
SQUARE ROOT - USING RECURRENCE RELATION
*/

#include<stdio.h>

int main()
{
	float x,a=1,c,d,e=0.000000001;
	scanf("%f",&x);
	if(x<0)
	a=-1.000000;
	else
	{
	c=1-x;
	a=x;
	d=c*-1;
	while(c>e || d>e)
		{
			a=a*(1+c*0.5);
			c=c*c*(0.75+0.25*c);
			d=c*-1;
		}
	}
	printf("root is %6f",a);
return 0;
}
```

Output

```
2
root is 1.414214
```

Program

Write a Program to find the largest element in the given array.

```
/*
LARGEST ELEMENT OF AN ARRAY OF MAX 50 Ellements
*/

#include<stdio.h>

int main()
{

        int arr[50],arr_size,i,max=0;

        printf("Enter the number of elements : ");
        scanf("%d",&arr_size);
        printf("\nEnter the elements of the array");
        for(i=0;i<arr_size;i++)
        {scanf("%d",&arr[i]);
        }
        for(i=0;i<arr_size;i++)
        {       if(max<arr[i])
                max=arr[i];
        }
        printf("The Largest Elemnt is : %d\n",max);
        return 0;
}
```

Output

```
Enter the number of elements : 4

Enter the elements of the array-1 10 20 13
The Largest Elemnt is : 20
```

Program

Write a Program to find the smallest element in the given array.

```
/*
SMALLEST ELEMENT OF AN ARRAY an max array of 50
*/

#include<stdio.h>

int main()
{
        int arr[50],arr_size,i,min=0;
        printf("Enter size of arrray under 50 : ");
        scanf("%d",&arr_size);
        printf("Enter the elements\n");
        i=0;
        for(;i<arr_size;++i)
                scanf("%d",&arr[i]);
        min=a[0];
        for(i=1;i<arr_size;++i)
                if(arr[i]<min)
                        min=arr[i];
        printf("The Smallest Elemnt is : %d\n",min);
        return 0;
}
```

Output

```
Enter size of arrray under 50 : 5
Enter the elements
-1 10 20 13 3
The Smallest Elemnt is : -1
```

Program

Write a Program to find average of all elements of the array.

```
/*
AVERAGE OF AN ARRAY of atmost 50 elements*/

#include <stdio.h>

int main()
{
        int a[50],n;
        float avg=0;
        int i;
        printf("number of elements: ");
        scanf("%d",&n);
        printf("\nEnter the elements of the array");
        i=0;
        for(;i<n;i++)
                {
                        scanf("%d",&a[i]);
                        avg=avg+a[i];
                }
        avg=avg/n;
        printf("average is : %f\n",avg);
        return 0;
}
```

Output

```
number of elements: 5

Enter the elements of the array10 20 30 40 50
average is : 30.00000
```

Program

Write a Program to find the largest prime number.

```
/*LARGEST PRIME NUMBER*/

#include<stdio.h>
#include<math.h>
int main()
{
	int flag=0;
	unsigned long long int i=0,j=0,root;
	i=i-1;
	for(;i>1;i--)//loop to decrement i by seting unsigned i to 0-1(set all 0's to 1)
	{
		flag=0;
		root=sqrt(i);
		for(j=1;j<root;j++)
		{	if(i%j!=0)
			{
				continue;
			}
			else
			{	flag=1;
				break;
			}
		}
		if(flag==0)
		{
			printf("%llu\n",i);
			break;
		}
	}

return 0;
}
```

Output

```
18446744073709551557
```

Program

Write a Program to sort the array elements using quick sort.

```
/*
SORTING
QuickSort is a Divide and Conquer algorithm
This programe implement Quick Sort
*/
#include<stdio.h>
void quicksort(int number[25],int first,int last){
   int i, j, pivot, temp;
   if(first<last){
      pivot=first;
      i=first;
      j=last;
      while(i<j){
         while(number[i]<=number[pivot]&&i<last)
            i++;
         while(number[j]>number[pivot])
            j--;
         if(i<j){
            temp=number[i];
            number[i]=number[j];
            number[j]=temp;
         }
      }

      temp=number[pivot];
      number[pivot]=number[j];
      number[j]=temp;
      quicksort(number,first,j-1);
      quicksort(number,j+1,last);
   }
}
int main(){
   int i, count, number[25];
   printf("How many elements are u going to enter?: ");
   scanf("%d",&count);
   printf("Enter %d elements: ", count);
   for(i=0;i<count;i++)
      scanf("%d",&number[i]);
   quicksort(number,0,count-1);
```

```
   printf("Order of Sorted elements: ");
   for(i=0;i<count;i++)
      printf(" %d",number[i]);

   return 0;
}
```

Output

```
How many elements are u going to enter?: 5
Enter 5 elements: -1 9 7 2 8
Order of Sorted elements:  -1 2 7 8 9
```

Program

Write a Program to find if the given number is perfect number or not.

```
/*
                        PERFECT NUMBERS
 Given number is a perfect number or not
*/
#include<stdio.h>
int main()
{
        int number,i=2,sum=1;
        printf("Enter number : ");
        scanf("%d",&number);
        for(;i<number;++i)
        {
                if(i%number==0)
                        sum=sum+i;
        }

        if(sum==number)
                printf("Perfect Number\n");
        else
                printf("Not a Perfect Number\n");
        return 0;
}
```

Output

```
Enter number : 6
Perfect Number
Enter number : 13
Not a Perfect Number
```

Program

Write a Program to implement singly linked list.

```
/*PERFORM VARIOUS OPRATIONS ON LINKED LIST LIKE
1)insert
2) deletion
3)Printing
4)reverseLL
5)search
 */
#include <stdio.h>
#include <stdlib.h>
struct node{
        int data;
        struct node *next;
};
struct node *head = NULL;
struct node *curr = NULL;
struct node * insert( struct node * head, int elem );
struct node* create_list(int elem);
void printLL( struct node * head );
struct node * reverseLL(struct node * head);
struct node * deleteNode(struct node * head, int elem);
struct node* search_in_list(int val, struct node **prev);
struct node * insertAtPosition(struct node * head, int elem, int loc );
struct node* create_list(int elem)
{
        struct node *ptr=(struct node *)malloc(sizeof(struct node));
        if (ptr==NULL)
        {  printf("\n Node creation failed \n");
                return NULL;
        }
        ptr->data = elem;
   ptr->next = NULL;
   head = curr = ptr;
   return ptr;
}
struct node* insert( struct node * head, int elem )
{       if(NULL == head)
   {       return (create_list(elem));
   }
```

```
        struct node *ptr=(struct node *)malloc(sizeof(struct node));
        if(NULL == ptr)
    {
       printf("\n Node creation failed \n");
       return NULL;
    }
    ptr->data = elem;
    ptr->next = NULL;
    curr->next=ptr;
    curr=ptr;
    return ptr;
}
int main()
{
        int ele,pos,input=1;
        struct node *result;
        while (input!=0)
        {
        printf("\n1. insert/create_list\n2. Print linked list\n3. Delete\n4. insert At
Position\n5. reversal\n0. exit\n");
        scanf("%d",&input);
        switch(input)
                {       case 1:
                        printf("enter inserting elmnt \n");
                        scanf("%d",&ele);
                        result=insert(head,ele);
                        break;
                        case 2:
                        printLL(head);
                        break;
                        case 3:
                        scanf("%d",&ele);
                        result=deleteNode(head,ele);
                        head=result;
                        break;
                        case 4:
                        scanf("%d",&ele);
                        printf("\n enter Position\n");
                        scanf("%d",&pos);
                        result=insertAtPosition(head,ele,pos);
                        head=result;
                        break;
                        case 5:
```

```
                        result=reverseLL(head);
                        head=result;
                                break;
                        default:
                                return 0;}
        }
        return 0;
}
void printLL( struct node * head )
{
        struct node *ptr = head;
                if(head==NULL)
                        printf("Empty linked list\n");
        while(ptr != NULL)
    {
        if (ptr->next!=NULL)
                printf("%d->",ptr->data);
        else
        {
                printf("%d",ptr->data);
        }
       ptr = ptr->next;
    }
    printf("\n");
}
struct node * deleteNode(struct node * head, int elem)
{struct node *prev = NULL;
    struct node *del = NULL;
         del = search_in_list(elem,&prev);
         //printf("deleteNode %d\n",del->data );
         if(del == NULL)
    {
         printf("Element not found\n");
       return head;
    }    else
    {
       if(prev != NULL)
           prev->next = del->next;
       if(del == curr)
            curr = prev;
       if(del == head)
       {  head = head->next;
       }
```

```
    }
    free(del);
    del = NULL;
    return head;
}
struct node* search_in_list(int val, struct node **prev)
{
   struct node *ptr = head;
    struct node *tmp = NULL;
    while(ptr != NULL)
    {
          if(ptr->data == val)
          {       *prev =tmp;
                  return ptr;
          }
          else
          {
                  tmp = ptr;
                  ptr=ptr->next;
          }
    }
    return NULL;
}
struct node * insertAtPosition(struct node * head, int elem, int loc )
{
          struct node *ptr = head;
          struct node *tmp = NULL;
          int set_flag_for_position=0;
          int loc_temp=loc;
          while(ptr != NULL)
    {
         if (--loc_temp)
         {       //printf("going in\n");
                 tmp = ptr;
                 ptr=ptr->next;
         }
         else
         {       set_flag_for_position=1;
                 break;
         }
    }
    if (set_flag_for_position==1)
    {             struct node *newnode=(struct node *)malloc(sizeof(struct node));
```

```
            if(NULL == newnode)
            {
                printf("\n Node creation failed \n");
                return NULL;
                  }
        else    {newnode->data=elem;
                            newnode->next=NULL;
                  }
        if(loc==1)
        {       newnode->next=head;
                head=newnode;
                return head;
        }
        else if(ptr->next==NULL)
        {
                ptr->next=newnode;
                curr=newnode;
                return head;
        }
        else
        {       newnode->next=ptr;
                tmp->next=newnode;
                return head;
        }
    }
    else
    {
        printf("Linked list short\n");
    }
    return head;
}
struct node * reverseLL(struct node * head)
{
        struct node* prev   = NULL;
    struct node* current = head;
    struct node* next;
    while (current != NULL)
    {
        next  = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }
```

```
    head = prev;
    return head;
}
```

Output

```
1. insert/create_list
2. Print linked list
3. Delete
4. insert At Position
5. reversal
0. exit
2
10->10->2->1->10->30

1. insert/create_list
2. Print linked list
3. Delete
4. insert At Position
5. reversal
0. exit
5
1. insert/create_list
2. Print linked list
3. Delete
4. insert At Position
5. reversal
0. exit
2
30->10->1->2->10->10
```

Program

Write a Program to detect/remove cycle in the linked list.

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

/* Link list node */
struct node
{
    int data;
    struct node* next;
};
/* Function to remove loop. Used by detectAndRemoveLoop()
*/
void removeLoop(struct node *, struct node *);

/* This function detects and removes loop in the list
  If loop was there in the list then it returns 1,
  otherwise returns 0 */
int detectAndRemoveLoop(struct node *list)
{
    struct node  *slow_p = list, *fast_p = list;
    while (slow_p && fast_p && fast_p->next)
    {
        slow_p = slow_p->next;
        fast_p  = fast_p->next->next;
        /* If slow_p and fast_p meet at some point then
there
           is a loop */
        if (slow_p == fast_p)
        {
            removeLoop(slow_p, list);
            /* Return 1 to indicate that loop is found */
            return 1;
        }
    }
    /* Return 0 to indeciate that ther is no loop*/
    return 0;
}

/* Function to remove loop.
 loop_node --> Pointer to one of the loop nodes
 head -->  Pointer to the start node of the linked list
*/
void removeLoop(struct node *loop_node, struct node
```

```
*head)
{
   struct node *ptr1;
   struct node *ptr2;

   /* Set a pointer to the beging of the Linked List and
      move it one by one to find the first node which is
      part of the Linked List */
   ptr1 = head;
   while(1)
   {
     /* Now start a pointer from loop_node and check if
it ever
        reaches ptr2 */
     ptr2 = loop_node;
     while(ptr2->next != loop_node && ptr2->next != ptr1)
     {
         ptr2 = ptr2->next;
     }
     /* If ptr2 reahced ptr1 then there is a loop. So
break the
        loop */
     if(ptr2->next == ptr1)
        break;
     /* If ptr2 did't reach ptr1 then try the next node
after ptr1 */
     else
       ptr1 = ptr1->next;
   }
   /* After the end of loop ptr2 is the last node of the
loop. So
     make next of ptr2 as NULL */
   ptr2->next = NULL;
}

/* UTILITY FUNCTIONS */
/* Given a reference (pointer to pointer) to the head
  of a list and an int, pushes a new node on the front
  of the list. */
void push(struct node** head_ref, int new_data)
{
    /* allocate node */
    struct node* new_node =
        (struct node*) malloc(sizeof(struct node));
    /* put in the data  */
```

```
    new_node->data  = new_data;
    /* link the old list off the new node */
    new_node->next = (*head_ref);

    /* move the head to point to the new node */
    (*head_ref)    = new_node;
}
/* Function to print linked list */
void printList(struct node *node)
{
    while(node != NULL)
    {
        printf("%d  ", node->data);
        node = node->next;
    }
}
/* Drier program to test above function*/
int main()
{
    /* Start with the empty list */
    struct node* head = NULL;

    push(&head, 10);
    push(&head, 4);
    push(&head, 15);
    push(&head, 20);
    push(&head, 50);

    /* Create a loop for testing */
    head->next->next->next->next->next = head->next-
>next;

    detectAndRemoveLoop(head);

    printf("Linked List after removing loop \n");
    printList(head);

    getchar();
    return 0;
}
```

Output

Linked List after removing loop
50 20 15 4 10

Program

Write a Program to represents polynomials using linked list

```
/*  Represent polynomials using a linked list.

Given a polynomial of the form a+ax+ax^2+ax^3+...+ax^n=
0,
the coefficient representation of
the polynomial is given by (a0,a1,a2...an).
Represent this as a linked list*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct node
{
    int cof;
    int exp;
    struct node *link;
};

struct node *create(struct node *q);
struct node *insert(struct node *ptr, struct node *p);
void display(char const *tag, struct node *ptr);
void err_exit(char const *tag);

struct node *create(struct node *q)
{
    int i, n;
    printf("enter the number of nodes: ");
    if (scanf("%d", &n) != 1)
        err_exit("Read error (number of nodes)");
    for (i = 0; i < n; i++)
    {
        struct node *ptr = (struct node
*)malloc(sizeof(struct node));
        if (ptr == 0)
            err_exit("Out of memory (1)");
        printf("enter the coefficient and exponent
respectively: ");
        if (scanf("%d%d", &ptr->cof, &ptr->exp) != 2)
            err_exit("Read error (coefficient and
exponent)");
        ptr->link = NULL;
```

```
        q = insert(ptr, q);
        display("after input", q);
    }
    return q;
}

struct node *insert(struct node *ptr, struct node *p)
{
    struct node *temp, *b;
    if (p == NULL)
        p = ptr;
    else
    {
        display("insert: p   = ", p);
        display("insert: ptr = ", ptr);
        if (p->exp < ptr->exp)
        {
            ptr->link = p;
            p = ptr;
        }
        else
        {
            temp = p;
            while ((temp->link != NULL) && (temp->link-
>exp < ptr->exp))
                display("insert: tmp = ", temp),
                temp = temp->link;
            display("insert: post loop", temp);
            b = temp->link;
            temp->link = ptr;
            ptr->link = b;
        }
    }
    return p;
}

void display(char const *tag, struct node *ptr)
{
    struct node *temp;
    const char *pad = "";
    temp = ptr;
    printf("%s: ", tag);
    while (temp != NULL)
    {
        printf("%s%d x ^ %d", pad, temp->cof, temp->exp);
```

```
        temp = temp->link;
        pad = " + ";
    }
    putchar('\n');
}

int main(void)
{
    printf("enter the first polynomial:\n");
    struct node *p1 = NULL, *p2 = NULL;

    p1 = create(p1);

    printf("enter the second polynomial:\n");
    p2 = create(p2);

    display("p1", p1);
    display("p2", p2);

    return 0;
}

void err_exit(char const *tag)
{
    fprintf(stderr, "%s\n", tag);
    exit(1);
}
```

Output

```
enter the first polynomial:
enter the number of nodes: 3
enter the coefficient and exponent respectively: 2 3
after input: 2 x ^ 3
enter the coefficient and exponent respectively: 9 2
insert: p   = : 2 x ^ 3
insert: ptr = : 9 x ^ 2
insert: post loop: 2 x ^ 3
after input: 2 x ^ 3 + 9 x ^ 2
enter the coefficient and exponent respectively: 3 3
insert: p   = : 2 x ^ 3 + 9 x ^ 2
insert: ptr = : 3 x ^ 3
insert: tmp = : 2 x ^ 3 + 9 x ^ 2
insert: post loop: 9 x ^ 2
after input: 2 x ^ 3 + 9 x ^ 2 + 3 x ^ 3
enter the second polynomial:
enter the number of nodes: 3 1
```

```
enter the coefficient and exponent respectively: 1 3
after input: 1 x ^ 1
enter the coefficient and exponent respectively: 1 3
insert: p   = : 1 x ^ 1
insert: ptr = : 3 x ^ 1
insert: post loop: 1 x ^ 1
after input: 1 x ^ 1 + 3 x ^ 1
enter the coefficient and exponent respectively: 3 1
insert: p   = : 1 x ^ 1 + 3 x ^ 1
insert: ptr = : 3 x ^ 3
after input: 3 x ^ 3 + 1 x ^ 1 + 3 x ^ 1
p1: 2 x ^ 3 + 9 x ^ 2 + 3 x ^ 3
p2: 3 x ^ 3 + 1 x ^ 1 + 3 x ^ 1
```

Program

Write a Program to implement AVL trees.

```
/*This program implement an AVL tree.*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int flag=0;
struct avlnode
{
    int data;
    struct avlnode *left;
    struct avlnode *right;
    int height;
};
struct avlnode* new_node(int data)
{
    struct avlnode* treenode = (struct
avlnode*)malloc(sizeof(struct avlnode));
    if (!treenode)
    {
        flag=1;
    }
    treenode->data = data;
    treenode->left = NULL;
    treenode->right = NULL;
    treenode->height = 1;
    return(treenode);
}

int getheight(struct avlnode *tnode)
{
    int temp=0;
    if (tnode != NULL)
        temp=tnode->height;
    return temp;
}
int getBalance(struct avlnode *tnode)
{
    if (tnode != NULL)
    return getheight(tnode->left) - getheight(tnode-
>right);
    else
    return 0;
```

```
}
struct avlnode *rightRotation(struct avlnode *treenode)
{
    struct avlnode *temp = treenode->left;
    struct avlnode *temp2 ;
    temp2= temp->right;
    // rotation
    temp->right = treenode;
    treenode->left = temp2;
    if(getheight(treenode->left)>getheight(treenode-
>right))
    {treenode->height=getheight(treenode->left) + 1;
    }
    else
    {
        treenode->height= getheight(treenode->right) +
1;
    }
    if(getheight(temp->left)>getheight(temp->right))
    {
        temp->height=getheight(temp->left)+1;
    }
    else
        temp->height=getheight(temp->right)+1;
    return temp;
}

struct avlnode *leftRotation(struct avlnode *treenode)
{
    struct avlnode *temp = treenode->right;
    struct avlnode *temp2 = temp->left;
    //rotation
    temp->left = treenode;
    treenode->right = temp2;

    //heights
    if(getheight(treenode->left)>getheight(treenode-
>right))
    {treenode->height=getheight(treenode->left) + 1;
    }
    else
    {
        treenode->height= getheight(treenode->right) +
1;
    }
```

```
    if(getheight(temp->left)>getheight(temp->right))
    {
        temp->height=getheight(temp->left)+1;
    }
    else
        temp->height=getheight(temp->right)+1;
    return temp;
}

struct avlnode * insertElement(struct avlnode *avlnode1,
int data)
{
    if (avlnode1 == NULL)
        return(new_node(data));
    int temp;
    temp=avlnode1->data;
    if (temp > data)
    {
    avlnode1->left = insertElement(avlnode1->left, data);
    }
    else
    {
    avlnode1->right = insertElement(avlnode1->right,
data);
    }
    if(getheight(avlnode1->left)>getheight(avlnode1-
>right))
    avlnode1->height = getheight(avlnode1->left)+1;
    else
    avlnode1->height = getheight(avlnode1->right)+ 1;
    int balance;
    balance= getBalance(avlnode1);
    if (balance > 1)
    {//case 1 left imbalaced
        if( data < avlnode1->left->data)
        return rightRotation(avlnode1);
        //case 3 left-right imbalaced
        else if(data > avlnode1->left->data)
        {
        avlnode1->left = leftRotation(avlnode1->left);
        return rightRotation(avlnode1);
        }
    }

    if(balance < -1)
```

```
    {            //case 2 right imbalaced

        if (data > avlnode1->right->data)
        {
            return leftRotation(avlnode1);
        }
        //case 4 right-left imbalaced

        else if(data < avlnode1->right->data)
        {
            avlnode1->right=rightRotation(avlnode1-
>right);
            return leftRotation(avlnode1);
        }
    }
    return avlnode1;
}
struct avlnode * minVal(struct avlnode* node)
{
    struct avlnode* current = node;
    while (current->left != NULL)
        {
            current = current->left;
        }
    return current;
}


struct avlnode* deleteElement(struct avlnode* node, int
data)
{
    if (node != NULL)
        {
            int temp;
            temp= node->data;
        if ( data < temp )
        node->left = deleteElement(node->left,data);
        else if( data > temp )
        node->right = deleteElement(node->right, data);
        else
        {
            if( (node->left == NULL) || (node->right ==
NULL) )
        {
            struct avlnode *temp = node->left ? node-
```

```
>left : node->right;


            if(temp == NULL)
            {
                temp = node;
                node = NULL;
            }
            else
            *node = *temp;

            free(temp);
        }
        else
        {
            struct avlnode* temp = minVal(node->right);


            node->data = temp->data;


            node->right = deleteElement(node->right,
temp->data);
        }
        }
        if (node == NULL)
        return node;
        if (getheight(node->left)>getheight(node-
>right))
        {
            node->height=getheight(node->left)+1;
        }
        else
        {
        node->height = getheight(node->right) + 1;
        }
        int balance;
        balance=getBalance(node);
        if (balance > 1)
        {
            if (getBalance(node->left) >= 0)
            {
                return rightRotation(node);
            }
            if (getBalance(node->left) < 0)
```

```
            {
                node->left= leftRotation(node->left);
                return rightRotation(node);
            }
        }
        if (balance < -1)
        {
            if (getBalance(node->right) <= 0)
            {
                return leftRotation(node);
            }
            if (getBalance(node->right) > 0)
            {
                node->right=rightRotation(node->right);
                return leftRotation(node);
            }
        }
        return node;

        }
    else
    {
        flag=1;
        return node;
    }
}
/*void preOrder(struct avlnode *root)
{
    if(root != NULL)
    {
        printf("%d\n ", root->data);
        preOrder(root->left);
        preOrder(root->right);
    }
}*/
int main()
{

    char op;
    int t,val;
    scanf("%d",&t);
    struct avlnode *root = NULL;
    while(t)
    {    fflush( stdout );
        getchar();
```

```
        scanf("%c",&op);
        switch(op)
        {

            case 'I':
                flag=0;
                scanf("%d",&val);
                root = insertElement(root,val);
                if(flag==1)
                {
                    printf("False\n");
                }
                else
                    printf("True\n");
                break;
            case 'D':
                flag=0;
                scanf("%d",&val);
                root = deleteElement(root,val);
                if(flag==1)
                {
                    printf("False\n");
                }
                else
                {
                    printf("True\n");
                }
                break;
            case 'H':
                printf("True");
                break;
            default:
                printf("invalid input");
                break;
        }
    t--;
    }
    //preOrder(root);

return 0;
}
```

Output

```
6
I 5
```

```
I 2
I 3
D 2
I 2
D 5

true
true
true
true
true
false
```

Program

Write a Program to find inverse of a matrix.

```
/* INVERSE OF A GIVEN M X N Matrix
max size [20]X[20]
*/
#include <stdio.h>
int determinant(int f[20][20],int x)
{
  int pr,c[20],d=0,b[20][20],j,p,q,t;
  if(x==2)
  {
    d=0;
    d=(f[1][1]*f[2][2])-(f[1][2]*f[2][1]);
    return(d);
   }
  else
  {
    for(j=1;j<=x;j++)
    {
      int r=1,s=1;
      for(p=1;p<=x;p++)
        {
          for(q=1;q<=x;q++)
            {
              if(p!=1&&q!=j)
              {
                b[r][s]=f[p][q];
                s++;
                if(s>x-1)
                 {
                   r++;
                   s=1;
                  }
               }
             }
         }
     for(t=1,pr=1;t<=(1+j);t++)
     pr=(-1)*pr;
     c[j]=pr*determinant(b,x-1);
     }
     for(j=1,d=0;j<=x;j++)
     {
       d=d+(f[1][j]*c[j]);
      }
```

```
     return(d);
   }
}

/*calculate minor of matrix and return to determinant's
function*/
float minor(float matrix[][max],int k)
{
      int m=1 , p , r , c , row=1 , column;
      column=k;
      for(r=2;r<=n;r++)
      {
          p=1;
          for(c=1;c<=n;c++)
          {
              if(r!=row &amp;&amp; c!=column)
              {
                  new_mat[m][p]=matrix[r][c];
                  p++;
              }
          }
          if(r!=row)
             m++;
      }
      n--;
      return new_mat[m][p];

}

/*calculate Transpose*/
float Transpose(float matrix[][max])
{
    for(int i=1;i<=n;i++)
       for(j=1;j<=n;j++)
          m_Transpose[i][j]=matrix[j][i];
    return m_Transpose[n][n];
}


int main(){

  int a[20][20],i,j,m;
  float determinant1=0;
  printf("Enter m value for mXm matrix\n");
  scanf("%d",&m);
```

```
  printf("Enter the %d * %d elements of matrix:\n",m,m);
  for(i=0;i<m;i++)
      for(j=0;j<m;j++)
           scanf("%d",&a[i][j]);

  printf("\nThe matrix is\n");
  for(i=0;i<m;i++){
      printf("\n");
      for(j=0;j<m;j++)
           printf("%d\t",a[i][j]);
  }

   determinant1 = determinant(a,m);

   printf("\nInverse of matrix is: \n\n");
   for(i=0;i<3;i++){
      for(j=0;j<3;j++)
           printf("%.2f\t",((a[(i+1)%3][(j+1)%3] *
a[(i+2)%3][(j+2)%3]) - (a[(i+1)%3][(j+2)%3]*a[(i+2)%3]
[(j+1)%3]))/ determinant1);
       printf("\n");
   }

   return 0;
}
```

Output

Enter m value for mXm matrix 3
Enter the 3 * 3 elements of matrix:

```
5
2
1
5
8
3
9
2

The matrix is

3   5   2
1   5   8
3   9   2
Inverse of matrix is:
```

```
0.70     -0.25    0.07
-0.09    -0.00    0.14
-0.34    0.25     -0.11
```

Program

Write a Program to find the kth largest number in given array.

```
/*Find K largest number in a given array.
t = no of testcases
n = no of elements in array
k = required kth largest number */
#include <stdio.h>
void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
int findklargest (int ar[], int start, int end, int k)
{
    int pivot = start, left, right;
    left = start;
    right = end;
    while(left<=right)
        {
            while(left<=right && ar[left]>=ar[pivot])
                left++;
            while(left<=right && ar[right]<=ar[pivot])
                right--;
            if(left<right)
            {
                swap(&ar[left], &ar[right]);
            }

        }
    swap(&ar[pivot], &ar[right]);
    if(k==right+1)
            return ar[right];
    else if (k>right+1)
            return findklargest(ar, right+1, end, k);
    else return findklargest(ar, start, right-1, k);
}
int main()
{
    int n, k, res, i, t;
    scanf ("%d" ,&t);
    while(t>0)
```

```
        {
            res =0;
            scanf ("%d" ,&n);
            scanf ("%d" ,&k);
            if(k<1||k>n)
                {
                printf("Error\n");
                continue;
                }
            int ar[n];
            for(i=0; i<n; i++)
                {
                scanf("%d", &ar[i]);
                }
            res = findklargest(ar, 0, n-1, k);
            printf("%d\n", res);
            t--;
        }
    return 0;
}
```

Output

```
1
5 3
2
10
11
15
17
11
```

Program

Write a Program to find the reversal of the string.

```
/*Find the reverse of the given string */

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

int main()
{
    char *str, *newstr;
    str = (char*) malloc(sizeof(char)*256);   //
Allocating memory to the string
    printf("Enter the string\n");
    fgets(str, 256, stdin);
    int i, len, j;
    i=0;
    len =strlen(str);
    newstr = (char*) malloc(len);     // Allocating
memory to the new string
    newstr[len]='\0';
    j=len-1;

    while (str[i]!='\0')              // Reversing the
string
            {
                newstr[j]=str[i];
                j--;
                i++;
            }
    printf("Revesed String");
    printf("%s\n", newstr );
    return 0;

}
```

Output

```
Enter the string
adfadf
Revesed String
fdafda
```

Program

Write a Program to lower To upper case .

```
#include <stdio.h>
#include <string.h>
//convert lower case letters to upper
int main()
{
    char str[100];
    scanf("%s",str);
    int i;
    for(i=0;i<strlen(str);i++)
    {
        if(str[i]>='a' && str[i]<='z')
            str[i]=str[i]-'a'+'A';
    }
    printf("%s\n",str);
    return 0;
}
```

Output
hello
HELLO