**Odd or Even**

#include<stdio.h>

int main()

{

int arr[50],n,e\_c=0,o\_c=0,i;

printf("Enter the number of elements: ");

scanf("%d",&n);

for(i=1;i<=n;i++)

scanf("%d",&arr[i]);

for(i=0;i<n;i++)

{

if(arr[i]%2==0)

e\_c++;

else

o\_c++;

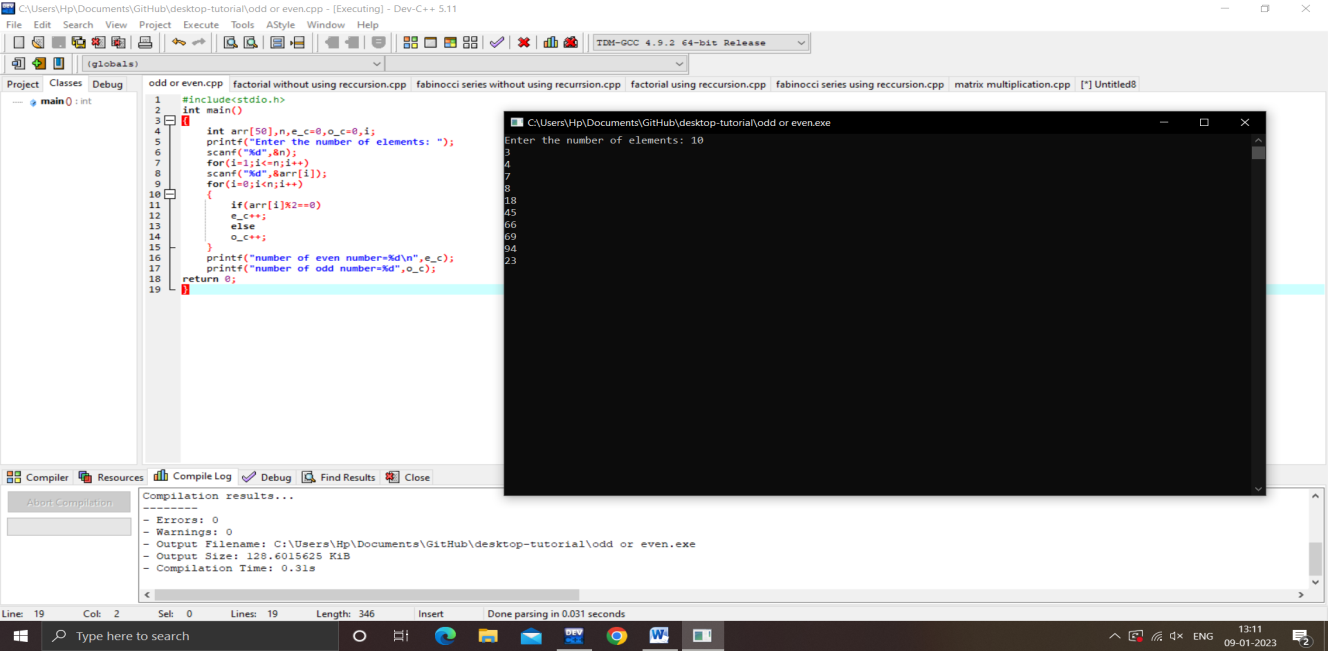
}

printf("number of even number=%d\n",e\_c);

printf("number of odd number=%d",o\_c);

return 0;

}



**Factorial without using recursion**

#include<stdio.h>

int main()

{

int n,i,fact=1;

printf("Enter any number: ");

scanf("%d",&n);

for(i=1;i<=n;i++)

{

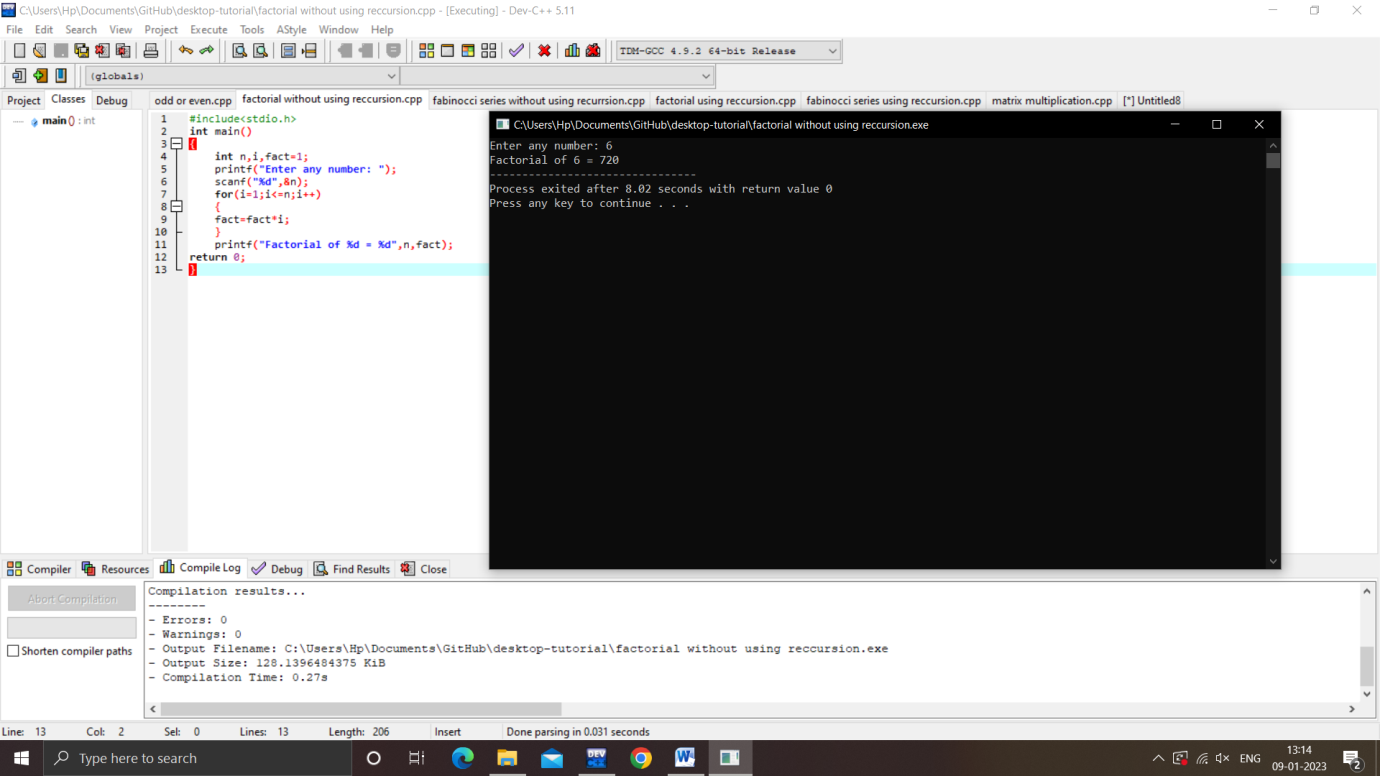
fact=fact\*i;

}

printf("Factorial of %d = %d",n,fact);

return 0;

}

****

**Fabinocci series without using recursion**

#include<stdio.h>

int main()

{

int a=0,b=1,c,i,n;

printf("Enter a number: ");

scanf("%d",&n);

for(i=2;i<n;i++)

{

c=a+b;

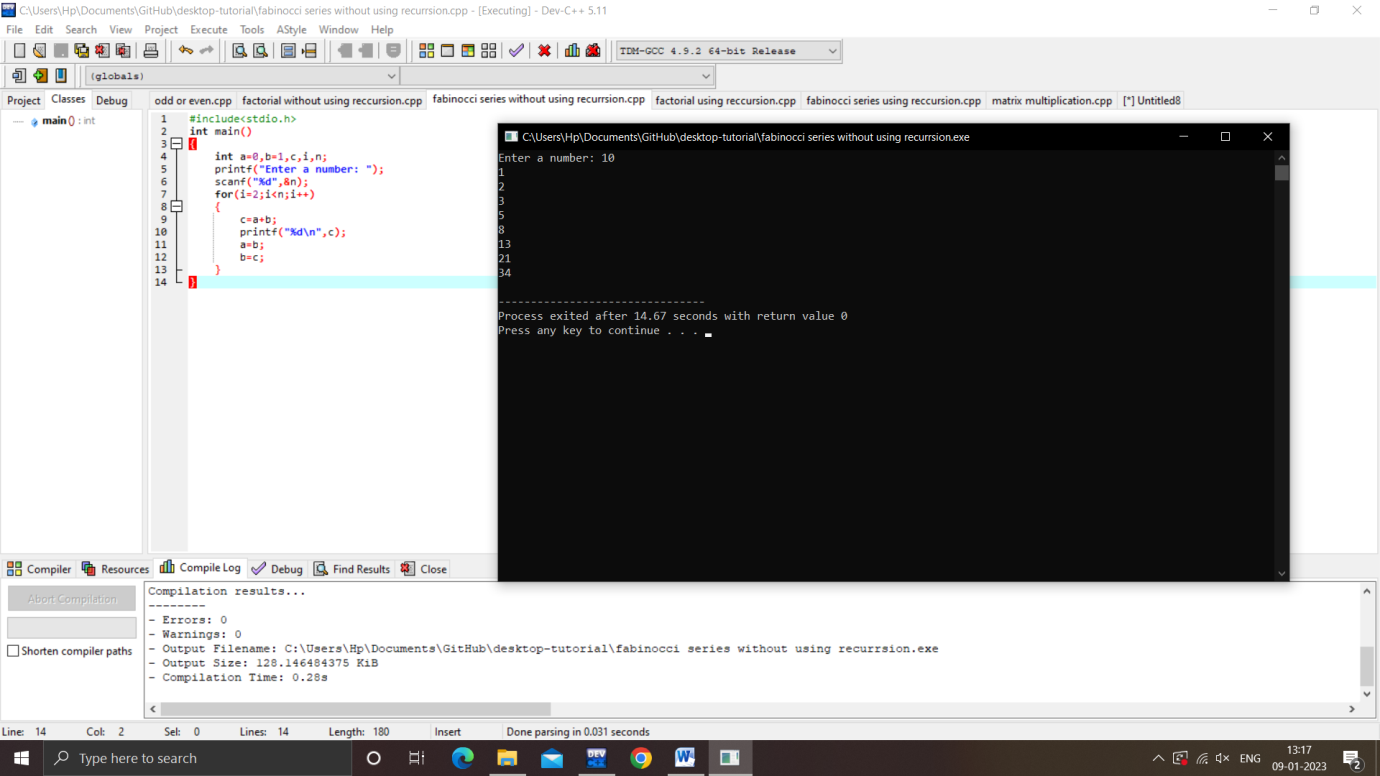
printf("%d\n",c);

a=b;

b=c;

}

}



**Factorial using recursion**

#include<stdio.h>

int fact(int n)

{

if(n==1)

return 1;

else

return(n\*fact(n-1));

}

int main()

{

int n,factorial;

printf("Enter any number: ");

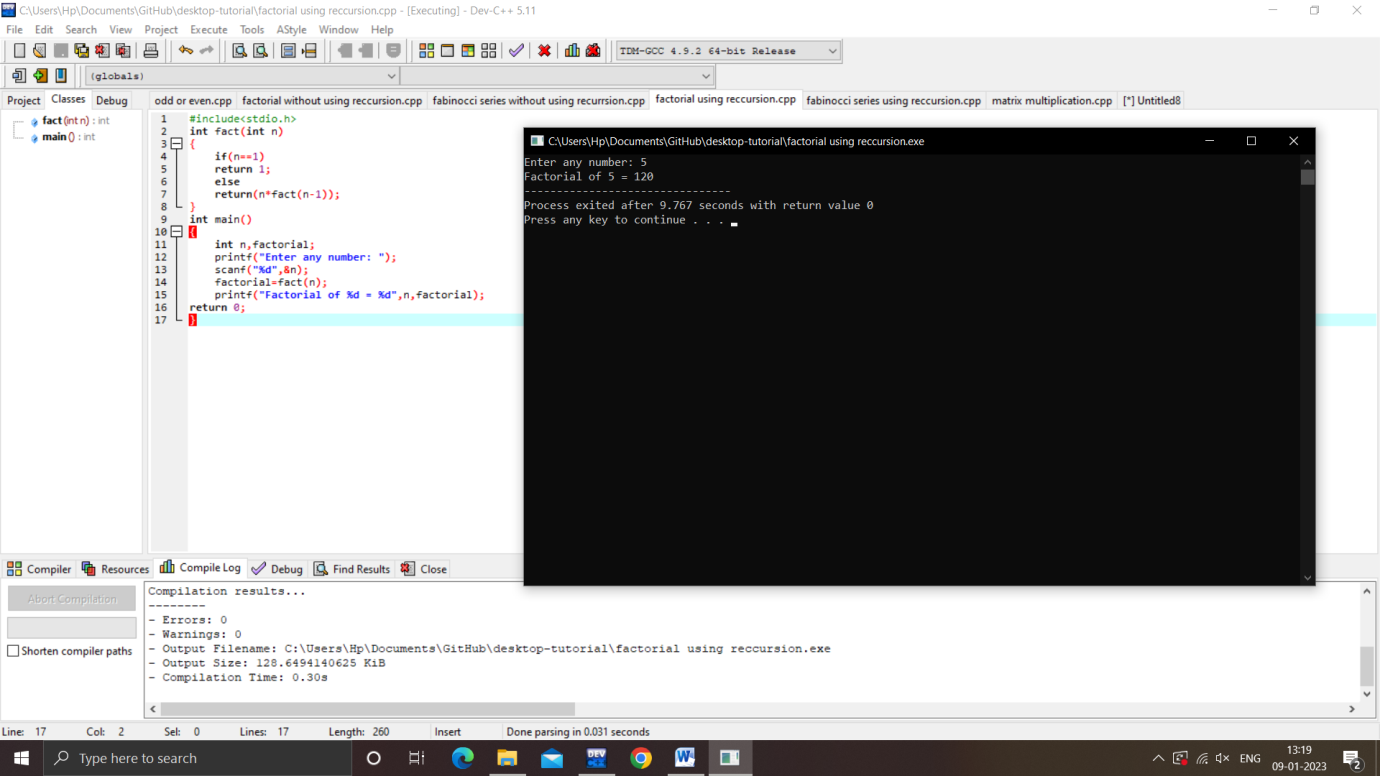
scanf("%d",&n);

factorial=fact(n);

printf("Factorial of %d = %d",n,factorial);

return 0;

}



**Fabinocci series using recursion**

#include<stdio.h>

int f(int n)

{

if(n==0)

return 0;

else if(n==1)

return 1;

else

return(f(n-1)+f(n-2));

}

int main()

{

int a=0,b=1,c,i,n;

printf("Enter a number: ");

scanf("%d",&n);

for(i=2;i<n;i++)

{

c=a+b;

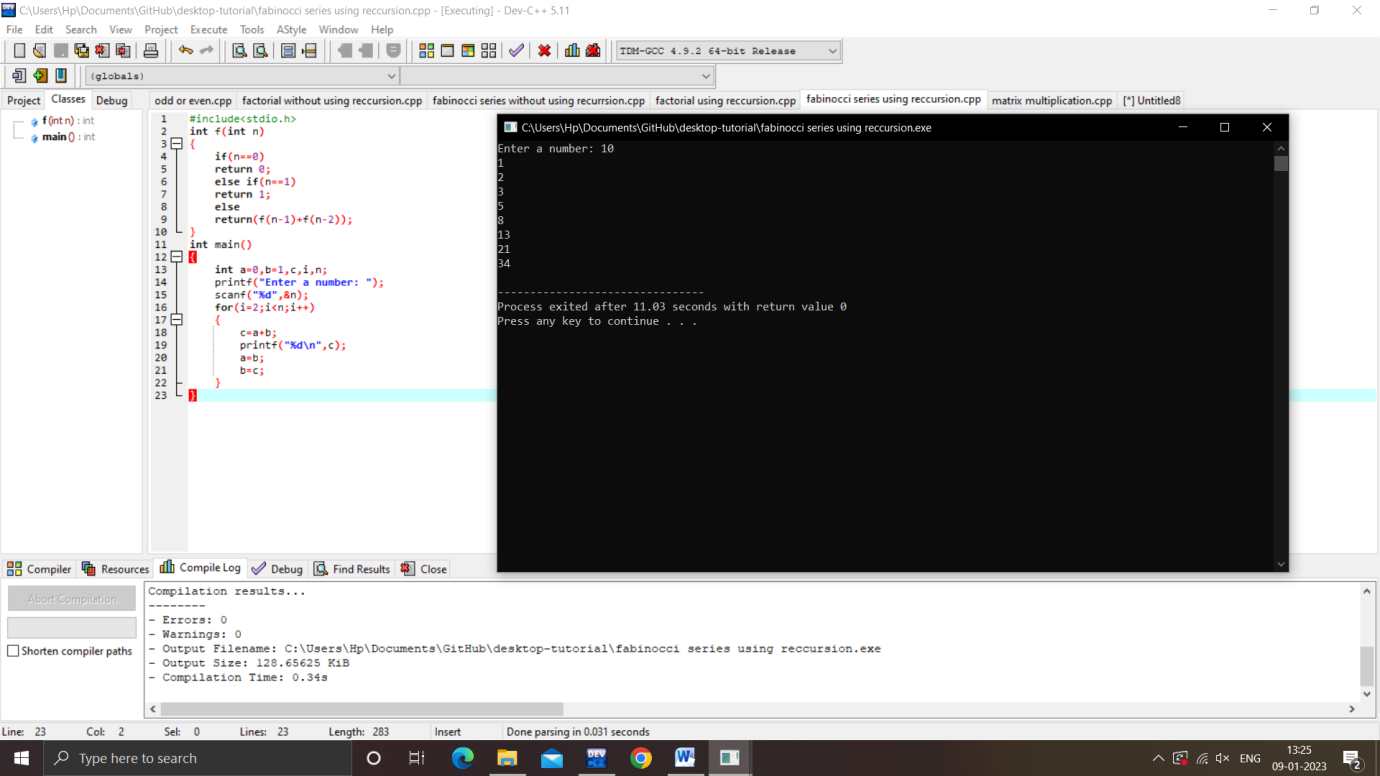
printf("%d\n",c);

a=b;

b=c;

}

}



**Matrix multiplication**

#include<stdio.h>

int main()

{

int a[10][10],b[10][10],c[10][10],i,j,k,n,m,l;

printf("Enter the no.of rows and columns of A: ");

scanf("%d %d",&n,&m);

printf("Enter the no.of rows and columns of B: ");

scanf("%d %d",&m,&l);

printf("Enter the elements of A: ");

for(i=0;i<n;i++)

for(j=0;j<m;j++)

scanf("%d",&a[i][j]);

printf("Enter the elements of B: ");

for(j=0;j<m;j++)

for(k=0;k<l;k++)

scanf("%d",&b[j][k]);

for(i=0;i<n;i++)

for(k=0;k<l;k++)

{

c[i][k]=0;

for(j=0;j<m;j++)

{

c[i][k]+=a[i][j]\*b[j][k];

}

}

for(i=0;i<n;i++)

{

for(k=0;k<l;k++)

{

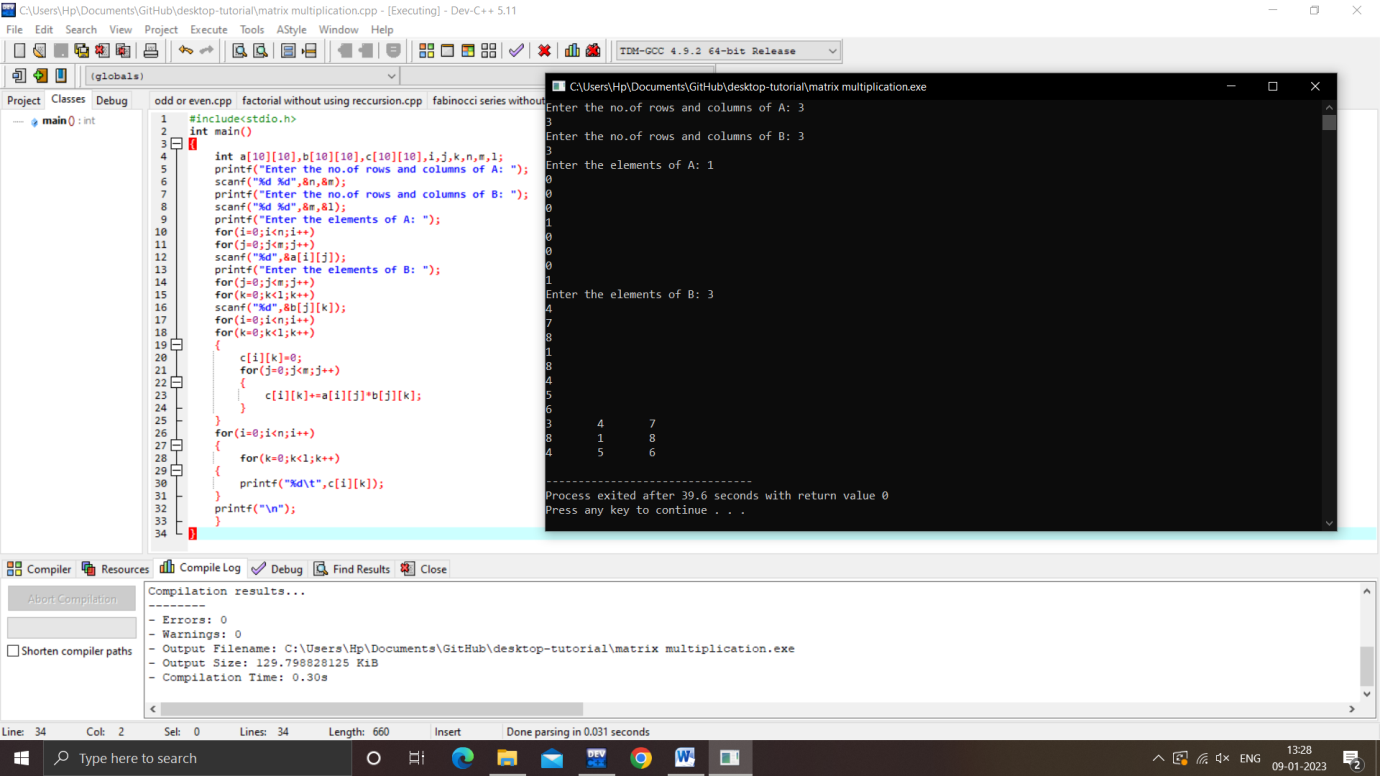
printf("%d\t",c[i][k]);

}

printf("\n");

}

}



Liner search in array

#include<stdio.h>

int main()

{

int a[10],i,n,flag=0,key;

printf("Enter the size of array: ");

scanf("%d",&n);

printf("Enter the elements: ");

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("Enter the key value: ");

scanf("%d",&key);

for(i=0;i<=n;i++)

{

if(a[i]==key)

{

flag=1;

break;

}

}

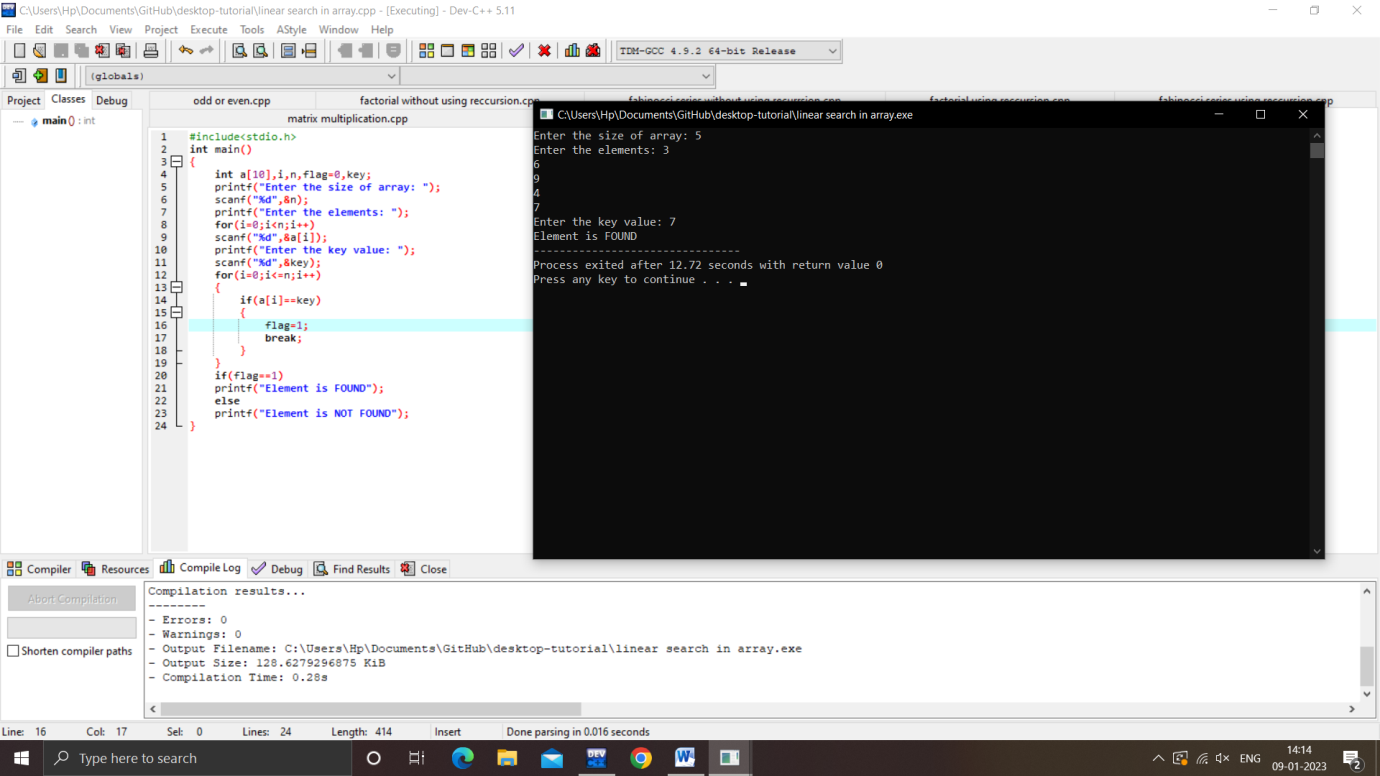
if(flag==1)

printf("Element is FOUND");

else

printf("Element is NOT FOUND");

}



**Binary search in array**

#include<stdio.h>

int main()

{

int a[20],n,i,key,flag=0,f,l,m;

printf("Enter the size of array: ");

scanf("%d",&n);

printf("ENTER THE ELEMENTS IN ASCENDING ORDER : ");

for(i=0;i<n;i++)

scanf("%d",&a[i]);

printf("Enter the key value: ");

scanf("%d",&key);

f=0;

l=n-1;

while (f<=l)

{

m=(f+l)/2;

if(a[m]==key)

{

flag=1;

break;

}

else if(a[m]<key)

f=m+1;

else if(a[m]>key)

l=m-1;

}

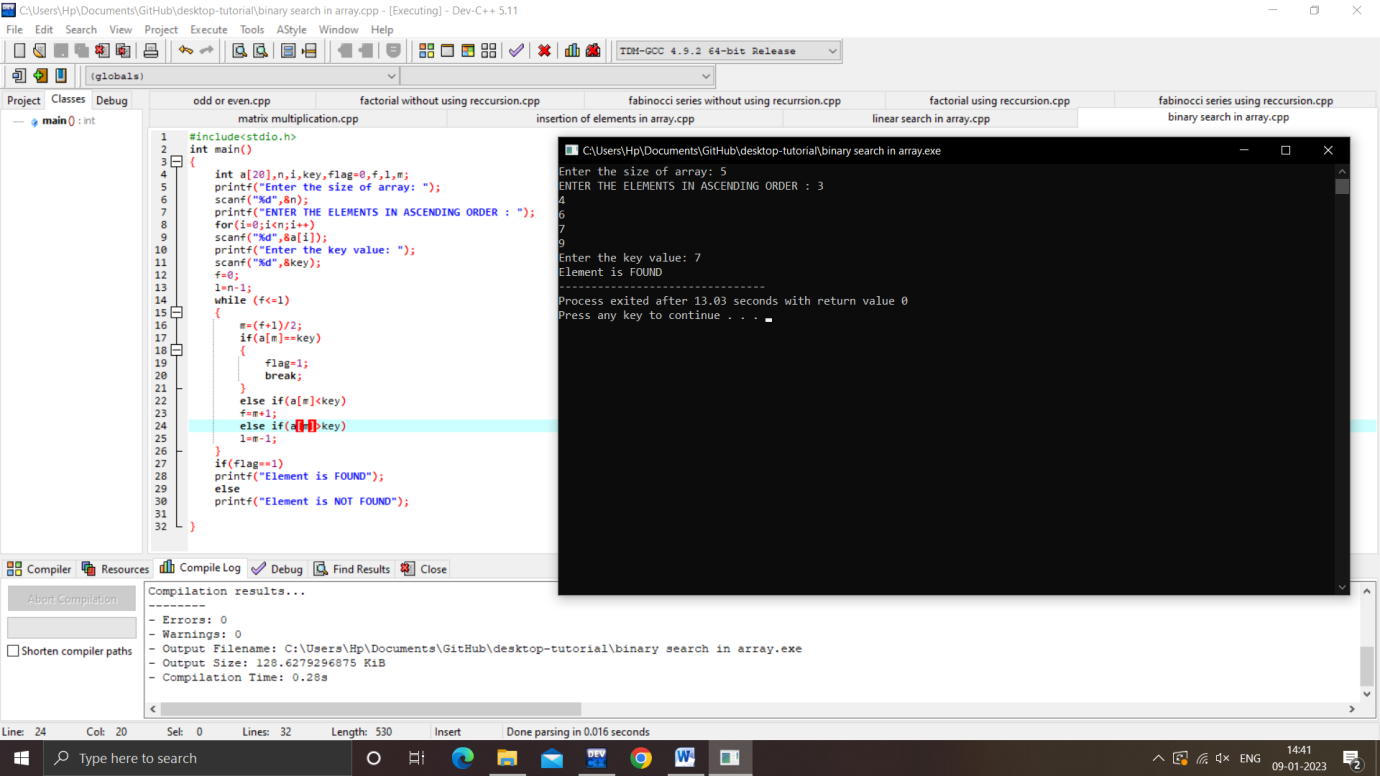
if(flag==1)

printf("Element is FOUND");

else

printf("Element is NOT FOUND");

}



**Array operations**

#include<stdio.h>

#include<stdlib.h>

int arr[100],i,n;

void Arr\_Insertion()

{

int pos,ele;

printf("Enter the position of element: ");

scanf("%d",&pos);

printf("Enter the element: ");

scanf("%d",&ele);

if(pos>n)

printf("INVALID INPUT");

else

{

for(i=n-1;i>=pos-1;i--)

arr[i+1]=arr[i];

arr[pos]=ele;

printf("array after insersion :\n");

for(i=0;i<=n;i++)

printf("%d\n",arr[i]);

}

}

void Arr\_Deletion()

{

int pos;

printf("Enter the position of element: ");

scanf("%d",&pos);

if(pos>n)

printf("DELETION NOT POSSIBLE");

else

{

for(i=pos;i<n;i++)

arr[i]=arr[i+1];

printf("array after deletion :\n");

for(i=0;i<=n-2;i++)

printf("%d\n",arr[i]);

}

}

int main()

{

int pos,ele,ch;

printf("Enter the size of the array: ");

scanf("%d",&n);

printf("Enter the elements in the array: ");

for(i=0;i<n;i++)

scanf("%d",&arr[i]);

do

{

printf("1 for insertion\n");

printf("2 for deletion\n");

printf("3 for display\n");

printf("Enter the choice:");

scanf("%d",&ch);

switch(ch)

{

case 1: Arr\_Insertion();

break;

case 2: Arr\_Deletion();

break;

case 3:

for(i=0;i<n;i++)

printf("%d\n",arr[i]);

break;

case 4:

exit(0);

default:

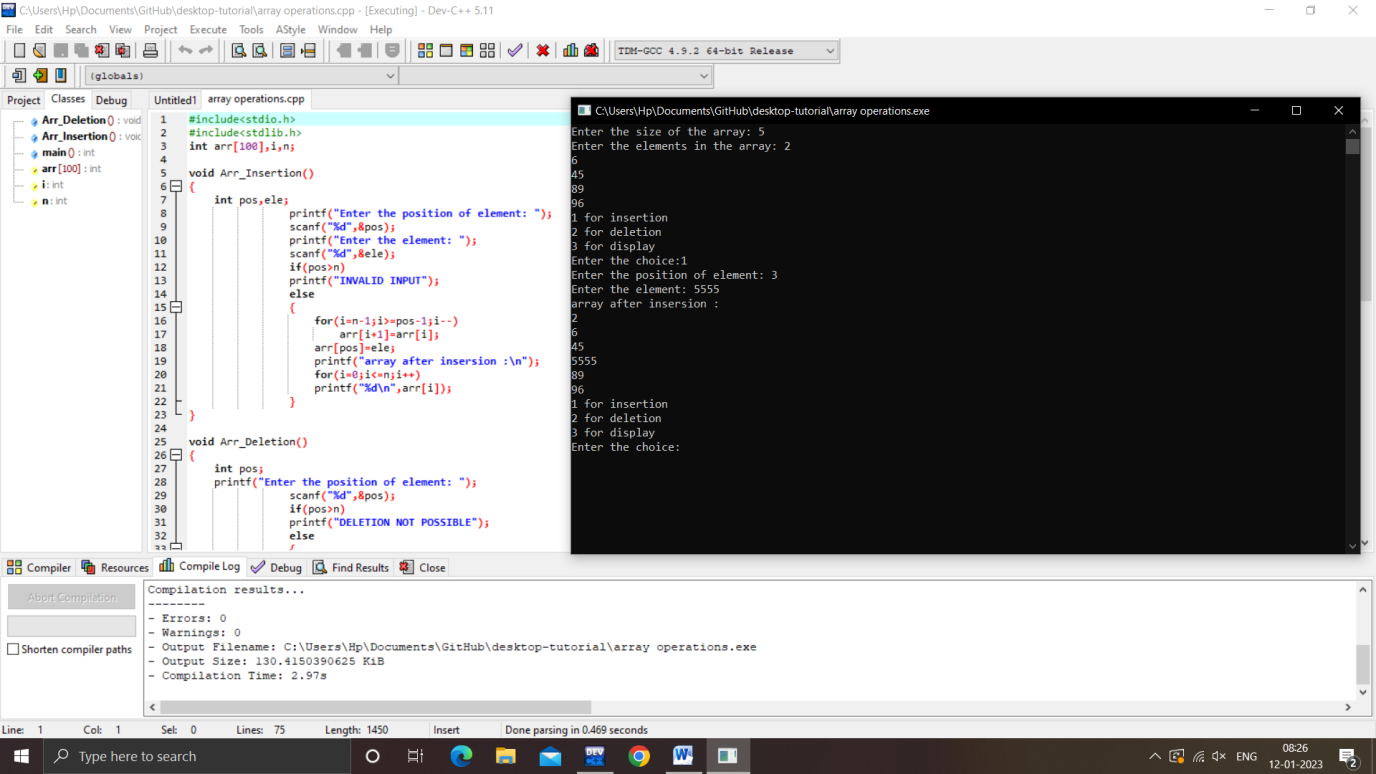
printf("Enter the number between 1 to 4 ");

}

}

while(ch>0 && ch<=4);

}



**Stack operations**

#include<stdio.h>

#include<stdlib.h>

int stk[10],i,n,top=-1;

void push()

{

int ele;

if(top==10-1)

{

printf("STACK IS FULL");

}

else

{

printf("Enter the element to be pushed: ");

scanf("%d",&ele);

top++;

stk[top]=ele;

}

}

void pop()

{

if(top==-1)

printf("STACK IS EMPTY");

else

{

printf("popped element is %d ",stk[top]);

top=top-1;

}

}

void peak()

{

printf("peak element is %d",stk[top]);

}

void display()

{

int i;

if(top==-1)

printf("STACK IS EMPTY");

else

{

for(i=top;i>=0;i--)

printf("%d\n",stk[i]);

}

}

int main()

{

int ch;

printf("STACK OPERATIONS");

do

{

printf("\n1.Push\n2.Pop\n3.Peak\n4.Display\n5.Exit");

printf("\n\nEnter the choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

push();

break;

case 2:

pop();

break;

case 3:

peak();

break;

case 4:

display();

break;

case 5:

exit(0);

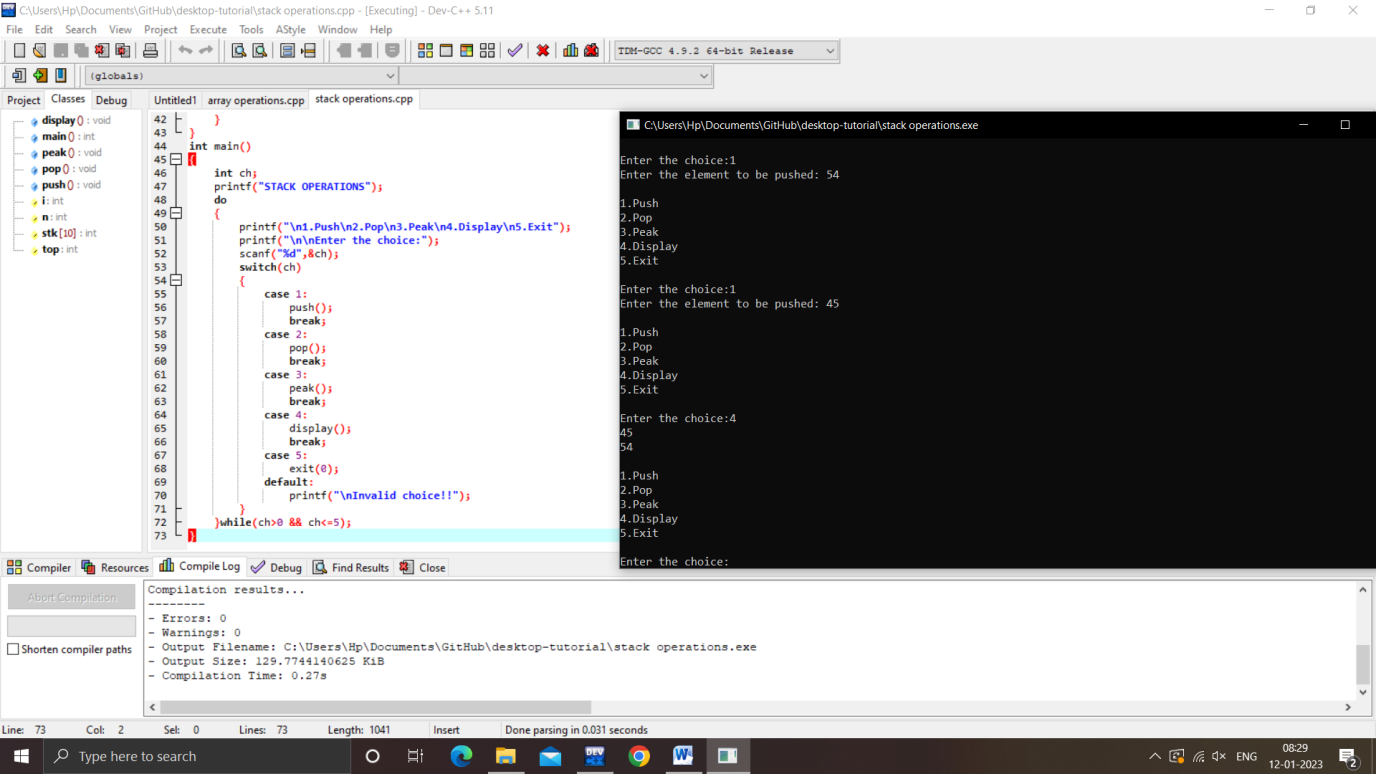
default:

printf("\nInvalid choice!!");

}

}while(ch>0 && ch<=5);

}



**Infix to postfix conversion**

#include<stdio.h>

#include<ctype.h>

#include<string.h>

char stk[20];

int top=-1;

void push(char c)

{

stk[++top]=c;

}

char pop()

{

return(stk[top--]);

}

int priority(char c)

{

if(c=='^'|| c=='&' || c=='|')

return 3;

else if (c=='/'|| c=='\*' || c=='%')

return 2;

else if(c=='+' || c=='-')

return 1;

else

return 0;

}

main()

{

char in[50],post[50],ch;

int i,j,l;

printf("Enter the string :");

gets(in);

l=strlen(in);

j=0;

for(i=0;i<l;i++)

{

if(isalpha(in[i]))

post[j++]=in[i];

else

{

if(in[i]=='(')

push(in[i]);

else if(in[i]==')')

while((ch=pop())!='(')

post[j++]=ch;

else

{

while(priority(in[i])<=priority(stk[top]))

post[j++]=pop();

push(in[i]);

}

}

}

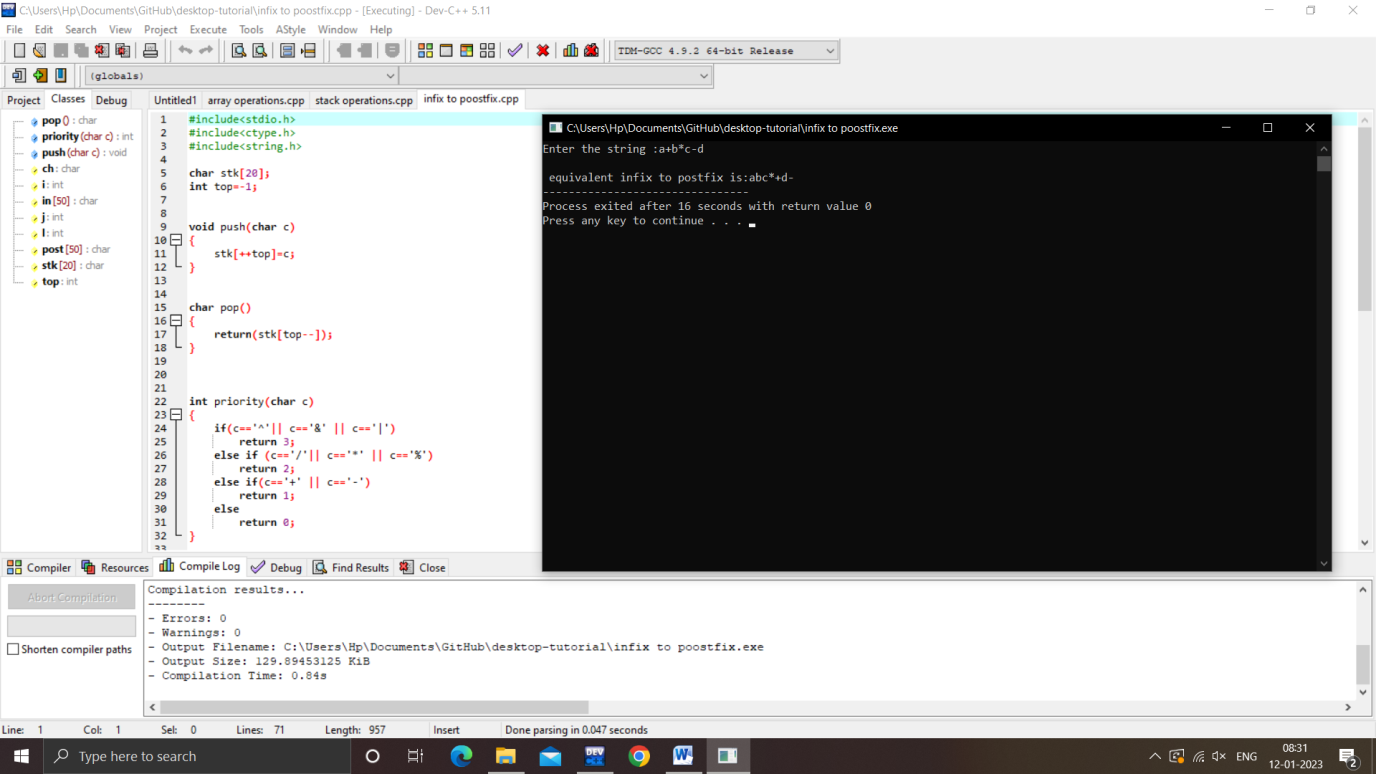
while(top!=-1)

post[j++]=pop();

post[j]='\0';

printf("\n equivalent infix to postfix is:%s",post);

}



**Postfix expression evaluation**

#include<stdio.h>

#include<conio.h>

#include<ctype.h>

#include<string.h>

int stk[20];

int top =-1;

void push(int val)

{

stk[++top] = val;

}

int pop()

{

return(stk[top--]);

}

main()

{

int op1,op2,len,result,i;

char postfix[30],op;

printf("\nEnter the postfix expression: ");

gets(postfix);

len=strlen(postfix);

for(i=0 ;i<len ; i++)

{

if( isdigit(postfix[i]))

push(postfix[i]-'0');

else

{

op=postfix[i];

op2=pop();

op1=pop();

switch(op)

{

case '+': push(op1+op2);

break;

case '-': push(op1-op2);

break;

case '\*': push(op1\*op2);

break;

case '/': push(op1/op2);

break;

case '%': push(op1%op2);

break;

}

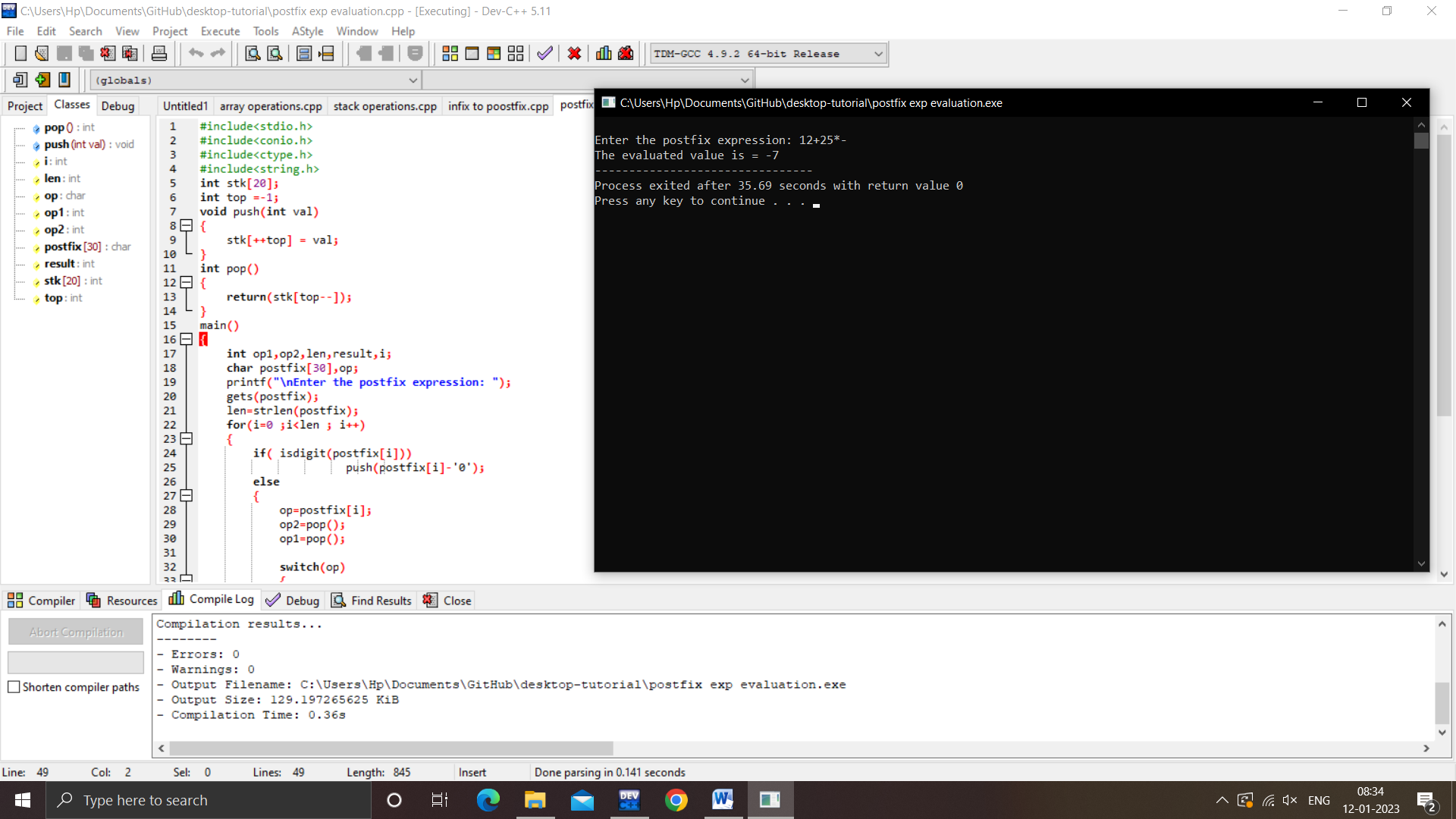
}

}

result=pop();

printf("The evaluated value is = %d",result);

}



**Queue operations**

#include<stdio.h>

#include<stdlib.h>

#define size 5

int que[20];

int i,f=-1,r=-1;

void enqueue()

{

int ele;

printf("Enter the element to be inserted : ");

scanf("%d",&ele);

if(r==size-1)

{

printf("QUEUE IS FULL\n");

}

else if(f==-1 && r==-1)

{

f=0;

r=0;

que[r]=ele;

}

else

{

r=r+1;

que[r]=ele;

}

}

void dequeue()

{

if(r==-1 && f==-1)

printf("QUEUE IS EMPTY\n");

else

{

f=f+1;

}

}

void display()

{

if(r==-1 && f==-1)

printf("QUEUE IS EMPTY\n");

else

{

for(i=f;i<=r;i++)

printf("%d\t",que[i]);

}

}

int main()

{

int ch;

do

{

printf("\nTHE QUEUE OPERATIONS ARE:\n");

printf(" 1.ENQUEUE\n 2.DEQUEUE\n 3.DISPLAY\n 4.EXIT\n");

printf("\nEnter your choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

enqueue();

break;

case 2:

dequeue();

break;

case 3:

display();

break;

case 4:

exit(0);

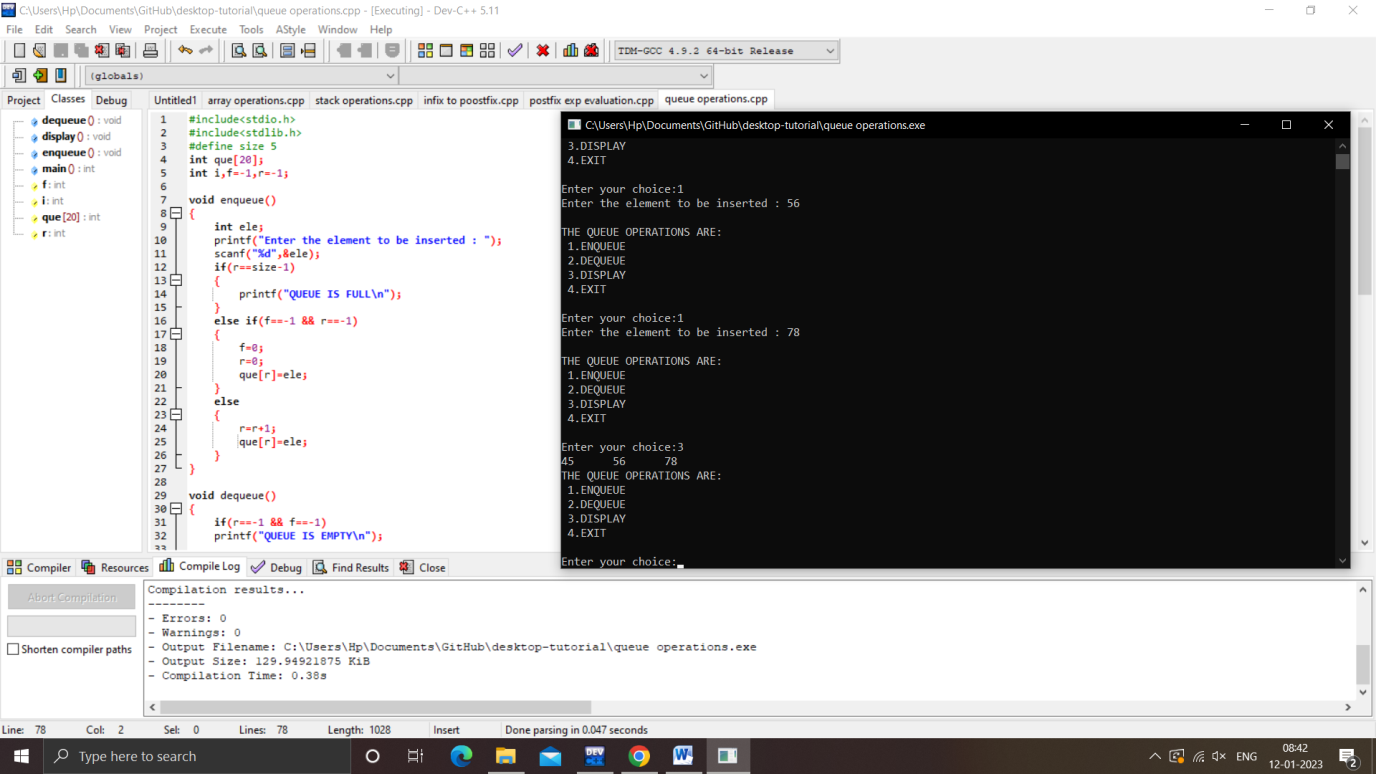
default:

printf("\nENTER A VALID CHOICE\n");

}

}while(ch>0 && ch<=4);

}



**Linked list operations**

#include <stdio.h>

#include <malloc.h>

#include <stdlib.h>

struct node

{

int data;

struct node \*next;

}\*p,\*newnode,\*head=NULL,\*t;

void create()

{

int i,n,ele;

printf("Enter the number of elements to be created: ");

scanf("%d",&n);

for(i=0;i<n;i++)

{

newnode = (struct node\*)malloc(sizeof(struct node));

printf("Enter the element : ");

scanf("%d",&ele);

newnode->data = ele;

newnode->next = NULL;

if(head==NULL)

{

head=newnode;

p=newnode;

}

else

{

for(p=head; p->next!= NULL; p=p->next);

p->next=newnode;

p=newnode;

}

}

}

void insert\_b()

{

int a;

printf("enter the value: ");

scanf("%d",&a);

newnode = (struct node\*)malloc(sizeof(struct node));

newnode->data = a;

newnode->next = head;

head = newnode;

}

void insert\_e()

{

newnode = (struct node\*)malloc(sizeof(struct node));

int a;

printf("enter the value: ");

scanf("%d",&a);

newnode->data = a;

newnode->next = NULL;

while(t->next != NULL)

{

t = t->next;

}

t->next = newnode;

}

void insert\_any()

{

struct node\* newnode = (struct node\*) malloc(sizeof(struct node));

int a,i,pos;

printf("enter the a value and pos: ");

scanf("%d%d",&a,&pos);

newnode->data = a;

struct node \*t = head;

for(int i=2; i < pos; i++) {

if(t->next != NULL) {

t = t->next;

}

}

newnode->next = t->next;

t->next = newnode;

}

void delete\_b()

{

int ele;

p=head;

head=p->next;

ele=p->data;

free(p);

printf("The deleted element = %d",ele);

}

void delete\_e()

{

int ele;

for(p=head;p->next!=NULL;p=p->next)

t=p;

t->next=NULL;

ele=p->data;

free(p);

printf("The deleted element = %d",ele);

}

void delete\_any\_position()

{

int pos,i,ele;

printf("\nEnter the Position: ");

scanf("%d",&pos);

for(p=head,i=1; i<pos; p=p->next,i++)

t=p;

t->next=p->next;

ele=p->data;

free(p);

printf("\nThe deleted element is ->:%d ",ele);

}

void display()

{

if(head==NULL)

printf("SLL is Empty");

else

for(p=head; p!=NULL; p=p->next)

printf("%d -> ",p->data);

}

int main()

{

int ch;

do

{

printf("\n1.Create\n2.insert\_b\n3.insert\_e\n4.insert\_any\n5.Display\n6.delete\_b\n7.delete\_e\n8.delete at any position\n9.exit\n");

printf("enter the choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

create();

break;

case 2:

insert\_b();

break;

case 3:

insert\_e();

break;

case 4:

insert\_any();

break;

case 5:

display();

break;

case 6:

delete\_b();

break;

case 7:

delete\_e();

break;

case 8:

delete\_any\_position();

break;

case 9:

exit(0);

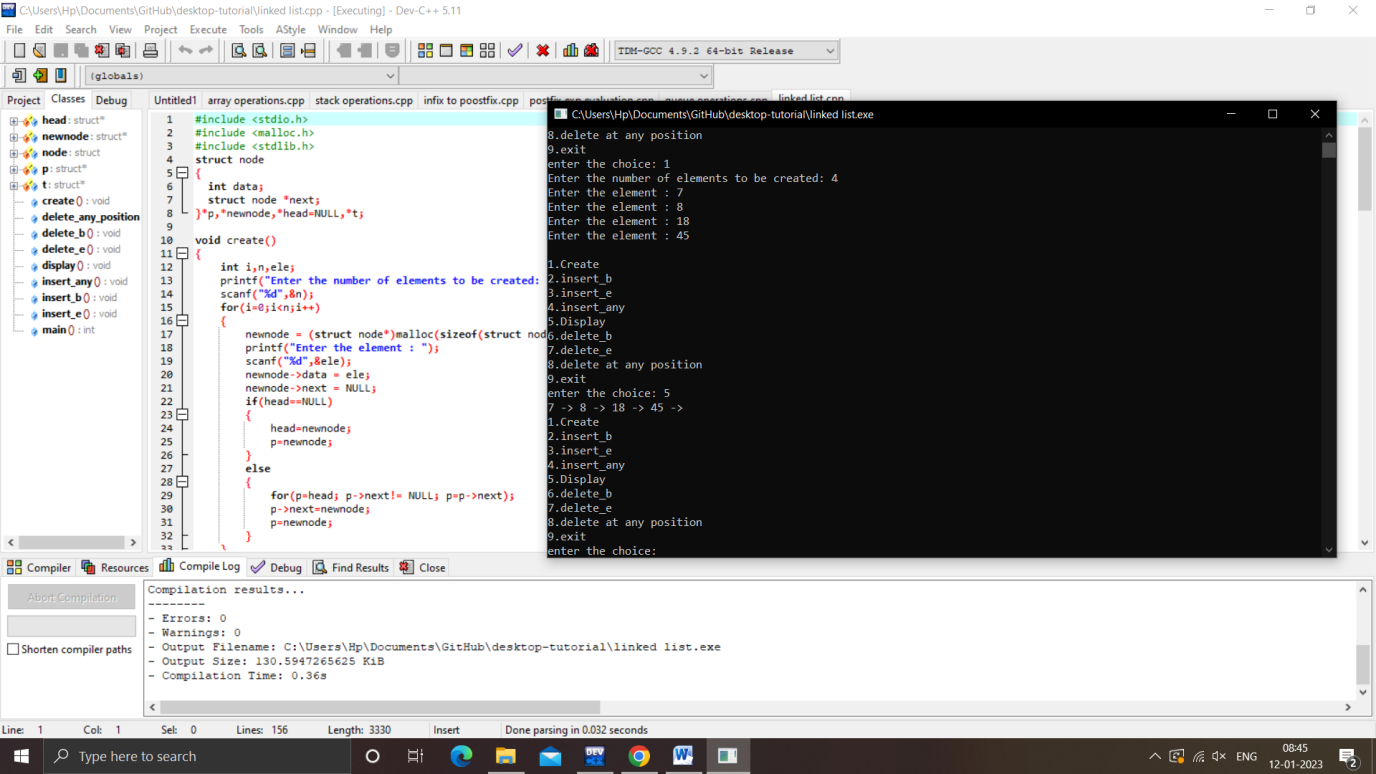
default:

printf("\n wrong input");

}

} while(ch>=1 && ch<=9);

}



**BST tree inorder , preorder , postorder**

#include<stdio.h>

#include<stdlib.h>

#include<malloc.h>

struct node

{

int data;

struct node \*l,\*r;

} \*root=NULL,\*newnode;

struct node\* create(struct node \*root, int ele)

{

if(root==NULL)

{

newnode = (struct node \*)malloc(sizeof(struct node));

newnode->data=ele;

newnode->l=NULL;

newnode->r=NULL;

return(newnode);

}

else if(ele < root->data)

root->l=create(root->l,ele);

else if(ele> root->data)

root->r=create(root->r,ele);

return(root);

}

void inorder(struct node \*root)

{

if(root==NULL)

return;

inorder(root->l);

printf("-(%d)- ",root->data);

inorder(root->r);

}

void preorder(struct node \*root)

{

if(root!=NULL)

{

printf("-(%d)- ",root->data);

preorder(root->l);

preorder(root->r);

}

}

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->l);

postorder(root->r);

printf("-(%d)- ",root->data);

}

}

int main()

{

int ch,ele;

do

{

printf("\n \*tree menu \*");

printf("\n\n1.create \n2.inorder \n3.preorder \n4.postorder \n5.exit");

printf("\n\n enter your choice(1-5):");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("enter the element:");

scanf("%d",&ele);

root=create(root,ele);

break;

case 2:

inorder(root);

break;

case 3:

preorder(root);

break;

case 4:

postorder(root);

break;

case 5:exit(0);

default:

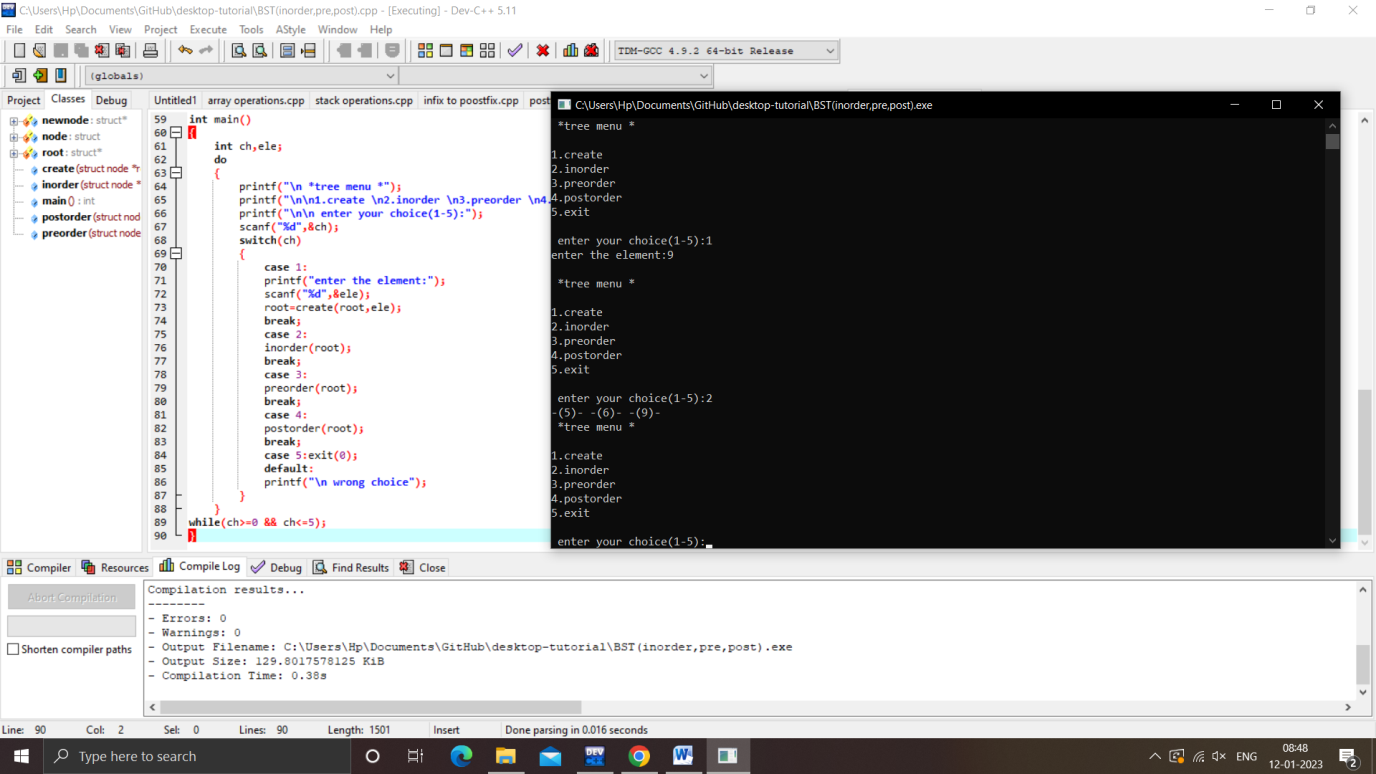
printf("\n wrong choice");

}

}

while(ch>=0 && ch<=5);

}



**Insertion sort**

#include<stdio.h>

int insertion\_sort(int A[],int n)

{

int i,j,temp;

for(i=1;i<n;i++)

{

j=i;

while(j>0 && A[j]<A[j-1])

{

temp = A[j-1];

A[j-1]=A[j];

A[j]=temp;

j--;

}

}

}

int main()

{

int i,n,A[100];

printf("\nEnter n:");

scanf("%d",&n);

printf("\nEnter Array elements:");

for(i=0;i<n;i++)

scanf("%d",&A[i]);

insertion\_sort(A,n);

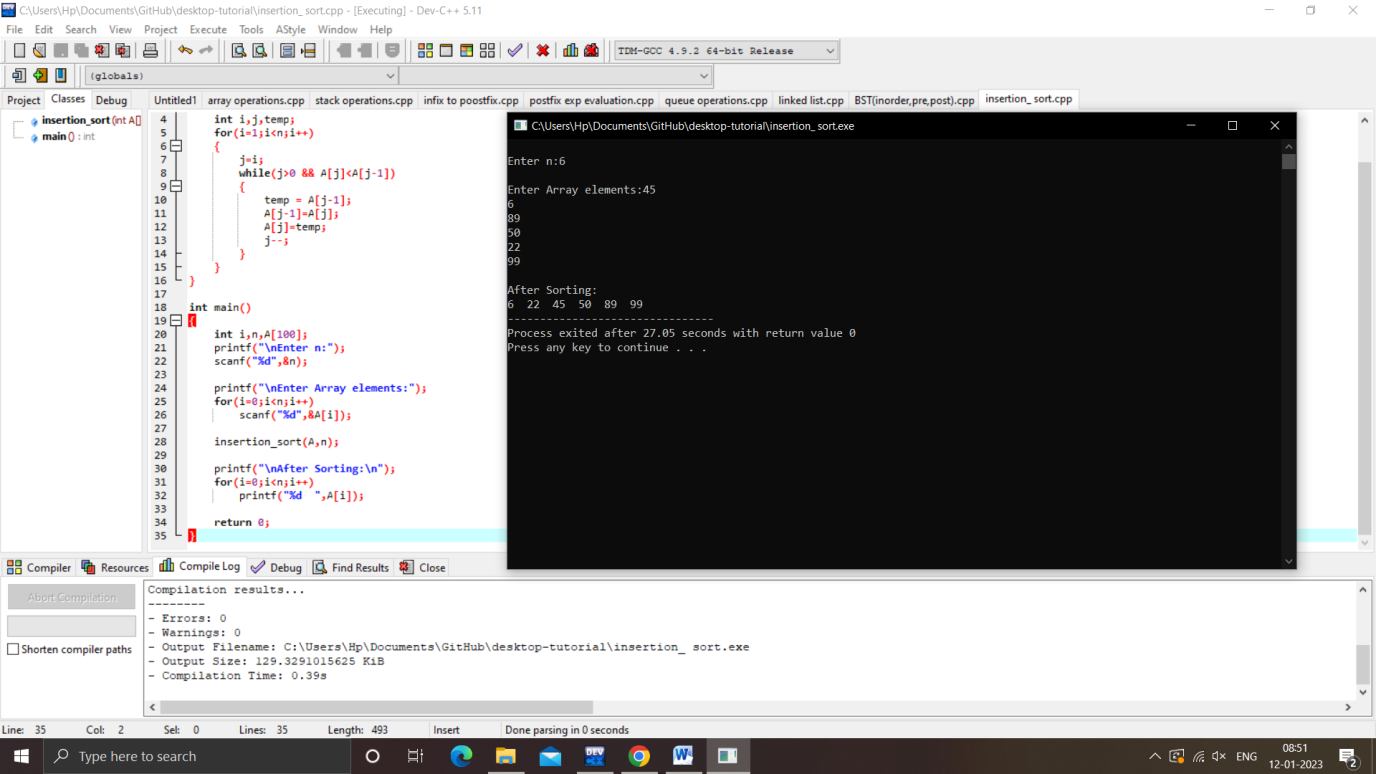
printf("\nAfter Sorting:\n");

for(i=0;i<n;i++)

printf("%d ",A[i]);

return 0;

}



**Heap sort**

#include<stdio.h>

heapify(int a[], int n, int i)

{

int root,l,r,t;

root = i;

l = 2\*i + 1;

r = 2\*i + 2;

if (l<n && a[l] > a[root])

root = l;

if (r<n && a[r] > a[root])

root = r;

if (root != i)

{

t=a[i];

a[i] = a[root];

a[root] = t;

heapify(a, n, root);

}

}

heapsort(int a[], int n)

{

int i,t;

for(i=n/2-1; i>=0; i--)

heapify(a, n, i);

for(i=n-1; i>=0; i--)

{

t= a[0];

a[0]= a[i];

a[i] = t;

heapify(a, i, 0);

}

}

int main()

{

int a[50],i,n;

printf("Enter total number of elements:");

scanf("%d", &n);

printf("Enter the elements:\n");

for(i = 0; i < n; i++)

scanf("%d", &a[i]);

heapsort(a,n);

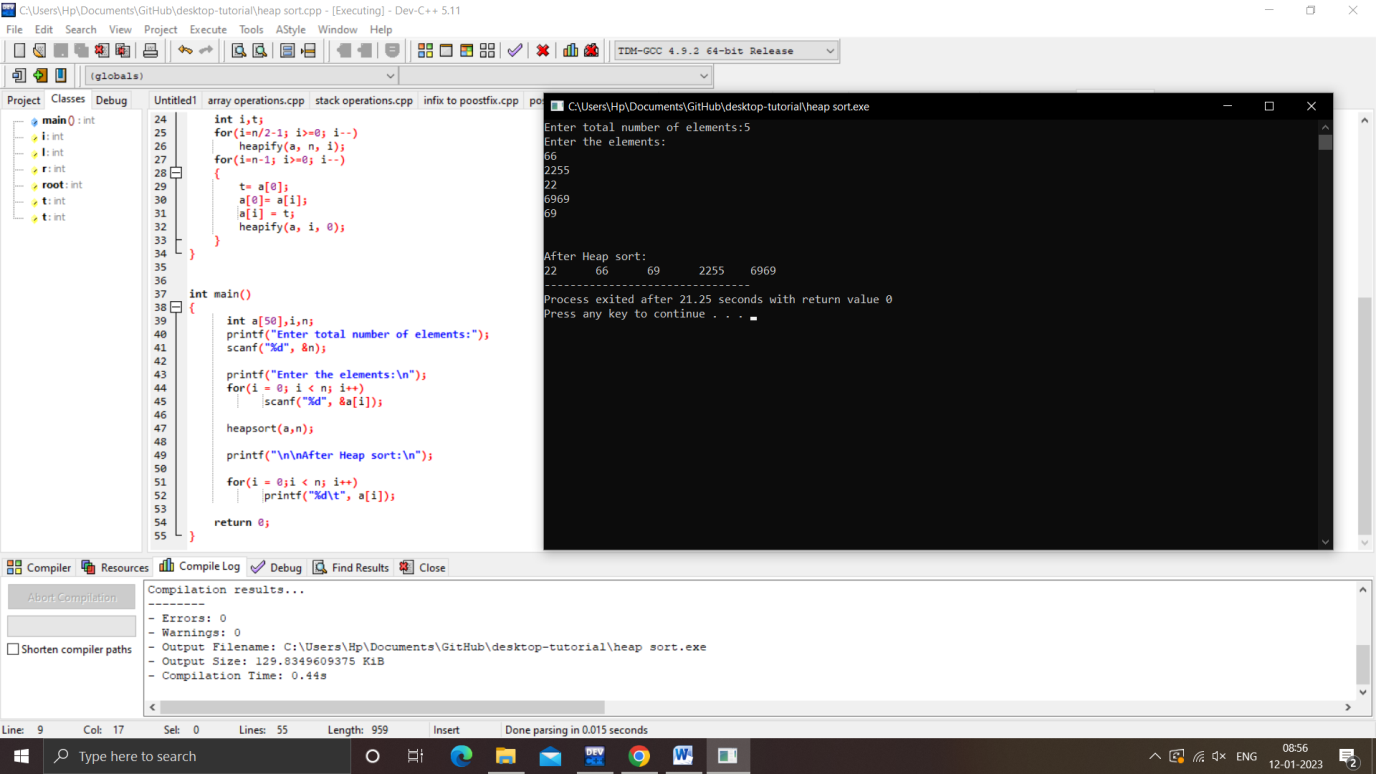
printf("\n\nAfter Heap sort:\n");

for(i = 0;i < n; i++)

printf("%d\t", a[i]);

return 0;

}



**Merge sort**

#include <stdio.h>

void merge\_Sort(int [], int, int, int);

void partition(int [],int, int);

main()

{

int a[50] , i, n;

printf("Enter total number of elements:");

scanf("%d", &n);

printf("Enter the elements:\n");

for(i = 0; i < n; i++)

scanf("%d", &a[i]);

partition( a, 0, n - 1);

printf("After merge sort:\n");

for(i = 0;i < n; i++)

printf("%d\t", a[i]);

}

void partition(int a[],int low,int high)

{

int mid;

if(low < high)

{

mid = (low + high)/2;

partition( a, low, mid);

partition(a, mid+1, high);

merge\_Sort(a, low, mid, high);

}

}

void merge\_Sort(int a[], int low, int mid, int high)

{

int i, j, k, lo, temp[50];

lo = low;

i = low;

j = mid + 1;

while ((lo <= mid) && (j <= high))

{

if (a[lo] <= a[j])

{

temp[i] = a[lo];

lo++;

}

else

{

temp[i] = a[j];

j++;

}

i++;

}

if (lo > mid)

{

for (k = j; k <= high; k++)

{

temp[i] = a[k];

i++;

}

}

else

{

for(k=lo;k<=mid;k++)

{

temp[i]=a[k];

i++;

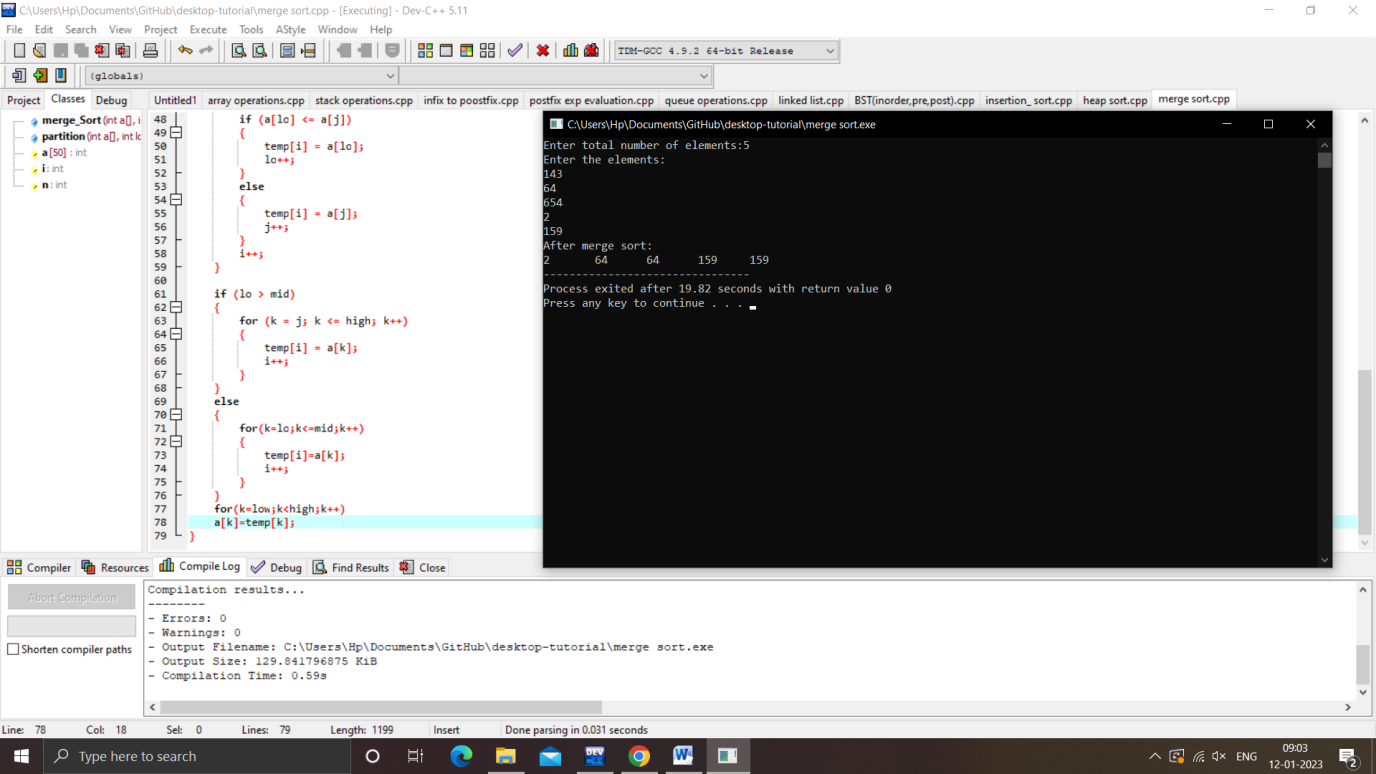
}

}

for(k=low;k<high;k++)

a[k]=temp[k];

}



**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("enter the number of elements:");

scanf("%d",&count);

printf("enter %d elements: ",count);

for(i=0;i<count;i++)

scanf("%d",&number[i]);

quicksort(number,0,count-1);

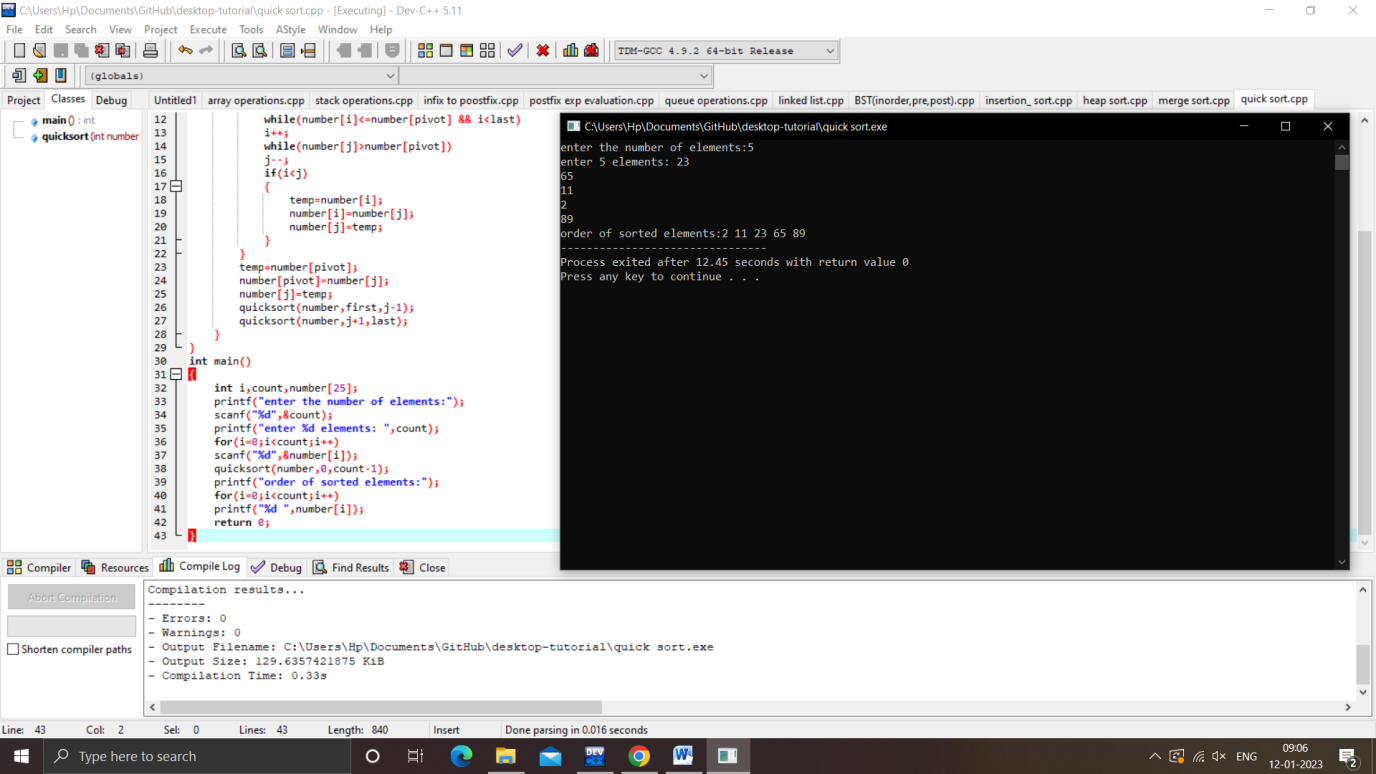
printf("order of sorted elements:");

for(i=0;i<count;i++)

printf("%d ",number[i]);

return 0;

}



**HASHING – separate chaining**

#include<stdio.h>

#include<malloc.h>

#include<stdlib.h>

#define size 10

struct hash

{

int key;

struct hash \*next;

}\*ht[size],\*newnode,\*p;

void initially()

{

int i;

for(i=0;i<size;i++)

{

ht[i]=NULL;

}

}

void insert()

{

int hi,ele;

printf("Enter the value of Key");

scanf("%d",&ele);

hi=ele%size;

newnode=(struct hash\*)malloc(sizeof(struct hash));

newnode->key = ele;

newnode->next=NULL;

if(ht[hi]==NULL)

{

ht[hi]=newnode;

}

else

{

for(p=ht[hi];p->next !=NULL ;p=p->next);

p->next = newnode;

}

}

void find()

{

int i,n,flag=0;

printf("enter element want to find");

scanf("%d",&n);

for(p=ht[n%size];p!=NULL;p=p->next)

{

if(p->key==n)

{ flag=1;

break;

}

}

if(flag==1)

printf("found");

else

printf("not found");

}

void display()

{

int i;

printf("\n\*\*The Hash table\*\*\n\n HashTable[Hash Key]\t Chain of Key's\n");

for(i=0;i<size;i++)

{

if(ht[i] != NULL)

{

printf("Hash\_Table[%d] = ",i);

for(p=ht[i]; p != NULL; p =p->next)

printf("->%d ",p->key);

}

printf("\n\n");

}

}

main()

{

int cho,i,ele;

initially();

do

{

printf("\n\*\* Main Menu \*\*");

printf("\n1.Insert\n2.Find\n3.Display\n4.Exit\n");

printf("Enter your choice :");

scanf("%d",&cho);

switch(cho)

{

case 1:

insert();

break;

case 2:

find();

break;

case 3:

display();

break;

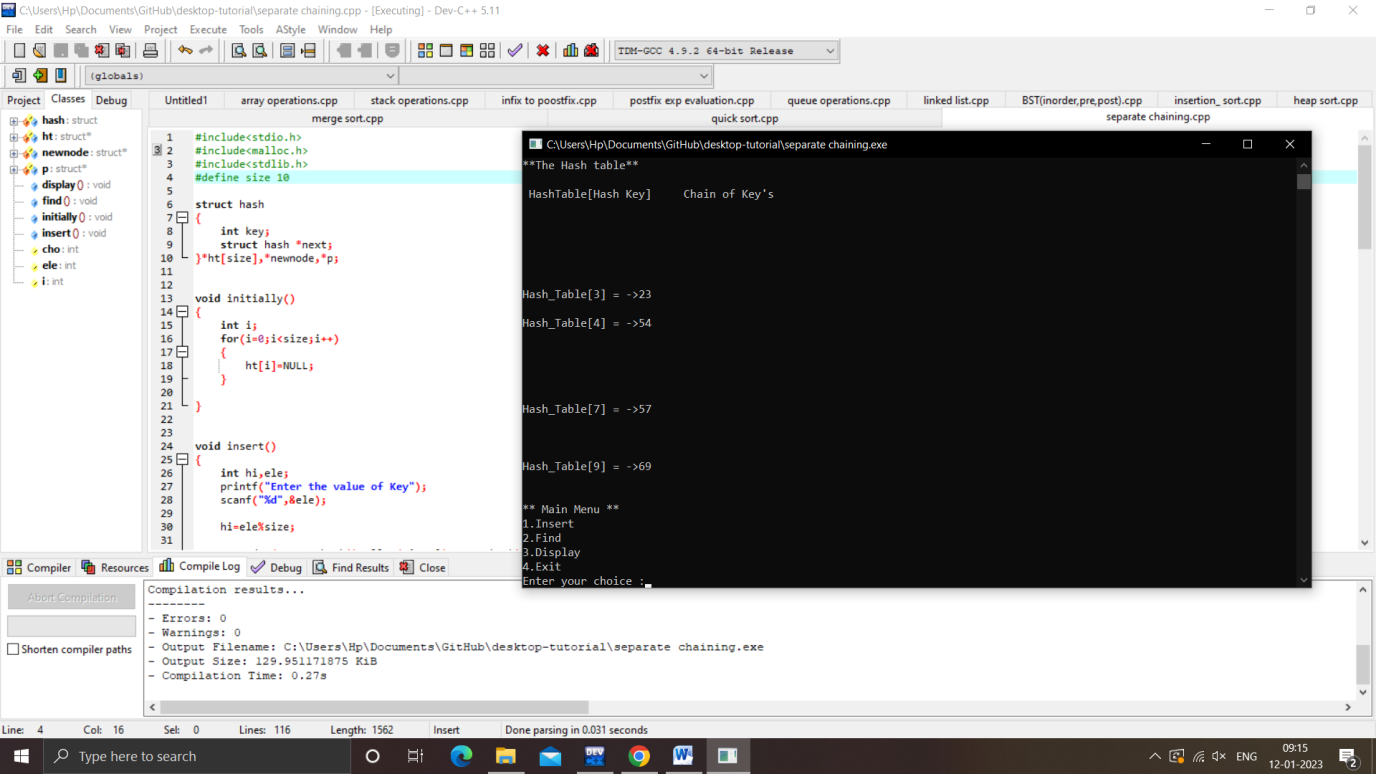
case 4:

exit(0);

}

}while(cho!=4);

}



**Avl tree**

#include<stdio.h>

#include<malloc.h>

struct node

{

int data,ht;

struct node \*left,\*right;

}\*root=NULL,\*temp;

struct node \*insert(struct node \*,int);

struct node \*Delete(struct node \*,int);

void preorder(struct node \*);

void inorder(struct node \*);

void postorder(struct node \*);

int height(struct node \*);

struct node \*rotateright(struct node \*);

struct node \*rotateleft(struct node \*);

struct node \*RR(struct node \*);

struct node \*LL(struct node \*);

struct node \*LR(struct node \*);

struct node \*RL(struct node \*);

int BF(struct node \*);

main()

{

int x,n,i,op;

do

{

printf("\n1)Create:\n2)Insert:\n3)Delete:\n4)Print:\n5)Quit:");

printf("\n\nEnterYour Choice:");

scanf("%d",&op);

switch(op)

{

case 1:printf("\nEnter no. of elements:");

scanf("%d",&n);

printf("\nEnter tree data:");

root=NULL;

for(i=0;i<n;i++)

{

scanf("%d",&x);

root=insert(root,x);

}

break;

case 2:

printf("\nEnter a data:");

scanf("%d",&x);

root=insert(root,x);

break;

case 3:

printf("\nEnter a data:");

scanf("%d",&x);

root=Delete(root,x);

break;

case 4:

printf("\nPreordersequence:\n");

preorder(root);

printf("\nInordersequence:\n");

inorder(root);

printf("\n");

}

}while(op!=5);

}

struct node\* insert(struct node \*root,int x)

{

if(root==NULL)

{

temp =(struct node\*)malloc(sizeof(struct node));

temp->data=x;

temp->left=NULL;

temp->right=NULL;

return (temp);

}

else if(x< root->data)

{

root->left=insert(root->left,x);

if(BF(root)==2)

{

if(x < root->left->data)

root=LL(root);

else

root=LR(root);

}

}

else if(x > root->data)

{

root->right=insert(root->right,x);

if(BF(root)==-2)

{

if(x>root->right->data)

root=RR(root);

else

root=RL(root);

}

}

root->ht=height(root);

return(root);

}

int height(struct node \*root)

{

int lh,rh;

if(root==NULL)

return(0);

if(root->left==NULL)

lh=0;

else

lh=1+root->left->ht;

if(root->right==NULL)

rh=0;

else

rh=1+root->right->ht;

if(lh>rh)

return(lh);

return(rh);

}

struct node \* rotateright(struct node \*x)

{

struct node \*y;

y=x->left;

x->left=NULL;

y->right=x;

x->ht=height(x);

y->ht=height(y);

return(y);

}

struct node \*rotateleft(struct node \*x)

{

struct node \*y;

y=x->right;

x->right=NULL;

y->left=x;

x->ht=height(x);

y->ht=height(y);

return(y);

}

struct node \*RR(struct node \*root)

{

root=rotateleft(root);

return(root);

}

struct node \*LL(struct node \*root)

{

root=rotateright(root);

return(root);

}

struct node \*LR(struct node \*root)

{

root->left=rotateleft(root->left);

root=rotateright(root);

return(root);

}

struct node \*RL(struct node \*root)

{

root->right=rotateright(root->right);

root=rotateleft(root);

return(root);

}

int BF(struct node \*root)

{

int lh,rh;

if(root==NULL)

return(0);

if(root->left==NULL)

lh=0;

else

lh=1+root->left->ht;

if(root->right==NULL)

rh=0;

else

rh=1+root->right->ht;

return(lh-rh);

}

void preorder(struct node \*root)

{

if(root!=NULL)

{

printf("%d(Bf=%d)\t",root->data,BF(root));

preorder(root->left);

preorder(root->right);

}

}

void inorder(struct node \*root)

{

if(root!=NULL)

{

inorder(root->left);

printf("%d(Bf=%d)\t",root->data,BF(root));

inorder(root->right);

}

}

void postorder(struct node \*root)

{

if(root!=NULL)

{

postorder(root->left);

postorder(root->right);

printf("%d(Bf=%d)\t",root->data,BF(root));

}

}

struct node \*Delete(struct node \*root,int x)

{

struct node \*p;

if(root==NULL)

{

return NULL;

}

else if(x> root->data)

{

root->right=Delete(root->right,x);

if(BF(root)==2)

if(BF(root->left)>=0)

root=LL(root);

else

root=LR(root);

}

else if(x<root->data)

{

root->left=Delete(root->left,x);

if(BF(root)==-2)

if(BF(root->right)<=0)

root=RR(root);

else

root=RL(root);

}

else

{

if(root->right!=NULL)

{

p=root->right;

while(p->left!= NULL)

p=p->left;

root->data=p->data;

root->right=Delete(root->right,p->data);

if(BF(root)==2)

if(BF(root->left)>=0)

root=LL(root);

else

root=LR(root);

}

else

return(root->left);

}

root->ht=height(root);

return(root);

}

