**Practical 1**

**AIM: Introduction to pointers.**

**(a) Call by Value (write a function that return max of two passed value)**

**(b) Call by reference.(write a function to swap value of two variable)**

**Code a:**

#include <stdio.h>

int max(int ,int);

int main()

{

int a,b,c;

printf("Enter the Numbers:");

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

c=max(a,b);

printf("Maximum Number is:%d",c);

return 0; }

int max(int x,int y)

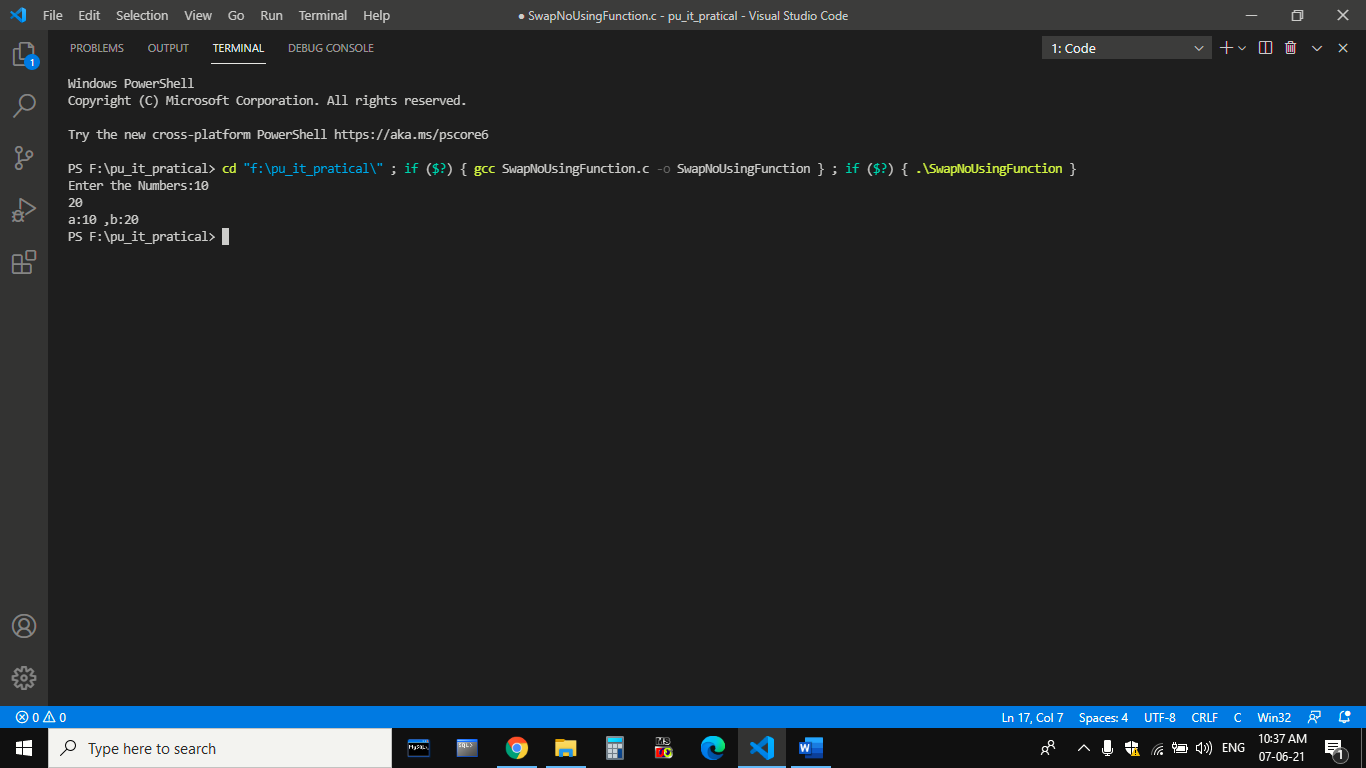
{ if (x>y)

{ return x; }

else

{ return y; } }

**Output:**



**Code b:**

#include <stdio.h>

void swap(int \*,int \*);

int main()

{

int a,b,c;

printf("Enter the Numbers:");

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

printf("before Swaping 1st value is :%d ,2nd value is :%d",a,b);

swap(&a,&b);

printf("\nAfter Swaping 1st value is :%d ,2nd value is :%d",a,b);

return 0;

}

void swap(int \*x,int \*y)

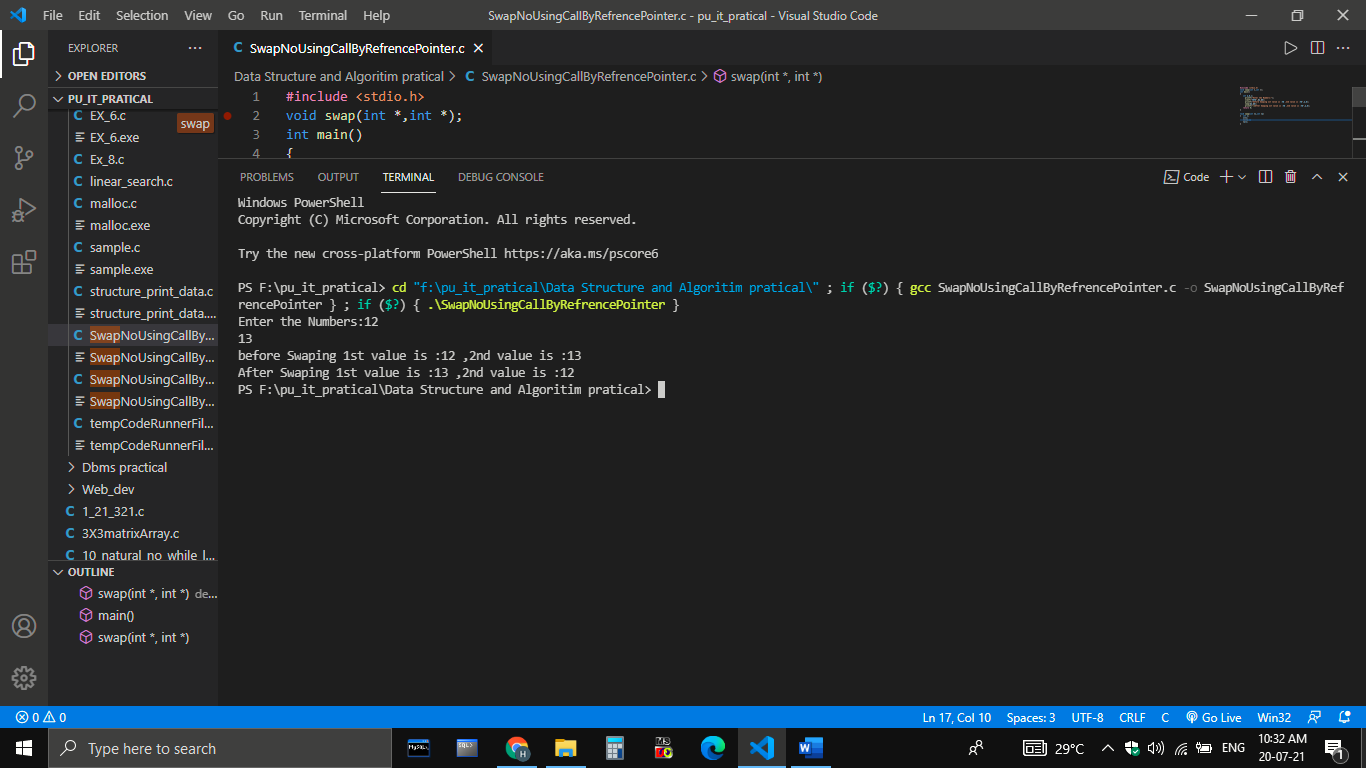
{ int z;

z=\*x;

\*x=\*y;

\*y=z; }

**Output:**



**Practical 2**

**Aim: Introduction to Dynamic Memory Allocation. DMA functions malloc(), calloc(), free() etc. (a) W.A.P. to create dynamic int array using malloc() and free() (b) W.A.P. to create dynamic char array using calloc() and free()**

**Code:**

#include<stdio.h>

#include<stdlib.h>

void main()

{

int i,n,\*p;

printf("Enter Size of array:");

scanf("%d",&n);

p=(int \*)malloc(n\*sizeof(int));

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

{

printf("Enter Elements:");

scanf("%d",p);

p++;

}

p=p-n;

printf("printing:");

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

{ printf("%d ",\*p);

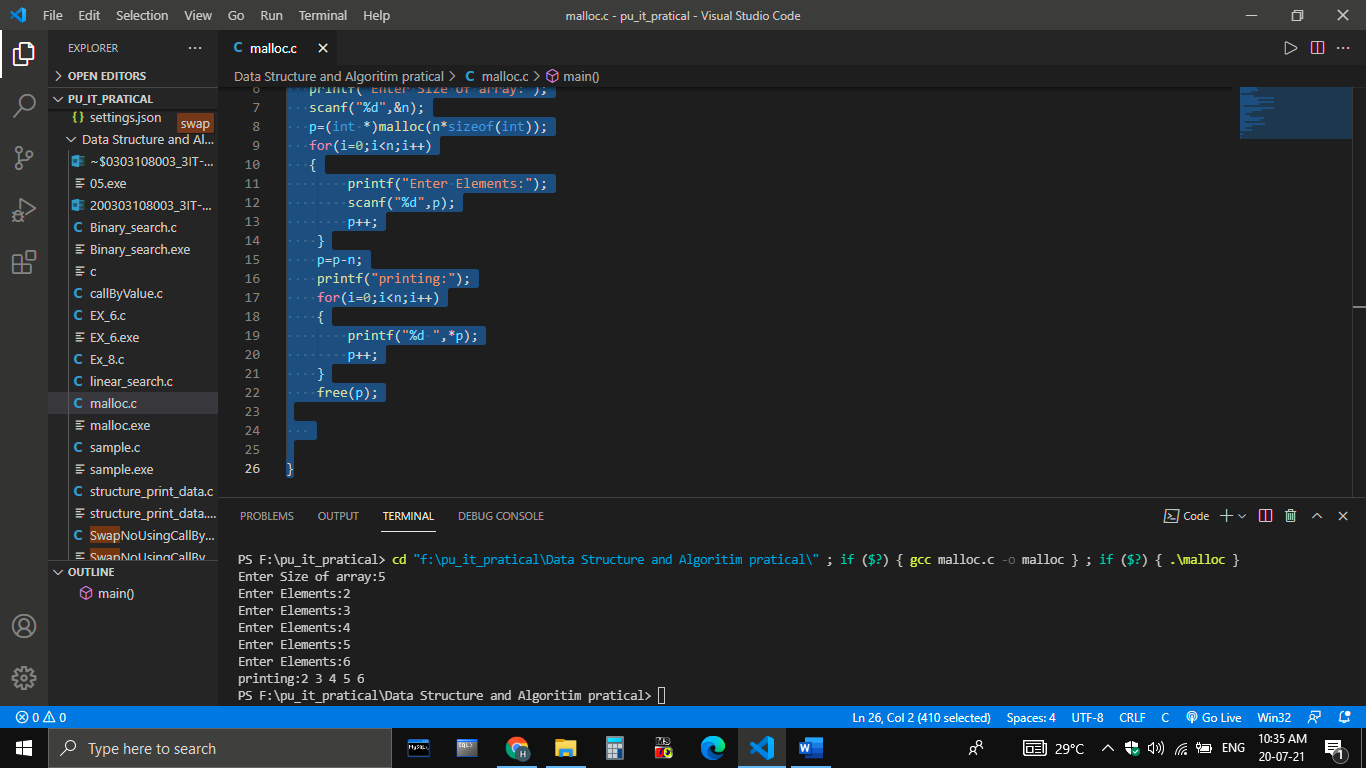
p++;

}

free(p);

}

**Output:**



**Practical 3**

**AIM:** **Write a program to implement structure in c.**

**Code:**

#include<stdio.h>

struct student

{ int eno;

char name[30];

char branch[10];

}s[3];

int main()

{

for(int i=0;i<3;i++)

{

printf("%d.Enter Name:\n",i+1);

gets(s[i].name);

printf("%d.Enter branch:\n",i+1);

gets(s[i].branch);

printf("%d.Enter eno:\n",i+1);

scanf("%d",&s[i].eno);

}

for(int i=0;i<3;i++)

{ printf("--------------------Printing the record of the student: %d--------------\n",i+1);

printf("Name:\t");

puts(s[i].name);

printf("Branch:\t");

puts(s[i].branch);

printf("Eno:\t");

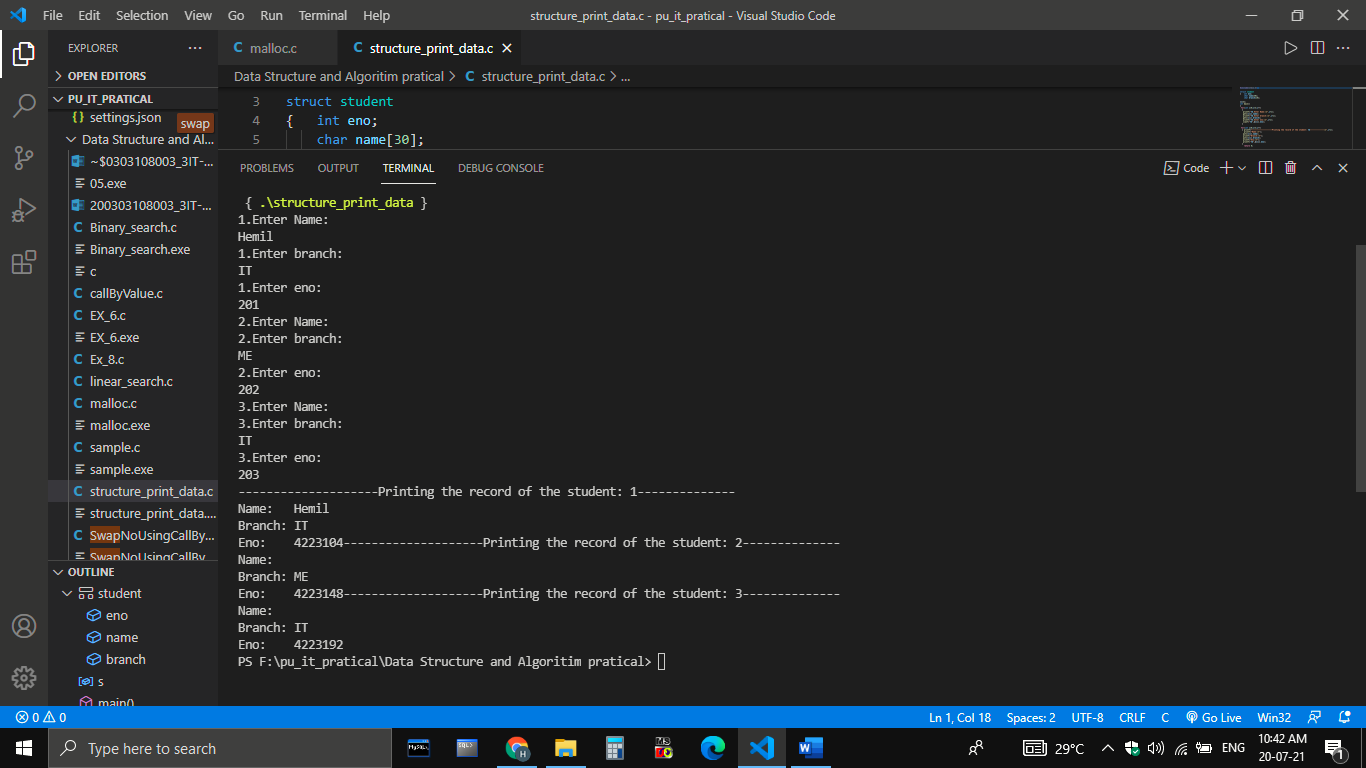
printf("%d",&s[i].eno);

}

return 0;

}

**Output:**



**Practical 4**

**AIM: Write a program to implement (a) linear Search (b) Binary Search**

**Code a)linear Search:**

#include<stdio.h>

#include<stdlib.h>

int main()

{

int i,a[10],n,key;

printf("Enter Size of array:");

scanf("%d",&n);

printf("Enter array:");

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

{

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

}

printf("Key Value:");

scanf("%d",&key);

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

{

if(a[i]==key)

{ printf("key value: %d at index is:%d",key,i);

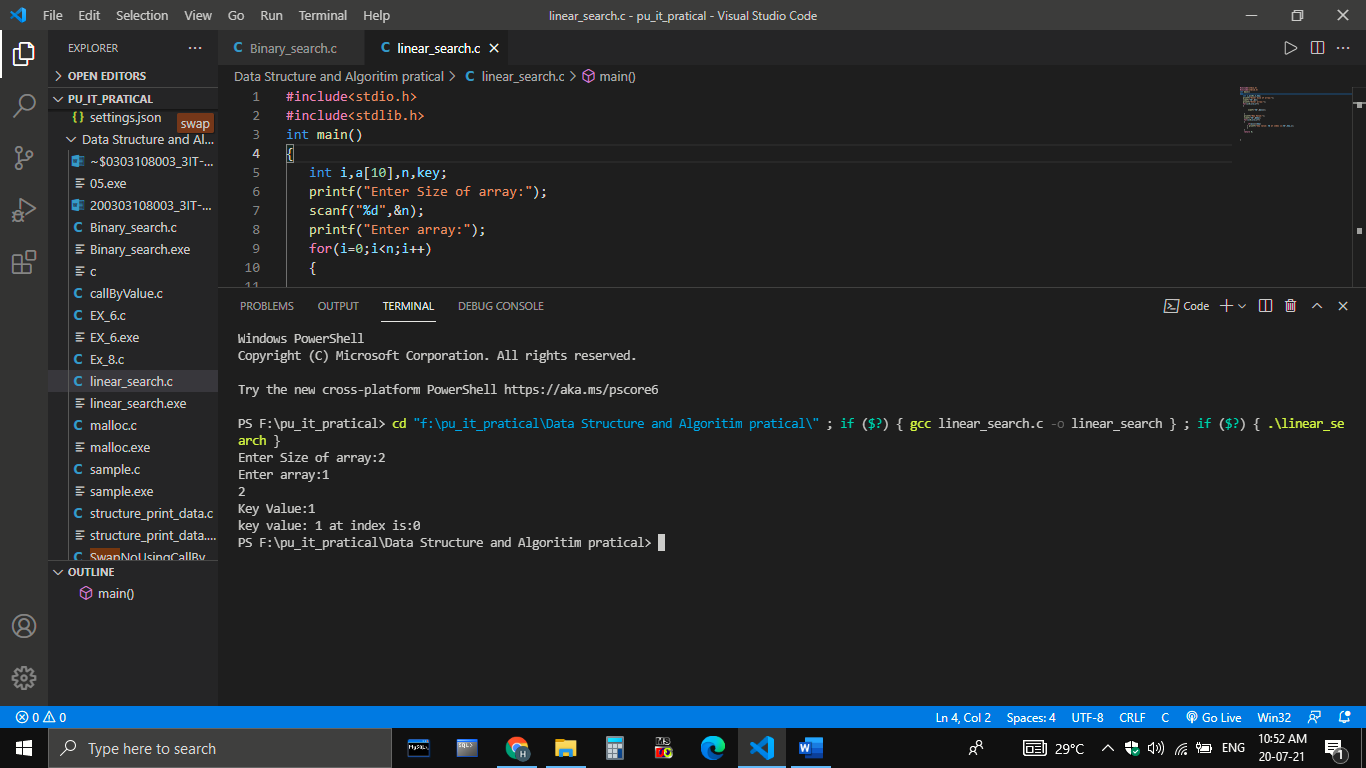
}

}

return 0;

}

**Output a:**



**Code b) Binary Search:**

#include<stdio.h>

int main()

{

int i,a[10],n,key;

printf("Enter Size of array:");

scanf("%d",&n);

printf("Enter array:");

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

{

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

}

printf("Enter Key Value:");

scanf("%d",&key);

int low=0,high=n,mid;

while (low <= high) {

mid=(low+high)/2;

if(a[mid]==key)

{

printf("Element %d found at location %d",key,mid);

break;

}

else if (a[mid]<key)

{

low=mid+1;

}

else if(key<a[mid]);

high=mid-1;

mid=(low+high)/2;

}

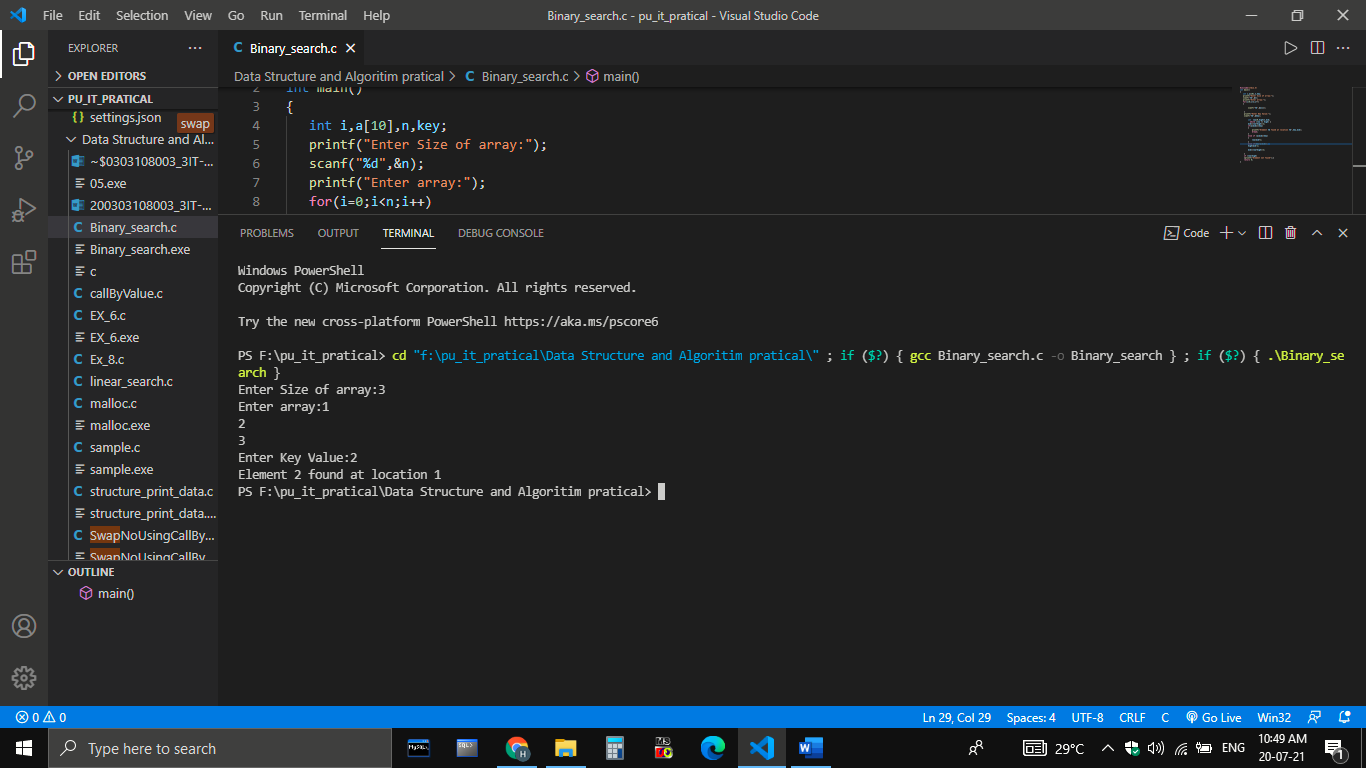
if (low>high)

{printf("Element not found");}

return 0;

}

**Output b:**



**Practical 6**

**AIM: Implement a program for stack that performs following operations using array. (a)PUSH (b) POP (c) PEEP (d) CHANGE (e) DISPLAY**

**Code:**

#include<stdio.h>

void push(int [],int \*,int , int );

void display(int [],int);

int pop(int [],int \*);

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

int peep(int [],int ,int );

int main()

{ int e,n=10,t=-1,s[10],c=0,i,in;

while(c!=6)

{ printf("1.Push\n2.Display\n3.Pop\n4.Change\n5.Peep\n6.Exit\nEnter Choice :");

scanf("%d",&c);

switch(c)

{

case 1: {

printf("Enter the Elements to push\n");

scanf("%d",&e);

push(s,&t,n,e);

break;

}

case 2: { printf("\nElements are : ");

display(s,t);

break;

}

case 3: {

e=pop(s,&t);

printf("%d is poped \n,e");

break;

}

case 4: {

printf("\nEnter Element e and index to change:");

scanf("%d%d",&e,&in);

change(s,t,in,e);

break;

}

case 5: {

printf("\nEnter Index to peep:\n");

scanf("%d",&in);

e=peep(s,t,in);

printf("\n%d is peeped\n",e);

break;

}

case 6: break;

default: printf("invalid choice,try again");

}

}

return 0;

}

void push(int s[],int \*t,int n,int e)

{

if(\*t>=n-1)

{

printf("\nStack is full");

}

else{

s[++\*t]=e;

}

}

void display(int s[],int t)

{

for(int i=0;i<=t;i++)

{

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

printf("\n");

}

}

int pop(int s[],int \*t)

{

if(\*t==-1)

{

return(-1);

}

else

{

printf("Removed element is %d:",s[\*t]);

\*t=\*t-1;

}

}

int peep(int s[],int t,int index)

{ if((t-index)<0)

{

printf("\nUnderflow");

return(-1);

}

else{

return(s[t-index]);

}

}

void change(int s[],int t,int index,int e)

{

if((t-index)<0)

{

printf("underflow");

}

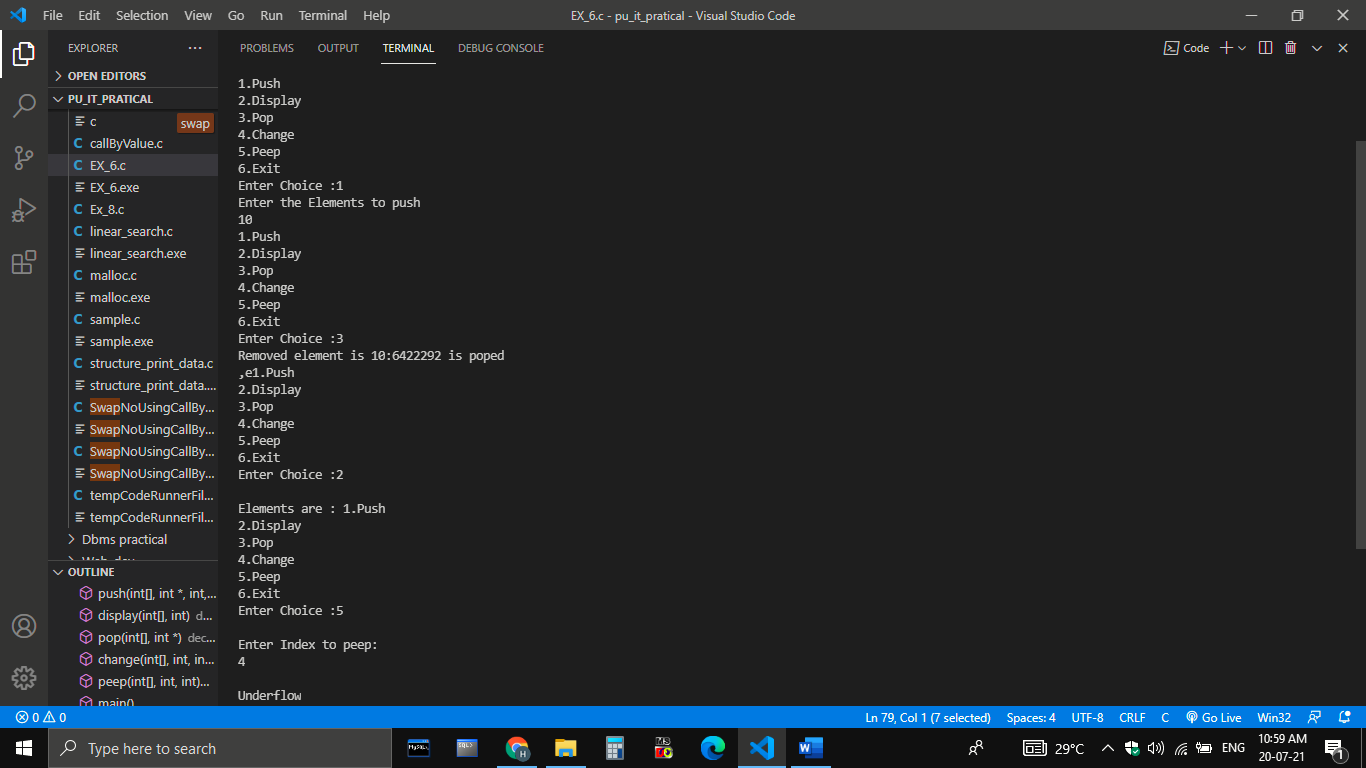
else{

s[t-index]=e;

}

}

**Output:**



**Practical 7**

**AIM :Implement a program to convert infix to postfix notation using stack.**

**Code:**

#include<stdio.h>

#include<ctype.h>

char stack[100];

int top = -1;

void push(char x)

{

stack[++top] = x;

}

char pop()

{

if(top == -1)

return -1;

else

return stack[top--];

}

int priority(char x)

{

if(x == '(')

return 0;

if(x == '+' || x == '-')

return 1;

if(x == '\*' || x == '/')

return 2;

return 0;

}

int main()

{

char exp[100];

char \*e, x;

printf("Enter the expression : ");

scanf("%s",exp);

printf("\n");

e = exp;

while(\*e != '\0')

{

if(isalnum(\*e))

printf("%c ",\*e);

else if(\*e == '(')

push(\*e);

else if(\*e == ')')

{

while((x = pop()) != '(')

printf("%c ", x);

}

else

{

while(priority(stack[top]) >= priority(\*e))

printf("%c ",pop());

push(\*e);

}

e++;

}

while(top != -1)

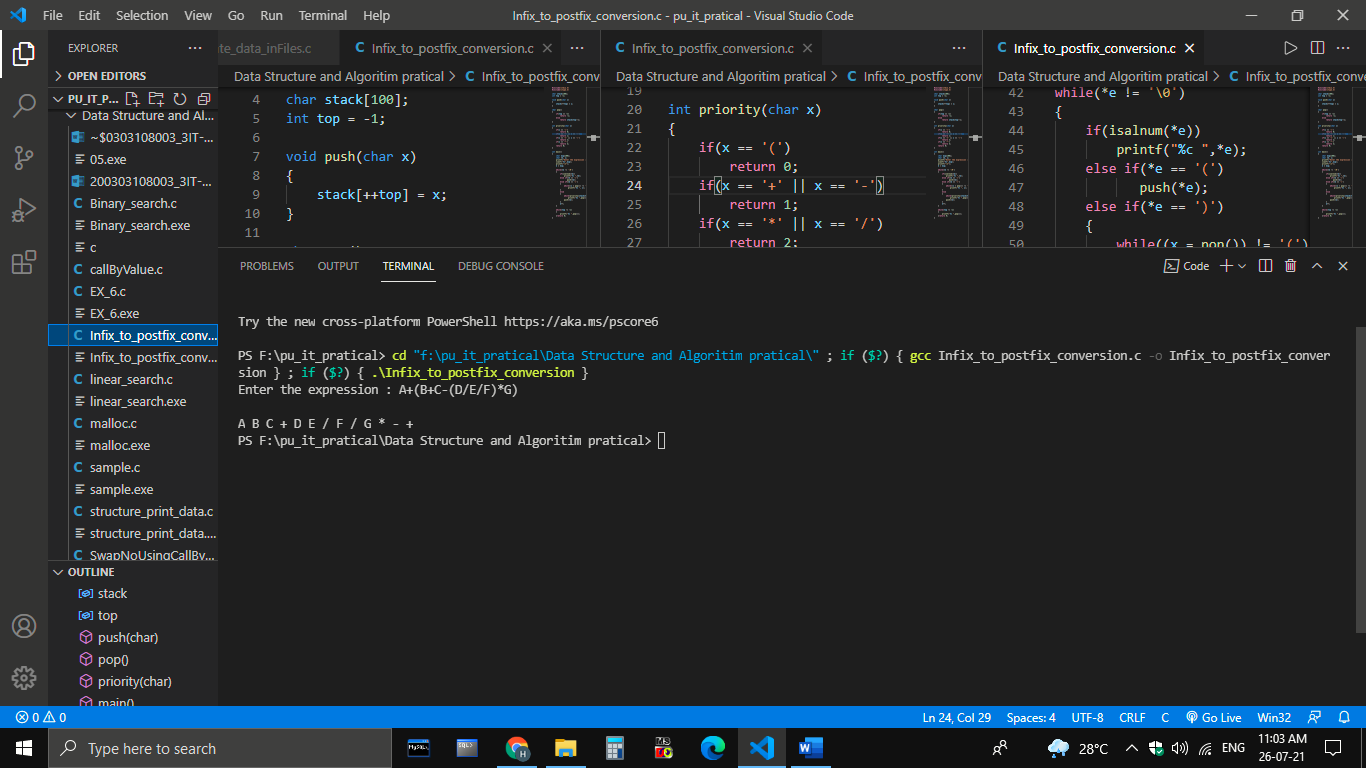
{

printf("%c ",pop());

}return 0;

}

**Output:**



**Practical 8**

**AIM: Implement a program to evaluate postfix notation**

**Code:**

#include<stdio.h> //standard input output functions

#include<conio.h> //console functions

#include<string.h> //string functions

#define MAX 50 //max size defined

int stack[MAX]; //a global stack

char post[MAX]; //a global postfix stack

int top=-1; //initializing top to -1

void pushstack(int tmp); //push function

void evaluate(char c); //calculate function

void main()

{

int i,l;

//clrscr();

printf("Insert a postfix notation :: ");

gets(post); //getting a postfix expression

l=strlen(post); //string length

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

{

if(post[i]>='0' && post[i]<='9')

{

pushstack(i); //if the element is a number push it

}

if(post[i]=='+' || post[i]=='-' || post[i]=='\*' ||

post[i]=='/' || post[i]=='^') //if element is an operator

{

evaluate(post[i]); //pass it to the evaluate

}

} //print the result from the top

printf("\n\nResult :: %d",stack[top]);

getch();

}

void pushstack(int tmp) //definiton for push

{

top++; //incrementing top

stack[top]=(int)(post[tmp]-48); //type casting the string to its integer value

}

void evaluate(char c) //evaluate function

{

int a,b,ans; //variables used

a=stack[top]; //a takes the value stored in the top

stack[top]='\0'; //make the stack top NULL as its a string

top--; //decrement top's value

b=stack[top]; //put the value at new top to b

stack[top]='\0'; //make it NULL

top--; //decrement top

switch(c) //check operator been passed to evaluate

{

case '+': //addition

ans=b+a;

break;

case '-': //subtraction

ans=b-a;

break;

case '\*': //multiplication

ans=b\*a;

break;

case '/': //division

ans=b/a;

break;

case '^': //power

ans=b^a;

break;

default:

ans=0; //else 0

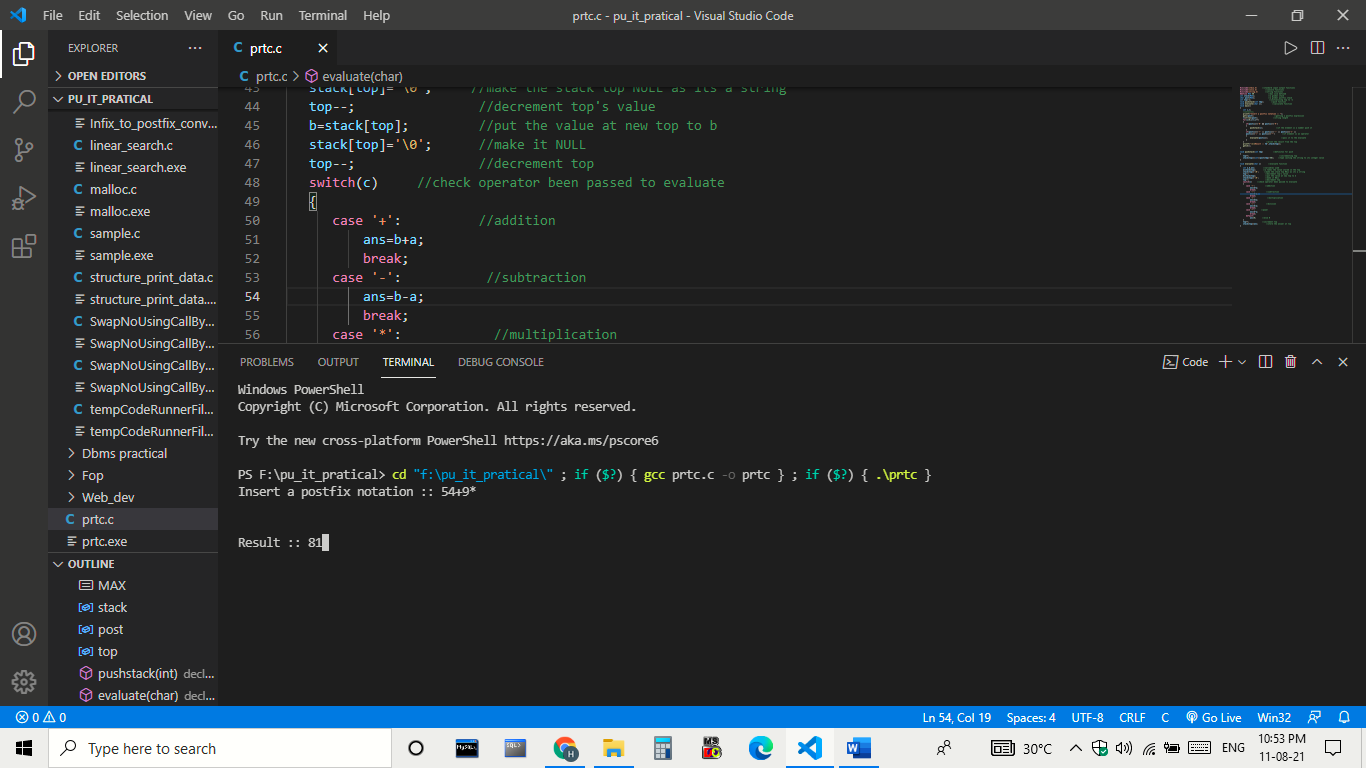
}

top++; //increment top

stack[top]=ans; //store the answer at top

}

**Output:**



**Practical 9**

**AIM: Write a program to implement QUEUE using arrays that performs following operations (a)INSERT (b) DELETE (c) DISPLAY**

**Code:**

#include<stdio.h>

#define n 5

int main()

{

int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;

printf("Queue using Array");

printf("\n1.Insertion(Enqueue) \n2.Deletion(Dequeue) \n3.Display \n4.Exit");

while(ch<4)

{

printf("\nEnter the Choice:");

scanf("%d",&ch);

switch(ch)

{

case 1:

if(rear==x)

printf("\n Queue is Full");

else

{

printf("\n Enter Enqueue Element %d:",j++);

scanf("%d",&queue[rear++]);

}

break;

case 2:

if(front==rear)

{

printf("\n Queue is empty");

}

else

{

printf("\n Deleted(Dequeue) Element is %d ",queue[front++]);

x++;

}

break;

case 3:

printf("\nQueue Elements are:\n ");

if(front==rear)

printf("\n Queue is Empty");

else

{

for(i=front; i<rear; i++)

{

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

printf("\n");

}

break;

default:

printf("Wrong Choice: please see the options");

}

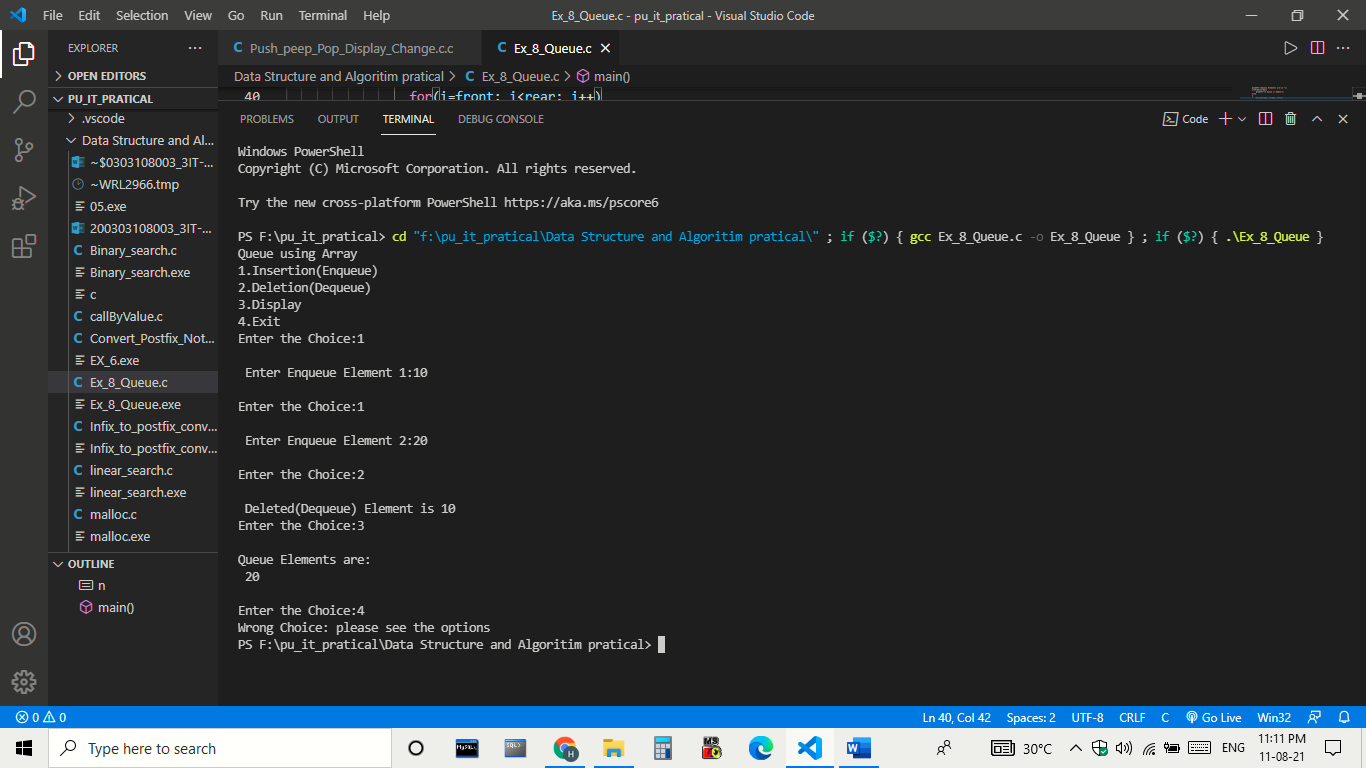
}

}

return 0;

}

**Output:**



**Practical 10**

**AIM:- Write a menu driven program to implement following operations on the singly linked list. (a) Insert a node at the front of the linked list. (b) Insert a node at the end of the linked list.**

**Code:-**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next; };

struct node \*head,\*newnode;

void insert\_first();

void insert\_last();

void display();

void main()

{ int c;

while(c!=4)

{ printf("\n1.insert\_first()\n2.insert\_last()\n3.display()\nEnter your Choice:");

scanf("%d",&c);

switch(c)

{

case 1: insert\_first();

break;

case 2: insert\_last();

break;

case 3: display();

break;

case 4: break;

default: printf(" Invalid Input");

} } }

void insert\_first()

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

if(newnode==NULL)

{

printf("\noverflow");

}

else{

printf("\nEnter data value to insert at first :");

scanf("%d",&newnode->data);

newnode->next=head;

head=newnode;

printf("\nNode Inserted");

} }

void insert\_last()

{ struct node \*newnode,\*temp;

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

if(newnode == NULL)

{ printf("\nOVERFLOW"); }

else

{ printf("\nEnter value to insert at last:");

scanf("%d",&newnode->data);

if(head == NULL)

{ newnode->next = NULL;

head = newnode;

printf("\nNode value inserted at last"); }

else {

temp = head;

printf("%d",\*temp);

while (temp->next != NULL)

{ temp = temp->next; }

temp->next = newnode;

newnode->next = NULL;

printf("\nNode value inserted at last");

} } }

void display()

{ struct node \*newnode;

newnode = head;

if(newnode == NULL)

{ printf("\nNothing to print"); }

else{

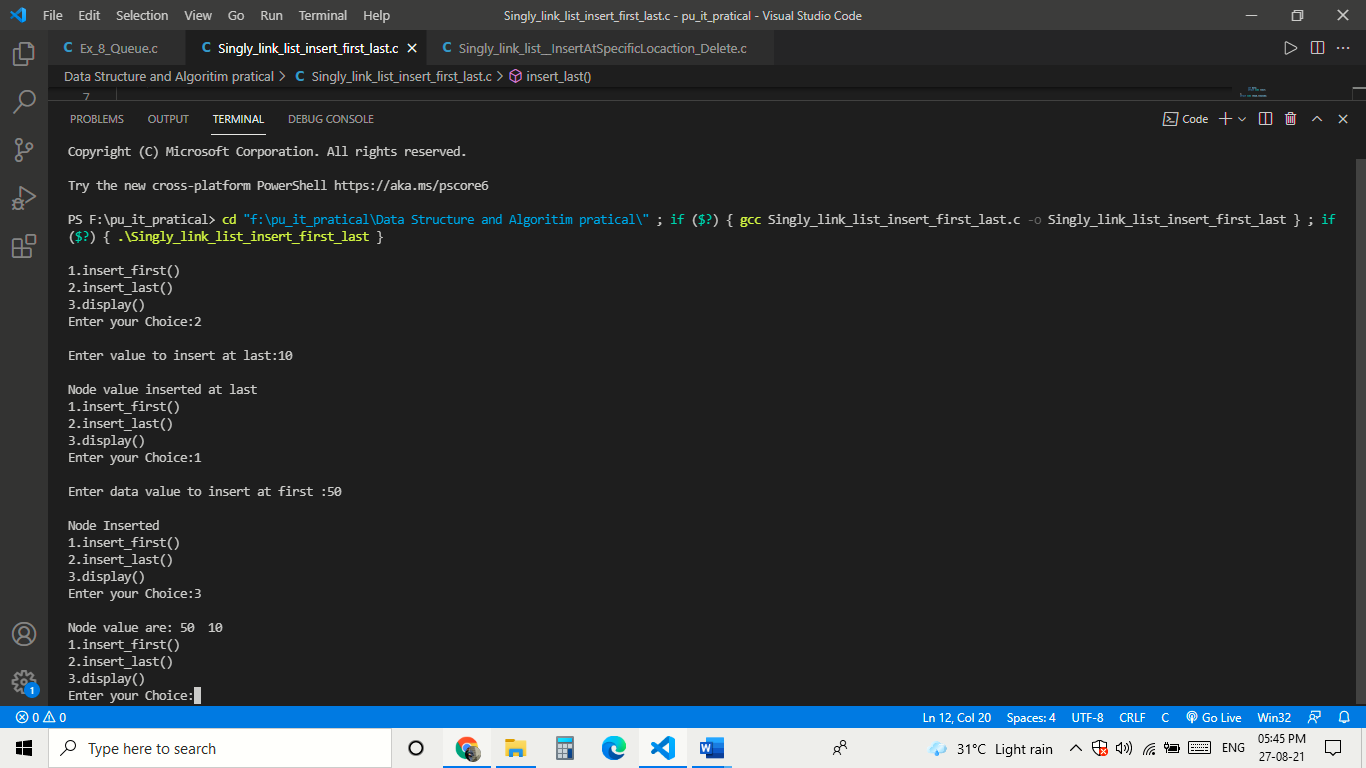
printf("\nNode value are:");

while (newnode!=NULL)

{ printf(" %d ",newnode->data);

newnode = newnode->next; } } }

**Output:**



**Practical-11**

**AIM:- Write a menu driven program to implement following operations on the singly linked list. (a) Insert a node at the specified position (b) Delete a first node of the linked list.**

**Code:-**

#include<stdio.h>

#include<stdlib.h>

struct node{

int data;

struct node \*next; };

struct node \*newnode, \*head;

void insert\_at\_specific\_loc();

void delete\_first();

void display();

void insert\_last();

void main()

{ int c;

while(c!=5)

{ printf("\n1.Insert node at specific location\n2.Delete from first\n3.Display\n4.insert\_last()\n5.Exit\nEnter your Choice:");

scanf("%d",&c);

switch(c)

{ case 1: insert\_at\_specific\_loc();

break;

case 2: delete\_first();

break;

case 3:display();

break;

case 4: insert\_last();

break;

case 5: break;

default:printf("\nInvalid Input"); } } }

void insert\_at\_specific\_loc()

{ int val,key;

printf("Enter the number after which you want to add newnode:-");

scanf("%d",&key);

struct node \*newnode,\*ptr,\*preptr;

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

if(newnode==NULL)

{ printf("no node is created"); }

printf("\nEnter value to be insert after a position :-\n");

scanf("%d",&val);

newnode->data=val;

if (head==NULL)

{ printf("o linked list exist so adding at first");

newnode->next=NULL;

head=newnode; }

else

{ ptr=head;

preptr=head;

ptr=ptr->next;

while(preptr->data!=key)

{ ptr=ptr->next;

preptr=preptr->next; }

newnode->next=preptr->next;

preptr->next=newnode;

} }

void insert\_last()

{ struct node \*newnode,\*temp;

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

if(newnode == NULL)

{ printf("\nOVERFLOW"); }

else

{ printf("\nEnter value to insert at last:");

scanf("%d",&newnode->data);

if(head == NULL)

{ newnode->next = NULL;

head = newnode;

printf("\nNode value inserted at last"); }

else

{ temp = head;

printf("%d",\*temp);

while (temp->next != NULL)

{ temp = temp->next; }

temp->next = newnode;

newnode->next = NULL;

printf("\nNode value inserted at last"); } } }

void delete\_first()

{ struct node \*ptr;

if(head==NULL)

{ printf("List is Empty:\n"); }

else

{ ptr=head;

head=head->next ;

printf("The deleted element is :%d",ptr->data);

free(ptr);

} }

void display()

{ struct node \*newnode;

newnode = head;

if(newnode == NULL)

{ printf("\nNothing to print"); }

else

{ printf("\nNode value are:");

while (newnode!=NULL)

{ printf(" %d ",newnode->data);

newnode = newnode->next;

} } }

**Output:**

