CS6104

DATA STRUCTURES

AND

ALGORITHMS

PROGRAMS

|  |
| --- |
| **NAME: BABITH SARISH S**  **REG.NO: 2021503009**  **DEPARTMENT: DEPARTMENT OF**  **COMPUTER TECHNOLOGY** |

**MULTIDIMENSIONAL ARRAY**

**To create a matrix and do necessary Operations(Insert,deleting diagonal elements,Search) .**

#include<stdio.h>

int main(void){

int arr[3][3],d,i,n,j,k,flag;

printf("Enter the value of n: ");

scanf("%d",&n);

//Insertion

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

for( j=0;j<n;j++){

printf("Enter element %d%d: ",i+1,j+1);

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

}

}

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

for(j=0;j<n;j++){

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

if(j==n-1)

printf("\n");

}

}

printf("Press 1 to delete the main diagonal elements... ");

scanf("%d",&d);

//Deletion

if(d==1){

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

for( j=0;j<n;j++){

if(i==j){

for( k=j;k<n-1;k++){

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

}

arr[i][k]='\0';

}

}

}

printf("After deletion...\n\n");

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

for(j=0;j<n-1;j++){

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

}

printf("\n");

}

}

//search

int num;flag=0;

printf("Enter the number to be searched: ");

scanf("%d",&num);

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

for(j=0;j<n-1;j++){

if(arr[i][j]==num){

flag=1;

break;

}

}}

if(flag==1)

printf("Element found!!\n");

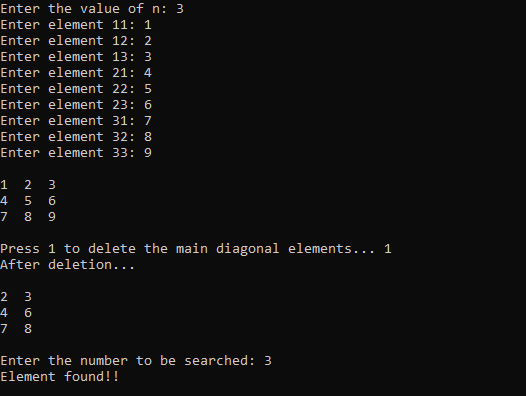
else

printf("Element not found!!\n");

return 0;

}

**OUTPUT:**

****

**To do matrix addition Operation using Multi-dimensional Array.**

**SOURCE CODE:**

#include<stdio.h>

int main(void){

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

printf("Enter the value of n: ");

scanf("%d",&n);

printf("Enter matrix A:\n");

//getting input

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

for( j=0;j<n;j++){

printf("a%d%d: ",i+1,j+1);

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

}

}

printf("Enter matrix B:\n");

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

for( j=0;j<n;j++){

printf("b%d%d: ",i+1,j+1);

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

}

}

//adding

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

for( j=0;j<n;j++){

c[i][j]=a[i][j]+b[i][j];

}

}

//printing

printf("Matrix Addition is:\n");

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

for( j=0;j<n;j++){

printf("%d ",c[i][j]);

if(j==n-1)

printf("\n");

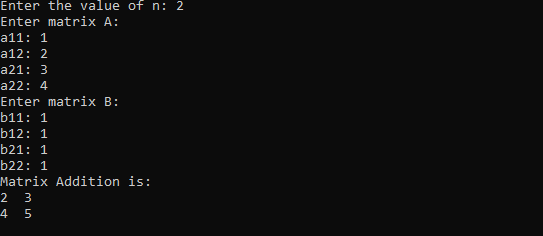
}

}

return 0;

}

**OUTPUT:**

****

**To do the matrix Multiplication Operation using Multi-dimensional Array.**

#include <stdio.h>

int main(){

int i, j, k, arr1[15][15], arr2[15][15], arr3[15][15], n, sum=0;

printf("Enter the value n:");

scanf("%d", &n);

printf("Enter array 1:\n");

//getting input

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

for(j=0; j<n; j++){

printf("Enter arr1[%d][%d]:", i+1, j+1);

scanf("%d", &arr1[i][j]);

}

}

printf("Enter array 2:\n");

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

for(j=0; j<n; j++){

printf("Enter arr2[%d][%d]:", i+1, j+1);

scanf("%d", &arr2[i][j]);

}

}

//calculating multiplication and printing

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

for(j=0; j<n; j++){

for(k=0; k<n; k++){

sum += ( arr1[i][j] \* arr2[j][i] );

}

arr3[i][j] = sum;

sum=0;

printf("%d ", arr3[i][j]);

}

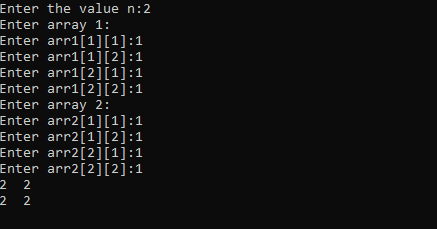
printf("\n");

}

return 0;

}

**OUTPUT:**

****

**To do the matrix Transpose Operation using Multi-dimensional Array.**

#include<stdio.h>

int main(void){

int a[3][3],b[3][3],i,j;

//getting

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

for( j=0;j<3;j++){

printf("Enter a%d%d: ",i+1,j+1);

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

}

}

printf("The given matrix is:\n");

//printing given matrix

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

for( j=0;j<3;j++){

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

if(j==2)

printf("\n");

}

}

//processing transpose of given matrix

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

for(j=0;j<3;j++){

b[i][j]=a[j][i];

}

}

//printing transpose of given matrix

printf("Transpose of the given matrix is:\n");

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

for( j=0;j<3;j++){

printf("%d ",b[i][j]);

if(j==2)

printf("\n");

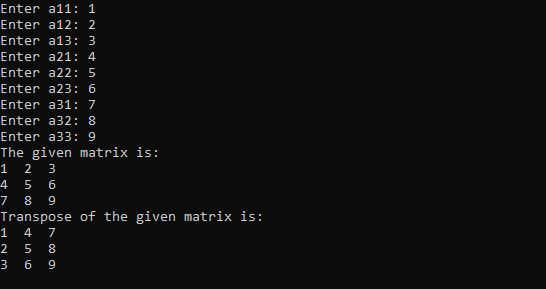
}

}

return 0;

}

**OUTPUT:**

****

**To create a Student record with Register Number , marks and Average using Multi-dimensional Array.**

#include<stdio.h>

int main(void){

long int arr[6][6],i,j,n;float avg;

printf("Enter the no of students: ");

scanf("%d",&n);

//getting input

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

for( j=0;j<6;j++){

if(j==0){

printf("Enter the Reg.no of student %d:",i+1);

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

}

else{

printf("Enter mark %d:",j);

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

}

}

}

//calculating average and printing

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

avg=0;

for( j=0;j<6;j++){

if(j>0)

avg+=arr[i][j];

}

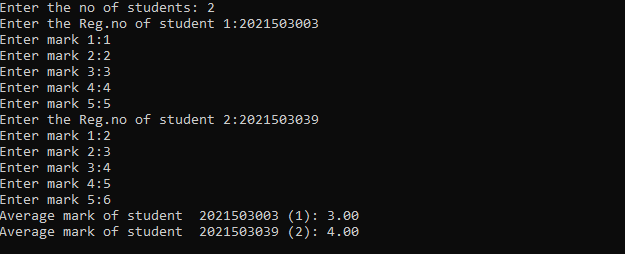
printf("Average mark of student %ld (%ld): %.2f\n",arr[i][0],i+1,avg/5);

}

return 0;

}

**OUTPUT:**



**To create a Student record with Register Number , Phone , CGPA and display using Multi-dimensional Array.**

**SOURCE CODE:**

#include <stdio.h>

int main(){

double student[20][5];

int n, i, j;

printf("Enter No of Students:");

scanf("%d",&n);

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

printf("Enter Reg No: ");

scanf("%lf", &student[i][0]);

printf("Enter CGPA: ");

scanf("%lf", &student[i][1]);

printf("Enter Phone Number: ");

scanf("%lf", &student[i][2]);

}

printf(" Reg No\t\tPhone Number\tCGPA\n");

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

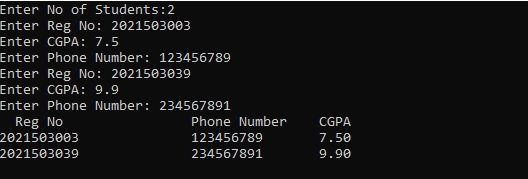
printf("%ld\t\t%ld\t%.2f\n",(long int)student[i][0], (long int)student[i][2],student[i][1]);

}

return 0;

}

**OUTPUT:**



STACK IMPLEMENTATION USING ARRAY

#include<stdio.h>

#define max 5

int top=0,n=0,i;

int stack[5];

int pop();

int display();

int push(int);

int main()

{

int choice,x=1;

while(x)

{

printf(" 1: push\t\t 2: pop\t\t 3: display\t\t 4: exit\n");

printf("Enter your choice:\t");

scanf("%d", &choice);

switch(choice)

{

case 1:

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

scanf("%d", &n);

push(n);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

x=0;

}

}

}

int push(var)

{

if(top>=max-1)

{

printf("The stack is full\n");

}

else

{

stack[top]=var;

top++;

}

}

int pop()

{

if(top==0)

{

printf("The stack is empty\n");

}

else

{

printf("The poped element is %d\t:", stack[top]);

stack[top]='\0';

top--;

printf("\n");

}

}

int display()

{

if(top==0)

{

printf("The stack is empty\n");

}

else

{

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

{

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

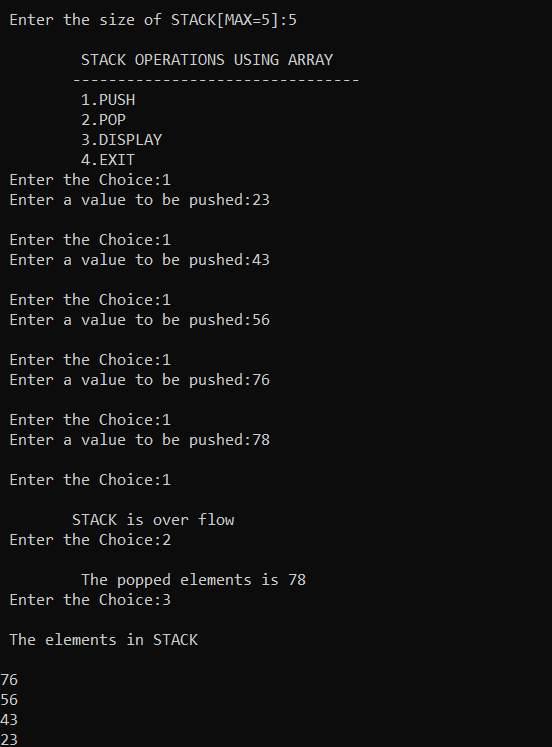
}

printf("\n");

}

}

**OUTPUT:**

****

QUEUE IMPLEMENTATION USING ARRAY

#include<stdio.h>

#define max 5

int back=0,n=0,i, front=0;

int stack[5];

int pop();

int display();

int push(int);

int main()

{

int choice,x=1;

while(x)

{

printf(" 1: push\t\t 2: pop\t\t 3: display\t\t 4: exit\n");

printf("Enter your choice:\t");

scanf("%d", &choice);

switch(choice)

{

case 1:

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

scanf("%d", &n);

push(n);

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

x=0;

}

}

}

int push(var)

{

if(front<max)

{

stack[front]=var;

front++;

printf("%d is pushed\n",var);

}

else

{

printf("The stack is full\n");

}

}

int pop()

{

if(back<front)

{

printf("%d is popped\n", stack[back]);

back+=1;

}

else

{

printf("The stack is empty\n");

}

}

int display()

{

if(back<front)

{

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

{

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

}

printf("\n");

}

else

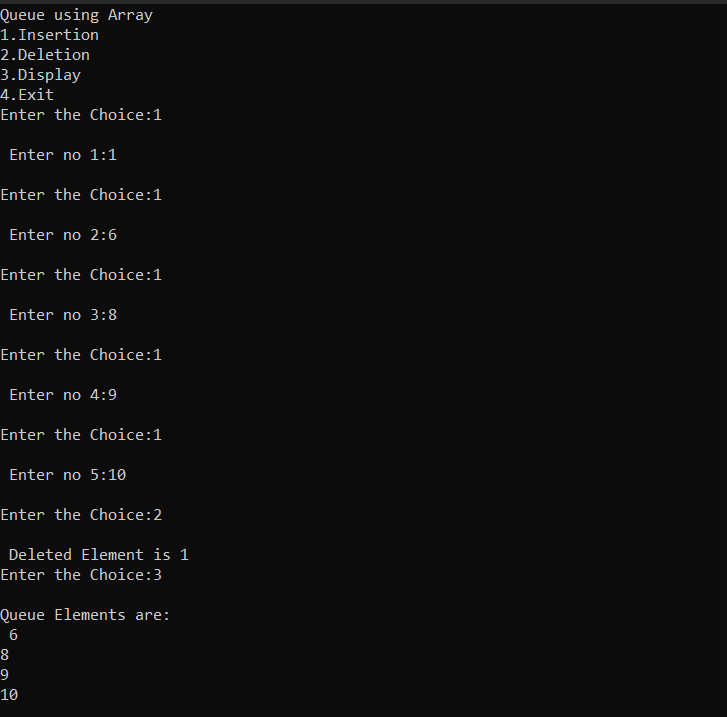
{

printf("The stack is empty\n");

}

}

**OUTPUT:**

****

IMPLEMENTATION OF SINGLY LINKED LIST

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* next;

}\*current, \*head=NULL;

void insertb(int);

void insertf(int);

void deletef();

void deleteb();

void afternum(int);

void beforenum(int);

void display();

void search(int);

int main()

{

int num,num2,num3,num4,num5, value, choice;

printf("1-insert front\t\t 2-insert back\t\t 3-delete front\t\t 4-delete back\t\t 5-afternum\t\t 6-beforenum\t\t 7-search\t\t 8-display\n");

while(choice!=9){

printf("\nEnter your operation:\n");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the num to be inserted: \n");

scanf("%d: ", &num2);

insertf(num2);

display();

break;

case 2:

printf("Enter the num to be inserted: \n");

scanf("%d: ", &num);

insertb(num);

display();

break;

case 3:

deletef();

display();

break;

case 4:

deleteb();

display();

break;

case 5:

printf("Enter the num to be inserted: \n");

scanf("%d: ", &num3);

afternum(num3);

display();

break;

case 6:

printf("Enter the num to be inserted: \n");

scanf("%d: ", &num4);

beforenum(num4);

display();

break;

case 7:

printf("Enter the num to be searched: \n");

scanf("%d: ", &num5);

search(num5);

break;

case 8:

display();

break;

case 9:

printf("Exit");

}

}

}

void insertb(int num)

{

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

Newnode->data = num;

Newnode->next = '\0';

if(head =='\0')

{

head = Newnode;

}

else

{

current = head;

while

(current->next!='\0')

{

current = current->next;

}

current->next = Newnode;

}

}

void insertf( int num2)

{

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

newnode->data=num2;

newnode->next=head;

head=newnode;

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

}

void deletef()

{

if(head=='\0')

{

printf("NO ELEMENTS");

}

else

{

current = head;

head = head->next;

free(current);

}

}

void deleteb()

{

if(head=='\0')

{

printf("NO ELEMENTS");

}

else

{

struct node\* neq;

current=head;

while(current->next!='\0')

{

neq = current;

current = current->next;

}

if(current!=head)

{

neq->next = '\0';

}

else

{

head='\0';

}

free(current);

}

}

void afternum(int num3)

{

int value;

struct node\* Newnode1 = (struct node\*)malloc(sizeof(struct node));

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

scanf("%d", &value);

current=head;

Newnode1->data = num3;

while(current->next!='\0')

{

if(current->data==value)

{

Newnode1->next=current->next;

current->next=Newnode1;

break;

}

current=current->next;

}

if(current->data!=value)

{

printf("no element exist");

}

}

void beforenum(int num4)

{

int value;

struct node\* Newnode1 = (struct node\*)malloc(sizeof(struct node));

struct node\* pre;

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

scanf("%d", &value);

current=head;

pre='\0';

Newnode1->data = num4;

while(current->next!='\0')

{

if(current->data==value)

{

Newnode1->next=pre->next;

pre->next=Newnode1;

break;

}

pre=current;

current=current->next;

}

if(current->data!=value)

{

printf("no element exist");

}

}

void search(int num5)

{

current=head;

int flag=0;

while(current->next!='\0')

{

if(current->data==num5)

{

printf("Element found");

flag=1;

}

current=current->next;

}

if(flag==0)

{

printf("Element not found");

}

}

void display()

{

printf("The inserted number is: ");

current = head;

if(head =='\0')

{

printf("NO ELEMENTS");

}

else

{

while(current->next !='\0')

{

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

current = current->next;

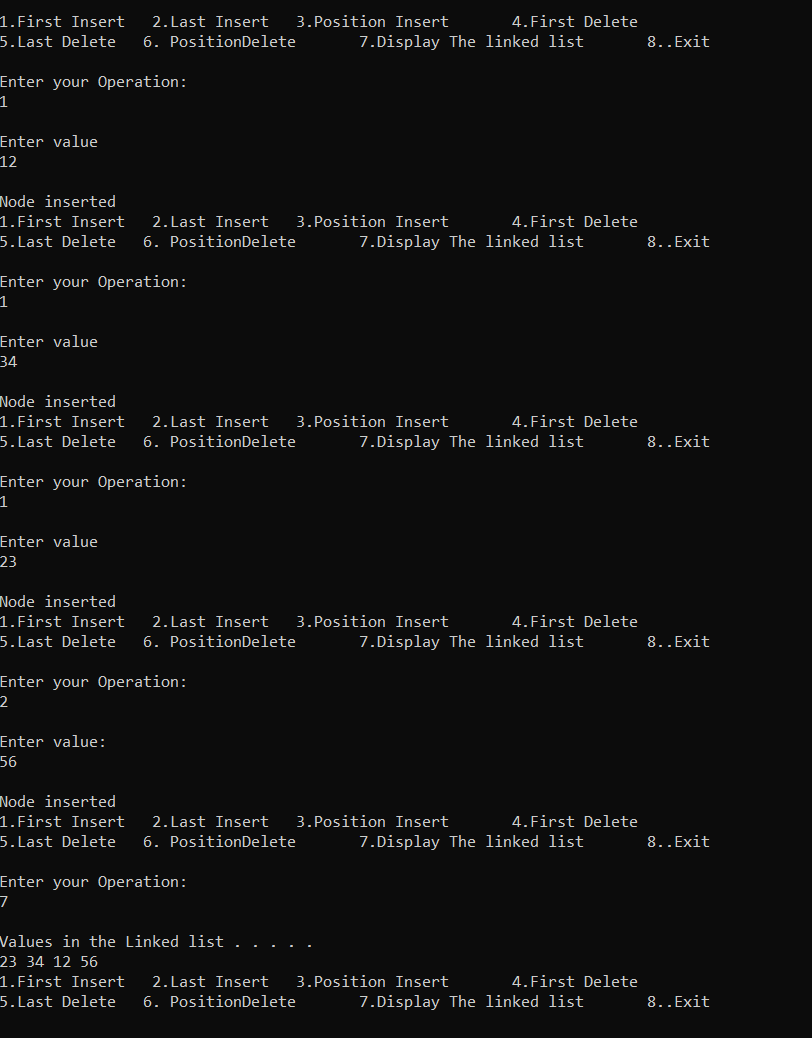
}

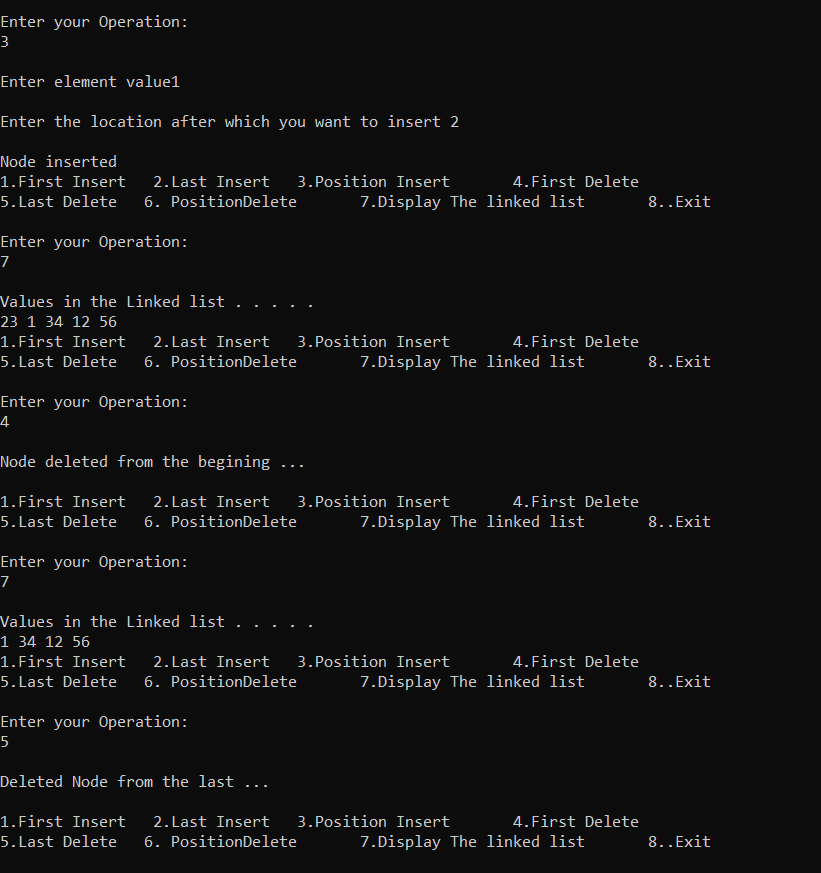
printf("%d",current->data);

}

}

**OUTPUT:**

****

****

STACK IMPLEMENTATION USING LINKED LIST

#include<stdio.h>

#include<stdlib.h>

struct node

{

int data;

struct node\* next;

}\*top;

void insert(int);

void display();

void del();

void search();

int main()

{

int num, ch,n ;

printf("1-insert\t\t 2-delete\t\t 3-search\t\t 4-display\n");

while(ch!=5)

{

printf("\nEnter your choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

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

scanf("%d", &num);

insert(num);

display();

break;

case 2:

del();

break;

case 3:

search();

break;

case 4:

display();

break;

}

}

}

void insert( int num)

{

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

if(Newnode=='\0')

{

printf("unable to push\n");

}

else

{

if(top=='\0')

{

Newnode->data=num;

Newnode->next='\0';

top=Newnode;

printf("Inserted\n");

}

else

{

Newnode->data=num;

Newnode->next=top;

top=Newnode;

printf("Inserted\n");

}

}

}

void display()

{

struct node\* temp;

temp=top;

if(top=='\0')

{

printf("No elements\n");

}

else

{

printf("\nThe elements are: ");

while(temp!='\0')

{

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

temp=temp->next;

}

printf("\n");

}

}

void search()

{

struct node\*ptr;

int item,i=0,flag;

ptr=top;

if(ptr==NULL)

{

printf("\nstack is empty");

}

else

{

printf("\nEnter the element to search:");

scanf("%d",&item);

while(ptr!=NULL)

{

if(ptr->data==item)

{

printf("\nElement found at %d position",i+1);

flag=0;

break;

}

else

{

flag=1;

}

ptr=ptr->next;

i++;

}

if(flag==1)

{

printf("\nElement not found");

}

}

}

void del()

{

int f=0;

struct node\* temp;

if(top=='\0')

{

printf("No elements to be deleted\n");

}

else

{

temp=top;

top=temp->next;

free(temp);

f=1;

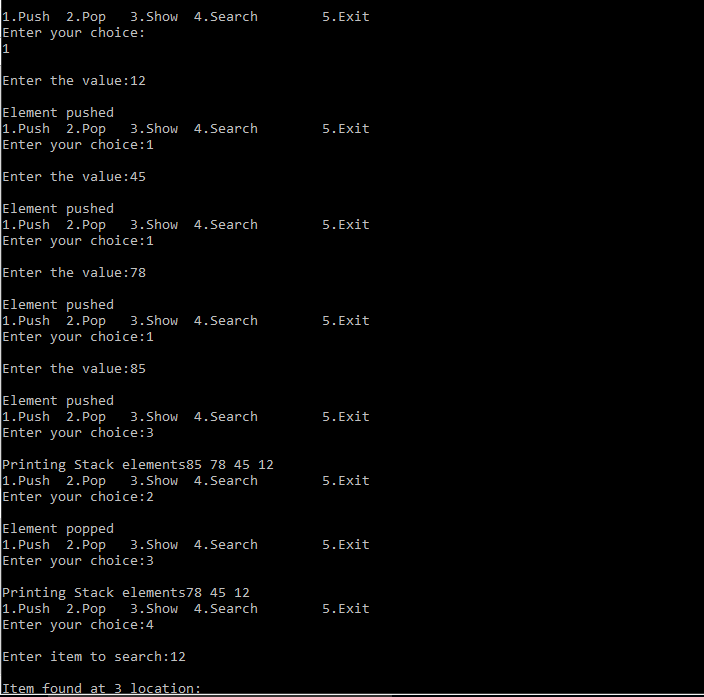
}

if(f==1)

printf("Deleted\n");

}

**OUTPUT:**

****

STACK IMPLEMENTATION USING QUEUE

#include<stdio.h>

#define N 5

void push();

void pop();

void display();

void send();

void receive();

void search();

int s1[N],s2[N],top1=-1,top2=-1,rear1=-1,rear2=-1;

int main(){

int ch;

printf("1 - Push\t\t 2 - Pop\t\t 3 - Display\t\t 4 - Search\n");

while(4){

printf("Enter your choice: ");

scanf("%d",&ch);

switch(ch){

case 1:

push();

break;

case 2:

pop();

break;

case 3:

display();

break;

case 4:

search();

break;

default:

printf("Invalid Choice \n");

}

}

}

void push(){

if(rear1==N-1)

printf("Stack is full \n");

else{

int data;

printf("Enter the data: ");

scanf("%d",&data);

if(top1==-1){

s1[++top1]=data;

rear1++;

}

else{

send();

s1[++top1]=data;

rear1++;

receive();

}

display();

}

}

void send(){

while(top1<=rear1){

if(top2==-1){

s2[++top2]=s1[top1++];

rear2++;

}

else{

s2[++rear2]=s1[top1++];

}

}

top1=-1;rear1=-1;

}

void receive(){

while(top2<=rear2){

s1[++rear1]=s2[top2++];

}

top2=-1;

rear2=-1;

}

void pop(){

if(top1==-1)

printf("Stack is Empty \n");

else if(top1==rear1){

printf("%d is removed from the Stack \n",s1[top1]);

top1=-1;rear1=-1;

display();

}

else{

printf("%d is removed from the Stack \n",s1[top1]);

int i;

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

s1[i]=s1[i+1];

rear1--;

display();

}

}

void display(){

if(top1==-1)

printf("Stack is Empty \n");

else{

int i;

for(i=0;i<=rear1;i++)

printf("%d ",s1[i]);

printf("\n");

}

}

void search(){

if(top1==-1)

printf("Stack is Empty \n");

else{

int i,flag=0,data;

printf("Enter the Value: ");

scanf("%d",&data);

for(i=0;i<=rear1;i++){

if(data==s1[i]){

flag=1;

break;}

}

if(flag==1)

printf("Element Found \n");

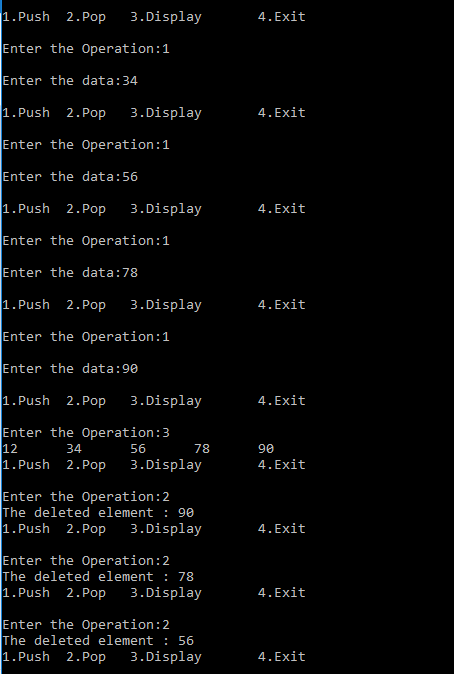
else

printf("Element not Found \n");

}

}

**OUTUT:**



QUEUE IMPLEMENTATION USING STACK

#include<stdio.h>

#include<stdlib.h>

/\*struct node

{

int data;

struct node\* next;

}top1=0, top2=0;\*/

#define MAX 5

void enqueue(int);

void dequeue();

void display();

int push1(int);

int push2(int);

int pop1();

int pop2();

int stack1[MAX],stack2[MAX],top1=-1, top2=-1,count=0,flag=0, i=0,var=0,rear;

int main()

{

int choice, num;

printf("1-enqueue\t\t 2-dequeue\t\t 3-display\t\t \n");

while(1)

{

printf("\nEnter the command: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

printf("Enter the num to insert: ");

scanf("%d",&num);

enqueue(num);

break;

case 2:

dequeue();

break;

case 3:

display();

break;

}

}

}

void display()

{

if(top1==-1)

{

printf("Stack is empty\n");

}

for(i=0;i<=top1;i++)

{

printf("%d ", stack1[i]);

}

}

void enqueue(num)

{

if(top1==MAX-1)

{

printf("Stack is full\n");

}

else

{

push1(num);

count++;

}

display();

}

int push1(num)

{

top1++;

stack1[top1]=num;

}

int pop1()

{

return(stack1[top1--]);

}

int push2(num)

{

top2++;

stack2[top2]=num;

}

int pop2()

{

return(stack2[top2--]);

}

void dequeue()

{

if(top1==-1)

{

printf("Stack is empty\n");

}

else

{

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

{

push2(pop1());

}

printf(" Deleted element: %d", stack2[top2]);

pop2();

count--;

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

{

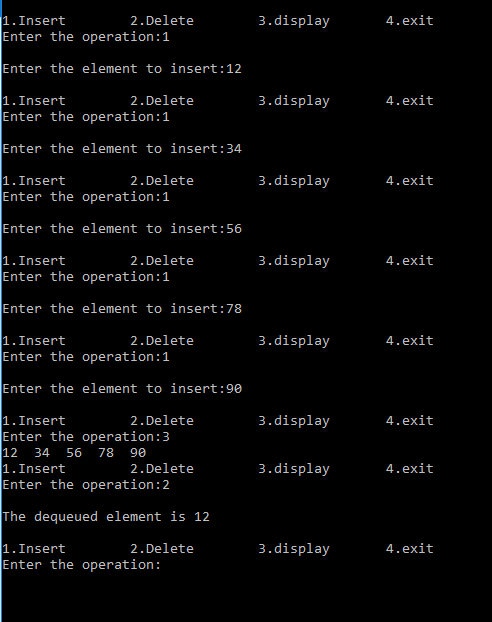
push1(pop2());

}

}

}

**OUTPUT:**

****

IMPLEMENTATION OF BINARY SEARCH TREE AND TRAVERSALS

#include<stdio.h>

#include<stdlib.h>

int num;

struct node

{

int data;

struct node \*left, \*right;

};

struct node\* Newnode(int data)

{

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

temp->right=NULL;

temp->left=NULL;

temp->data=num;

}

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

{

if(node==NULL)

return Newnode(data);

if(data < node->data)

node->left=insert(node->left, data);

else

node->right=insert(node->right, data);

}

struct node\*minvalue(struct node\* node)

{

struct node\* current = node;

while(current && current->data != NULL)

current = current->left;

return current;

}

struct node\* Deletenode(struct node\* root, int data)

{

if(root==NULL)

return root;

if(data < root->data )

root->left=Deletenode(root->left, data);

if(data > root->data )

root->right=Deletenode(root->right, data);

else

{

if (root->left == NULL)

{

struct node \*temp = root->right;

free(root);

return temp;

}

else if(root->right==NULL)

{

struct node \*temp = root->left;

free(root);

return temp;

}

struct node\* temp = minvalue(root->right);

root->data=temp=data;

root->right=Deletenode(root->right, temp->data);

}

return root;

}

void inorder(struct node \*root)

{

if (root != NULL)

{

inorder(root->left);

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

inorder(root->right);

}

}

void preorder(struct node \*root)

{

if (root != NULL)

{

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

preorder(root->left);

preorder(root->right);

}

}

void postorder(struct node \*root)

{

if (root != NULL)

{

postorder(root->left);

postorder(root->right);

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

}

}

int main()

{

int choice;

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

root=NULL;

printf("1-insert\t\t 2-delete\t\t 3-inorder\t\t 4-preorder\t\t 5-postorder\n");

while(1)

{

printf("\nEnter the operation: ");

scanf("%d", &choice);

switch(choice)

{

case 1:

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

scanf("%d", &num);

root=insert(root,num);

break;

case 2:

printf("Enter the number to be deleted: ");

scanf("%d", &num);

root=Deletenode(root,num);

break;

case 3:

printf("INORDER TRAVERSAL: ");

inorder(root);

break;

case 4:

printf("PREORDER TRAVERSAL: ");

preorder(root);

break;

case 5:

printf("POSTORDER TRAVERSAL: ");

postorder(root);

break;

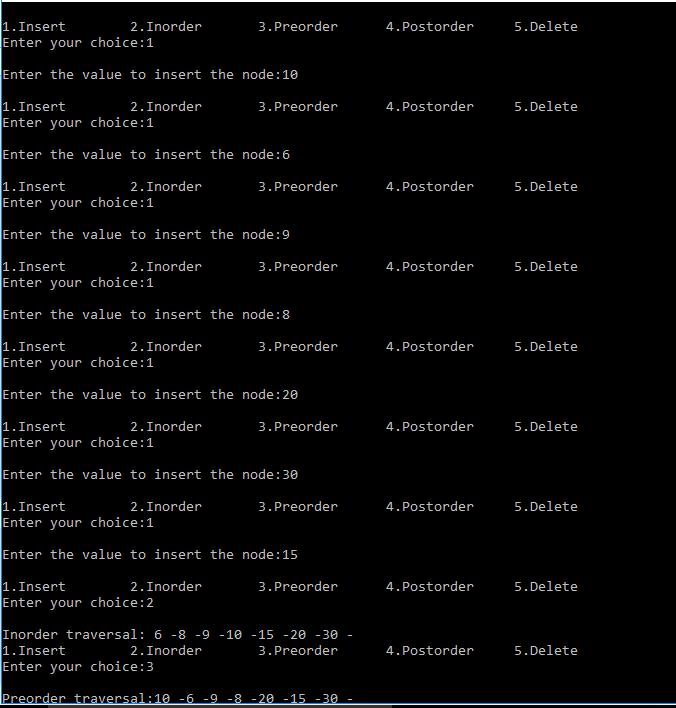
default:

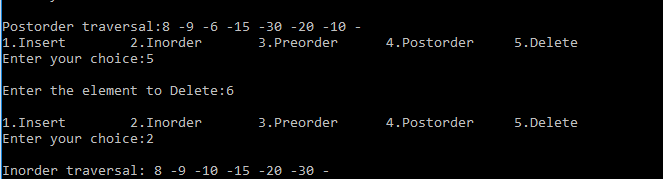
break;

}

}

}

**OUTPUT:**



CONVERSION OF INFIX TO POSTFIX

#include<stdio.h>

#include<ctype.h>

int top=-1;

char stack[100];

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 value[20];

char \*ch, var;

printf("Enter the expression: ");

scanf("%s", value);

ch=value;

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

{

if(isalnum(\*ch))

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

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

push(\*ch);

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

{

while((var=pop())!='(')

printf("%c", var);

}

else

{

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

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

push(\*ch);

}

ch++;

}

while(top!=-1)

{

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

}

return 0;

}

}

**OUTPUT:**



COMBINE TWO LISTS AND SORT WITHOUT DUPLICATES

#include<stdio.h>

int main()

{

int choice,arr1[5], arr2[5], new[10], duplicate[10],sum=0,avg=0,demo=0, a[10], b[5], c[5],i,j,k=0, var=0,condition=0,count=0, flag=0, ch=0;

//input the values from the user

printf("Enter the elements in the first list: \n");

for(i=0;i<5;i++)

{

scanf("%d",&arr1[i]);

}

printf("Enter the elements in the second list: \n");

for(i=0;i<5;i++)

{

scanf("%d",&arr2[i]);

}

//concordinate the two lists

printf("\nconcordinated list with duplicates: \n");

for(i=0;i<5;i++)

{

new[i]=arr1[i];

}

for(i=0;i<5;i++)

{

new[i+5]=arr2[i];

}

//displaying the concordinated list

for(i=0;i<10;i++)

{

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

}

//duplicating the concordinated list

for(i=0;i<10;i++)

{

duplicate[i]=new[i];

}

//sorting

for(i=0; i<10;i++)

{

for(j=i+1;j<10;j++)

{

if(new[i]>new[j])

{

ch=new[i];

new[i]=new[j];

new[j]=ch;

}

}

}

printf("\nsorting:");

for(i=0;i<10;i++)

{

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

}

//removing the duplicates

printf("\nwithout duplicates: ");

for(i=0;i<10;i++)

{

//var=duplicate[i];

for(j=0;j<10;j++)

{

if(duplicate[i]==duplicate[j])

{

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

a[k]=duplicate[j];

k++;

count++;

}

}

}

//sum and mean

for(i=0;i<10;i++)

{

sum+=new[i];

avg=sum/10;

}

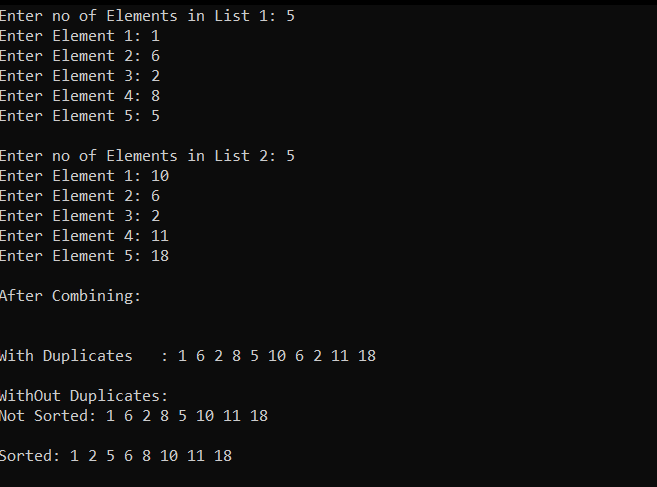
printf("\nsum is: %d\n",sum );

printf("average is: %d", avg);

return 0;

}

**OUTPUT:**



IMPLEMENTAION OF REVERSE AND COPY OF A STACK

#include<stdio.h>

int main(){

int a[20],b[20],c[40],d[40],i,l1,l2,n,flag,j=0,k=0,m=0,temp;

printf("Enter no of Elements in List 1: ");

scanf("%d",&l1);

for(i=0;i<l1;i++){

printf("Enter Element %d: ",i+1);

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

printf("\nEnter no of Elements in List 2: ");

scanf("%d",&l2);

for(i=0;i<l2;i++){

printf("Enter Element %d: ",i+1);

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

//with Duplicates

for(i=0;i<l1+l2;i++){

if(i<l1)

c[j++]=a[k++];

else

c[j++]=b[m++];}

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

printf("\t----->With Duplicates : ");

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

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

printf("\n");

//without duplicates

j=0;k=0;m=0;

for(i=0;i<l1+l2;i++){

flag=0;

if(i<l1){

for(n=0;n<j;n++){

if(a[k]==c[n]){ flag=1;}}

if(flag==0) d[j++]=a[k++];

else k++;}

else {

for(n=0;n<j;n++){ if(b[m]==c[n]){flag=1;}}

if(flag==0)d[j++]=b[m++];

else m++;}}

printf("\n\t----->WithOut Duplicates:");

printf("\n\n\t\t\t----->Not Sorted: ");

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

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

printf("\n");

//Sorting

for(i=0;i<j-1;i++){

for(k=0;k<j-i-1;k++){

if(d[k]>d[k+1]){

temp=d[k];d[k]=d[k+1];d[k+1]=temp;}}}

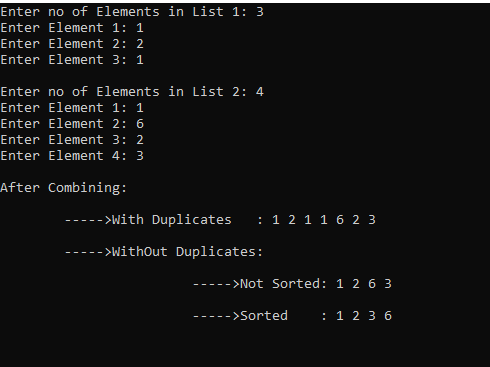
printf("\n\t\t\t----->Sorted : ");

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

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

printf("\n\n\n");}

**OUTPUT:**



IMPLEMENTATION OF PARENTHESIS VALIDITY CHECKER

#include<stdio.h>

int main()

{

int var1=0, var2=0,i;

char exp[20];

printf("Enter the expression: ");

scanf("%d", exp);

for(i=0; i<strlen(exp);i++)

{

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

{

var1++;

}

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

{

var2++;

}

}

if(var1==var2)

{

printf("Valid Expression");

}

else

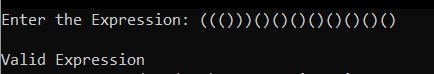
{

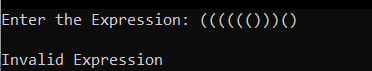
printf("Invalid Expression");

}

}

**OUTPUT:**





IMPLEMENTATION OF BFS AND DFS

#include<stdio.h>

#include<stdlib.h>

#include<stdbool.h>

struct Queue{

int data;

struct Queue\* next;

}\*front,\*rear,\*temp;

void Enqueue(int);

int Dequeue();

void Display();

struct stack{

int data;

struct stack\* link;

}\*top;

void push(int);

int pop();

struct mygraph{

int numnodes;

bool \*\*edges;

};

typedef struct mygraph graph;

graph\* create\_graph(int numnodes);

void destroy\_graph(graph\* g);

void print\_graph(graph\* g);

bool add\_edge(graph \*g,int from\_node,int to\_node);

bool remove\_edge(graph \*g,int from\_node,int to\_node);

bool has\_edge(graph \*g,int from\_node,int to\_node);

void Depth\_First\_Traversal(graph\* g,int visited[],int vertex);

void Breadth\_First\_Traversal(graph\* g,int visited[],int vertex,int arr[],int\* index,int\* count);

graph\* create\_graph(int numnodes)

{

graph\* g=malloc(sizeof(\*g));

g->numnodes=numnodes;

g->edges=calloc(sizeof(bool\*),g->numnodes);

for(int i=0;i<g->numnodes;i++)

{

g->edges[i]=calloc(sizeof(bool),g->numnodes);

}

return g;

}

void destroy\_graph(graph\* g)

{

for(int i=0;i<g->numnodes;i++)

{

if(g->edges[i]!=NULL)

{

free(g->edges[i]);

}

}

free(g->edges);

free(g);

}

bool add\_edge(graph\* g,int from\_node,int to\_node)

{

if(has\_edge(g,from\_node,to\_node))

return false;

g->edges[from\_node][to\_node]=true;

return true;

}

bool remove\_edge(graph\* g,int from\_node,int to\_node)

{

if(has\_edge(g,from\_node,to\_node)){

g->edges[from\_node][to\_node]=false;

return true;

}

return false;

}

bool has\_edge(graph\* g,int from\_node,int to\_node)

{

return g->edges[from\_node][to\_node];

}

void print\_graph(graph \*g)

{

printf("\nDIGRAPH:\n");

for(int from=0;from<g->numnodes;from++)

{

for(int to=0;to<g->numnodes;to++)

{

if(g->edges[from][to])

{

printf("%d->%d\n",from,to);

}

}

}

printf("\n");

}

void Depth\_First\_Traversal(graph\* g,int visited[],int vertex)

{

//----------------using stack--------------------

push(vertex);

while(!isempty1())

{

int node=pop();

if(!visited[node]){

printf("%d\t",node);

visited[node]=1;

for(int j=0;j<g->numnodes;j++)

{

if(!visited[j] && g->edges[node][j])

{

push(j);

}

}

}

}

}

bool Linear\_search(int arr[],int key,int size)

{

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

if(arr[i]==key)

return true;

return false;

}

void Breadth\_First\_Traversal(graph\* g,int visited[],int vertex,int arr[],int\* index,int\* count)

{

printf("%d\t",vertex);

visited[vertex]=1;

Enqueue(vertex);

while(!isempty())

{

int node=Dequeue();

for(int j=0;j<g->numnodes;j++)

{

if(!visited[j] && g->edges[node][j]){

printf("%d\t",j);

visited[j]=1;

Enqueue(j);

}

}

}

}

int main()

{

front=NULL;

rear=NULL;

int numnodes,ch,from,to,index=-1,count=0;

bool ret\_val;

printf("Enter the max values of nodes:");

scanf("%d",&numnodes);

graph \*g1=create\_graph(numnodes);

//printf("%d\t",g1->numnodes);

int visited[numnodes];

int arr[g1->numnodes];

printf("\n1.Add edge\n2.Remove edge\n3.Display\n4.Breadth First Traversal\n5.Depth\_First\_Traversal\n5.Exit\n");

do{

printf("\nEnter the choice: ");

scanf("%d",&ch);

switch(ch)

{

case 1:

printf("---ADDING EDGES---\n");

printf("Enter the FROM node and TO node:\n");

scanf("%d%d",&from,&to);

ret\_val=add\_edge(g1,from,to);

if(ret\_val)

printf("Edge added\n");

else

printf("Edge not added\n");

break;

case 2:

printf("---REMOVING EDGES---\n");

printf("Enter FROM node and TO node:");

scanf("%d%d",&from,&to);

ret\_val=remove\_edge(g1,from,to);

if(ret\_val)

printf("Edge removed\n");

else

printf("Edge does not exit to be removed\n");

break;

case 3:

printf("---DISPLAYING---");

print\_graph(g1);

break;

case 4:

printf("---BFS TRAVERSAL---");

printf("\nEnter the vertex of traversal:\n");

scanf("%d",&from);

memset(visited,0,numnodes\*sizeof(int));

Breadth\_First\_Traversal(g1,visited,from,arr,&index,&count);

break;

case 5:

printf("--- DFS TRAVERSAL---");

printf("\nEnter the vertex of traversal:\n");

scanf("%d",&from);

memset(visited,0,g1->numnodes\*sizeof(int));

Depth\_First\_Traversal(g1,visited,from);

break;

case 6:

break;

default:

printf("Invalid choice!!\n");

}}while(ch!=6);

// print\_graph(g1);

destroy\_graph(g1);

}

void Enqueue(int x)

{

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

temp->data=x;

temp->next=NULL;

if(rear==NULL && front==NULL)

{

front=temp;

rear=temp;

}

else{

rear->next=temp;

rear=temp;

}

}

int Dequeue()

{

if(front==rear)

{

int item=front->data;

front=NULL;

rear=NULL;

return item;

}

if(front==NULL){

printf("Queue is empty\n");

return NULL;}

int item=front->data;

temp=front;

front=front->next;

return item;

}

int isempty()

{

if(front==NULL)

return 1;

return 0;

}

void push(int item)

{

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

newnode->data=item;

newnode->link=top;

top=newnode;

}

int pop()

{

int item;

item=top->data;

top=top->link;

return item;

}

int isempty1()

{

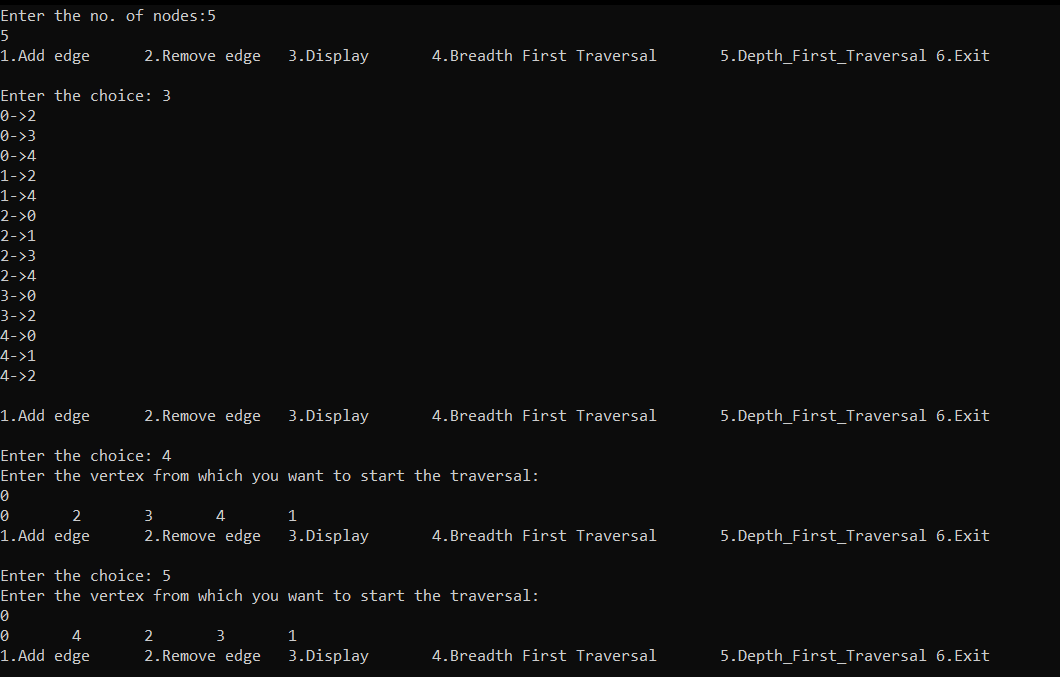
if(top==NULL)

return 1;

return 0;

}

**OUTPUT:**

****

MERGE SORT

#include<stdio.h>

int arr[100],i,j,k,l,r,m,n,n1,n2;

int mergesort(int,int);

int merge(int, int, int);

int mergesort(int l,int r)

{

if(l<r)

{

m=(l+r)/2;

mergesort(l,m);

mergesort(m+1,r);

merge(l,m,r);

}

}

int merge(int l,int m,int r)

{

n1=m-l+1;

n2=r-m;

k=l;

int L[n1], R[n2];

for(i=0;i<n1;i++)

{

L[i]=arr[l+i];

}

for(j=0;j<n2;j++)

{

R[j]=arr[m+1+j];

}

i=0;

j=0;

while(i<n1 && j<n2)

{

if(L[i]<=R[j])

{

arr[k++]=L[i++];

}

else

{

arr[k++]=R[j++];

}

}

while(i<n1)

{

arr[k++]=L[i++];

}

while(j<n2)

{

arr[k++]=R[j++];

}

}

int main()

{

printf("MERGE SORT\n");

printf("Enter the no of elements: ");

scanf("%d",&n);

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

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

{

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

}

l=0;

r=n-1;

mergesort(l,r);

printf("\nSorted array: ");

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

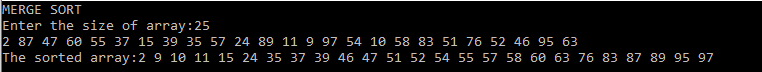
{

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

}

}

**OUTPUT:**



QUICK SORT

#include <stdio.h>

int n,arr[100],i,j;

void swap(int \*a, int \*b) {

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition(int array[], int low, int high)

{

int pivot = array[high];

int i = (low - 1);

for ( j = low; j < high; j++) {

if (array[j] <= pivot)

{

i++;

swap(&array[i], &array[j]);

}

}

swap(&array[i + 1], &array[high]);

return (i + 1);

}

void quickSort(int array[], int low, int high)

{

if (low < high)

{

int pi = partition(array, low, high);

quickSort(array, low, pi - 1);

quickSort(array, pi + 1, high);

}

}

void printArray(int array[], int size) {

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

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

}

printf("\n");

}

int main()

{

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

scanf("%d",&n);

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

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

{

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

}

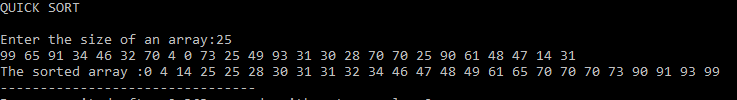
quickSort(arr, 0, n - 1);

printf("Sorted array in ascending order: \n");

printArray(arr, n);

}

**OUTPUT:**



STRASSEN MATRIX MULTIPLICATION

#include<stdio.h>

#include<conio.h>

#include<math.h>

int power(int);

int strassen(int\*,int\*,int\*,int,int);

int main()

{

int i,j,k,n1,n2,n3,n,m;

printf("FIRST MATRIX\n");

printf("Enter no of rows and columns:\n");

scanf("%d%d",&n1,&n2);

int A[n1][n2];

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

for(i=0;i<n1;i++)

{

for(j=0;j<n2;j++)

{

//printf("A[i][j]:",i+1,j+1);

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

}

}

printf("SECOND MATRIX\n");

printf("\nEnter the no. of columns:\n");

scanf("%d",&n3);

int B[n2][n3];

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

for(i=0;i<n2;i++)

{

for(j=0;j<n3;j++)

{

//printf("B[i][j]",i+1,j+1);

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

}

}

//creating square matrix of order 2^n for all and initializing all elements to 0 except prefixed

if(n1>=n2 && n1>=n3)

n=n1;

else if(n2>=n1 && n2>=n3)

n=n2;

else

n=n3;

int o=1;

while(n>power(o))

o=o+1;

m=power(o);

int a[m][m],b[m][m],c[m][m];

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

{

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

{

a[i][j]=0;

b[i][j]=0;

}

}

for(i=0;i<n1;i++)

for(j=0;j<n2;j++)

a[i][j]=A[i][j];

for(i=0;i<n2;i++)

for(j=0;j<n3;j++)

b[i][j]=B[i][j];

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

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

c[i][j]=0;

strassen(a,b,c,m,m);

/\*printf("Elements of the FIRST MATRIX: \n");

for(i=0;i<n1;i++)

{

for(j=0;j<n2;j++)

{

printf("%d\t",&A[i][j]);

}

printf("\n");

}

printf("Elements of the SECOND MATRIX: \n");

for(i=0;i<n2;i++)

{

for(j=0;j<n3;j++)

{

printf("%d\t",&B[i][j]);

}

printf("\n");

}\*/

printf("Result matrix\n");

for(i=0;i<n1;i++){

for(j=0;j<n3;j++){

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

}

printf("\n");

}

}

int power(int n)

{

int i,p=1;

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

p=2\*p;

return p;

}

int strassen(int\* A,int\* B,int\* C, int m, int n){

if(m==2)

{

int p=(\*A+\*(A+n+1))\*(\*B+\*(B+n+1));//p=(A[0][0]+A[1][1])\*(B[0][0]+B[1][1])

int q=(\*(A+n)+\*(A+n+1))\*(\*B);//q=(A[1][0]+A[1][1])\*B[0][0]

int r=(\*A)\*(\*(B+1)-\*(B+n+1)); //r=A[0][0]\*(B[0][1]-B[1][1])

int s=(\*(A+n+1))\*(\*(B+n)-\*B);// s=A[1][1]\*(B[1][0]-B[0][0])

int t=(\*A+\*(A+1))\*(\*(B+n+1));// t=(A[0][0]+A[0][1])\*B[1][1]

int u=(\*(A+n)-\*A)\*(\*B+\*(B+1)); //u=(A[1][0]-A[0][0])\*(B[0][0]+B[0][1])

int v=(\*(A+1)-\*(A+n+1))\*(\*(B+n)+\*(B+n+1)); // v=(A[0][1]-A[1][1])\*(B[1][0]+B[1][1])

\*C=\*C+p+s-t+v;

\*(C+1)=\*(C+1)+r+t;

\*(C+n)=\*(C+n)+q+s;

\*(C+n+1)=\*(C+n+1)+p+r-q+u;

}

else{

m=m/2;

strassen(A,B,C,m,n);

strassen(A,B+m,C+m,m,n);

strassen(A+m,B+m\*n,C,m,n);

strassen(A+m,B+m\*(n+1),C+m,m,n);

strassen(A+m\*n,B,C+m\*n,m,n);

strassen(A+m\*n,B+m,C+m\*(n+1),m,n);

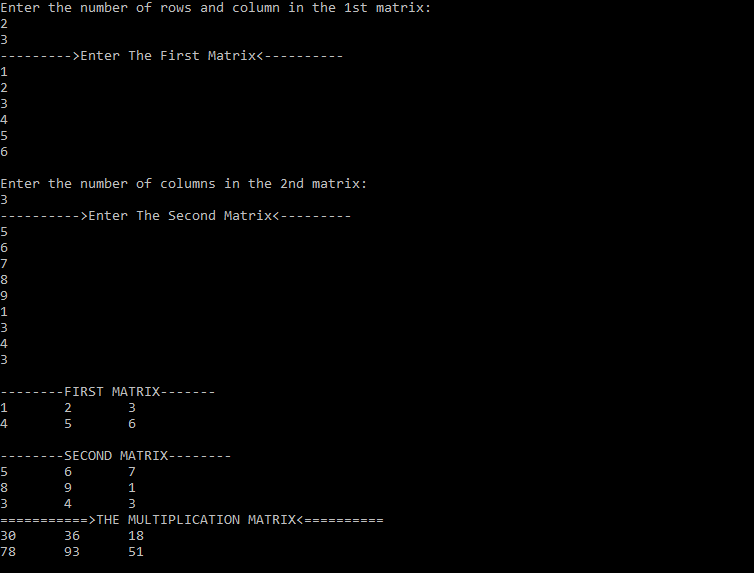
strassen(A+m\*(n+1),B+m\*n,C+m\*n,m,n);

strassen(A+m\*(n+1),B+m\*(n+1),C+m\*(n+1),m,n);

}

}

**OUTPUT:**



KRUSKAL’S ALGORITHM

#include <stdio.h>

#define MAX 30

int Graph[MAX][MAX], n;

void insertEdge(int m, int l, int w){

Graph[m-1][l-1] = w;

Graph[l-1][m-1] = w;

}

typedef struct edge {

int u, v, w;

} edge;

typedef struct edge\_list {

edge data[MAX];

int n;

} edge\_list;

edge\_list elist;

edge\_list spanlist;

void kruskalAlgo();

int find(int belongs[], int vertexno);

void applyUnion(int belongs[], int c1, int c2);

void sort();

void print();

void kruskalAlgo() {

int belongs[MAX], i, j, cno1, cno2;

elist.n = 0;

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

for (j = 0; j < i; j++) {

if (Graph[i][j] != 0) {

elist.data[elist.n].u = i;

elist.data[elist.n].v = j;

elist.data[elist.n].w = Graph[i][j];

elist.n++;

}

}

sort();

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

belongs[i] = i;

spanlist.n = 0;

for (i = 0; i < elist.n; i++) {

cno1 = find(belongs, elist.data[i].u);

cno2 = find(belongs, elist.data[i].v);

if (cno1 != cno2) {

spanlist.data[spanlist.n] = elist.data[i];

spanlist.n = spanlist.n + 1;

applyUnion(belongs, cno1, cno2);

}

}

}

int find(int belongs[], int vertexno) {

return (belongs[vertexno]);

}

void applyUnion(int belongs[], int c1, int c2) {

int i;

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

if (belongs[i] == c2)

belongs[i] = c1;

}

void sort() {

int i, j;

edge temp;

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

for (j = 0; j < elist.n - 1; j++)

if (elist.data[j].w > elist.data[j + 1].w) {

temp = elist.data[j];

elist.data[j] = elist.data[j + 1];

elist.data[j + 1] = temp;

}

}

void print() {

int i, cost = 0;

for (i = 0; i < spanlist.n; i++) {

printf("\n%d - %d : %d", spanlist.data[i].u, spanlist.data[i].v, spanlist.data[i].w);

cost = cost + spanlist.data[i].w;

}

printf("\nSpanning tree cost: %d", cost);

}

int main() {

int i, j, total\_cost, e, m ,l, w;

printf("Enter no of vertices and edges: ");

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

for(i=0; i<e; i++){

printf("Enter two vertices and edge cost: ");

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

insertEdge(m,l,w);

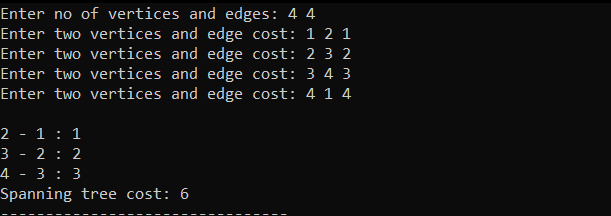
}

kruskalAlgo();

print();

}

**OUTPUT:**

****

PRIM’S ALGORITHM

#include <limits.h>

#include <stdbool.h>

#include <stdio.h>

#define MAX 10

int n,v,i,count;

int minKey(int key[], bool mstSet[])

{

int min = INT\_MAX, min\_index;

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

if (mstSet[v] == false && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

int printMST(int parent[], int graph[n][n])

{

printf("Edge \tWeight\n");

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

printf("%d - %d \t%d \n", parent[i], i,

graph[i][parent[i]]);

}

void primMST(int graph[n][n])

{

int parent[n];

int key[n];

bool mstSet[n];

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

for (count = 0; count < n - 1; count++)

{

int u = minKey(key, mstSet);

mstSet[u] = true;

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

{

if (graph[u][v] && mstSet[v] == false && graph[u][v] < key[v])

parent[v] = u, key[v] = graph[u][v];

}

}

printMST(parent, graph);

}

int main()

{

int e, i, j, m, l, w;

printf("Enter no of vertices and edges: ");

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

int graph[n][n];

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

for(j=0; j<n; j++){

graph[i][j] = 0;

}

}

for(i=0; i<e; i++){

printf("Enter two vertices and edge cost: ");

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

graph[m-1][l-1] = w;

graph[l-1][m-1] = w;

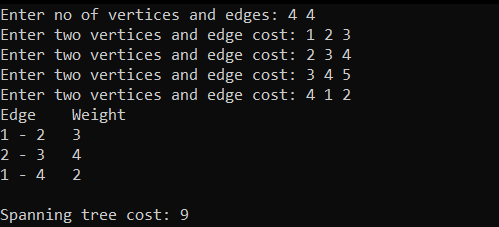
}

primMST(graph);

return 0;

}

**OUTPUT:**

****

KNAPSACK PROBLEM

#include<stdio.h>

int maxi(int a, int b)

{

return (a > b)? a : b;

}

int knapsack( int max, int profit[], int weight[], int val)

{

int i, j;

int K[val+1][max+1];

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

{

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

{

if (i==0 || j==0)

K[i][j] = 0;

else if (weight[i-1] <= j)

K[i][j] = maxi(profit[i-1] + K[i-1][j-weight[i-1]], K[i-1][j]);

else

K[i][j] = K[i-1][j];

}

}

return K[val][max];

}

int main()

{

int val,profit[100], weight[100],max,i;

printf("KNAPSACK PROBLEM AND SOLUTION\n");

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

scanf("%d",&val);

printf("Enter %d profits and their corresponding weights\n",val);

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

{

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

}

printf("Enter the maximum weight of the sack: ");

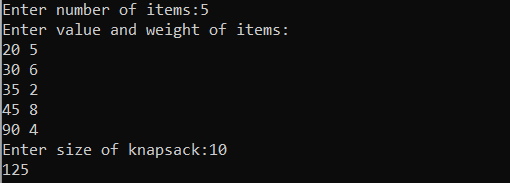
scanf("%d", &max);

printf("The maximum profit obtainable for the given data is: %d",knapsack(max,profit,weight,val));

return 0;

}

**OUTPUT:**



MATRIX CHAIN MULTIPLICATION USING DYNAMIC PROGRAMMING

//2.MULTIPLICATION:

#include <limits.h>

#include <stdio.h>

int block(int arr[],int i, int j)

{

if (i == j)

return 0;

int k;

int min =INT\_MAX;

int number;

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

{

number = (block(arr, i, k)

+ block(arr, k + 1, j)

+ arr[i - 1] \* arr[k] \* arr[j]);

if (number < min)

min = number;

}

return min;

}

int main()

{

int arr[100],n;

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

scanf("%d",&n);

printf("Enter an array: ");

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

{

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

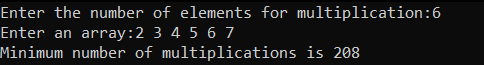
}

printf("Minimum number of multiplications %d: ",block(arr,1,n-1));

return 0;

}

**OUTPUT:**



LONGEST COMMON SUBSEQUENCE USING DYNAMIC PROGRAMMING

//3.COMMON SUBSEQUENCE:

#include <stdio.h>

#include <string.h>

int i, j, m, n, table[20][20];

char arr[20], str[20], b[20][20];

void lcsAlgo() {

m = strlen(arr);

n = strlen(str);

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

{

table[i][0] = 0;

}

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

{

table[0][i] = 0;

}

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

{

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

{

if (arr[i - 1] == str[j - 1])

{

table[i][j] = table[i - 1][j - 1] + 1;

}

else if (table[i - 1][j] >= table[i][j - 1])

{

table[i][j] = table[i - 1][j];

}

else

{

table[i][j] = table[i][j - 1];

}

}

}

int index = table[m][n];

char lcsAlgo[index + 1];

lcsAlgo[index] = '\0';

int i = m, j = n;

while (i> 0 && j > 0) {

if (arr[i - 1] == str[j - 1]) {

lcsAlgo[index - 1] = arr[i - 1];

i--;

j--;

index--;

}

else if (table[i - 1][j] > table[i][j - 1])

i--;

else

j--;

}

printf("FIRST: %s \nSECOND : %s \n", arr, str);

printf("LCS: %s", lcsAlgo);

}

int main()

{

printf("\nEnter the first sequence of character:");

scanf("%s",arr);

printf("\nEnter the second sequence of character:");

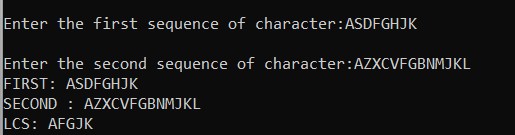
scanf("%s",str);

lcsAlgo();

printf("\n");

}

**OUTPUT:**



N QUEEN PROBLEM

#include<stdio.h>

#include<conio.h>

#include<math.h>

int a[30],count=0;

int place(int pos)

{

int i;

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

{

if((a[i]==a[pos])||((abs(a[i]-a[pos])==abs(i-pos))))

return 0;

}

return 1;

}

void sol(int n)

{

int i,j;

count++;

printf("\n\nSolution -->%d:\n",count);

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

{

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

{

if(a[i]==j)

printf("Q\t");

else

printf("+\t");

}

printf("\n");

}

}

void find(int n)

{

int k=1;

a[k]=0;

while(k!=0)

{

a[k]=a[k]+1;

while((a[k]<=n)&&!place(k))

a[k]++;

if(a[k]<=n)

{

if(k==n)

sol(n);

else

{

k++;

a[k]=0;

}

}

else

k--;

}

}

int main() {

int i,n;

printf("N QUEEN PROBLEM BACKTRACKING\n");

printf("\nEnter the number of Queens in the chess board:");

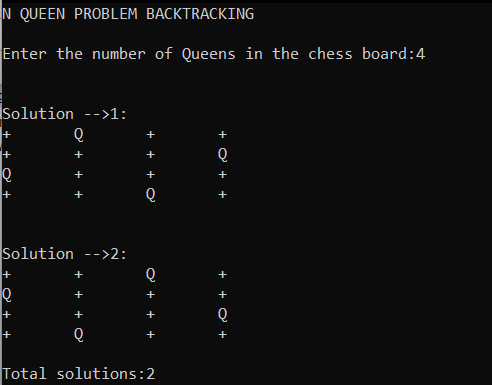
scanf("%d",&n);

find(n);

printf("\nTotal solutions:%d",count);

return 0;

}

**OUTPUT:**  


SUM OF SUBSETS

#include <stdio.h>

#include <stdlib.h>

static int total\_nodes;

void printSubset(int A[], int size)

{

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

{

printf("%\*d", 5, A[i]);

}

printf("\n");

}

int comparator(const void \*pLhs, const void \*pRhs)

{

int \*lhs = (int \*)pLhs;

int \*rhs = (int \*)pRhs;

return \*lhs > \*rhs;

}

void subset\_sum(int s[], int t[],

int s\_size, int t\_size,

int sum, int ite,

int const target\_sum)

{

total\_nodes++;

if( target\_sum == sum )

{

printSubset(t, t\_size);

if( ite + 1 < s\_size && sum - s[ite] + s[ite+1] <= target\_sum )

{

subset\_sum(s, t, s\_size, t\_size-1, sum - s[ite], ite + 1, target\_sum);

}

return;

}

else

{

if( ite < s\_size && sum + s[ite] <= target\_sum )

{

for( int i = ite; i < s\_size; i++ )

{

t[t\_size] = s[i];

if( sum + s[i] <= target\_sum )

{

subset\_sum(s, t, s\_size, t\_size + 1, sum + s[i], i + 1, target\_sum);

}

}

}

}

}

void generateSubsets(int s[], int size, int target\_sum)

{

int \*tuplet\_vector = (int \*)malloc(size \* sizeof(int));

int total = 0;

// sort the set

qsort(s, size, sizeof(int), &comparator);

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

{

total += s[i];

}

if( s[0] <= target\_sum && total >= target\_sum )

{

subset\_sum(s, tuplet\_vector, size, 0, 0, 0, target\_sum);

}

free(tuplet\_vector);

}

int main()

{

int n,target;

printf("SUM OF SUBSET BACKTRACKING PROGRAM\n");

int weights[100];

printf("\nEnter the number of elements in the set:");

scanf("%d",&n);

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

{

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

}

printf("\nEnter the target value of the subset:");

scanf("%d",&target);

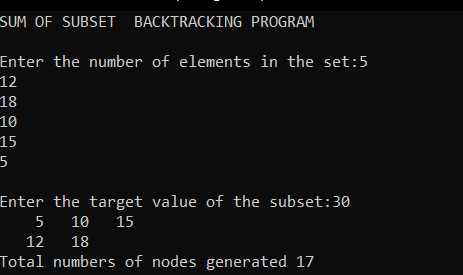
generateSubsets(weights, n, target);

printf("Total numbers of nodes generated %d\n", total\_nodes);

return 0;

}

**OUTPUT:**



0/1 KNAPSACK PROBLEM USING BRANCH AND BOUND

#include<stdio.h>

int max(int a, int b) {

if(a>b){

return a;

} else {

return b;

}

}

int knapsack(int W, int wt[], int val[], int n) {

int i, w;

int knap[n+1][W+1];

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

for (w = 0; w <= W; w++) {

if (i==0 || w==0)

knap[i][w] = 0;

else if (wt[i-1] <= w)

knap[i][w] = max(val[i-1] + knap[i-1][w-wt[i-1]], knap[i-1][w]);

else

knap[i][w] = knap[i-1][w];

}

}

return knap[n][W];

}

int main() {

printf("0/1 KNAPSACK PROBLEM BRANCH AND BOUND METHOD\n");

int profit[100],weight[100],cap,n;

printf("\nEnter the Number of items:");

scanf("%d",&n);

printf("\nEnter the profit and weight ");

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

{

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

}

printf("\nEnter the capacity of the sack:\n");

scanf("%d",&cap);

printf("The solution is : %d", knapsack(cap, weight, profit, n));

return 0;

}

**OUTPUT:**