# TASK 1:

#include<iostream>

#include<stdlib.h>

#include<time.h>

using namespace std;

class Game

{

int Cost;

char Arr[15][15];

int s, s1;

public:

Game()

{

Cost = s = s1 = 0;

for (int i = 0; i < 15; i++)

{

for (int j = 0; j < 15; j++)

{

Arr[i][j] = ' ';

}

}

srand(time(0));

int x, y;

for (int i = 0; i < 50; i++)

{

while (true)

{

x = rand() % 14;

y = rand() % 14;

if (x != 0 && y != 0)

break;

}

Arr[x][y] = 'X';

}

Arr[0][0] = 'S';

Arr[14][14] = 'E';

}

int getcost()

{

return Cost;

}

void disp()

{

for (int i = 14; i >= 0; i--)

{

for (int j = 0; j < 15; j++)

{

cout << Arr[i][j] << " | ";

}

cout << endl;

}

}

bool Upward()

{

Arr[s][s1] = ' ';

s = s + 1;

if (s > 14) {

s = s - 14;

}

if (Arr[s][s1] == 'X')

{

return 0;

}

Arr[s][s1] = 'S';

Cost = Cost + 2;

return 1;

}

bool Right()

{

Arr[s][s1] = ' ';

s1 = s1 + 1;

if (s1 > 14)

{

s1 = s1 - 14;

}

if (Arr[s][s1] == 'X')

{

return 0;

}

Arr[s][s1] = 'S';

Cost = Cost + 2;

return 1;

}

bool Dignal()

{

Arr[s][s1] = ' ';

s1 = s1 + 1;

s = s + 1;

if (s > 14)

{

s = s - 14;

}

if (s1 > 14)

{

s1 = s1 - 14;

}

if (Arr[s][s1] == 'X')

{

return 0;

}

Arr[s][s1] = 'S';

Cost = Cost + 3;

return 1;

}

bool Comp()

{

if (s == 14 && s1 == 14)

{

return 1;

}

return 0;

}

};

int main()

{

int x;

bool fg;

Game obj;

while (!obj.Comp())

{

system("cls");

obj.disp();

cout << endl << "Press 1 to move Upward....." << endl;

cout << "Press 2 to move Right....." << endl;

cout << "Press 3 to move dignally....." << endl;

cout << "Enter : ";

cin >> x;

switch (x)

{

case 1:

{

fg = obj.Upward();

if (fg == 0)

{

system("cls");

cout << "You hit the Wall....!" << endl;

cout << "Game is over" << endl;

system("pause");

return 0;

}

break;

}

case 2:

fg = obj.Right();

if (fg == 0)

{

system("cls");

cout << "You hit the Wall....!" << endl;

cout << "Game is over" << endl;

system("pause");

return 0;

}

break;

case 3:

fg = obj.Dignal();

if (fg == 0)

{

system("cls");

cout << "You hit the Wall....!" << endl;

cout << "Game is over" << endl;

system("pause");

return 0;

}

break;

default:

cout << "Wrong input......" << endl;

break;

}

}

system("cls");

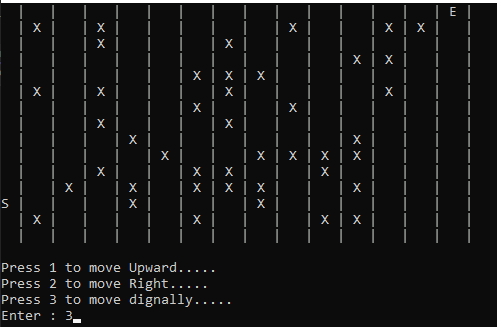
cout << "You have won...." << endl;

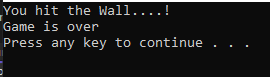
cout << "Total cost : " << obj.getcost() << endl;

system("pause");

}

# OUTPUT:





# TASK 2:

#include<iostream>

using namespace std;

void setval(int a[][6], int n)

{

bool flag;

int x, y,weight;

cout << "e.g (First Node,Second Node)\n";

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

{

flag = false;

while (!flag)

{

cout << "\nEdge (" << i + 1 << ")\n";

cout << "1st Node -> ";

cin >> x;

cout << "2st Node -> ";

cin >> y;

cout << "Enter Weight Of Edge -> ";

cin >> weight;

if (x > 0 && x <= 6 && y > 0 && y <= 6)

{

a[x - 1][y - 1] = weight;

a[y - 1][x - 1] = weight;

flag = true;

}

else

{

cout << "\nWrong Input! enter Again." << endl;

flag = false;

}

}

}

}

void display(int a[][6], int n)

{

cout << "\nEntered Edges Is A={";

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

{

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

{

if (a[i][j] != -1)

cout << "(" << i + 1 << "," << j + 1 << ")";

if (j == n - 1 && i == n - 1)

{

cout << "}";

break;

}

if (a[i][j] != -1)

cout << ",";

}

}

char ch = 'A';

cout << "\n\nAdjacency Matrix:\n";

cout << " 1 2 3 4 5 6" << endl;

cout << " A B C D E F" << endl;

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

{

cout << ch++ << " | ";

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

{

if (a[i][j] != -1)

{

if (a[i][j] > 9)

cout << "+" << a[i][j];

else

cout << "+" << a[i][j] << " ";

}

else

cout << a[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int nodes = 6;

int a[6][6];

int num;

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

{

for (int j = 0; j < nodes; j++)

{

a[i][j] = -1;

}

}

cout << "How Many Edges Do You Want To Add in Graph : ";

cin >> num;

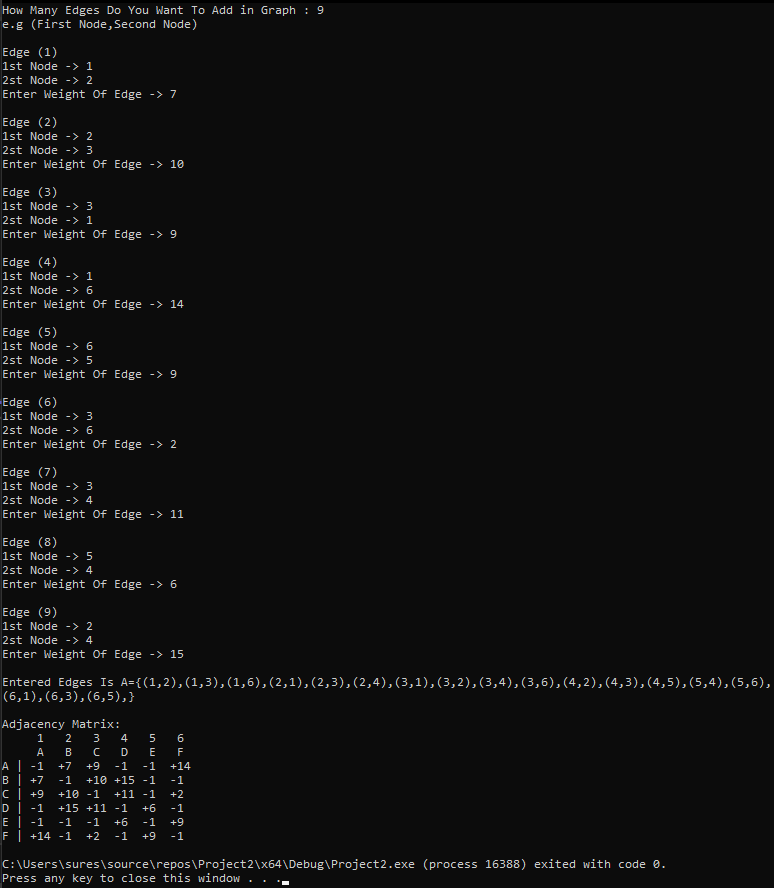
setval(a, num);

display(a, 6);

return 0;

}

# Output: -



# TASK 2 (b1):

#include<iostream>

using namespace std;

void setval(int a[][8], int n)

{

bool flag;

int x, y;

cout << "e.g (First Node,Second Node)\n";

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

{

flag = false;

while (!flag)

{

cout << "\nEdge (" << i + 1 << ")\n";

cout << "1st Node ->";

cin >> x;

cout << "2st Node ->";

cin >> y;

if (x >= 0 && x <= 8 && y >= 0 && y <= 8)

{

a[x - 1][y - 1] = 1;

a[y - 1][x - 1] = 1;

flag = true;

}

else

{

cout << "\nWrong Input! enter Again." << endl;

flag = false;

}

}

}

}

void display(int a[][8],int n)

{

cout << "\nEntered Edges Is A={";

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

{

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

{

if (a[i][j] == 1)

cout << "(" << i + 1 << "," << j + 1 << ")";

if (j == n - 1 && i == n - 1)

{

cout << "}";

break;

}

if (a[i][j] == 1)

cout << ",";

}

}

cout << "\n\nAdjacency Matrix:\n";

cout << " 1 2 3 4 5 6 7 8" << endl;

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

{

cout << i + 1 << " ";

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

{

cout << a[i][j] << " ";

}

cout << endl;

}

}

int main()

{

int nodes = 8;

int a[8][8];

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

{

for (int j = 0; j < nodes; j++)

{

a[i][j] = 0;

}

}

cout << "Enter Edges:\n";

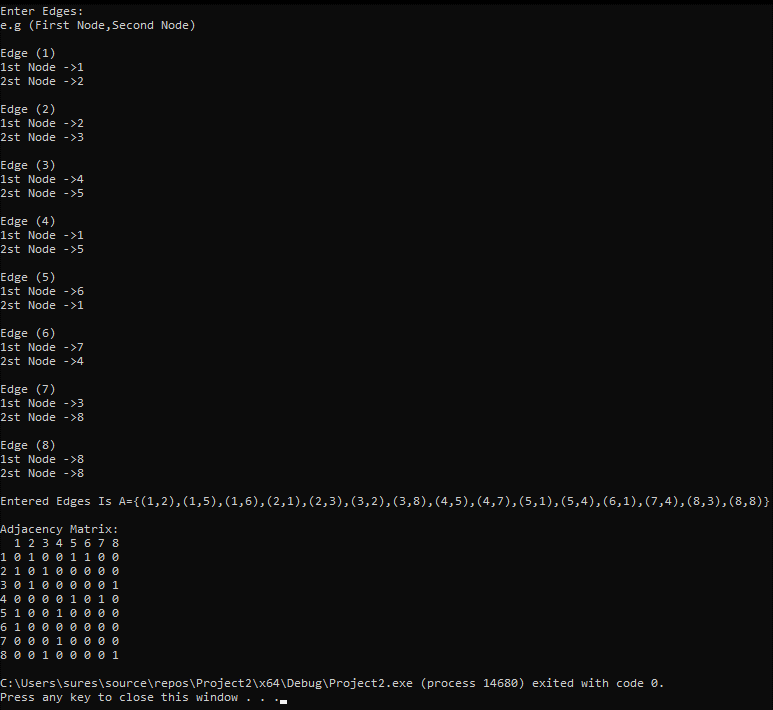
setval(a, 8);

display(a, 8);

return 0;

}

# Output: -



# TASK 2 (b2):

#include<iostream>

using namespace std;

void setval(int a[][8], int n)

{

bool flag;

int x, y;

cout << "e.g (First Node,Second Node)\n";

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

{

flag = false;

while (!flag)

{

cout << "\nEdge (" << i + 1 << ")\n";

cout << "1st Node ->";

cin >> x;

cout << "2st Node ->";

cin >> y;

if (x >= 0 && x <= 8 && y >= 0 && y <= 8)

{

a[x - 1][y - 1] = 1;

a[y - 1][x - 1] = 1;

flag = true;

}

else

{

cout << "\nWrong Input! enter Again." << endl;

flag = false;

}

}

}

}

void display(int a[][8],int n)

{

cout << "\nEntered Edges Is A={";

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

{

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

{

if (a[i][j] == 1)

cout << "(" << i + 1 << "," << j + 1 << ")";

if (j == n - 1 && i == n - 1)

{

cout << "}";

break;

}

if (a[i][j] == 1)

cout << ",";

}

}

cout << "\nAdjancy List Given Below:\n";

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

{

cout << i+1;

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

{

if (a[i][j]==1)

{

cout <<" -> " << j + 1;

}

}

cout << endl;

}

}

int main()

{

int nodes = 8;

int a[8][8];

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

{

for (int j = 0; j < nodes; j++)

{

a[i][j] = 0;

}

}

cout << "Enter Edges:\n";

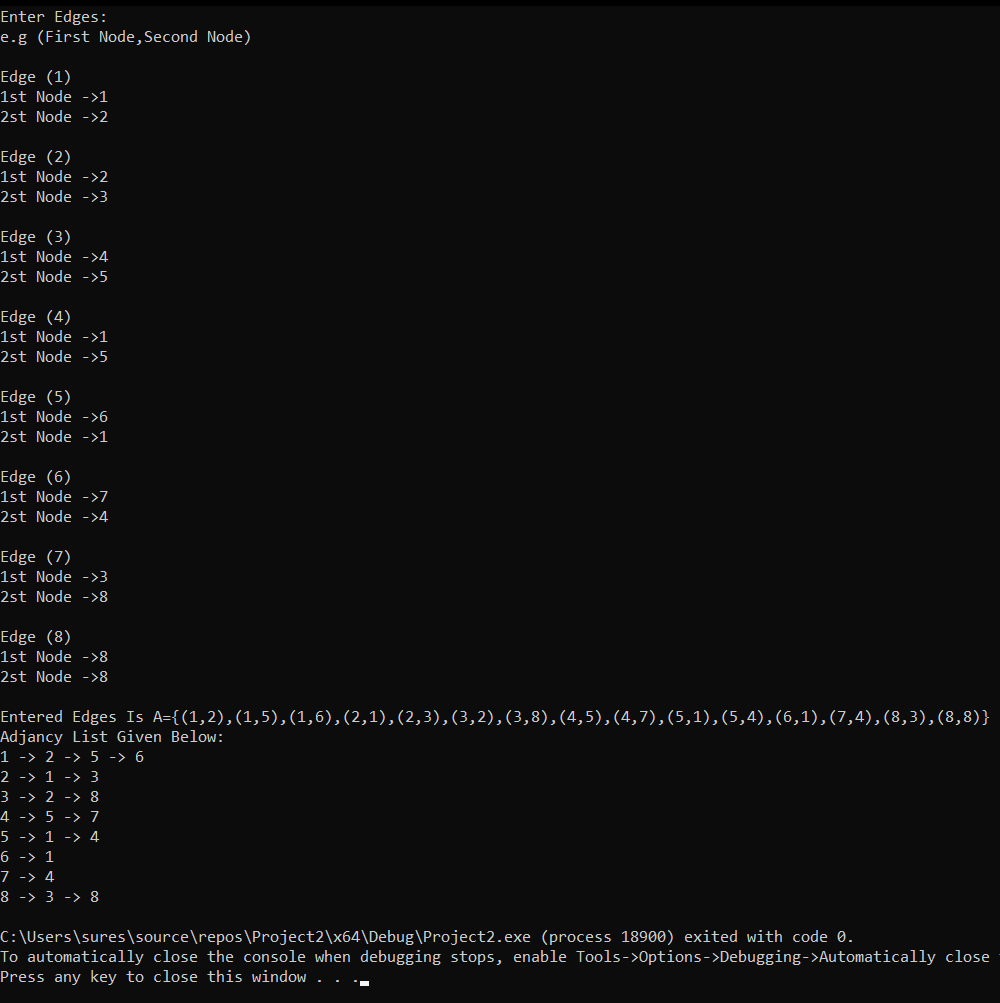
setval(a, 8);

display(a, 8);

return 0;

}

# Output: -



# TASK 3:

#include <iostream>

using namespace std;

int V;

struct Node

{

int data;

Node\* Next;

Node()

{

data = 0;

Next = NULL;

}

Node(int v) :Node()

{

data = v;

}

};

Node\* Np[10];

class Stack

{

private:

Node\* Top;

public:

Stack()

{

Top = NULL;

}

void Push(int Val)

{

Node\* n;

n = new Node;

n->data = Val;

n->Next = NULL;

if (Top == NULL)

Top = n;

else

{

n->Next = Top;

Top = n;

}

}

Node\* Pop()

{

if (IsEmpty() == false)

{

Node\* Temp = Top;

Top = Top->Next;

return Temp;

}

else

{

cout << "The Stack Is Empty." << endl;

Top = NULL;

}

}

char Peek()

{

if (IsEmpty() == false)

return Top->data;

else

{

cout << "The Stack Is Empty." << endl;

return 0;

}

}

bool IsEmpty()

{

if (Top == NULL)

return true;

else

return false;

}

void DFS\_Traversal()

{

bool f[10];

Node\* Temp;

int D, Val;

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

f[i] = false;

Stack obj;

cout << "Your Vertices: ";

for (int i = 1; i <= V; i++)

cout << i << " ";

cout << "\n\nEnter Starting Vertex: ";

cin >> Val;

while (Val < 1 || Val > V)

{

cout << "Invalid Starting Vertex added"

<< "\nEnter Again: ";

cin >> Val;

}

cout << "Your Vertices: ";

for (int i = 1; i <= V; i++)

{

cout << i << " ";

}

cout << endl << "Starting Vertex: " << Val << endl << endl;

cout << "DFS Traversal: ";

obj.Push(Val);

while (!obj.IsEmpty())

{

D = obj.getTop()->data;

obj.Pop();

f[D] = true;

cout << D << " ";

Temp = Np[D];

while (Temp)

{

D = Temp->data;

if (!f[D])

{

obj.Push(D);

f[D] = true;

}

Temp = Temp->Next;

}

}

}

Node\* getTop()

{

return Top;

}

};

class Queue

{

Node\* front;

Node\* rare;

public:

Queue()

{

front = NULL;

rare = NULL;

}

bool IsEmpty()

{

if (rare == NULL)

return true;

else

return false;

}

void Enqueue(int Val)

{

Node\* N;

N = new Node;

N->data = Val;

N->Next = NULL;

if (rare == NULL)

{

rare = N;

front = N;

}

else

{

N->Next = rare;

rare = N;

}

}

int Dequeue()

{

if (IsEmpty() == false)

{

int Temp = front->data;

if (rare == front)

{

delete rare;

front = NULL;

rare = NULL;

return Temp;

}

else

{

Node\* Foo1 = rare;

while (Foo1->Next != front)

{

Foo1 = Foo1->Next;

}

delete front;

front = Foo1;

front->Next = NULL;

return Temp;

}

}

else

{

cout << "The Stack Is Empty!" << endl;

rare = front = NULL;

return -1;

}

}

void Display()

{

if (IsEmpty() == false)

{

Queue Obj;

int temp;

while (IsEmpty() == false)

{

temp = Dequeue();

cout << temp << " ";

Obj.Enqueue(temp);

}

while (Obj.IsEmpty() == false)

{

Enqueue(Obj.Dequeue());

}

}

else

cout << "Queues is Empty...!" << endl;

}

int Peek()

{

if (IsEmpty() == false)

return front->data;

else

{

cout << "Is Empty!" << endl;

return -1;

}

}

void BFS\_Traversal()

{

bool f[10];

Node\* Temp;

int D, f1;

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

{

f[i] = false;

}

Stack obj;

cout << "Please Enter Starting Vertex: ";

cin >> f1;

while (f1 < 1 || f1 > V)

{

cout << "Invalid Input...!"

<< "Please Enter Again: ";

cin >> f1;

}

cout << "Your Vertices Are: ";

for (int i = 1; i <= V; i++)

{

cout << i << " ";

}

cout << endl;

cout << "Starting Vertex: " << f1 << endl << endl;

cout << "BFS Traversal: ";

obj.Push(f1);

while (!obj.IsEmpty())

{

D = obj.getTop()->data;

obj.Pop();

f[D] = true;

cout << D << " ";

Temp = Np[D];

while (Temp)

{

D = Temp->data;

if (!f[D])

{

obj.Push(D);

f[D] = true;

}

Temp = Temp->Next;

}

}

}

Node\* getFront()

{

return front;

}

};

void Graph()

{

cout << "Please Enter Total Numbers of Vertices: ";

cin >> V;

while (V < 1 || V > 10)

{

cout << "Invalid Input...!" << endl << "Please Enter Total Numbers of Vertices(1-10): ";

cin >> V;

}

cout << endl << endl;

cout << "Your Vertices are: ";

for (int i = 1; i <= V; i++)

{

cout << i << " ";

}

cout << endl << endl << "Press Enter to Continue...!";

int total, num;

Node\* f, \* f1;

for (int i = 1; i <= V; i++)

{

f1 = NULL;

cout << "\nPlease Enter Adjacent Vertices of Vertex (" << i << ") : ";

cin >> total;

cout << "\n\n";

for (int j = 1; j <= total; j++)

{

cout << "Enter Adjacent Vertex Number: ";

cin >> num;

f = new Node(num);

if (!Np[i])

Np[i] = f1 = f;

else

{

f1->Next = f;

f1 = f;

}

}

}

}

int main()

{

int option = 0;

Stack obj1;

Queue obj2;

do

{

cout << "\nPress 1 To Input In Graph," << endl;

cout << "Press 2 To Display The Graph By DFS-Traversal," << endl;

cout << "Press 3 To Display The Graph By BFS-Traversal," << endl;

cout << "Press 0 To Terminate The Program." << endl;

cout << "\nSeleect you choice : ";

cin >> option;

switch (option)

{

case 1:

Graph();

break;

case 2:

obj1.DFS\_Traversal();

break;

case 3:

obj2.BFS\_Traversal();

break;

case 0:

exit(0);

break;

default:

cout << "Invalid Input...!" << endl;

break;

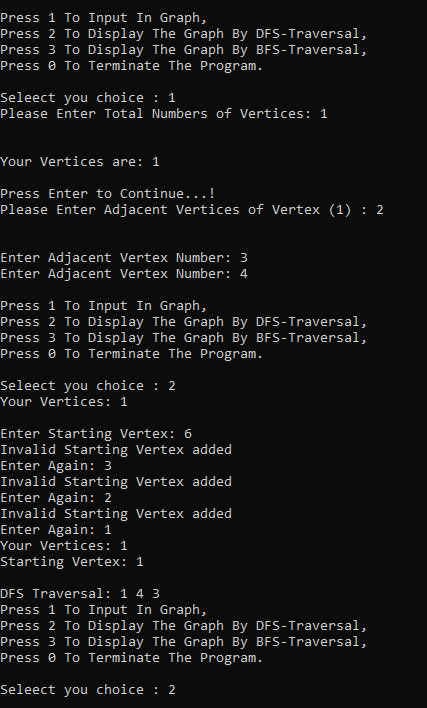
}

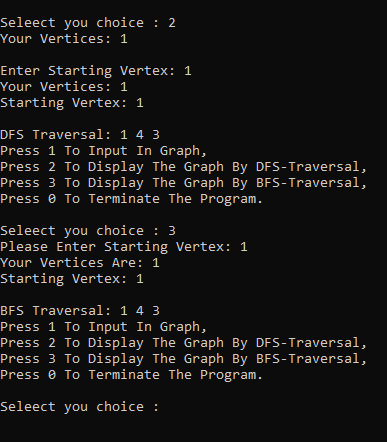
} while (option != 0);

return 0;

}

# Output:





# TASK 4:

#include<iostream>

using namespace std;

class minheap

{

int\* arr;

int cap;

int size;

public:

minheap(int c)

{

size = 0;

cap = c;

arr = new int[cap];

cout << "Heap Is Created." << endl;

}

void print()

{

cout << "\n------Heap-------\n";

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

cout << arr[i] << " ";

cout << endl;

}

int parent(int i)

{

return ((i - 1) / 2);

}

int left(int i)

{

return ((2 \* i) + 1);

}

int right(int i)

{

return ((2 \* i) + 2);

}

void swap(int &arr, int& a)

{

int num;

num = arr;

arr = a;

a = num;

}

void insert(int val)

{

if (size == cap)

{

cout << "OverFlow." << endl;

return;

}

size++;

int temp = size - 1;

arr[temp] = val;

while (temp != 0 && arr[parent(temp)] > arr[temp])

{

swap(arr[temp], arr[parent(temp)]);

temp = parent(temp);

}

}

void SwapLast(int num, int newval)

{

arr[num] = newval;

while (num != 0 && arr[parent(num)] > arr[num])

{

swap(arr[num], arr[parent(num)]);

num = parent(num);

}

}

int Rearrange()

{

int num;

if (size<=0)

return INT\_MIN;

if (size == 1)

{

size--;

return arr[0];

}

num = arr[0];

arr[0] = arr[size - 1];

size--;

MinHeapify(0);

return num;

}

void MinHeapify(int val)

{

int l\_val = left(val);

int r\_val = right(val);

int smallest = val;

if (l\_val < size && arr[l\_val] < arr[val])

smallest = l\_val;

if (r\_val < size && arr[r\_val] < arr[smallest])

smallest = r\_val;

if (smallest != val)

{

swap(arr[val], arr[smallest]);

MinHeapify(smallest);

}

}

void deletekey(int num)

{

int min = -1;

SwapLast(num, min);

Rearrange();

cout << "Smallest Number Is Deleted In Heap : " << endl;

}

};

int main()

{

int num, val, option;

cout << "Enter Size Of Heap : ";

cin >> num;

minheap obj(num);

do

{

cout << "Select Option, Otherwise,\n" << endl;

cout << "1). Insertion," << endl;

cout << "2). Delete," << endl;

cout << "3). Display," << endl;

cout << "0). Exit.\n" << endl;

cin >> option;

switch (option)

{

case 1:

{

cout << "\nEnter Value To Insertion In Heap : ";

cin >> val;

obj.insert(val);

break;

}

case 2:

{

obj.deletekey(0);

break;

}

case 3:

{

cout << endl;

obj.print();

break;

}

case 0:

break;

default:

cout << "\nWrong Input!------- Try Again--------" << endl;

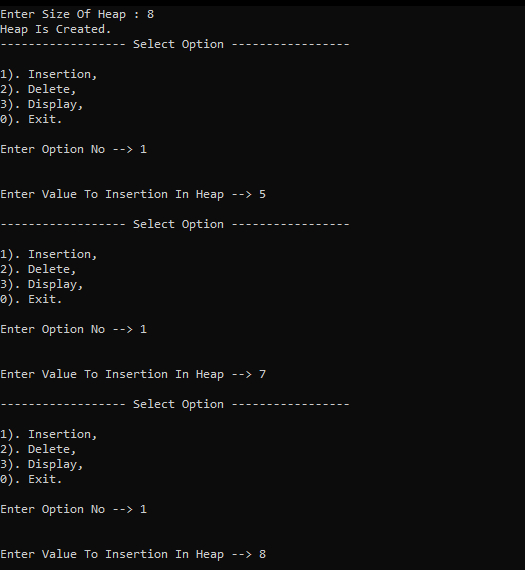
}

} while (true);

return 0;

}

# Output:



Text

Description automatically generated

Text

Description automatically generated

