**Muhammad Abdullah**

**SE(3A) | 19F-0916**

Data Structures Lab

Heap Sort

**Question # 1:**

**PROGRAM:**

#include <iostream>

using namespace std;

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

void Delete(int temp)

{

Decrease\_Key(temp, 0);

Extract\_Min();

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

int Left(int temp)

{

return (2 \* temp + 1);

}

int Right(int temp)

{

return (2 \* temp + 2);

}

void Heapify(int temp)

{

int Lowest = temp;

int Left\_Node = Left(temp);

int Right\_Node = Right(temp);

if (Left\_Node < Size && Array[Left\_Node] < Array[temp])

{

Lowest = Left\_Node;

}

if (Right\_Node < Size && Array[Right\_Node] < Array[Lowest])

{

Lowest = Right\_Node;

}

if (Lowest != temp)

{

swap(&Array[temp], &Array[Lowest]);

Heapify(Lowest);

}

}

void Decrease\_Key(int temp, int New)

{

Array[temp] = New;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

int Extract\_Min()

{

if (Size <= 0)

{

return -1;

}

if (Size == 1)

{

Size--;

return Array[0];

}

int Root = Array[0];

Array[0] = Array[Size - 1];

Size--;

Heapify(0);

return Root;

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

int main()

{

int choice = 0, Size = 0, Value = 0, i = 1;

cout << endl << " Enter the Size of the Heap : ";

cin >> Size;

Heap heap(Size);

while (i != 0)

{

system("cls");

cout << "---------------------------------------------" << endl;

cout << " Press 1 to Insert a Node !" << endl;

cout << " Press 2 to Delete a Node !" << endl;

cout << " Press 3 to Get Root Value !" << endl;

cout << " Press 4 to To Decrease a Value !" << endl;

cout << " Press 0 to Exit from Program !" << endl;

cout << "---------------------------------------------" << endl;

cout << endl << " Option Choosed : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl<<"Enter Value To Make a Node : ";

cin >> Value;

heap.Insert(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Value To Delete a Node : ";

cin >> Value;

heap.Delete(Value);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Root Value is : ";

cout << heap.Extract\_Min();

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl<<"Enter the Value to Change : ";

cin >> Value;

int V;

cout << "Enter the New Value : ";

cin >> V;

heap.Decrease\_Key(Value, V);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You Have Exited From the System !" << endl;

i = 0;

system("Pause");

break;

}

default:

{

cout << "Invalid Entry Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

A picture containing text, monitor, screenshot, indoor

Description automatically generated

**Menu Driven Program:**

A picture containing text, screenshot, monitor, indoor

Description automatically generated

**ROOT VALUE**

A picture containing text, screenshot, monitor, indoor

Description automatically generated

**Question # 2:**

**PROGRAM:**

#include <iostream>

using namespace std;

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

void Delete(int temp)

{

Decrease\_Key(temp, 0);

Extract\_Min();

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

int Left(int temp)

{

return (2 \* temp + 1);

}

int Right(int temp)

{

return (2 \* temp + 2);

}

void Heapify(int temp[], int Temp)

{

int Highest = Temp;

int Left\_Node = Left(Temp);

int Right\_Node = Right(Temp);

if (Left\_Node < Size && Array[Left\_Node] > Array[Highest])

{

Highest = Left\_Node;

}

if (Right\_Node < Size && Array[Right\_Node] > Array[Highest])

{

Highest = Right\_Node;

}

if (Highest != Temp)

{

swap(&Array[Temp], &Array[Highest]);

Heapify(Array,Highest);

}

}

void Decrease\_Key(int temp, int New)

{

Array[temp] = New;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

int Extract\_Min()

{

if (Size <= 0)

{

return -1;

}

if (Size == 1)

{

Size--;

return Array[0];

}

int Root = Array[0];

Array[0] = Array[Size - 1];

Size--;

Heapify(Array, 0);

return Root;

}

void Heap\_Sort(int Array[], int temp)

{

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

Heapify(Array,temp);

for (int i = temp - 1; i >= 0; i--) {

swap(&Array[0], &Array[i]);

Heapify(Array, 0);

}

}

void Print\_Sort(int Array[], int temp)

{

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

cout << Array[i] << " ";

cout << endl;

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

int main()

{

int choice = 0, Size = 0, Value = 0, i = 1;

cout << endl << " Enter the Size of the Heap : ";

cin >> Size;

Heap heap(Size);

while (i != 0)

{

system("cls");

cout << "---------------------------------------------" << endl;

cout << " Press 1 to Insert a Node !" << endl;

cout << " Press 2 to Delete a Node !" << endl;

cout << " Press 3 to Get Root Value !" << endl;

cout << " Press 4 to Decrease a Value !" << endl;

cout << " Press 5 to Perfrom Heap Sort !" << endl;

cout << " Press 6 to Dsplay Sort Values !" << endl;

cout << " Press 0 to Exit from Program !" << endl;

cout << "---------------------------------------------" << endl;

cout << endl << " Option Choosed : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl<<"Enter Value To Make a Node : ";

cin >> Value;

heap.Insert(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Value To Delete a Node : ";

cin >> Value;

heap.Delete(Value);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl<<"Root Value is : ";

cout << heap.Extract\_Min();

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl<<"Enter the Value to Change : ";

cin >> Value;

int V;

cout << "Enter the New Value : ";

cin >> V;

heap.Decrease\_Key(Value, V);

cout << endl;

system("pause");

break;

}

case 5:

{

cout << endl ;

heap.Heap\_Sort(heap.Array, heap.Size);

cout << endl;

system("pause");

break;

}

case 6:

{

cout << endl << " Value is/are : ";

heap.Print\_Sort(heap.Array, heap.Size);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You Have Exited From the System !" << endl;

i = 0;

system("Pause");

break;

}

default:

{

cout << "Invalid Entry Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**Initializing**

A picture containing text, monitor, indoor, screenshot

Description automatically generated

**Main Values**

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**Sorted Values**

**A picture containing text, monitor, indoor, electronics

Description automatically generated**

**Question # 3:**

**PROGRAM:**

#include <iostream>

using namespace std;

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

void Delete(int temp)

{

Decrease\_Key(temp, 0);

Extract\_Min();

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

int Left(int temp)

{

return (2 \* temp + 1);

}

int Right(int temp)

{

return (2 \* temp + 2);

}

void Heapify(int temp)

{

int Lowest = temp;

int Left\_Node = Left(temp);

int Right\_Node = Right(temp);

if (Left\_Node < Size && Array[Left\_Node] < Array[temp])

{

Lowest = Left\_Node;

}

if (Right\_Node < Size && Array[Right\_Node] < Array[Lowest])

{

Lowest = Right\_Node;

}

if (Lowest != temp)

{

swap(&Array[temp], &Array[Lowest]);

Heapify(Lowest);

}

}

void Decrease\_Key(int temp, int New)

{

Array[temp] = New;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

int Extract\_Min()

{

if (Size <= 0)

{

return -1;

}

if (Size == 1)

{

Size--;

return Array[0];

}

int Root = Array[0];

Array[0] = Array[Size - 1];

Size--;

Heapify(0);

return Root;

}

void Largest\_Values(int Array[], int size, int No\_of\_Values)

{

Heap \*heap = new Heap(No\_of\_Values);

for (int i = No\_of\_Values; i < size; i++)

{

if (Array[0] > Array[i])

continue;

else

{

Array[0] = Array[i];

heap->Heapify(0);

}

}

cout << endl << "Required Largest Value/s : ";

for (int i = 0; i < No\_of\_Values; i++)

{

cout << Array[i] << " ";

}

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

int main()

{

int choice = 0, Size = 0, Value = 0, i = 1;

cout << endl << " Enter the Size of the Heap : ";

cin >> Size;

Heap heap(Size);

while (i != 0)

{

system("cls");

cout << "---------------------------------------------" << endl;

cout << " Press 1 to Insert a Node !" << endl;

cout << " Press 2 to Delete a Node !" << endl;

cout << " Press 3 to Get Root Value !" << endl;

cout << " Press 4 to To Decrease a Value !" << endl;

cout << " Press 5 to To Get Largest Value/s !" << endl;

cout << " Press 0 to Exit from Program !" << endl;

cout << "---------------------------------------------" << endl;

cout << endl << " Option Choosed : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value To Make a Node : ";

cin >> Value;

heap.Insert(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl << "Enter Value To Delete a Node : ";

cin >> Value;

heap.Delete(Value);

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl << "Root Value is : ";

cout << heap.Extract\_Min();

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Enter the Value to Change : ";

cin >> Value;

int V;

cout << "Enter the New Value : ";

cin >> V;

heap.Decrease\_Key(Value, V);

cout << endl;

system("pause");

break;

}

case 5:

{

cout << endl << "Enter Number of Largest Value/s you Want : ";

cin >> Value;

heap.Largest\_Values(heap.Array, heap.Size, Value);

cout << endl << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You Have Exited From the System !" << endl;

i = 0;

system("Pause");

break;

}

default:

{

cout << "Invalid Entry Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**VALUES ADDED WERE 6,7,8,9 and Ans is:**

**A picture containing text, screenshot, monitor, electronics

Description automatically generated**

**Question # 4:**

**PROGRAM:**

#include <iostream>

using namespace std;

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

int Max\_Heap(int Array[], int temp)

{

for (int i = 0; i <= (temp - 2) / 2; i++)

{

if (Array[2 \* i + 1] > Array[i])

return 0;

if (2 \* i + 2 < temp && Array[2 \* i + 2] > Array[i])

return 0;

}

return 1;

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

int main()

{

int choice = 0, Size = 0, Value = 0, i = 1;

cout << endl << " Enter the Size of the Heap : ";

cin >> Size;

Heap heap(Size);

while (i != 0)

{

system("cls");

cout << "------------------------------------------------" << endl;

cout << " Press 1 to Insert a Node !" << endl;

cout << " Press 2 to Check Either it is Max Heap or Not !" << endl;

cout << " Press 0 to Exit from Program !" << endl;

cout << "------------------------------------------------" << endl;

cout << endl << " Option Choosed : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value To Make a Node : ";

cin >> Value;

heap.Insert(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl;

int a;

a = heap.Max\_Heap(heap.Array, heap.Size);

if (a == 1)

{

cout << endl << "Yes, This is Max Heap !" << endl;

}

else

{

cout << endl << "No, This is Not Max Heap !" << endl;

}

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You Have Exited From the System !" << endl;

i = 0;

system("Pause");

break;

}

default:

{

cout << "Invalid Entry Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE MENU**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**CHECK BY ENTRING VALUES…ANS IS NO BECAUSE IT IS A MIN HEAP:**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**Question # 5:**

**PROGRAM:**

#include <iostream>

using namespace std;

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

int Size\_of\_Tree(Node\* Root)

{

if (Root == NULL)

return 0;

return 1 + Size\_of\_Tree(Root->Left\_Node) + Size\_of\_Tree(Root->Right\_Node);

}

int Heap\_or\_Not(Node\* Root, int i, int n)

{

if (Root == NULL)

return 1;

if (i >= n)

return 0;

if ((Root->Left\_Node && Root->Left\_Node->Data <= Root->Data) || (Root->Right\_Node && Root->Right\_Node->Data <= Root->Data))

return 0;

return Heap\_or\_Not(Root->Left\_Node, 2 \* i + 1, n) && Heap\_or\_Not(Root->Right\_Node, 2 \* i + 2, n);

}

bool Heap\_or\_Not(Node\* Root)

{

int i = 0;

return Heap\_or\_Not(Root, i, Size\_of\_Tree(Root));

}

};

int main()

{

Bst\_Tree Tree;

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "-------------------------------------------------" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Check Either it is Heap or Not !" << endl;

cout << " Press 0 to Exit through System !" << endl;

cout << "-------------------------------------------------" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl;

int a = Tree.Heap\_or\_Not(Tree.Root);

if (a == 0)

{

cout << endl << "Given Tree is Not a Heap !!" << endl;

}

else

{

cout << endl << "Given Tree is a Heap !!" << endl;

}

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE**

**A picture containing text, monitor, electronics, screenshot

Description automatically generated**

**Entered Values Were 5,4,3 hence it is a Heap**

**A picture containing text, electronics, monitor, screenshot

Description automatically generated**

**Question # 6:**

**PROGRAM:**

#include <iostream>

using namespace std;

int Checker\_Array[10] = { 0 };

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

void Print(int Array[], int temp)

{

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

cout << Array[i] << " ";

cout << endl;

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

Root = NULL;

}

Node \*Root;

Node \*Insertion(Node \*Current, int data) // Insertion of Nodes

{

if (Current == NULL)

{

Current = new Node;

Current->Data = data;

Current->Left\_Node = NULL;

Current->Right\_Node = NULL;

return Current;

}

else if (data < Current->Data) // If Data is lesser than Root

{

Current->Left\_Node = Insertion(Current->Left\_Node, data);

}

else if (data >= Current->Data) // If Data is greater than Root

{

Current->Right\_Node = Insertion(Current->Right\_Node, data);

}

return Current;

}

void In\_Order(Node \*C, int i) // In order traversal

{

if (Root == NULL)

{

cout << endl << "Tree is Empty !" << endl;

return;

}

else

{

if (C == NULL)

return;

In\_Order(C->Left\_Node, i);

cout << C->Data << " ";

Checker\_Array[i++] = C->Data;

In\_Order(C->Right\_Node, i);

}

}

};

int main()

{

Bst\_Tree Tree;

int a = sizeof Checker\_Array;

Heap heap(a);

int choice = 0, value = 0, i = 1;

while (i != 0)

{

system("cls");

cout << "-------------------------------------------------" << endl;

cout << " Press 1 to Enter a Node !" << endl;

cout << " Press 2 to Convert BST into Min Heap !" << endl;

cout << " Press 0 to Exit through System !" << endl;

cout << "-------------------------------------------------" << endl;

cout << " Enter Choice : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value for the Node :";

cin >> value;

Tree.Root = Tree.Insertion(Tree.Root, value);

system("pause");

break;

}

case 2:

{

cout << endl << "In Order of BST is : ";

Tree.In\_Order(Tree.Root, 0);

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

{

heap.Insert(Checker\_Array[i]);

}

cout << endl << "BST Converted into HEAP !" << endl;

cout << "Now, Data of Heap is : ";

heap.Print(heap.Array, heap.Size);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You have exited from the system !" << endl;

i = 0;

break;

}

default:

cout << endl << "Invalid Entry !" << endl;

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE MENU**

**A picture containing text, monitor, screenshot, indoor

Description automatically generated**

**After Converting it into HEAP**

**A picture containing text, monitor, screenshot, electronics

Description automatically generated**

**Question # 7:**

**PROGRAM:**

#include <iostream>

using namespace std;

class Heap

{

public:

int \*Array;

int Storage;

int Size;

Heap(int storage)

{

Storage = storage;

Array = new int[Storage];

Size = 0;

}

void Insert(int index)

{

if (Size == Storage)

{

cout << endl << " Storage is Full. Error = Overflow !!" << endl;

return;

}

int temp = ++Size - 1;

Array[temp] = index;

while (temp != 0 && Array[Parent(temp)] > Array[temp])

{

swap(&Array[temp], &Array[Parent(temp)]);

temp = Parent(temp);

}

}

void Heapify(int Array[], int Counter ,int Size)

{

int Highest = Size;

int Left\_Node = Left(Size);

int Right\_Node = Right(Size);

if (Left\_Node < Size && Array[Left\_Node] > Array[Size])

{

Highest = Left\_Node;

}

if (Right\_Node < Size && Array[Right\_Node] > Array[Highest])

{

Highest = Right\_Node;

}

if (Highest != Size)

{

swap(&Array[Size], &Array[Highest]);

Heapify(Array, Highest, Size);

}

}

int Parent(int temp)

{

return (temp - 1) / 2;

}

int Left(int temp)

{

return (2 \* temp + 1);

}

int Right(int temp)

{

return (2 \* temp + 2);

}

void Min\_To\_Max(int Array[], int Size)

{

for (int i = (Size - 2) / 2; i >= 0; --i)

Heapify(Array, i, Size);

}

void Print(int Array[], int temp)

{

for (int i = temp - 1; i >= 0 ; i--)

cout << Array[i] << " ";

cout << endl;

}

void swap(int \*First, int \*Second)

{

int temp = \*First;

\*First = \*Second;

\*Second = temp;

}

};

int main()

{

int choice = 0, Size = 0, Value = 0, i = 1;

cout << endl << " Enter the Size of the Heap : ";

cin >> Size;

Heap heap(Size);

while (i != 0)

{

system("cls");

cout << "------------------------------------" << endl;

cout << " Press 1 to Insert a Node !" << endl;

cout << " Press 2 to Covert Min to Max !" << endl;

cout << " Press 3 to Print Values !" << endl;

cout << " Press 0 to Exit from Program !" << endl;

cout << "------------------------------------" << endl;

cout << endl << " Option Choosed : ";

cin >> choice;

switch (choice)

{

case 1:

{

cout << endl << "Enter Value To Make a Node : ";

cin >> Value;

heap.Insert(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl;

heap.Min\_To\_Max(heap.Array, heap.Size);

cout << "Min Heap Converted to Max Heap !!" << endl;

cout << endl << "Values Are : ";

heap.Print(heap.Array, heap.Size);

system("pause");

break;

}

case 3:

{

cout << endl << "Values Are : ";

heap.Print(heap.Array, heap.Size);

cout << endl;

system("pause");

break;

}

case 0:

{

cout << endl << "You Have Exited From the System !" << endl;

i = 0;

system("Pause");

break;

}

default:

{

cout << "Invalid Entry Try Again !" << endl;

system("pause");

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE MENU**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**

**After Converting Min to Max With Values:**

**A picture containing text, monitor, indoor, screenshot

Description automatically generated**