**Muhammad Abdullah**

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

DS LAB ASSIGNMENT

DS SFML ASSIGNMENT

**QUESTION # 1:  
PROGRAM:**

#include <iostream>

#include "SFML/Graphics.hpp"; // Including SFML

using namespace std;

using namespace sf;

RenderWindow window(VideoMode(1024, 768), " Singly Linked List"); // Making Window

Font font;

int x\_axis = 100;

int y\_axis = 200; // axis for later use

struct Node

{

int data; // Link List Structure

Node \*NextNode;

};

class Fun // class code

{

public:

Fun()

{

font.loadFromFile("comic.TTF"); // Load Font File

Head = NULL;

}

Node \*Head;

void Add\_Node\_First(int val);

void Add\_Node\_Last(int val);

void Add\_Node\_Random(int val);

void Delete\_Node\_First();

void Delete\_Node\_Last();

void Delete\_Node\_Random();

void Search\_Node\_Data(int val);

void Display();

void Display\_Graphics();

};

void Fun::Add\_Node\_First(int val) // To Add Node at first

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

temp->NextNode = current;

Head = temp;

}

}

void Fun::Add\_Node\_Last(int val) // To Add Node At Last

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->NextNode = NULL;

}

else

cout << endl << "!!! Create Link List First !!!" << endl;

}

void Fun::Add\_Node\_Random(int val) // To Add Node Randomly

{

Node \*temp = NULL, \*current = Head;

int opt = 0, counter = 0;

cout << endl << "Enter Postion to add a Node : ";

cin >> opt;

if (Head == NULL)

{

cout << endl << "Create List First !!" << endl;

}

else

{

Node \*temp1 = Head;

while (temp1 != NULL)

{

counter++;

temp1 = temp1->NextNode;

}

if (opt == 1)

{

if (Head == NULL)

{

temp = new Node;

temp->data = val;

temp->NextNode = NULL;

Head = temp;

}

else

{

temp = new Node;

temp->data = val;

current = Head;

Head = temp;

temp->NextNode = current;

}

}

else if (opt == counter)

{

while (current->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp->data = val;

current->NextNode = temp;

temp->NextNode = NULL;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp->data = val;

temp->NextNode = current->NextNode;

current->NextNode = temp;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Delete\_Node\_First() // To Delete Node from start

{

Node \*temp = NULL;

if (Head != NULL)

{

temp = new Node;

temp = Head;

Head = Head->NextNode;

free(temp);

cout << endl << "First Node has been deleted !" << endl;

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Last() // To Delete Node from End

{

Node \*temp = NULL, \*current = Head;

if (Head != NULL)

{

while (current->NextNode->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode->NextNode;

current->NextNode = NULL;

free(temp);

cout << endl << "Last Node has been deleted !" << endl;

}

else

cout << "Link List is Empty ! " << endl;

}

void Fun::Delete\_Node\_Random() // To delete Node Randomely

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

int opt = 0, counter = 0;

cout << "Enter the Position of Node to Delete it : ";

cin >> opt;

while (current != NULL)

{

counter++;

current = current->NextNode;

}

if (opt == 1)

{

temp = new Node;

temp = Head;

Head = Head->NextNode;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt == counter)

{

current = Head;

while (current->NextNode->NextNode != NULL)

{

current = current->NextNode;

}

temp = new Node;

temp = current->NextNode->NextNode;

current->NextNode = NULL;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp = current->NextNode;

current->NextNode = temp->NextNode;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

break;

}

current = current->NextNode;

}

}

}

}

void Fun::Search\_Node\_Data(int val) // To Search a Node

{

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

RectangleShape rectangle; // Making Rectangle

Text text; // Using to show values

Text found; // Using to show searched value

RectangleShape line; // Using to make connecting lines

Text singly; // Using to show SINGLY LINKED LIST

CircleShape triangle(6, 3); //Using to make Arrows

RectangleShape FL; //FL = Found Line (FOR searched value)

singly.setFont(font);

singly.setCharacterSize(50);

singly.setPosition(300, 50);

singly.setString("Singly Linked List"); // Setting singly

singly.setFillColor(Color::Cyan);

singly.setOutlineColor(Color::White);

singly.setOutlineThickness(2);

Node \*temp = Head;

window.clear();

int i = 0;

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4); // setting text for values

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->data));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis); // setting rectangle attributes

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18); // setting lines for connection

triangle.setPosition(x\_axis + 100, y\_axis + 13);

triangle.setFillColor(Color::White); // setting arrow position to point out

triangle.setRotation(90.f);

found.setFont(font);

found.setCharacterSize(25);

found.setFillColor(Color::White); // setting found for searched values

found.setOutlineColor(Color::White);

found.setOutlineThickness(1);

found.setString("Value Found");

found.setPosition(x\_axis - 35 , y\_axis + 75);

FL.setSize(Vector2f(50, 2)); // for searched line

FL.setFillColor(Color::White);

FL.setRotation(90.f);

FL.setPosition(x\_axis + 35, y\_axis + 20);

if (temp->data % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else // even odd difference

rectangle.setFillColor(Color::Red);

if (temp->data == val)

{

window.draw(FL); // showing data if found

window.draw(found);

i = 1;

}

window.draw(singly);

window.draw(rectangle);

window.draw(text); // drawing the required graphics

window.draw(line);

window.draw(triangle);

temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black); // setting for NULL text

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White); // setting to store NULL Text

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100; // setting default value of x

window.display();

system("pause");

}

}

void Fun::Display()

{

Node \*temp = Head; // Displaying in Console

cout << endl << "DATA : ";

while (temp != NULL)

{

cout << " " << temp->data << " ";

temp = temp->NextNode;

}

cout << endl;

}

void Fun::Display\_Graphics() // Displaying in SFML

{

RectangleShape rectangle;

Text text;

RectangleShape line;

Text singly;

CircleShape triangle(6, 3);

singly.setFont(font);

singly.setCharacterSize(50);

singly.setPosition(300, 50);

singly.setString("Singly Linked List");

singly.setFillColor(Color::Cyan);

singly.setOutlineColor(Color::White);

singly.setOutlineThickness(2);

Node \*temp = Head;

window.clear();

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->data));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18);

triangle.setPosition(x\_axis + 100, y\_axis + 13);

triangle.setFillColor(Color::White);

triangle.setRotation(90.f);

if (temp->data % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else

rectangle.setFillColor(Color::Red);

window.draw(singly);

window.draw(rectangle);

window.draw(text);

window.draw(line);

window.draw(triangle);

temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5 , y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100;

window.display();

system("pause");

}

int main()

{

Fun List;

int opt = 1, val = 0;

while (opt != 0)

{

system("cls");

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

cout << " | Press 1 to Add Node on Start |" << endl;

cout << " | Press 2 to Add Node on Last |" << endl;

cout << " | Press 3 to Add Node Randomly |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 4 to Delete First Node |" << endl;

cout << " | Press 5 to Delete Last Node |" << "\t\t\tSINGLY LINK LIST" << endl;

cout << " | Press 6 to Delete Random Node |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 7 to Search Data in Node |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 9 to Display Link List |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 10 to Display Graphics |" << endl << " |\t\t\t\t |" << endl;

cout << " | Press 0 to Exit from the system |" << endl;

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

cout << endl << " Option Choosen : ";

cin >> opt;

cout << endl;

switch (opt)

{

case 1:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_First(val);

break;

}

case 2:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Last(val);

break;

}

case 3:

{

cout << "Enter any data to Enter in List : ";

cin >> val;

List.Add\_Node\_Random(val);

break;

}

case 4:

{

List.Delete\_Node\_First();

cout << endl;

system("pause");

break;

}

case 5:

{

List.Delete\_Node\_Last();

cout << endl;

system("pause");

break;

}

case 6:

{

List.Delete\_Node\_Random();

cout << endl;

system("pause");

break;

}

case 7:

{

cout << "Enter value to search in Link List : ";

cin >> val;

List.Search\_Node\_Data(val);

cout << endl;

system("pause");

break;

}

case 9:

{

List.Display();

cout << endl;

system("pause");

break;

}

{

case 10:

List.Display\_Graphics();

cout << endl;

system("pause");

break;

}

case 0:

{

opt = 0;

cout << endl << "You have exited from Link List !" << endl;

break;

}

default:

cout << "Invalid Entry, Press any key to try again !!" << endl;

system("pause");

break;

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE**

A picture containing text, monitor, computer, screenshot

Description automatically generated

**DISPLAYING LINKED LIST**

A picture containing text, monitor, indoor, computer

Description automatically generated

**SEARCHING IN LINKED LIST**

A picture containing text, monitor, computer, computer

Description automatically generated

**QUESTION # 2:  
PROGRAM:**

#include <iostream>

#include "SFML/Graphics.hpp"; // Including SFML

using namespace std;

using namespace sf;

RenderWindow window(VideoMode(1024, 768), " QUEUES"); // Making Window

Font font;

int x\_axis = 100;

int y\_axis = 200; // axis for later use

struct Node

{

int Value;

Node \*NextNode; // Initillizing Elements

};

class Queue

{

public:

Node \*front, \*rear;

Queue()

{

font.loadFromFile("comic.TTF");

front = NULL, rear = NULL;

}

void enqueue(int value); //ENQUEUE FUNCTION

void dequeue(); //DEQUEUE FUNCTION

void Search\_Node\_Data(int val);

void display(); // Display Function for ease

void Display\_Graphics(); // Display with graphics

};

void Queue::enqueue(int value)

{

Node \*temp = new Node; // Enqueue Function

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

{

temp->Value = value;

temp->NextNode = NULL;

front = temp;

rear = temp;

}

else

{

temp->Value = value;

temp->NextNode = NULL;

rear->NextNode = temp;

rear = temp;

}

}

void Queue::dequeue() // Dequeue Function

{

Node \*temp = new Node;

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

{

cout << endl << "Following Value is Dequeued : " << front->Value;

temp = front;

front = front->NextNode;

free(temp);

}

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

{

cout << "Queue is Empty" << endl;

}

else if (front == rear)

{

cout << endl << "Following Value is Dequeued : " << front->Value;

temp = front;

free(temp);

front = NULL;

rear = NULL;

}

}

void Queue::display() // Displaying Queue

{

Node \*temp = front;

if (temp == NULL)

{

cout << "Queue is empty" << endl;

}

else

{

while (temp != NULL)

{

cout << temp->Value << " ";

temp = temp->NextNode;

}

}

}

void Queue::Search\_Node\_Data(int val) // To Search a Node

{

if (front == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

RectangleShape rectangle; // Making Rectangle

Text text; // Using to show values

Text found; // Using to show searched value

RectangleShape line; // Using to make connecting lines

Text singly; // Using to show SINGLY LINKED LIST

CircleShape triangle(6, 3); //Using to make Arrows

RectangleShape FL; //FL = Found Line (FOR searched value)

singly.setFont(font);

singly.setCharacterSize(50);

singly.setPosition(250, 50);

singly.setString("! QUEUES Implementation !"); // Setting singly

singly.setFillColor(Color::Cyan);

singly.setOutlineColor(Color::White);

singly.setOutlineThickness(2);

Node \*temp = front;

window.clear();

int i = 0;

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4); // setting text for values

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->Value));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis); // setting rectangle attributes

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18); // setting lines for connection

triangle.setPosition(x\_axis + 70, y\_axis + 24);

triangle.setFillColor(Color::White);

triangle.setRotation(270.f);

found.setFont(font);

found.setCharacterSize(25);

found.setFillColor(Color::White); // setting found for searched values

found.setOutlineColor(Color::White);

found.setOutlineThickness(1);

found.setString("Value Found");

found.setPosition(x\_axis - 35, y\_axis + 75);

FL.setSize(Vector2f(50, 2)); // for searched line

FL.setFillColor(Color::White);

FL.setRotation(90.f);

FL.setPosition(x\_axis + 35, y\_axis + 20);

if (temp->Value % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else // even odd difference

rectangle.setFillColor(Color::Red);

if (temp->Value == val)

{

window.draw(FL); // showing data if found

window.draw(found);

i = 1;

}

window.draw(singly);

window.draw(rectangle);

window.draw(text); // drawing the required graphics

window.draw(line);

window.draw(triangle);

temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black); // setting for NULL text

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White); // setting to store NULL Text

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100; // setting default value of x

window.display();

system("pause");

}

}

void Queue::Display\_Graphics() // Displaying in SFML

{

RectangleShape rectangle;

Text text;

RectangleShape line;

Text queue;

CircleShape triangle(6, 3);

Text Prime;

RectangleShape PL;

queue.setFont(font);

queue.setCharacterSize(50);

queue.setPosition(250, 50);

queue.setString("! Queue Implementation !");

queue.setFillColor(Color::Cyan);

queue.setOutlineColor(Color::White);

queue.setOutlineThickness(2);

Node \*temp = front;

window.clear();

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->Value));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18);

triangle.setPosition(x\_axis + 70, y\_axis + 24);

triangle.setFillColor(Color::White);

triangle.setRotation(270.f);

Prime.setFont(font);

Prime.setCharacterSize(25);

Prime.setFillColor(Color::White); // setting Prime for searched values

Prime.setOutlineColor(Color::White);

Prime.setOutlineThickness(1);

Prime.setString("Prime");

Prime.setPosition(x\_axis , y\_axis + 65);

PL.setSize(Vector2f(50, 2)); // for searched line

PL.setFillColor(Color::White);

PL.setRotation(90.f);

PL.setPosition(x\_axis + 35, y\_axis + 20);

if (temp->Value % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else

rectangle.setFillColor(Color::Red);

bool isPrime = true;

if (temp->Value == 0 || temp->Value == 1)

{

isPrime = false;

}

else

{

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

{

if (temp->Value % i == 0)

{

isPrime = false;

break;

}

}

}

if (isPrime)

{

window.draw(Prime);

window.draw(PL);

}

window.draw(queue);

window.draw(rectangle);

window.draw(text);

window.draw(line);

window.draw(triangle);

temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100;

window.display();

system("pause");

}

int main()

{

Queue queue;

int value;

int choice = 1;

int Opt;

while (choice != 0)

{

system("cls");

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

cout << "| Press 1 To Enqueue |" << endl;

cout << "| Press 2 To Dequeue |" << endl << "|\t\t\t |" << endl;

cout << "| Press 3 To Search a Node |" << endl << "|\t\t\t |\t\t\tQueues Implementation" << endl; //Menu Driven Program

cout << "| Press 4 To Display |" << endl;

cout << "| Press 5 To Display Graphics |" << endl << "|\t\t\t |" << endl;

cout << "| Press 0 To Exit |" << endl;

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

cout << "Enter Your Option : ";

cin >> Opt;

switch (Opt)

{

case 1:

{

cout << endl << "Enter Value to Enqueue : ";

cin >> value;

queue.enqueue(value);

cout << endl;

system("pause");

break;

}

case 2:

{

cout << endl;

queue.dequeue();

cout << endl << endl;

system("pause");

break;

}

case 3:

{

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

cin >> value;

queue.Search\_Node\_Data(value);

cout << endl;

system("pause");

break;

}

case 4:

{

cout << endl << "Displaying : ";

queue.display();

cout << endl << endl;

system("pause");

break;

}

case 5:

{

queue.Display\_Graphics();

cout << endl;

system("pause");

break;

}

case 0:

{

choice = 0;

break;

}

default:

{

cout << "Invalid Entry" << endl;

break;

}

}

}

cout << endl << endl;

system("pause");

}

**MAIN CODE**

A picture containing text, monitor, computer, screenshot

Description automatically generated

**DISPLAYING VALUES**

A picture containing text, indoor, monitor, computer

Description automatically generated

**SEARCHING A NODE**

A picture containing text, monitor, indoor, screenshot

Description automatically generated

**QUESTION # 3:  
PROGRAM:**

#include <iostream>

#include "SFML/Graphics.hpp"; // Including SFML

using namespace std;

using namespace sf;

RenderWindow window(VideoMode(1024, 768), " QUEUES"); // Making Window

Font font;

int x\_axis = 100;

int y\_axis = 200; // axis for later use

struct Node //Link List based Implementation of Stack

{

int value;

Node \*Next\_Node = NULL;

};

class Stack // Creating Class

{

public:

Stack()

{

font.loadFromFile("comic.TTF");

Head = NULL;

}

Node \*Head;

void push(int Value); //push function

int pop(); //pop function

void Delete\_Node\_Random();

void Search\_Node\_Data(int val);

void Display(); // This is just to confirm output

void Display\_Graphics();

};

void Stack::push(int Value) //Push Function

{

if (Head == NULL)

{

Node \*temp = new Node;

temp->value = Value;

temp->Next\_Node = NULL;

Head = temp;

}

else

{

Node \*temp = new Node;

temp->value = Value;

temp->Next\_Node = Head;

Head = temp;

}

}

int Stack::pop() //Pop Function

{

if (Head != NULL)

{

Node \*temp = new Node;

int tem;

temp = Head;

Head = Head->Next\_Node;

cout << endl << temp->value << " is Poped !" << endl;

tem = temp->value;

free(temp);

return tem;

}

else

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

}

void Stack::Display() //Displaying for Checking

{

Node \*temp = Head;

if (temp != NULL)

{

cout << "Values = ";

while (temp != NULL)

{

cout << temp->value << " ";

temp = temp->Next\_Node;

}

}

else

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

}

void Stack::Delete\_Node\_Random() // To delete Node Randomely

{

Node \*temp = NULL, \*current = Head;

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

int opt = 0, counter = 0;

cout << "Enter the Position of Node to Delete it : ";

cin >> opt;

while (current != NULL)

{

counter++;

current = current->Next\_Node;

}

if (opt == 1)

{

temp = new Node;

temp = Head;

Head = Head->Next\_Node;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt == counter)

{

current = Head;

while (current->Next\_Node->Next\_Node != NULL)

{

current = current->Next\_Node;

}

temp = new Node;

temp = current->Next\_Node->Next\_Node;

current->Next\_Node = NULL;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

}

else if (opt > 1 && opt < counter)

{

current = Head;

for (int i = 2; i < counter - 1; i++)

{

if (i == opt)

{

temp = new Node;

temp = current->Next\_Node;

current->Next\_Node = temp->Next\_Node;

free(temp);

cout << endl << opt << " Node has been deleted !" << endl;

break;

}

current = current->Next\_Node;

}

}

}

}

void Stack::Search\_Node\_Data(int val) // To Search a Node

{

if (Head == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

RectangleShape rectangle; // Making Rectangle

Text text; // Using to show values

Text found; // Using to show searched value

RectangleShape line; // Using to make connecting lines

Text queue; // Using to show Stack with LINKED LIST

CircleShape triangle(6, 3); //Using to make Arrows

RectangleShape FL; //FL = Found Line (FOR searched value)

queue.setFont(font);

queue.setCharacterSize(50);

queue.setPosition(250, 50);

queue.setString("! Stack Implementation !");

queue.setFillColor(Color::Cyan);

queue.setOutlineColor(Color::White);

queue.setOutlineThickness(2);

Node \*temp = Head;

window.clear();

int i = 0;

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 380, y\_axis + 35);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->value));

rectangle.setSize(Vector2f(80, 40));

rectangle.setPosition(x\_axis + 350, y\_axis + 30);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(55, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 387, y\_axis - 27);

line.setRotation(90.f);

triangle.setPosition(x\_axis + 380, y\_axis - 27);

triangle.setFillColor(Color::White);

found.setFont(font);

found.setCharacterSize(25);

found.setFillColor(Color::White); // setting found for searched values

found.setOutlineColor(Color::White);

found.setOutlineThickness(1);

found.setString("Value Found");

found.setPosition(x\_axis + 150, y\_axis + 35);

FL.setSize(Vector2f(50, 2)); // for searched line

FL.setFillColor(Color::White);

FL.setPosition(x\_axis + 298, y\_axis + 50);

if (temp->value % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else // even odd difference

rectangle.setFillColor(Color::Red);

if (temp->value == val)

{

window.draw(FL); // showing data if found

window.draw(found);

i = 1;

}

window.draw(queue);

window.draw(rectangle);

window.draw(text); // drawing the required graphics

window.draw(line);

window.draw(triangle);

temp = temp->Next\_Node;

y\_axis = y\_axis + 100;

}

y\_axis = 100; // setting default value of x

window.display();

system("pause");

}

}

void Stack::Display\_Graphics() // Displaying in SFML

{

RectangleShape rectangle;

Text text;

RectangleShape line;

Text queue;

CircleShape triangle(6, 3);

queue.setFont(font);

queue.setCharacterSize(50);

queue.setPosition(250, 50);

queue.setString("! Stack Implementation !");

queue.setFillColor(Color::Cyan);

queue.setOutlineColor(Color::White);

queue.setOutlineThickness(2);

Node \*temp = Head;

window.clear();

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 380, y\_axis + 35);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->value));

rectangle.setSize(Vector2f(80, 40));

rectangle.setPosition(x\_axis + 350, y\_axis + 30);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(55, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 387, y\_axis - 27);

line.setRotation(90.f);

triangle.setPosition(x\_axis + 380, y\_axis - 27);

triangle.setFillColor(Color::White);

if (temp->value % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else

rectangle.setFillColor(Color::Red);

window.draw(queue);

window.draw(rectangle);

window.draw(text);

window.draw(line);

window.draw(triangle);

temp = temp->Next\_Node;

y\_axis = y\_axis + 100;

}

y\_axis = 100;

window.display();

system("pause");

}

int main()

{

int Opt = 0, Choice = 0, Value = 0;

Stack Object;

while (Choice != -1)

{

system("cls");

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

cout << "| Press 1 to Push Value in Stack |" << endl;

cout << "| Press 2 to Pop Value from Stack |" << endl << "|\t\t\t\t\t|" << endl;

cout << "| Press 3 to Delete Value from Stack |" << endl << "|\t\t\t\t\t|" << endl;

cout << "| Press 4 to Search Values in Stack |" << endl << "|\t\t\t\t\t|\t\t\tStack Implementation" << endl;

cout << "| Press 5 to Display Stack |" << endl;

cout << "| Press 6 to Display Stack Graphically |" << endl << "|\t\t\t\t\t|" << endl;

cout << "| Press 0 to Exit from the Program |" << endl;

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

cout << endl << "Enter Choice : ";

cin >> Opt;

switch (Opt)

{

case 1:

{

cout << endl << "Enter Value in Stack : ";

cin >> Value;

Object.push(Value);

cout << endl;

system("pause");

break;

}

case 2:

{

Object.pop();

cout << endl;

system("pause");

break;

}

case 3:

{

cout << endl;

Object.Delete\_Node\_Random();

system("pause");

break;

}

case 4:

{

cout << endl << "Enter Value to Search : ";

cin >> Value;

Object.Search\_Node\_Data(Value);

system("pause");

break;

}

case 5:

{

cout << endl;

Object.Display();

system("pause");

break;

}

case 6:

{

Object.Display\_Graphics();

cout << endl;

system("pause");

break;

}

case 0:

{

Choice = -1;

break;

}

default:

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

system("pause");

}

}

cout << endl << endl;

system("pause");

}

**Main Code**

A picture containing text, monitor, screenshot, indoor

Description automatically generated

**DISPLAYING STACK**

A picture containing text, screenshot, monitor

Description automatically generated

**SEARCHING IN STACK**

A picture containing text, monitor, screenshot, black

Description automatically generated

**QUESTION # 4:  
PROGRAM:**

#include <iostream>

#include "SFML/Graphics.hpp"; // Including SFML

using namespace std;

using namespace sf;

RenderWindow window(VideoMode(1000, 600), " BST Implementation"); // Making Window

Font font;

int x\_axis = 250;

int y\_axis = 0; // axis for later use

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

font.loadFromFile("comicbd.TTF");

Root = NULL;

}

Node \*Root;

Text text;

RectangleShape line\_right;

RectangleShape line\_left;

Text BST;

CircleShape circle;

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

void Delete\_Node(Node \*root, int Key); // Deletion of Nodes

void Bst\_Tree::Search\_Node\_Data(int val); // Searching of Nodes

void Bst\_Tree::Display\_Graphics(Node \*temp, int L ); // Graphical Representation

};

Node \*Bst\_Tree::Insertion(Node \*Current, int data)

{

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 Bst\_Tree::Delete\_Node(Node \*root, int Key) //Deleting Nodes

{

Node \*temp = root;

if (temp == NULL)

{

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

return;

}

else if (temp->Left\_Node->Data == Key && temp->Left\_Node != NULL)

{

Node \*temp1 = temp->Left\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Left\_Node = NULL;

free(temp1);

return;

}

else if (temp->Right\_Node->Data == Key && temp->Right\_Node != NULL)

{

Node \*temp1 = temp->Right\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Right\_Node = NULL;

free(temp1);

return;

}

else if (temp->Left\_Node == NULL && temp->Right\_Node == NULL)

{

if (Key == temp->Data)

{

Node \*temp1 = Root;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

else

cout << endl << "There is NO Such Node in the Tree !" << endl;

}

else

{

Delete\_Node(temp->Left\_Node, Key);

if (Key == temp->Data)

{

Node \*temp1 = temp;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

Delete\_Node(temp->Right\_Node, Key);

}

}

void Bst\_Tree::Search\_Node\_Data(int val) // To Search a Node

{

if (Root == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

RectangleShape rectangle; // Making Rectangle

Text text; // Using to show values

Text found; // Using to show searched value

RectangleShape line; // Using to make connecting lines

Text BST; // Using to show BST TREE

CircleShape triangle(6, 3); //Using to make Arrows

RectangleShape FL; //FL = Found Line (FOR searched value)

BST.setFont(font);

BST.setCharacterSize(50);

BST.setPosition(250, 50);

BST.setString("! BST Implementation !"); // Setting BST

BST.setFillColor(Color::Cyan);

BST.setOutlineColor(Color::White);

BST.setOutlineThickness(2);

Node \*temp = Root;

window.clear();

int i = 0;

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4); // setting text for values

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->Data));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis); // setting rectangle attributes

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18); // setting lines for connection

triangle.setPosition(x\_axis + 70, y\_axis + 24);

triangle.setFillColor(Color::White);

triangle.setRotation(270.f);

found.setFont(font);

found.setCharacterSize(25);

found.setFillColor(Color::White); // setting found for searched values

found.setOutlineColor(Color::White);

found.setOutlineThickness(1);

found.setString("Value Found");

found.setPosition(x\_axis - 35, y\_axis + 75);

FL.setSize(Vector2f(50, 2)); // for searched line

FL.setFillColor(Color::White);

FL.setRotation(90.f);

FL.setPosition(x\_axis + 35, y\_axis + 20);

if (temp->Data % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else // even odd difference

rectangle.setFillColor(Color::Red);

if (temp->Data == val)

{

window.draw(FL); // showing data if found

window.draw(found);

i = 1;

}

window.draw(BST);

window.draw(rectangle);

window.draw(text); // drawing the required graphics

window.draw(line);

window.draw(triangle);

//temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black); // setting for NULL text

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White); // setting to store NULL Text

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100; // setting default value of x

window.display();

system("pause");

}

}

void Bst\_Tree::Display\_Graphics(Node \*temp, int L = 1) // Displaying in SFML

{

y\_axis = 200;

BST.setFont(font);

BST.setCharacterSize(50);

BST.setPosition(250, 50);

BST.setString("! BST Implementation !"); // Setting BST

BST.setFillColor(Color::Cyan);

BST.setOutlineColor(Color::White);

BST.setOutlineThickness(2);

if (temp != NULL)

{

Display\_Graphics(temp->Left\_Node, L + 1);

for (int i = 0; i < L && temp != Root; i++)

{

y\_axis = y\_axis + 50;

}

circle.setRadius(20);

circle.setPosition(Vector2f(x\_axis - 5, y\_axis - 5));

text.setFont(font);

text.setCharacterSize(20);

text.setFillColor(Color::White);

text.setOutlineColor(Color::White);

text.setOutlineThickness(1);

if (temp->Data % 2 == 0)

{

circle.setFillColor(Color::Blue);

}

else

circle.setFillColor(Color::Red);

text.setPosition(Vector2f(x\_axis + 8, y\_axis + 4));

text.setString(to\_string(temp->Data));

line\_right.setSize(Vector2f(30, 2));

line\_right.setFillColor(Color::White);

line\_right.setPosition(circle.getPosition().x + circle.getRadius() \* 2, circle.getPosition().y + circle.getRadius() + 10);

line\_right.setRotation(45.f);

line\_left.setSize(Vector2f(30, 2));

line\_left.setFillColor(Color::White);

line\_left.setPosition(circle.getPosition().x - (circle.getRadius() / 2) + 15, circle.getPosition().y + circle.getRadius() + 10);

line\_left.setRotation(135.f);

if (temp == Root)

{

line\_right.setSize(Vector2f(100, 2));

line\_left.setSize(Vector2f(100, 2));

}

window.draw(BST);

window.draw(circle);

window.draw(text);

window.draw(line\_right);

window.draw(line\_left);

x\_axis = x\_axis + 50;

Display\_Graphics(temp->Right\_Node, L + 1);

}

}

int main() // Main Code

{

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 Delete Any Node From Tree !" << endl;

cout << " Press 3 to Search Any Node From Tree !" << endl;

cout << " Press 4 to Display Tree Graphically !" << endl;

cout << " Press 0 To Exit From The 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 << "Enter Value to Delete Respective Node :";

cin >> value;

Tree.Delete\_Node(Tree.Root, value);

system("pause");

break;

}

case 3:

{

cout << endl << "Enter Value to Search Respective Node :";

cin >> value;

Tree.Delete\_Node(Tree.Root, value);

system("pause");

break;

}

case 4:

{

window.clear();

cout << endl;

Tree.Display\_Graphics(Tree.Root);

window.display();

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, indoor, screenshot

Description automatically generated

**TREE IMPLEMENTATION**

A picture containing text, indoor, computer, screenshot

Description automatically generated

**QUESTION # 5:  
PROGRAM:**

#include <iostream>

#include "SFML/Graphics.hpp"; // Including SFML

using namespace std;

using namespace sf;

RenderWindow window(VideoMode(1000, 600), " BST Implementation"); // Making Window

Font font;

int x\_axis = 250;

int y\_axis = 0; // axis for later use

struct Node // Node Construction

{

int Data;

Node \*Left\_Node;

Node \*Right\_Node;

};

class Bst\_Tree // Class for Tree Formation

{

public:

Bst\_Tree()

{

font.loadFromFile("comicbd.TTF");

Root = NULL;

}

Node \*Root;

Text text;

RectangleShape line\_right;

RectangleShape line\_left;

Text BST;

CircleShape circle;

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

void Delete\_Node(Node \*root, int Key); // Deletion of Nodes

void Bst\_Tree::Search\_Node\_Data(int val); // Searching of Nodes

void Bst\_Tree::Mirror\_BST(Node\* Curr);

void Bst\_Tree::Display\_Graphics(Node \*temp, int L ); // Graphical Representation

};

Node \*Bst\_Tree::Insertion(Node \*Current, int data)

{

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 Bst\_Tree::Delete\_Node(Node \*root, int Key) //Deleting Nodes

{

Node \*temp = root;

if (temp == NULL)

{

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

return;

}

else if (temp->Left\_Node->Data == Key && temp->Left\_Node != NULL)

{

Node \*temp1 = temp->Left\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Left\_Node = NULL;

free(temp1);

return;

}

else if (temp->Right\_Node->Data == Key && temp->Right\_Node != NULL)

{

Node \*temp1 = temp->Right\_Node;

cout << endl << "Required Node is Deleted !" << endl;

temp->Right\_Node = NULL;

free(temp1);

return;

}

else if (temp->Left\_Node == NULL && temp->Right\_Node == NULL)

{

if (Key == temp->Data)

{

Node \*temp1 = Root;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

else

cout << endl << "There is NO Such Node in the Tree !" << endl;

}

else

{

Delete\_Node(temp->Left\_Node, Key);

if (Key == temp->Data)

{

Node \*temp1 = temp;

cout << endl << "Required Node is Deleted !" << endl;

Root = NULL;

free(temp1);

return;

}

Delete\_Node(temp->Right\_Node, Key);

}

}

void Bst\_Tree::Search\_Node\_Data(int val) // To Search a Node

{

if (Root == NULL)

{

cout << endl << "Link List is Empty !" << endl;

}

else

{

RectangleShape rectangle; // Making Rectangle

Text text; // Using to show values

Text found; // Using to show searched value

RectangleShape line; // Using to make connecting lines

Text BST; // Using to show BST TREE

CircleShape triangle(6, 3); //Using to make Arrows

RectangleShape FL; //FL = Found Line (FOR searched value)

BST.setFont(font);

BST.setCharacterSize(50);

BST.setPosition(250, 50);

BST.setString("! BST Implementation !"); // Setting BST

BST.setFillColor(Color::Cyan);

BST.setOutlineColor(Color::White);

BST.setOutlineThickness(2);

Node \*temp = Root;

window.clear();

int i = 0;

while (temp != NULL)

{

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 25, y\_axis + 4); // setting text for values

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black);

text.setOutlineThickness(1);

text.setString(to\_string(temp->Data));

rectangle.setSize(Vector2f(70, 40));

rectangle.setPosition(x\_axis, y\_axis); // setting rectangle attributes

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White);

rectangle.setOutlineThickness(2);

line.setSize(Vector2f(50, 2));

line.setFillColor(Color::White);

line.setPosition(x\_axis + 70, y\_axis + 18); // setting lines for connection

triangle.setPosition(x\_axis + 70, y\_axis + 24);

triangle.setFillColor(Color::White);

triangle.setRotation(270.f);

found.setFont(font);

found.setCharacterSize(25);

found.setFillColor(Color::White); // setting found for searched values

found.setOutlineColor(Color::White);

found.setOutlineThickness(1);

found.setString("Value Found");

found.setPosition(x\_axis - 35, y\_axis + 75);

FL.setSize(Vector2f(50, 2)); // for searched line

FL.setFillColor(Color::White);

FL.setRotation(90.f);

FL.setPosition(x\_axis + 35, y\_axis + 20);

if (temp->Data % 2 == 0)

{

rectangle.setFillColor(Color::Blue);

}

else // even odd difference

rectangle.setFillColor(Color::Red);

if (temp->Data == val)

{

window.draw(FL); // showing data if found

window.draw(found);

i = 1;

}

window.draw(BST);

window.draw(rectangle);

window.draw(text); // drawing the required graphics

window.draw(line);

window.draw(triangle);

//temp = temp->NextNode;

x\_axis = x\_axis + 100;

}

text.setFont(font);

text.setCharacterSize(25);

text.setPosition(x\_axis + 5, y\_axis + 4);

text.setFillColor(Color::Black);

text.setOutlineColor(Color::Black); // setting for NULL text

text.setOutlineThickness(1);

text.setString("NULL");

rectangle.setSize(Vector2f(77, 40));

rectangle.setPosition(x\_axis, y\_axis);

rectangle.setFillColor(Color::Magenta);

rectangle.setOutlineColor(Color::White); // setting to store NULL Text

rectangle.setOutlineThickness(2);

window.draw(rectangle);

window.draw(text);

x\_axis = 100; // setting default value of x

window.display();

system("pause");

}

}

void Bst\_Tree::Display\_Graphics(Node \*temp, int L = 1) // Displaying in SFML

{

y\_axis = 200;

BST.setFont(font);

BST.setCharacterSize(50);

BST.setPosition(250, 50);

BST.setString("! BST Implementation !"); // Setting BST

BST.setFillColor(Color::Cyan);

BST.setOutlineColor(Color::White);

BST.setOutlineThickness(2);

if (temp != NULL)

{

Display\_Graphics(temp->Left\_Node, L + 1);

for (int i = 0; i < L && temp != Root; i++)

{

y\_axis = y\_axis + 50;

}

circle.setRadius(20);

circle.setPosition(Vector2f(x\_axis - 5, y\_axis - 5));

text.setFont(font);

text.setCharacterSize(20);

text.setFillColor(Color::White);

text.setOutlineColor(Color::White);

text.setOutlineThickness(1);

if (temp->Data % 2 == 0)

{

circle.setFillColor(Color::Blue);

}

else

circle.setFillColor(Color::Red);

text.setPosition(Vector2f(x\_axis + 8, y\_axis + 4));

text.setString(to\_string(temp->Data));

line\_right.setSize(Vector2f(30, 2));

line\_right.setFillColor(Color::White);

line\_right.setPosition(circle.getPosition().x + circle.getRadius() \* 2, circle.getPosition().y + circle.getRadius() + 10);

line\_right.setRotation(45.f);

line\_left.setSize(Vector2f(30, 2));

line\_left.setFillColor(Color::White);

line\_left.setPosition(circle.getPosition().x - (circle.getRadius() / 2) + 15, circle.getPosition().y + circle.getRadius() + 10);

line\_left.setRotation(135.f);

if (temp == Root)

{

line\_right.setSize(Vector2f(100, 2));

line\_left.setSize(Vector2f(100, 2));

}

window.draw(BST);

window.draw(circle);

window.draw(text);

window.draw(line\_right);

window.draw(line\_left);

x\_axis = x\_axis + 50;

Display\_Graphics(temp->Right\_Node, L + 1);

}

}

void Bst\_Tree:: Mirror\_BST(Node\* Curr)

{

if (Curr == NULL)

return;

struct Node\* temp;

Mirror\_BST(Curr->Left\_Node);

Mirror\_BST(Curr->Right\_Node);

temp = Curr->Left\_Node;

Curr->Left\_Node = Curr->Right\_Node;

Curr->Right\_Node = temp;

}

int main() // Main Code

{

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 Delete Any Node From Tree !" << endl;

cout << " Press 3 to Search Any Node From Tree !" << endl;

cout << " Press 4 to Make Mirror of the Tree !" << endl;

cout << " Press 5 to Display Tree Graphically !" << endl;

cout << " Press 0 To Exit From The 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 << "Enter Value to Delete Respective Node :";

cin >> value;

Tree.Delete\_Node(Tree.Root, value);

system("pause");

break;

}

case 3:

{

cout << endl << "Enter Value to Search Respective Node :";

cin >> value;

Tree.Delete\_Node(Tree.Root, value);

system("pause");

break;

}

case 4:

{

cout << endl;

Tree.Mirror\_BST(Tree.Root);

cout << "Tree has been Mirrored !" << endl;

system("pause");

break;

}

case 5:

{

window.clear();

cout << endl;

Tree.Display\_Graphics(Tree.Root);

window.display();

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, indoor, electronics

Description automatically generated

**ORIGINAL TREE**

A picture containing text, monitor, indoor, screenshot

Description automatically generated

**MIRROR TREE**

A picture containing text, monitor, indoor, screen

Description automatically generated