

DATA STructures FInal Project

[Made by Ahmed Kasteer]



January 2, 2022

Roll Number: 20F-0336

Section: 3D

This project was made after studying and learning Data Structures in my third semester under the course Instructors Ma’am Saba Aslam and Sir Usman Joiya. This project was made individually and there were no acts of Plagiarism to be followed. The main idea is to enhance and learn all the data structures we have followed up till now and implement them using graphics library of your choice. This has enhanced my learning skills especially related to Stack, Queue, Binary Search Tree and AVL Tree. This code was implemented using SFML 2.5.1 32bit edition. The main reason I chose to use SFML was because I am familiar with it since I have used this library for Visual Studio in my earlier final projects of courses as well that is in Programming Fundamentals in my first semester where we had to design Space Shooting Game and in Object Oriented Programming in my second semester where we had to design Cricket Buzz management system. Linker libraries used for SFML were 3 out of 5 named as SFML Graphics, SFML Window and SFML System.

SOURCE CODE

#include <iostream>

#include <cstdlib>

#include <SFML/Graphics.hpp>

#include <string>

#include<queue>

#include<vector>

using namespace std;

using namespace sf;

//functions for implementation

int input(string show);

void stack1();

void que();

void bst();

void avl\_tree();

struct node

{

int key = -1;

node\* pnext = nullptr;

};

struct NODE//struct use of binary search tree

{

int level = 1;

int key;

NODE\* left = NULL, \* right = NULL, \* parent = NULL;

CircleShape\* circle = new CircleShape;

RectangleShape\* leftturn = new RectangleShape;

RectangleShape\* rightturn = new RectangleShape;

Text\* text = new Text;

NODE(int key)

{

this->key = key;

}

};

class BST

{

public:

NODE\* root = NULL;

NODE\* temp = NULL;

struct NODE\* input1(int k, int x, int y, int a, int b)

{

struct NODE\* temp;

temp = new NODE(k);

temp->circle->setRadius(20.0);

temp->circle->setFillColor(Color::White);

temp->circle->setPosition(Vector2f(x, y));

temp->text->setPosition(Vector2f(x + 10, y + 10));

temp->left = NULL;

temp->right = NULL;

return temp;

}

void input(int key)

{

root = input2(key, root, 250, 50, 260, 85);

}

struct NODE\* input2(int k, NODE\* node, int x = 250, int y = 50, int a = 260, int b = 85)

{

if (node == NULL)

{

return input1(k, x, y, a, b);

}

if (k < node->key)//left link

{

// int minus = -5;

if (node->left)

{

// minus -= (node->left->level \* 10);

}

node->left = input2(k, node->left, x - 50 /\*minus\*/, y + 50, a - 50 /\*+ minus\*/, b

+ 50);

node->left->parent = node;

node->left->level = node->left->parent->level + 1;

node->left->parent->leftturn->setSize(sf::Vector2f(70, 5));

node->left->parent->leftturn->setFillColor(sf::Color::Red);

node->left->parent->leftturn->setRotation(130);

node->left->parent->leftturn->setPosition(sf::Vector2f(a, b));

}

else if (k > node->key)//right link

{

// int minus = 5;

if (node->left)

{

// minus += (node->left->level \* 10);

}

node->right = input2(k, node->right, x + 50 /\*+ minus\*/, y + 50, a + 50 /\*+

minus\*/, b + 50);

node->right->parent = node;

node->right->level = node->right->parent->level + 1;

node->right->parent->rightturn->setSize(sf::Vector2f(50, 5));

node->right->parent->rightturn->setFillColor(sf::Color::Red);

node->right->parent->rightturn->setRotation(230);

node->right->parent->rightturn->setPosition(sf::Vector2f(a + 50/\* + minus\*/, b

+ 40));

}

return node;

}

void del(int key)

{

delet(root, key);

}

int searching(int k)

{

return search(root, k);

}

bool search(struct NODE\* node, int k)

{

if (node == NULL)

{

return false;

}

if (node->key == k)

{

node->circle->setFillColor(Color::Red);

return true;

}

if (node->key < k)

{

node->circle->setFillColor(Color::Yellow);

return search(node->right, k);

}

else

{

node->circle->setFillColor(Color::Yellow);

return search(node->left, k);

}

}

NODE\* delet(NODE\* ptr, int data) {

// if no node

if (ptr == nullptr)

{

cout << "BST is empty!";

return nullptr;

}

else {

//case 1:

// if entered data is smaller so, moving to left sub-tree

if (data < ptr->key) {

ptr->left = delet(ptr->left, data);

}

// if entered data is bigger so, moving to right sub-tree

else if (data > ptr->key) {

ptr->right = delet(ptr->right, data);

}

//case 2:

//if there is an intermidiate node so, we have to swipe data and

//then delete

else {

//this can be done by either replacing minimum from

//right sub - tree

//or maximum from left sub-tree

if (ptr->right && ptr->left) {

temp = minValueNode(ptr->right);

//copying value fo that it will become easy to

//delete

//a leaf instead of intermidiate node

ptr->key = temp->key;

//now deleteing

ptr->right = delet(ptr->right, temp->key);

}

//case 3:

//if the node have one or zero chlid

//connect parent with grandchild

else {

temp = ptr;

if (ptr->left == nullptr) {

ptr = ptr->right;

}

else if (ptr->right == nullptr) {

ptr = ptr->left;

}

free(temp);

}

}

return ptr;

}

}

struct NODE\* minValueNode(struct NODE\* node)

{

struct NODE\* current = node;

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

{

current = current->left;

}

return current;

}

void setColors()

{

TRAVERSE(root);

}

void TRAVERSE(struct NODE\* node)

{

if (!node)

return;

node->circle->setFillColor(Color::Cyan);

TRAVERSE(node->left);

TRAVERSE(node->right);

}

void Display(RenderWindow\* linear, NODE\* root)

{

if (!root)

{

cout << "Nothing to Display";

return;

}

queue<NODE\*> obj;

obj.push(root);

while (!obj.empty())

{

NODE\* temp = obj.front();

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error Loading Font" << endl;

system("pause");

}

string str = to\_string(temp->key);

temp->text->setFont(font);

temp->text->setCharacterSize(20);

temp->text->setFillColor(Color::Black);

temp->text->setString(str);

linear->draw(\*temp->circle);

if (temp->left)

linear->draw(\*temp->leftturn);

if (temp->right)

linear->draw(\*temp->rightturn);

cout << temp->key << " ";

obj.pop();

linear->draw(\*temp->text);

if (temp->left)

obj.push(temp->left);

if (temp->right)

obj.push(temp->right);

}

}

};

struct avl

{

int level = 1;

int key;

avl\* left = NULL, \* right = NULL, \* parent = NULL;

CircleShape\* circle = new CircleShape;

RectangleShape\* leftone = new RectangleShape;

RectangleShape\* rightone = new RectangleShape;

Text\* text = new Text;

avl(int key)

{

this->key = key;

string str;

str = to\_string(key);

text->setString(str);

text->setFillColor(Color::Black);

}

};

class AVL

{

public:

avl\* roots = NULL;

int searching(int k)

{

return search(roots, k);

}

void insert(int k)

{

roots = input2(roots, k, 250, 50, 260, 85);

}

void setColors()

{

TRAVERSE(roots);

}

void TRAVERSE(struct avl\* node)

{

if (!node)

return;

node->circle->setFillColor(Color::Yellow);

TRAVERSE(node->left);

TRAVERSE(node->right);

}

struct avl\* input1(int k, int x, int y, int a, int b)

{

struct avl\* temp;

temp = new avl(k);

temp->circle->setRadius(20.0);

temp->circle->setFillColor(Color::Yellow);

temp->circle->setPosition(Vector2f(x, y));

temp->text->setPosition(Vector2f(x + 10, y + 10));

temp->left = NULL;

temp->right = NULL;

return temp;

}

struct avl\* input2(struct avl\* node, int k, int x, int y, int a, int b)

{

if (node == NULL)

{

return input1(k, x, y, a, b);

}

if (k < node->key)//left link

{

int minus = -5;

if (node->left)

{minus -= (node->left->level \* 10);

}

node->left = input2(node->left, k, x - 50 + minus, y + 50, a - 50 + minus, b + 50);

node->left->parent = node;

node->left->level = node->left->parent->level + 1;

node->left->parent->leftone->setSize(sf::Vector2f(50, 5));

node->left->parent->leftone->setFillColor(sf::Color::Green);

node->left->parent->leftone->setRotation(130);

node->left->parent->leftone->setPosition(sf::Vector2f(a, b));

node = balance(node, x, y, a, b);

}

else if (k > node->key)//right link

{

int minus = 5;

if (node->left)

{

minus += (node->left->level \* 10);

}

node->right = input2(node->right, k, x + 50 + minus, y + 50, a + 50 + minus, b +

50);

node->right->parent = node;

node->right->level = node->right->parent->level + 1;

node->right->parent->rightone->setSize(sf::Vector2f(50, 5));

node->right->parent->rightone->setFillColor(sf::Color::Green);

node->right->parent->rightone->setRotation(230);

node->right->parent->rightone->setPosition(sf::Vector2f(a + 50 + minus, b +

40));

node = balance(node, x, y, a, b);

}

return node;

}

void preorder(struct avl\* node)

{

if (node == NULL)

{

return;

}

cout << node->key << " ";

preorder(node->left);

preorder(node->right);

}

void postorder(struct avl\* node)

{

if (node == NULL)

{

return;

}

postorder(node->left);

postorder(node->right);

cout << node->key << " ";

}

void Inorder(struct avl\* node)

{

if (node == NULL)

{

return;

}

Inorder(node->left);

cout << node->key << " ";

Inorder(node->right);

}

int max(int a, int b)

{

if (a > b)

{

return a;

}

else

{

return b;

}

}

int height(struct avl\* node)

{

if (node == NULL)

{

return 0;

}

else

{

return 1 + max(height(node->left), height(node->right));//choosing maximum

//height of height of left subtreeand right subtree

}

}

int diff(struct avl\* node)

{

int d = 0; int l = 0;

int r = 0;

if (node != NULL)

{

if (node->left != NULL)

l = height(node->left);

if (node->right != NULL)

r = height(node->right);

}

d = l - r;

return d;

}

void del(int key)

{

deleteNode(roots, key, 375, 80, 375, 115);

}

struct avl\* right(struct avl\* node, int x, int y, int a, int b)

{

avl\* temp;

if (node->right != NULL)

{

temp = node->right;

node->right = temp->left;

node->text->setPosition(Vector2f(a - 50, b + 40));

node->circle->setPosition(Vector2f(a - 50, b + 40));

temp->text->setPosition(Vector2f(a, b - 40));

temp->circle->setPosition(Vector2f(a, b - 40));

if (temp->right)

{

temp->right->leftone->setPosition(Vector2f(a - 50, b + 40));

temp->right->rightone->setPosition(Vector2f(a, b - 40));

temp->right->circle->setPosition(Vector2f(a + 50, b + 40));

temp->right->text->setPosition(Vector2f(a + 50, b + 40));

}

temp->left = node;

return temp;

}

else

{

temp = node;

temp = temp->left;

}

return temp;

}

struct avl\* left(struct avl\* node, int x, int y, int a, int b)

{

avl\* temp = NULL;

if (node->left != NULL)

{

temp = node->left;

node->left = temp->right;

node->text->setPosition(Vector2f(a + 50, b + 40));

node->circle->setPosition(Vector2f(a + 50, b + 40));

temp->text->setPosition(Vector2f(a, b));

temp->circle->setPosition(Vector2f(a, b));

if (temp->left)

{

temp->left->leftone->setPosition(Vector2f(a + 50, b + 40));

temp->left->rightone->setPosition(Vector2f(a, b + 40));

temp->left->circle->setPosition(Vector2f(a - 50, b + 40));

temp->left->text->setPosition(Vector2f(a - 50, b + 40));

}

temp->right = node;

return temp;

}

else

{

temp = node;

}

return temp;

}

struct avl\* l\_right(struct avl\* node, int x, int y, int a, int b)

{

avl\* temp;

temp = node->left;

node->left = right(temp, x, y, a, b);

return left(node, x, y, a, b);

}

struct avl\* r\_left(struct avl\* node, int x, int y, int a, int b)

{

avl\* temp;

temp = node->right;

node->right = left(temp, x, y, a, b);

return right(node, x, y, a, b);

}

struct avl\* balance(struct avl\* node, int x, int y, int a, int b)

{

int s = 0;

s = diff(node);

if (s > 1)

{

if (diff(node->left) > 0)

{

node = left(node, x, y, a, b);

}

else

{

node = l\_right(node, x, y, a, b);

}

}

else if (s < -1)

{

if (diff(node->left) > 0)

{

node = right(node, x, y, a, b);

}

else

{

node = r\_left(node, x, y, a, b);

}

}

return node;

}

avl\* minValueNode(avl\* node)

{

avl\* current = node;

while (current->left != NULL)

{

current = current->left;

}

return current;

}

bool search(struct avl\* node, int k)

{

if (node == NULL)

{

return false;

}

if (node->key == k)

{

node->circle->setFillColor(Color::Red);

return true;

}

if (node->key < k)

{

node->circle->setFillColor(Color::Blue);

return search(node->right, k);

}

else

{

node->circle->setFillColor(Color::Blue);

return search(node->left, k);

}

}

avl\* deleteNode(avl\* root, int key, int x, int y, int a, int b)

{

if (root == NULL)

return root;

if (key < root->key)

root->left = deleteNode(root->left, key, x, y, a, b);

else if (key > root->key)

root->right = deleteNode(root->right, key, x, y, a, b);

else

{

if ((root->left == NULL) || (root->right == NULL))

{

avl\* temp = root->left ? root->left : root->right;

if (temp == NULL)

{

temp = root;

root = NULL;

}

else

{

\*root = \*temp;

}

free(temp);

}

else

{

avl\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key, x, y, a, b);

}

}

balance(root, x, y, a, b);

return root;

}

};

int main()

{

sf::RenderWindow window1(sf::VideoMode(1000, 640), "Main Menu");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error font not loading:\n";

}

Text text[15];

string str[15];

int option = 0;

str[0] = "Stack DATA STRUCTURE ";

str[1] = "Queue DATA STRUCTURE ";

str[2] = "Binary Search Tree ";

str[3] = "AVL Tree";

for (int i = 0; i < 12; ++i)

{

text[i].setFont(font);

text[i].setString(str[i]);

text[i].setCharacterSize(50);

text[i].setFillColor(Color::Yellow);

text[i].setPosition(170, i \* 70);

window1.display();

window1.draw(text[i]);

window1.display();

}

while (option < 5)

{

option = input("Press 1-4 to select option: ");

switch (option)

{

case 1:

stack1();

break; case 2:

que();

break;

case 3:

bst();

break;

case 4:

avl\_tree();

break;

default:

break;

}

}

while (window1.isOpen())

{

sf::Event event;

while (window1.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window1.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window1.close();

return 0;

}

}

}

window1.clear();

}

return 0;

}

void check(int num)

{

sf::RenderWindow window(sf::VideoMode(700, 100), "check!");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error";

}

Text text;

if (num == 1)

{

text.setFont(font);

text.setString("Value not found.");

text.setCharacterSize(35);

text.setStyle(Text::Bold | Text::Underlined);

text.setFillColor(Color::Yellow);

text.setPosition(150, 40);

}

else if (num == 2)

{

text.setFont(font);

text.setString("Value found.");

text.setCharacterSize(35);

text.setStyle(Text::Bold | Text::Underlined);

text.setFillColor(Color::Red);

text.setPosition(200, 40);

}

else if (num == 3)

{

text.setFont(font);

text.setString("Empty.");

text.setCharacterSize(35);

text.setStyle(Text::Bold | Text::Underlined);

text.setFillColor(Color::Red);

text.setPosition(200, 40);

}

else if (num == 4)

{

text.setFont(font);

text.setString("Not Empty.");

text.setCharacterSize(35);

text.setStyle(Text::Bold | Text::Underlined);

text.setFillColor(Color::Red);

text.setPosition(200, 40);

}

while (window.isOpen())

{

sf::Event event;

while (window.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window.close();

}

}

}

window.clear(Color::Blue);

window.draw(text);

window.display();

}

}

void bst()

{

sf::RenderWindow window1(sf::VideoMode(1000, 640), "Binary Search Tree");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error";

}

Text text;

int option = 1;

text.setFont(font);

text.setString("Binary search tree visualization");

text.setCharacterSize(25);

text.setFillColor(Color::Red);

text.setPosition(400, 0);

window1.draw(text);

window1.display();

BST obj;

do

{

option = input("1 Enter Data: 2 Delete Data");

if (option == 1)

{

int data = input("enter the data u want to enter: ");

obj.input(data);

// obj.Display(&window1, obj.root);

if (!obj.root)

{

cout << "Nothing to Display";

return;

}

queue<NODE\*> obj1;

obj1.push(obj.root);

while (!obj1.empty())

{

NODE\* temp = obj1.front();

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error Loading Font" << endl;

system("pause");

}

string str = to\_string(temp->key);

temp->text->setFont(font);

temp->text->setCharacterSize(20);

temp->text->setFillColor(Color::Black);

temp->text->setString(str);

window1.draw(\*temp->circle);

//window1.display();

if (temp->left)

window1.draw(\*temp->leftturn);

//window1.display();

if (temp->right)

window1.draw(\*temp->rightturn);

//window1.display();

cout << temp->key << " ";

obj1.pop();

window1.draw(\*temp->text);

if (temp->left)

obj1.push(temp->left);

if (temp->right)

obj1.push(temp->right);

window1.display();

}

}

else if (option == 2)

{

int data = input("enter the data u want to del: ");

obj.del(data);

window1.clear();

// obj.Display(&window1, obj.root);

if (!obj.root)

{

cout << "Nothing to Display";

return;

}

queue<NODE\*> obj1;

obj1.push(obj.root);

while (!obj1.empty())

{

NODE\* temp = obj1.front();

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error Loading Font" << endl;

system("pause");

}

string str = to\_string(temp->key);

temp->text->setFont(font);

temp->text->setCharacterSize(20);

temp->text->setFillColor(Color::Black);

temp->text->setString(str);

window1.draw(\*temp->circle);

//window1.display();

if (temp->left)

window1.draw(\*temp->leftturn);

//window1.display();

if (temp->right)

window1.draw(\*temp->rightturn);

//window1.display();

cout << temp->key << " ";

obj1.pop();

window1.draw(\*temp->text);

if (temp->left)

obj1.push(temp->left);

if (temp->right)

obj1.push(temp->right);

window1.display();

}

}

} while (option != 0);

while (window1.isOpen())

{

sf::Event event;

while (window1.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window1.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window1.close();

}

}

}

}

}

void stack1()

{

sf::RenderWindow window1(sf::VideoMode(1000, 640), "stack!");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error";

}

Text text;

text.setFont(font);

text.setString("->->stack visualization<-<-");

text.setCharacterSize(25);

text.setFillColor(Color::Red);

text.setPosition(400, 0);

int option, x = 50, y = 50, count = 0;

int size = 0;

Text tex;

RectangleShape shape;

string str;

string str1;

int\* l = new int[20];

for (int i = 0; i < 20; i++) {

l[i] = -1;

}

int j = 0;

do

{

option = input(" 1: for push \n 2: for pop \n 3: search\n 4: is empty \n esc: for exit");

if (option == 1)

{

window1.draw(text);

window1.display();

str.clear();

int data = input("enter the data");

str = to\_string(data);

l[j] = data;

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

j++;

size++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

else if (option == 2)

{

int data = input("enter the data u want to del:");

l[j - 1] = 0;

j = j - 1;

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}

}

else if (option == 3)

{

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}

int data, checker = 0;

data = input("enter the data u want to search: ");

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

{

if (data == l[i])

{

checker = 1;

check(2);

}

}

if (checker == 0)

{

check(1);

}

}

else if (option == 4)

{

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}int checker = 0;

if (l[0] == -1)

{

checker = 1;

check(3);

}

if (checker == 0)

{

check(4);

}

}

} while (option != 0);

while (window1.isOpen())

{

sf::Event event;

while (window1.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window1.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window1.close();

}

}

}

}

}

void que()

{

sf::RenderWindow window1(sf::VideoMode(1000, 640), "Que!");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error loading font";

}

Text text, t1, t2;

text.setFont(font);

text.setString("->->Que visulization:: <-<-");

text.setCharacterSize(25);

text.setFillColor(Color::Red);

text.setPosition(0, 0);

int option, x = 50, y = 50, count = 0;

int size = 0;

Text tex;

RectangleShape shape;

string str;

string str1;

int\* l = new int[20];

for (int i = 0; i < 20; i++) {

l[i] = -1;

}

int j = 0, index = 0;

do

{

option = input(" 1: for push \n 2: for pop \n 3: search\n 4: is empty \n esc: for exit");

if (option == 1)

{

window1.draw(text);

window1.display();

str.clear();

int data = input("enter the data");

str = to\_string(data);

l[j] = data;

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

j++;

size++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

else if (option == 2)

{

l[index] = 0;

index = index + 1;

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}

}

else if (option == 3)

{

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}

int data, checker = 0;

data = input("enter the data u want to search: ");

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

{

if (data == l[i])

{

checker = 1;

check(2);

}

}

if (checker == 0)

{

check(1);

}

}

else if (option == 4)

{

x = 50, y = 50, count = 0;

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

{

if (l[k] != -1) {

str.clear();

str = to\_string(l[k]);

window1.draw(text);

window1.display();

shape.setSize(Vector2f(50, 30));

shape.setFillColor(Color::White);

shape.setOutlineColor(Color::Red);

shape.setOutlineThickness(3.f);

shape.setPosition(x \* count + 10, y);

window1.draw(shape);

tex.setFont(font);

tex.setFillColor(Color::Red);

tex.setString(str);

tex.setPosition(x \* count + 25, y);

window1.draw(tex);

window1.display();

count++;

if (count == 20)

{

count = 0;

y = y + 60;

}

}

}

int checker = 0;

if (l[0] == -1)

{

checker = 1;

check(3);

}

if (checker == 0)

{

check(4);

}

}

} while (option != 0);

while (window1.isOpen())

{

sf::Event event;

while (window1.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window1.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window1.close();

}

}

}

}

}

void avl\_tree()

{

sf::RenderWindow window1(sf::VideoMode(1000, 640), "AVL Tree");

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error";

}

Text text;

int option = 1;

text.setFont(font);

text.setString("->->avl tree visualization<-<-");

text.setCharacterSize(15);

text.setFillColor(Color::Red);

text.setPosition(500, 0);

window1.draw(text);

window1.display();

AVL obj;

do

{

option = input("1: Input data. 2 Delete Data. 3. Search data");

if (option == 1)

{

int data = input("enter the data u want to enter: ");

obj.insert(data);

if (!obj.roots)

{

cout << "Nothing to Display";

return;

}

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error Loading Font" << endl;

system("pause");

}

queue<avl\*> obj1;

obj1.push(obj.roots);

while (!obj1.empty())

{

avl\* temp = obj1.front();

temp->text->setFont(font);

temp->text->setCharacterSize(15);

temp->text->setString(to\_string(temp->key));

window1.draw(\*temp->circle);

if (temp->left)

window1.draw(\*temp->leftone);

if (temp->right)

window1.draw(\*temp->rightone);

window1.draw(\*temp->text);

cout << temp->key << " ";

obj1.pop();

if (temp->left)

obj1.push(temp->left);

if (temp->right)

{

obj1.push(temp->right);

}

window1.display();

}

}

else if (option == 2)

{

int data = input("enter the data u want to del: ");

obj.del(data);

window1.clear();

if (!obj.roots)

{

cout << "Nothing to Display";

return;

}

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error Loading Font" << endl;

system("pause");

}

queue<avl\*> obj1;

obj1.push(obj.roots);

while (!obj1.empty())

{

avl\* temp = obj1.front();

temp->text->setFont(font);

temp->text->setCharacterSize(20);

temp->text->setString(to\_string(temp->key));

window1.draw(\*temp->circle);

if (temp->left)

window1.draw(\*temp->leftone);

if (temp->right)

window1.draw(\*temp->rightone);

window1.draw(\*temp->text);

cout << temp->key << " ";

obj1.pop();

if (temp->left)

obj1.push(temp->left);

if (temp->right)

obj1.push(temp->right);

}

}

else if (option == 3)

{

int data = input("enter the num u want to search");

obj.searching(data);

}

} while (option != 0);

while (window1.isOpen())

{

sf::Event event;

while (window1.pollEvent(event))

{

if (event.type == sf::Event::Closed)

{

window1.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window1.close();

}

}

}

}

}

int input(string show)

{

RenderWindow window(sf::VideoMode(1000, 300), "input function");

//window.setSize(Vector2u(0,0));

Font font;

if (!font.loadFromFile("Alien Eclipse Italic.ttf"))

{

cout << "Error";

}

Text text;

text.setFont(font);

text.setString(show);

text.setCharacterSize(25);

text.setFillColor(Color::White);

text.setPosition(30, 50);

RectangleShape rectangle;

rectangle.setSize(Vector2f(100, 25));

rectangle.setFillColor(Color::Transparent);

rectangle.setPosition(750, 50);

// rectangle.setPosition(Vector2f(750, 55));

string data;

Text EnteredData;

EnteredData.setFont(font);

EnteredData.setCharacterSize(18);

EnteredData.setString(data);

EnteredData.setFillColor(Color::White);

EnteredData.setPosition(Vector2f(752, 55));

Text text1;

text1.setFont(font);

text1.setString("You only enter NUMBERS");

text1.setFillColor(Color::Red);

text1.setPosition(Vector2f(150, 200));

Text text2;

text2.setFont(font);

text2.setString("entering number limit is full");

text2.setFillColor(Color::Red);

text2.setPosition(Vector2f(150, 250));

bool moreOrLess = false;

bool extra = false;

bool point = false;

bool negative = false;

while (window.isOpen())

{

Event event;

while (window.pollEvent(event))

{

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::BackSpace)

{

if (data.length() > 0)

{

data.erase(data.length() - 1);

}

if (data.length() == 0)

{

negative = false;

}

int points = 0;

for (int i = 0; i < data.length(); i++)

{

if (data[i] != '.')

points++;

}

if (points == data.length())

{

point = true;

}

moreOrLess = false;

extra = false;

}

}

if (event.type == Event::EventType::Closed)

{

window.close();

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Escape)

{

window.close();

return 0;

}

}

if (event.type == Event::EventType::KeyPressed)

{

if (event.key.code == Keyboard::Key::Return)

{

window.close();

float num = stof(data);

return num;

}

}

if (event.type == sf::Event::TextEntered)

{

if (event.text.unicode < 128)

{

cout << "ASCII entered: " <<

static\_cast<char>(event.text.unicode) << std::endl;

if (event.text.unicode == 45)

{

if (negative == false)

{

data = data + (char)event.text.unicode;

negative = true;

}

}

else if (event.text.unicode == 46)

{

if (point == false)

{

data = data + (char)event.text.unicode;

}

point = true;

}

else if (event.text.unicode >= 48 && event.text.unicode <= 57)

{

data = data + (char)event.text.unicode;

moreOrLess = false;

negative = true;

}

else

{

moreOrLess = true;

}

}

}

}

EnteredData.setString(data);

window.clear(Color::Black);

window.draw(text);

window.draw(rectangle);

window.draw(EnteredData);

if (moreOrLess)

{

window.draw(text1);

}

if (extra)

{

window.draw(text2);

}

window.display();

}

}

SNAPSHOTS

Text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Text

Description automatically generated with medium confidence

Graphical user interface, text

Description automatically generated

Graphical user interface, background pattern

Description automatically generated with medium confidence

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, application

Description automatically generated

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text

Description automatically generated

Graphical user interface, text, application

Description automatically generated

Rotation causes the graphics to look like this:

Graphical user interface, text, application

Description automatically generated

Graphical user interface

Description automatically generated with medium confidence