סדנא ב- C++ – 150018

תרגיל בית מספר 9

Template ועצים – תרגיל מסכם

שים/י לב:

א. הקפד/י על קְרִיאוּת התכנית ועל עימוד (Indentation).
ב. הקפד/י לבצע בדיוק את הנדרש בכל שאלה.
ג. בכל אחת מהשאלות יש להגדיר פונקציות במידת הצורך עבור קריאות התכנית.
ד. יש להגיש את התרגיל על פי ההנחיות להגשת תרגילים (המופיע באתר הקורס) וביניהם:
השתמש/י בשמות משמעותיים עבור המשתנים.
יש לתעד את התכנית גם עבור פונקציות אותם הנך מגדיר/ה וכן על תנאים ולולאות וקטעי קוד מורכבים, ובנוסף, **דוגמת הרצה לכל תכנית בסוף הקובץ**!
הגשה יחידנית - אין להגיש בזוגות.

הערה חשובה: תרגיל מסכם זה הינו חובה להגשה עד למועד הבחינה בתאריך: 22/06/2021 – בנוסף, תרגיל זה ינוקד כפול בציון הסופי. בהצלחה!

שאלה מס' 1:

א. הוסף/י למחלקה **Tree** (מחלקה גנרית אשר הובאה בהרצאה) את המתודות הבאות:

- **int height();** מתודה המחשבת ומחזירה את גובה העץ (לצורך התרגיל נגדיר: עץ ריק גובהו 1-, עץ עם שורש בלבד גובהו 0 וכו')
- **void reflect();** מתודה המחליפה בין הבנים של כל קודקוד בעץ ויוצרת עץ חדש שהוא תמונת הראי של העץ המקורי.
- **void breadthScan()** מתודה הסורקת לרוחב את צמתי העץ ומדפיסה את ערכי הצמתים רמה אחר רמה (החל מהשורש) משמאל לימין. מתודה זו תיעזר במבנה נתונים תור (יש לכלול את מבנה הנתונים תור בהגדרתה הגנרית וכן המחלקה היורשת מתור הממשת את תור ע"י ווקטור או רשימה)

ב. הוסף/י למחלקה **SearchTree** היורשת מ-**Tree** את המתודות הבאות:

- **void remove(T val);** מתודה המוחקת מהעץ את הקודקוד אשר מכיל את הנתון val.
- **T successor(T val);** מתודה המחזירה את העוקב של הנתון val במידה ולא נמצא אחד כזה – המתודה תזרוק שגיאה:

no successor

- **void deleteDuplicates();** מחיקת נתונים כפולים מהעץ

הנחיות:

- בתרגיל זה **אין** להשתמש במבנים המוגדרים ב-STL
- במחלקה היורשת, SearchTree, השימוש בתכונה root תיעשה באופן הבא:
Tree<T>::root

- במחלקה היורשת, SearchTree, הגדרת מצביע מטיפוס Node תיעשה באופן הבא:

  typename tree<T>::Node*
- המחלקות Tree ו-SearchTree מובאות כנספחים(לאחר שאלה מס' 2) לשאלה זו כפי שהובאו בהרצאה (כמובן, יש להוסיף עליהם את הנדרש בשאלה)
- המחלקות Queue ו-QueueVector מובאות כנספחים(לאחר שאלה מס' 2) לשאלה זו – שים/י לב – מחלקות אלו מובאות כמחלקות לא גנריות – יש להגדירן באופן גנרי עבור השאלה.
- שים/י לב, איברים זהים נמצאים בתת עץ השמאלי של הקדקוד ולא בתת העץ הימני
- עבור המתודה remove – מחיקת קודקוד מהעץ – יש להבחין בשלושה מקרים:
  א. קודקוד שהינו עלה (כלומר, אין לו בנים)
  ב. קודקוד עם בן יחיד
  ג. קודקוד עם שני בנים
- את המתודות על עצים לרב יש לממש באופן רקורסיבי
- עבור מימוש המתודות השונות – ניתן להיעזר באלגוריתמים אותם למדת בקורס מבנה נתונים ותכניות א'

ג. צרף/י את המחלקות שהגדרת בסעיפים לעיל לתכנית הראשית הבאה והראה/י את נכונותם:

```
#include <iostream>
using namespace std;
#include "SearchTree.h"
enum { END, ADD, SEARCH, REMOVE, BREADTHSCAN, HEIGHT, SUCCESSOR, DELETEDUP,
REFLECT };
int main()
{
	SearchTree<int> T1;
	cout << "enter 10 numbers:\n";
	int x, y, z;
	for (int i = 0; i < 10; i++)
	{
		cin >> x;
		T1.add(x);
	}
	cout << "inorder: ";
	T1.inOrder();
	cout << "\nenter 0-8:\n";
	cin >> x;
	while (x != END)
	{
		switch (x)
		{
		case ADD: cout << "enter a number: ";
			cin >> y;
			T1.add(y);
			cout << "after adding " << y << ": ";
			T1.inOrder();
			cout << endl;
			break;
		case SEARCH: cout << "enter a number: ";
			cin >> y;
			if (T1.search(y))
				cout << "exist" << endl;
			else
```

```
                        cout << "no exist" << endl;
                break;
            case REMOVE:cout << "enter a number: ";
                cin >> y;
                try
                {
                        T1.remove(y);
                        cout << "after removing " << y << ": ";
                        T1.inOrder();
                        cout << endl;
                }
                catch (const char* str)
                {
                        cout << str << endl;
                }
                break;
            case BREADTHSCAN: cout << "breadth scan: ";
                T1.breadthScan();
                cout << endl;
                break;
            case HEIGHT:cout << "height of tree: " << T1.height() << endl;
                break;
            case SUCCESSOR:cout << "enter a number: ";
                cin >> y;
                try
                {
                        z = T1.successor(y);
                        cout << "successor of " << y << " is: " << z <<
endl;

                }
                catch (const char* str)
                {
                        cout << str << endl;
                }
                break;
            case DELETEDUP: cout << "after delete duplicate: ";
                T1.deleteDuplicates();
                T1.inOrder();
                cout << endl;
                break;
            case REFLECT:T1.reflect();
                cout << "reflected tree: ";
                T1.inOrder();
                T1.reflect();
                cout << endl;
                break;
            }
            cout << "enter 0-8:\n";
            cin >> x;
        }
        return 0;
        }
```

<u>שאלה מס' 2:</u>

הגדר/י מחלקה לייצוג ספרים בספרייה.
המחלקה תכלול את התכונות הבאות:

- קוד ספר (int)
- שם הסופר (string)
- שם הספר (string)

בנוסף, המחלקה תכלול לפחות את המתודות הבאות:

- constructor עם ערכי ברירת מחדל עבור כל התכונות (0 ומחרוזות ריקות בהתאמה)
- אופרטורים =!,==,=>=,<=,>,<. כל אופרטורי ההשוואה יחזירו בהתאם לסדר לקסיקוגרפי (אלפא-ביתי, מילוני) של שמות הסופרים תחילה ולאחר מכן שמות הספרים – במידה ושניהם שווים יש להשוות בין הקודים השונים (עבור אופרטור == וכן =! יש לבדוק השוואה בין כל התכונות)
- אופרטורים <<,>> הקלט והפלט יתבצעו לפי סדר: קוד הספר, שם הסופר ושם הספר.

כתוב/י תכנית ראשית אשר תנהל את רישום הספרים בספרייה. לצורך כך, השתמש/י במחלקת עץ חיפוש כמבנה הנתונים המאחסן את רשימת הספרים.

<u>היעזר/י במחלקות המוגדרות בשאלה מס' 1</u>

בכל שלב יש להציג למשתמש תפריט לבחירת פעולה (מתוך a-e), התוכנית תמשיך כל עוד לא התקבל הקלט e המסמן את סיום התוכנית.

a. הוספת ספר לספרייה
b. מחיקת ספר מהספרייה
c. חיפוש ספר בספרייה
d. הדפסה אלפא-בתית של כל הספרים הנמצאים בספרייה
e. יציאה - סיום התוכנית

במהלך ריצת התכנית ניתן להשתמש בהודעות פלט הבאות בלבד:

enter a-e:
enter a book - עבור הוספה:
exist - עבור הודעה לאחר חיפוש, במקרה שהספר קיים בספרייה
not exist - עבור מקרה של חיפוש או מחיקה, במידה והתקבל ספר שאינו קיים במערכת
error – למקרה שהתקבלה אות שאינה בתחום

דוגמא להרצת התכנית:

```
enter a-e:
a
enter a book:
2 b b
enter a-e:
a
enter a book:
5 e e
enter a-e:
a
enter a book:
1 a a
enter a-e:
a
enter a book:
4 d d
enter a-e:
a
enter a book:
7 g g
enter a-e:
a
enter a book:
3 c c
enter a-e:
b
enter a book:
5 e e
enter a-e:
d
1 a a
2 b b
3 c c
4 d d
7 g g
enter a-e:
e
```

בהצלחה רבה!!

נספחים לשאלה מס' 1:

המחלקה Tree:

```
#pragma once
#include <iostream>
using namespace std;

//-----------------------------------
//  class Tree (Binary Trees)
// process nodes in Pre/In/Post  order
//-----------------------------------
template <class T>
class Tree
{
protected:
	//-------------------------------------------------------
	//  inner class Node
	//      a single Node from a binary tree
	//-------------------------------------------------------
	class Node
	{
	public:
		Node* left;
		Node* right;
		T value;
		Node(T val) : value(val), left(NULL), right(NULL) {}
		Node(T val, Node* l, Node* r): value(val), left(l), right(r) {}
	};		//end of Node class

	//data member of tree
	Node* root;

public:
	Tree() { root = NULL; }	// initialize tree
	~Tree();
	int isEmpty() const;
	void clear() { clear(root); root = NULL; }
	void preOrder() { preOrder(root); }
	void inOrder() { inOrder(root); }
	void postOrder() { postOrder(root); }

	virtual void process(T val) { cout << val << " "; }
	virtual void add(T val) = 0;
	virtual bool search(T val) = 0;
	virtual void remove(T val) = 0;


private:
	//private function for not give acsses to user
	void  clear(Node* current);
	void  preOrder(Node* current);
	void  inOrder(Node* current);
	void  postOrder(Node* current);

};
```

```
//-----------------------------------------------------------
//  class Tree implementation
//-----------------------------------------------------------
template <class T>
Tree<T>::~Tree() // deallocate tree
{
	if (root != NULL)
		clear(root);
}
template <class T>
void Tree<T>::clear(Node* current)
{
	if (current)
	{	// Release memory associated with children
		if (current->left)
			clear(current->left);
		if (current->right)
			clear(current->right);
		delete current;
	}
}

template <class T>
int Tree<T>::isEmpty() const
{
	return  root == NULL;
}

// preOrder processing of tree rooted at current
template <class T>
void Tree<T>::preOrder(Node* current)
{	// visit Node, left child, right child
	if (current)
	{	// process current Node
		process(current->value);
		// then visit children
		preOrder(current->left);
		preOrder(current->right);
	}
}

// inOrder processing of tree rooted at current
template <class T>
void Tree<T>::inOrder(Node* current)
{	// visit left child, Node, right child
	if (current)
	{
		inOrder(current->left);
		process(current->value);
		inOrder(current->right);
	}
}

// postOrder processing of tree rooted at current
template <class T>
void Tree<T>::postOrder(Node* current)
{	// visit left child, right child, node
	if (current)
	{
		postOrder(current->left);
		postOrder(current->right);
```

```
                process(current->value);
        }
}
```

המחלקה SearchTree:

```
#pragma once
#include "Tree.h"
#include <iostream>
using namespace std;

template<class T>
class SearchTree : public Tree<T>
{

public:
        void add(T value);
        bool  search(T value)
        {
                return search(Tree<T>::root, value);
        }

        }

private:
        void add(typename Tree<T>::Node* current, T val);
        bool  search(typename Tree<T>::Node* current, T val);
};

template <class T>
void SearchTree<T>::add(T val)
{
        // add value to binary search tree
        if (!Tree<T>::root)
        {
                Tree<T>::root = new typename Tree<T>::Node(val);
                return;
        }
        add(Tree<T>::root, val);
}

template <class T>
void SearchTree<T>::add(typename Tree<T>::Node* current, T val)
{
        if (current->value < val)

                if (!current->right)
                {
                        current->right = new typename Tree<T>::Node(val);
                        return;
                }
                else add(current->right, val);
        else

                if (!current->left)
                {
                        current->left = new typename Tree<T>::Node(val);
                        return;
                }
                else add(current->left, val);
}
```

```
template <class T>
bool SearchTree<T>::search(typename Tree<T>::Node* current, T val)
{
	// see if argument value occurs in tree
	if (!current)
		return false; // not found
	if (current->value == val)
		return true;
	if (current->value < val)
		return search(current->right, val);
	else
		return search(current->left, val);
}
```

המחלקה Queue:

```
#pragma once
#include <iostream>
using namespace std;
class Queue
{
public:
	virtual ~Queue() {};
	virtual void clear() = 0;
	virtual void enqueue(int value) = 0;
	virtual int dequeue() = 0;
	virtual int front() = 0;
	virtual bool isEmpty() const = 0;
};
```

המחלקה QueueVector:

```
#pragma once
#include "Queue.h"
class QueueVector : public Queue
{
public:
	QueueVector(int max);
	//QueueVector(const QueueVector&);
	void clear() override;
	int dequeue() override;
	void enqueue(int value) override;
	int front() override;
	bool isEmpty() const override;
private:
	int* data;
	int capacity;
	int nextSlot;
	int firstUse;
};
QueueVector::QueueVector(int size) {
	capacity = size + 1;
	data = new int[capacity];
	clear();
}
void QueueVector::clear() {
	nextSlot = 0;
	firstUse = 0;
}
```

```
int QueueVector::dequeue()
{
	if (isEmpty()) throw "Queue is empty\n";
	int dataloc = firstUse;
	++firstUse %= capacity;
	return data[dataloc];
}
void QueueVector::enqueue(int val) {
	if ((nextSlot + 1) % capacity == firstUse)
		throw "the Queue is full\n";
	data[nextSlot] = val;
	++nextSlot %= capacity;
}
int QueueVector::front() {
	if (isEmpty())
		throw "Queue is empty\n";
	return data[firstUse];
}
bool QueueVector::isEmpty() const {
	return nextSlot == firstUse;
}
```