**Logo, company name

Description automatically generated**

**Department of (Computer Science)**

**Pak-Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur, Pakistan**

**COMP-112L Data Structure** **& Algorithm Lab**

**Lab Journal**

**Class: BS Computer Science**

**Name: Ahmed Raza**

**Registration No.: B20F0436CS031**

**Semester: 4th**

**Submission Date: 26th April 2022**

**Submitted to: Engr. Rafi-Ullah**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Instructor Signature**

**Lab No. 10**

**Binary trees, insertion/deletion in BT & tree traversal**

**Objectives:**

In this lab we will be discussing about Binary trees, insertion/deletion in binary trees and tree traversal in detail. This is one of the most important concepts in Data Structure C++ language. Tree is a nonlinear data structure that models a hierarchical organization.

**Tools/Software Required:**

* All the tasks are implemented on DEV C++.

**Introduction:**

**TREES**

Tree is a nonlinear data structure that models a hierarchical organization. The characteristic features are that each element may have several successors (called its “children”) and every element except one (called the “root”) has a unique predecessor (called its “parent”). A Binary Tree is made of nodes, where each node contains a "left" pointer, a "right" pointer, and a data element. The "root" pointer points to the topmost node in the tree. The left and right pointers recursively point to smaller "subtrees" on either side. A null pointer represents a binary tree with no elements the empty tree. A binary tree is either empty (represented by a null pointer), or is made of a single node, where the left and right pointers (recursive definition ahead) each point to a binary tree. A rooted tree is a tree in which a special ("labeled") node is singled out. This node is called the "root" or (less commonly) "eve" of the tree. Rooted trees are equivalent oriented trees.

**Traversals of Binary Trees:**

**Depth First Search**:

* **Preorder** traversal sequence: F, B, A, D, C, E, G, I, H (root, left, right).
* **Inorder** traversal sequence: A, B, C, D, E, F, G, H, I (left, root, right); note how this produces a sorted sequence
* **Postorder** traversal sequence: A, C, E, D, B, H, I, G, F (left, right, root)

**Breadth First:**

* **Levelorder** traversal sequence: F, B, G, A, D, I, C, E, H.

**Lab Tasks:**

**Task 1:**

Perform a Pre-order, Post-order and In-order traversing on the following BST.

**Code:**

**#include<iostream>**

**using namespace std;**

**class TreeNode**

**{**

**public:**

**int value;**

**TreeNode \*left;**

**TreeNode \*right;**

**bool printGivenLevel(TreeNode\*, int num);**

**void insertNode(int num);**

**};**

**TreeNode \*root=NULL;**

**//INSERT**

**void TreeNode :: insertNode(int num)**

**{**

**TreeNode \*temp= new TreeNode; // Pointer to traverse the**

**TreeNode \*nodePtr= new TreeNode; // Create a new node**

**temp->value = num;**

**temp->left = temp->right = NULL;**

**if (!root) // Is the tree empty?**

**root = temp;**

**else**

**{**

**nodePtr = root;**

**while (nodePtr != NULL)**

**{**

**if (num < nodePtr->value)**

**{**

**if (nodePtr->left)**

**nodePtr = nodePtr->left;**

**else**

**{**

**nodePtr->left = temp;**

**break;**

**}**

**}**

**else if (num > nodePtr->value)**

**{**

**if(nodePtr->right)**

**nodePtr = nodePtr->right;**

**else**

**{**

**nodePtr->right = temp;**

**break;**

**}**

**}**

**else**

**{**

**cout << "Duplicate value found in tree.\n";**

**break;**

**}}}}**

**//INORDER**

**void displayInorder(TreeNode \*nodePtr)**

**{**

**if(nodePtr)**

**{**

**displayInorder(nodePtr->left);**

**cout<<nodePtr->value<<",";**

**displayInorder(nodePtr->right);**

**} }**

**//PREORDER**

**void displayPreorder(TreeNode \*nodePtr)**

**{**

**if(nodePtr)**

**{**

**cout<<nodePtr->value<<",";**

**displayPreorder(nodePtr->left);**

**displayPreorder(nodePtr->right);**

**}}**

**//POSTORDER**

**void displayPostorder(TreeNode \*nodePtr)**

**{**

**if(nodePtr)**

**{**

**displayPostorder(nodePtr->left);**

**displayPostorder(nodePtr->right);**

**cout<<nodePtr->value<<",";**

**}}**

**int main() {**

**int n,num;**

**cout << "Inserting nodes. "<<endl;**

**cout<<"Enter number of nodes you want to enter: ";**

**cin>>n;**

**while(n>0)**

**{**

**cin>>num;**

**root->insertNode(num);**

**n--;**

**}**

**cout << "Done.\nInorder Display:\n"; displayInorder(root);**

**cout << "\nPreorder Display:\n"; displayPreorder(root);**

**cout << "\nPostorder Display:\n"; displayPostorder(root);**

**return 0;**

**}**

**Output:**

**Text

Description automatically generated**

**Task # 02:**

You have to maintain information for a shop owner. For each of the products sold in his/hers shop the following information is kept: a unique code, a name, a price, amount in stock, date received, and expiration date. For keeping track of its stock, the clerk would use a computer program based on a search tree data structure. Write a program to help this person, by implementing following the following

operations:

• Insert an item with all its associated data.

• Find an item by its code, and support updating of the item found.

• List valid items in lexicographic order of their names.

• List expired items in lexicographic order of their names.

• List all items.

• Delete an item given by its code.

• Delete all expired items.

• Create a separate search tree for expired items.

• Save stock in file.

• Exit

**Code:**

**#include<iostream>**

**#include<iomanip>**

**#include<fstream>**

**#include<cstring>**

**using namespace std;**

**class Item {**

**public:**

**string code;**

**string name;**

**string price;**

**string amount;**

**string dateReceived;**

**string expiryDate;**

**Item\* left;**

**Item\* right;**

**Item(){**

**code = price = amount = name = dateReceived = expiryDate = "";**

**left = right = NULL;**

**}**

**Item(string c, string n, string p, string a, string r, string e){**

**code = c;**

**name = n;**

**price = p;**

**amount = a;**

**dateReceived = r;**

**expiryDate = e;**

**left = right = NULL;**

**}**

**void display(){**

**cout<<std::left**

**<<setw(5)<<code**

**<<setw(15)<<name**

**<<setw(10)<<price**

**<<setw(10)<<amount**

**<<setw(20)<<dateReceived**

**<<setw(20)<<expiryDate<<endl;**

**}**

**};**

**class BinaryTree {**

**private:**

**Item\* root;**

**int numberOfItems;**

**public:**

**BinaryTree(){**

**root = NULL;**

**numberOfItems = 0;**

**}**

**Item\* getRoot(){**

**return root;**

**}**

**int getNumberOfItems(){**

**return numberOfItems;**

**}**

**void insert(Item\* newItem){**

**if (!root)**

**root = newItem;**

**else {**

**Item\* currItem = root;**

**while (currItem != NULL){**

**if (newItem->code < currItem->code) {**

**if (currItem->left)**

**currItem = currItem->left;**

**else{**

**currItem->left = newItem;**

**numberOfItems++;**

**break;**

**}**

**} else if (newItem->code > currItem->code){**

**if (currItem->right)**

**currItem = currItem->right;**

**else {**

**currItem->right = newItem;**

**numberOfItems++;**

**break;**

**}**

**} else {**

**cout << "Duplicate Code. Code must be unique"<<endl;**

**break;**

**}**

**}**

**}**

**}**

**void find(Item\* currItem, int code){**

**if(currItem){**

**if (stoi( currItem->code ) == code){**

**cout<<"Item with code "<<code<<" found"<<endl;**

**cout<<left<<setw(5)<<"Code"<<setw(15)<<"Name"<<setw(10)<<"Price"<<setw(10)<<"Amount"<<setw(20)<<"Date Received"<<setw(20)<<"Expiry Date"<<endl;**

**currItem->display(); char choice;**

**cout<<"\n\nWould you like to update this item(Y/N)? : ";**

**cin>>choice;**

**if (toupper( choice ) == 'Y')**

**update(currItem);**

**return;**

**}**

**find(currItem->left, code);**

**find(currItem->right, code);**

**}**

**}**

**void update(Item\* currItem){**

**cout<<"Enter new name : ";**

**cin.ignore(); getline(cin, currItem->name);**

**cout<<"Enter new Price : ";**

**cin>>currItem->price;**

**cout<<"Enter new Amount : ";**

**cin>>currItem->amount;**

**cout<<"Enter new Date received : ";**

**cin.ignore();getline(cin, currItem->dateReceived);**

**cout<<"Enter new Expiry date : ";**

**getline(cin, currItem->expiryDate);**

**}**

**void display(Item\* currItem){**

**if (currItem){**

**currItem->display();**

**display(currItem->left);**

**display(currItem->right);**

**}**

**}**

**void displayValidItems(Item\* currItem){**

**if (currItem){**

**if (strcmp(currItem->dateReceived.c\_str(), currItem->expiryDate.c\_str()) <= 0)**

**currItem->display();**

**displayValidItems(currItem->left);**

**displayValidItems(currItem->right);**

**}**

**}**

**void displayExpiredItems(Item\* currItem){**

**if (currItem){**

**if (strcmp(currItem->dateReceived.c\_str(), currItem->expiryDate.c\_str()) > 0)**

**currItem->display();**

**displayExpiredItems(currItem->left);**

**displayExpiredItems(currItem->right);**

**}**

**}**

**Item\* deleteItem(Item\* currentItem, int code){**

**if (currentItem == NULL)**

**return currentItem;**

**if ( code < stoi( currentItem->code ) ){**

**currentItem->left = deleteItem(currentItem->left, code);**

**} else if ( code > stoi( currentItem->code ) ){**

**currentItem->right = deleteItem(currentItem->right, code);**

**} else {**

**if (currentItem->left == NULL && currentItem->right == NULL){**

**delete currentItem;**

**currentItem = NULL;**

**} else if (currentItem->left == NULL){**

**Item\* temp = currentItem;**

**currentItem = currentItem->right;**

**delete temp;**

**} else if (currentItem->right == NULL){**

**Item\* temp = currentItem;**

**currentItem = currentItem->left;**

**delete temp;**

**} else{**

**Item\* temp = findSmallestItem(currentItem->right);**

**currentItem->code = temp->code;**

**currentItem->right = deleteItem(currentItem->right, stoi( currentItem->code ));**

**}**

**numberOfItems--;**

**}**

**root = currentItem;**

**return currentItem;**

**}**

**Item\* deleteAllExpiredItems(Item\* currItem){**

**if (currItem){**

**if ( strcmp( currItem->dateReceived.c\_str(), currItem->expiryDate.c\_str() ) > 0 )**

**deleteItem(root, stoi( currItem->code ));**

**deleteAllExpiredItems(currItem->left);**

**deleteAllExpiredItems(currItem->right);**

**}**

**}**

**Item\* findSmallestItem(Item\* currentItem){**

**if (currentItem == NULL)**

**return NULL;**

**else if ( currentItem->left ==NULL)**

**return currentItem;**

**return findSmallestItem(currentItem->left);**

**}**

**void putDataInFile(ofstream& file, Item\* currItem){**

**if (!currItem)**

**return;**

**putDataInFile(file, currItem->left);**

**file<<"\n"<<currItem->code<<","<<currItem->name<<","<<currItem->price<<","<<currItem->amount<<","<<currItem->dateReceived<<","<<currItem->expiryDate<<",";**

**putDataInFile(file, currItem->right);**

**}**

**void saveStockInFile(){**

**ofstream file("stock.txt", ios::out);**

**putDataInFile(file, root);**

**file.close();**

**}**

**void putExpiredInSeparateTree(BinaryTree &tree2, Item\* currItem){**

**if (!currItem)**

**return;**

**putExpiredInSeparateTree(tree2,currItem->left);**

**if ( strcmp( currItem->dateReceived.c\_str(), currItem->expiryDate.c\_str() ) > 0 )**

**tree2.insert(currItem);**

**putExpiredInSeparateTree(tree2, currItem->right);**

**}**

**};**

**int displayMenu();**

**Item\* createItem();**

**int main(){**

**BinaryTree BT;**

**BinaryTree binaryTreeOfExpiredItems;**

**ifstream checkFile("stock.txt", ios::in);**

**if(!checkFile.good()){**

**ofstream file("stock.txt", ios::out);**

**file.close();**

**checkFile.close();**

**}**

**ifstream file("stock.txt", ios::in);**

**string data;**

**getline(file, data);**

**while(!file.eof()){**

**Item\* item = new Item();**

**getline(file, item->code, ',');**

**getline(file, item->name, ',');**

**getline(file, item->price, ',');**

**getline(file, item->amount, ',');**

**getline(file, item->dateReceived, ',');**

**getline(file, item->expiryDate, ',');**

**BT.insert(item);**

**getline(file,data);**

**}**

**file.close();**

**while (true)**

**switch(displayMenu()){**

**case 1:**

**BT.insert( createItem() );**

**break;**

**case 2:{**

**int code;**

**cout<<"Enter code of item to find : ";**

**cin>>code;**

**BT.find(BT.getRoot(), code);**

**break;**

**}**

**case 3:**

**cout<<"=====Valid Items======\n";**

**cout<<left<<setw(5)<<"Code"<<setw(15)<<"Name"<<setw(10)<<"Price"<<setw(10)<<"Amount"<<setw(20)<<"Date Received"<<setw(20)<<"Expiry Date"<<endl;**

**BT.displayValidItems(BT.getRoot());**

**break;**

**case 4:**

**cout<<"=====Expired Items======\n";**

**cout<<left<<setw(5)<<"Code"<<setw(15)<<"Name"<<setw(10)<<"Price"<<setw(10)<<"Amount"<<setw(20)<<"Date Received"<<setw(20)<<"Expiry Date"<<endl;**

**BT.displayExpiredItems(BT.getRoot());**

**break;**

**case 5:**

**cout<<left<<setw(5)<<"Code"<<setw(15)<<"Name"<<setw(10)<<"Price"<<setw(10)<<"Amount"<<setw(20)<<"Date Received"<<setw(20)<<"Expiry Date"<<endl;**

**BT.display(BT.getRoot());**

**break;**

**case 6:{**

**int code;**

**cout<<"Enter Code of Item You Want To Delete : ";**

**cin>>code;**

**BT.deleteItem(BT.getRoot(), code);**

**cout<<"=====Successfully Deleted Item of Code "<<code<<"====="<<endl;**

**break;**

**}**

**case 7:{**

**BT.deleteAllExpiredItems(BT.getRoot());**

**cout<<"=====Expired Items Successfully Deleted====="<<endl;**

**break;**

**}**

**case 8:{**

**BT.putExpiredInSeparateTree(binaryTreeOfExpiredItems, BT.getRoot());**

**cout<<"=====Expired Items Successfully Put Into Separate Binary Tree====="<<endl;**

**break;**

**}**

**case 9:{**

**BT.saveStockInFile();**

**cout<<"=====Stock Successfully Stored in 'stock.txt' File====="<<endl;**

**break;**

**}**

**case 10:{**

**cout<<"Exiting Program"<<endl;**

**exit(0);**

**break;**

**}**

**}**

**}**

**int displayMenu(){**

**int choice;**

**cout<<"\n\n==================================================================\n";**

**cout<<"1 -> Insert an Item\n"**

**<<"2 -> Find Item and Update it\n"**

**<<"3 -> List valid items\n"**

**<<"4 -> List expired items\n"**

**<<"5 -> List all items\n"**

**<<"6 -> Delete item by its code\n"**

**<<"7 -> Delete all expired items\n"**

**<<"8 -> Create a separate Binary Tree for expired Items\n"**

**<<"9 -> Save current stock in file\n"**

**<<"10 -> Exit"<<endl;**

**do {**

**cout<<"Enter Your Choice : ";**

**cin>>choice;**

**if (choice < 1 || choice > 10)**

**cout<<"Please enter a number between 1 and 10"<<endl;**

**} while (choice < 1 || choice > 10);**

**cout<<"\n\n\n";**

**return choice;**

**}**

**Item\* createItem(){**

**Item\* newItem = new Item();**

**cout<<"Enter code : ";**

**cin>>newItem->code;**

**cout<<"Enter name : ";**

**cin.ignore();**

**getline(cin,newItem->name);**

**cout<<"Enter price : ";**

**cin>>newItem->price;**

**cout<<"Enter amount : ";**

**cin>>newItem->amount;**

**cout<<"Enter date received (yyyy/mm/dd) : ";**

**cin>>newItem->dateReceived;**

**cout<<"Enter expiry date (yyyy/mm/dd) : ";**

**cin>>newItem->expiryDate;**

**return newItem;**

**}**

**Output:**

**INSERT ITEM:**

**Text

Description automatically generated**

**EXPIRED ITEM PUT IN SEPARATE FILE:**

**Text

Description automatically generated**

**SAVE ITME IN FILE:**

**Text

Description automatically generated**

**DISPLAY ITEM:**

**Text

Description automatically generated**

**Results & Observations:**

In this Lab I've learned about the concept of Binary trees & also understand that by using recursion, we can simplify implementation of binary tress to a great extent & returned to a main function. In the first task, I've used traversing concept by that how can we perform in-order, preorder & post-order by using function. In the second task, I’ve used tree node in which I’m inserting node that’s actually inserting the item then displaying it and delete it by using delete function.