|  |
| --- |
| **Computer Engineering Department - ITU** |
| **CE200L: Data Structures & Algorithms Lab** |

|  |  |
| --- | --- |
| **Course Instructor: Usama Bin Shakeel** | **Dated: 27/10/2022** |
| **Teaching Assistant: Muhammad Sufyan Ashraf** | **Semester: Fall 2022** |
| **Lab Engineer: Nadir Abbas** | **Batch: BSCE2021** |

# **Lab 9B. Full Tree Implementation with Pointers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Name** | **Roll number** | **Report**  **(out of 100)** | **Scaled to 10** | **Total**  **(out of 10)** |
| NIMRA MAQBOOL | BSCE21012 |  |  |  |

Checked on: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

## **Objective**

The objective of this lab is to provide the knowledge of basic data structures and their implementations.

## **Equipment and Component**

|  |  |  |
| --- | --- | --- |
| **Component Description** | **Value** | **Quantity** |
| Computer | Available in lab | 1 |

## **Conduct of Lab**

1. Students are required to perform this experiment individually.
2. In case the lab experiment is not understood, the students are advised to seek help from the course instructor, lab engineers, assigned teaching assistants (TA) and lab attendants.

## **Theory and Background**

In computer science, a **binary tree** is a tree data structure in which each node has at most two children, which are referred to as the left child and the right child.

A **pointer** is a variable that stores the address of another variable. Unlike other variables that hold values of a certain type, pointer holds the address of a variable. For example, an integer variable holds (or you can say stores) an integer value, however an integer pointer holds the address of a integer variable. A binary tree in which all nodes except leaf nodes have two children. A full Binary tree is a special type of binary tree in which every parent node/internal node has either two or no children. It is also known as a proper binary tree.

**Templates** are a feature of the C++ programming language that allows functions and classes to operate with generic types. This allows a function or class to work on many different data types without being rewritten for each one.

**Lab Task**

**Task A**

As you have implemented the complete binary tree before, Now implements full binary tree with pointers. Two pointers for each tree node for two children Check using them if the tree is full or not. Implement the following functions:

Insert

Update

Delete

Display

Check

Make all necessary functions and handle all corner cases. Make a menu driven program.

|  |
| --- |
| // Paste your code here  **FUNCTION.H:**  #include<iostream>  using namespace std;  class node { public:  int data;  node \*Left;  node \*Right;  node \*Parent;   node() {  Left = nullptr;  Right = nullptr;  }   node(int Data) {  data = Data;  Left = nullptr;  Right = nullptr;  Parent = nullptr;   }   node \*Insert(node \*temp, int value) {  if (temp == nullptr) {  temp = new node();  temp->data = value;  temp->Left = nullptr;  temp->Right = nullptr;  temp->Parent = nullptr;  } else if (temp->data < value) {  temp->Right = Insert(temp->Right,  value); //if the given data is greater than the data given in node then insert at right side  temp->Right->Parent = temp;  } else {  temp->Left = Insert(temp->Left,  value); //if the data is smaller than the data in node then it is written at left side of the tree  temp->Left->Parent = temp;  }  return temp;  }   void printPREOrder(node \*temp) {  if (temp == nullptr)  return;  cout << temp->data << " "; //if node is not null then print its value  printPREOrder(temp->Left); //recursive call for the left side of the node  printPREOrder(temp->Right); //recursive call for the right side of the node  cout << " ";  }    node \*Search(node \*temp, int value) {  if (temp == nullptr) //if temp is null  return nullptr;  if (temp->data == value) //if the value of temp is equal to the value given the return that temp  return temp;   if (temp->data > value) { //if given node is smaller than the current node  return Search(temp->Left, value);  } else {  return Search(temp->Right, value);  //if current node value is less than the given node  }  }   int FindMin(node \*temp) {  if (temp == nullptr) {  return 0;  }  if (temp->Left == nullptr) {  return temp->data;  } else {  return FindMin(temp->Left);  }   }   int minFromRight(node \*temp) {  if (temp->Right != nullptr) {  return FindMin(temp->Right); // to find the min value in the right sub tree  }  }   node \*Remove(node \*temp, int value) {  temp = Search(temp, value); //calling the search function  if (temp == nullptr) //if node is null  return nullptr;  if (temp->data == value) { //if leaf node  if (temp->Left == nullptr && temp->Right == nullptr)  temp = nullptr; // have only one child at right  else if (temp->Left == nullptr && temp->Right != nullptr) {  temp->Right->Parent = temp->Parent;  temp = temp->Right;  } // have only one child at left  else if (temp->Left != nullptr && temp->Right == nullptr) {  temp->Left->Parent = temp->Parent;  temp = temp->Left;  } // have two children (left and right)  else {  int a = minFromRight(temp); //find the min from right side   temp->data = a; //copy that value in it  temp->Right = Remove(temp->Right, a); //delete the old one  }  }  else if (temp->data, Right = Remove(temp->Right, value)); //if the node data is small  else {  temp->Left = Remove(temp->Left, value); } //if data is greater  return temp;  }   int countNumNodes(node \*temp) {  if (temp == nullptr) {  return 0;  }  return (1 + countNumNodes(temp->Left) +  countNumNodes(temp->Right)); //it is counting the node in recursive call  }   bool checkComplete(node \*temp, int index, int numberNodes) {   if (temp == nullptr) {  return true;  } //if tree is empty   if (index >= numberNodes) {  return false;  } //if index is greater   return (checkComplete(temp->Left, 2 \* index + 1, numberNodes) &&  // it is checking that the nodes of left is equal the nodes of right through recursive call  checkComplete(temp->Right, 2 \* index + 2, numberNodes));  }   void update(int key, node \*temp) {  if (temp == nullptr) //if it is empty  return;  temp->data = key / (temp->data);  update(key, temp->Right); // Go to the RIGHT tree  update(key, temp->Left); //go to LEFT side of tree  } };  **MAIN.CPP:**  // // Created by Lenovo on 10/27/2022. // #include<iostream> #include "Functions.h"  using namespace std;  int main() {  node n(0);  node \*root = nullptr; // Creating an empty tree  int opt;  do {   cout << "\nENTER OPTIONS." << endl;  cout << "1.INSERT." << endl;  cout << "2.DELETE." << endl;  cout << "3.UPDATE." << endl;  cout << "4.CHECK IF THE TREE IS BINARY OR NOT." << endl;  cout << "5.EXIT." << endl;  cin >> opt;  if (opt == 1) {  root = n.Insert(root, 12);  root = n.Insert(root, 19);  root = n.Insert(root, 5);  root = n.Insert(root, 6);  root = n.Insert(root, 1);  root = n.Insert(root, 13);  cout << "THE TREE IS = ";  n.printPREOrder(root);  }  if (opt == 2) {  n.Remove(root, 5);  cout << "THE TREE IS = ";  n.printPREOrder(root);  }  if (opt == 3) {  int node\_count = n.countNumNodes(root);  int index = 0;   if (n.checkComplete(root, index, node\_count))  cout << "\nThe Binary Tree is complete\n";  else  cout << "\nThe Binary Tree is not complete\n";  }  if (opt == 4) {  n.update(3, root);  cout << "AFTER PUTTING 3 AT 3 INDEX AND ZEROS AT OTHER THE TREE IS ." << endl;  cout << "THE TREE IS = ";  n.printPREOrder(root);  }  if (opt == 5) {  cout << "YOU CHOOSE TO EXIT." << endl;  exit(4);  }   } while (opt >= 1 && opt <= 5); }  // Paste your output here |

#### **Assessment Rubric for Lab**

**Method for assessment:**

Lab reports and instructor observation during lab sessions. Outcome assessed:

a. Ability to conduct experiments, as well as to analyze and interpret data (P) b. Ability to function on multi-disciplinary teams (A)

c. Ability to use the techniques, skills, and modern engineering tools necessary for engineering practice (P)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Performance metric** | **Task** | **CLO** | **Description** | **Max marks** | **Exceeds expectation** | **Meets expectation** | **Does not meet expectation** | **Obtained marks** |
| 1. Realization of experiment (a) | 1 | 1 | Functionality | 40 | Executes without errors excellent user prompts, good use of symbols, spacing in output. Through testing has been completed (35-40) | Executes without errors, user prompts are understandable, minimum use of symbols or spacing in output. Some testing has been completed (20-34) | Does not execute due to syntax errors, runtime errors, user prompts are misleading or non-existent. No testing has been completed (0-19) |  |
| 2. Teamwork (b) | 1 | 3 | Group Performance | 5 | Actively engages and cooperates with other group member(s) in effective manner (4-5) | Cooperates with other group member(s) in a reasonable manner but conduct can be improved (2-3) | Distracts or discourages other group members from conducting the experiment (0-1) |  |
| 3. Conducting experiment (a, c) | 1 | 1 | On Spot Changes | 10 | Able to make changes (8-10) | Partially able to make changes (5-7) | Unable to make changes (0-4) |  |
| 1 | 1 | Viva | 10 | Answered all questions (8-10) | Few incorrect answers (5-7) | Unable to answer all questions (0-4) |  |
| 4. Laboratory safety and disciplinary rules (a) | 1 | 3 | Code commenting | 5 | Comments are added and does help the reader to understand the code (4-5) | Comments are added and does not help the reader to understand the code (2-3) | Comments are not added (0-1) |  |
| 5. Data collection (c) | 1 | 3 | Code Structure | 5 | Excellent use of white space, creatively organized work, excellent use of variables and constants, correct identifiers for constants, No line-wrap (4-5) | Includes name, and assignment, white space makes the program fairly easy to read. Title, organized work, good use of variables (2-3) | Poor use of white space (indentation, blank lines) making code hard to read, disorganized and messy (0-1) |  |
| 6. Data analysis (a, c) | 1 | 4 | Algorithm | 20 | Solution is efficient, easy to understand, and maintain (15-20) | A logical solution that is easy to follow but it is not the most efficient (6-14) | A difficult and inefficient solution (0-5) |  |
| 7. Computer use (c) | 1 | 2 | Documentation & Github Submissions | 5 | Timely (4-5) | Late (2-3) | Not done (0-1) |  |
|  | Max Marks (total): | | | 100 | Obtained Marks (total): | | |  |

Lab Engineer Signature: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_