|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***National University of Computer and Emerging Sciences, Lahore Campus*** | | | | |
| C:\Users\saif\AppData\Local\Microsoft\Windows\Temporary Internet Files\Content.Word\final design.jpg | **Course:** | Data Structures | **Course Code:** | CS 201 |
| **Program:** | BS(CS) | **Semester:** | Fall 2019 |
| **Duration:** | 15 Minutes | **Total Marks:** | 10 |
| **Paper Date:** | 24 Oct 2019 | **Exam** | Quiz 3 |
| **Section:** | **D** |  |  |
| **Instruction/Notes:** | Solve the exam on this question paper. | | | |

**Question:** A binary tree used in representing arithmetic expressions is called an Expression tree. The internal nodes are operators and the leaves are operands. The sub-trees are subexpression. The precedence in the expression tree is bottom up that is operators closer to the operands (leaves) are evaluated first. You can assume that expression can have only +,-, \* and / operators.

You are given a template-based class for Binary Tree and BinaryNode. **Write a C++ program that takes a Binary Expression tree and prints the expression and returns the result. You can use built-in string function (std::stoi ) for converting string to integer if required. std::stoi function takes in string and returns integer.**

|  |  |  |
| --- | --- | --- |
| **template<class T>**  **class BNode{**  **T data**  **BNode<T> \*left, \*right;**  **}**  **template<class T>**  **class BinaryTree{**  **private:**  **BNode<T> \* root;**  **public:**  **… // functions**  **}** | **Input** | **Output**  **You program should print**  **( (5 + (3 \* 2)) - (18 / (3 \* 3)) )**  **and return**  **9**  **To caller** |

**void BinaryTree::EvaluateExpression()**

**{**

**EvaluateExpression (root);**

**}**

**int BinaryTree:: EvaluateExpression (BNode \* t)**

**{**

**if (t)**

**{**

**if (t->left == null && t->right == null)** //left or right child is null so it is operand ---------- 1 marks

**{**

**cout<< t-> data;** // print data ---------- 1 marks

**return std::stoi(t->data);** // convert data for arithmetic ---------- 1 marks

**}**

**else** //print and evaluate an expression

**{**

**cout<< “( “;** // manage starting bracket ---------- 1 marks

**int val1 = EvaluateExpression (t→left);** // left recursive call---------- 0.5 marks

**cout<< t-> data;**  //output operator---------- 1 marks

**int val2 = EvaluateExpression (t→right);** // right recursive call---------- 0.5 marks **cout<< ”)”;** // manage ending bracket---------- 1 marks

**if (t->data == “\*”)** // data arithmetics---------- 3 marks

**return val1\*val2;**

**else if (t->data == “/”)**

**return val1/val2;**

**else if (t->data == “+”)**

**return val1+val2;**

**else if (t->data == “-”)**

**return val1-val2;**

**else return 1;**

**}**

**}**

**}**

If you are an operator then you have expressions (brackets) as well as operands or sub-expression