# Comp 4603

# Advanced C++

|  |  |  |  |
| --- | --- | --- | --- |
| Assignment | 3 | Part | 1 |

Student Name: Alisher Shamayev

BCIT ID: A01182685

Task 1: Your definition of Queue<T>

Graphical user interface

Description automatically generated

Queue.hpp

|  |
| --- |
| //  // Queue.hpp  // Lab3P1  //  // Created by GUW06 on 2023-01-26.  //  #ifndef Queue\_hpp  #define Queue\_hpp  #include <stdio.h>  #include <iostream>  #include <list>  **template** <**typename** T>  **class** Queue {  **private**:  std::list<T> queue;  **public**:  **void** enqueue(T element) ;  T dequeue();  **int** getSize() **const**;  };  #endif /\* Queue\_hpp \*/ |

Queue.cpp

|  |
| --- |
| //  // Queue.cpp  // Lab3P1  //  // Created by GUW06 on 2023-01-26.  //  #include "Queue.hpp"  #include <iostream>  **template** <**typename** T>  **void** Queue<T>::enqueue(T element) {  queue.push\_back(element);  }  **template**<**typename** T>  T Queue<T>::dequeue() {  **if** (queue.empty()) {  **throw** std::out\_of\_range("Queue is empty");  }  T element = queue.front();  queue.pop\_front();  **return** element;  }  **template**<**typename** T>  **int** Queue<T>::getSize()**const**{  **return** queue.size();  } |

Now, let’s move on to BST, and here’s the Node definition:

Text

Description automatically generated

Task 2: Using given BST node setup, define your In-order traversal function

|  |
| --- |
| **template**<**typename** T>  **void** BST<T>::inorder()  {  // your code  inorderPointer( element);  }  **template**<**typename** T>  **void** BST<T>::inorderPointer(Node\* current){  **if**(current==**NULL**)  cout<<"BST is NULL"<<endl;  inorderPointer(current->left);  inorderPointer(current->right);  } |

Task 3:Using given BST node setup, define your search function

|  |
| --- |
| **template**<**typename** T>  **bool** BST<T>::search(T element)  {  // your code goes here  TreeNode<T>\* root=**new** TreeNode<T>(element);  **while** (root != **nullptr**) {  **if** (root->element == element) {  **return** **true**;  }  **if** (root->element < element) {  root = root->right;  } **else** {  root = root->left;  }  }  **return** **false**;  } |

Task 4: Using given BST node setup, define your count function to return the count of nodes (this is not only for BST, can be used in any Tree)

|  |
| --- |
| **template**<**typename** T>  **int** BST<T>::count()  {  // your code goes here  TreeNode<T>\* root= **new** TreeNode<T>(element);  **int** leftSide=0;  **int** rightSide=0;  **int** nodeRoot=1;  **if**(root==**nullptr**) **return** 0;  **while**(root!=**nullptr**){  left= root->left;  right=root->right;  leftSide++;  rightSide++;  }  **return** nodeRoot + leftSide + rightSide;  } |