CSII PRAC EXAM for FINAL

Templated average function with exception handler

Write a function template for the average function. The function takes in an array of elements of a generic data type (int, long, float, double,…) and the size of the array. The function returns the average of the elements in the array. The return type is double. (Note: if the array size is less than or equal to zero, throw exception of integer type indicating the size of the array is invalid.)

Assume the existence of a class Err, and use this class (create an object) if and when you encounter an exception:

class Err {

 public:

    int ErrorCode;

    string ErrorDescription;

}

ANSWER:

template <class T>

double average(int size, T\* arr){

// - OR - double average(int size, T arr[] ){

if (size <= 0){

Err anError;

anError.ErrorCode = 5001;

anError.ErrorDescription = “no element in the array”;

throw anError;

}

T total;

for (int i=0; i<size; i++){

total += arr[i];

}

return total/(static\_cast<double>(size));

}

Binary search tree insert

The following the class template for a binary tree that can hold values of any data type. Your task is to **complete the coding for the member function that performs the insert** for this tree class. **Use the program's given constructs and variables as appropriate.**

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#include <iostream>

using namespace std;

template <class T>

class BinaryTree

{

public:

   struct TreeNode

   {

      T value;

      TreeNode \*left;

      TreeNode \*right;

   };

   TreeNode \*root;

   void insert(TreeNode \*&, TreeNode \*&);

   void destroySubTree(TreeNode \*);

   void deleteNode(T, TreeNode \*&);

   void makeDeletion(TreeNode \*&);

   void displayInOrder(TreeNode \*);

   void displayPreOrder(TreeNode \*);

   void displayPostOrder(TreeNode \*);

public:

   BinaryTree()      // Constructor

      { root = nullptr; }

   ~BinaryTree()  // Destructor

      { destroySubTree(root); }

   void insertNode(T);

   bool searchNode(T);

   void remove(T);

   void displayInOrder()

      { displayInOrder(root); }

   void displayPreOrder()

      { displayPreOrder(root); }

   void displayPostOrder()

      { displayPostOrder(root); }

};

// Write the insert function

// YOUR CODE HERE

|  |  |
| --- | --- |
| **Answer Key** |  |
| template <class T>  void BinaryTree<T>::insert(TreeNode \*&nodePtr, TreeNode \*&newNode)  {      if (nodePtr == nullptr)      {          nodePtr = newNode;      }      else if (newNode->value < nodePtr->value)      {          insert(nodePtr->left, newNode);      }      else      {          insert(nodePtr->right, newNode);      }  } | |

Binary Search Tree post order

The following the class template for a binary tree that can hold values of any data type. Your task is to **complete the coding for the function that performs an inorder, postorder or preorder display** for this tree class. **Use the program's given constructs and variables as appropriate.**

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#include <iostream>

using namespace std;

template <class T>

class BinaryTree

{

public:

   struct TreeNode

   {

      T value;

      TreeNode \*left;

      TreeNode \*right;

   };

   TreeNode \*root;

   void insert(TreeNode \*&, TreeNode \*&);

   void destroySubTree(TreeNode \*);

   void deleteNode(T, TreeNode \*&);

   void makeDeletion(TreeNode \*&);

   void displayInOrder(TreeNode \*);

   void displayPreOrder(TreeNode \*);

   void displayPostOrder(TreeNode \*);

public:

   BinaryTree()      // Constructor

      { root = nullptr; }

   ~BinaryTree()  // Destructor

      { destroySubTree(root); }

   void insertNode(T);

   bool searchNode(T);

   void remove(T);

   void displayInOrder()

      { displayInOrder(root); }

   void displayPreOrder()

      { displayPreOrder(root); }

   void displayPostOrder()

      { displayPostOrder(root); }

};

|  |  |
| --- | --- |
| **Answer Key**  template <class T>  void BinaryTree<T>::displayPostOrder(TreeNode \*nodePtr)  {     if (nodePtr)     {        displayPostOrder(nodePtr->left);        displayPostOrder(nodePtr->right);        cout << nodePtr->value << endl;     }  } |  |

A recursive palindrome function

A palindrome is a string that reads the same forwards or backwards; for example dad, mom, deed (i.e., reversing a palindrome produces the same string). Write a recursive, bool-valued function, isPalindrome that accepts a string and returns whether the string is a palindrome.

A string, s, is a palindrome if:

* s is the empty string or s consists of a single letter (which reads the same back or forward), or
* the first and last characters of s are the same, and the rest of the string (i.e., the second through next-to-last characters ) form a palindrome.

|  |  |
| --- | --- |
| **Answer Key**  bool isPalindrome(string s) {  if (s.length() <= 1) return true;  return s[0] == s[s.length()-1] && isPalindrome(s.substr(1, s.length()-2));  }  Other CORRECT solutions  bool isPalindrome(string word)  {  if(word.length() < 2) {  return true;  }  else if(word[0] == word[word.length() - 1]){  return isPalindrome(word.substr(1, word.length() - 2));  }  return false;  }   * OR -   bool isPalindrome(string s){  if(s.length() < 2)  return true;  else if (s[0] == s[s.length() - 1])  return isPalindrome(s.substr(1, s.length() - 2));    return false;  }   * OR -   bool isPalindrome(string s) {  if (s.length() <= 1) return true;  return s[0] == s[s.length()-1] && isPalindrome(s.substr(1, s.length()-2));  }   * OR -   bool isPalindrome ( string s )  {  if(s.length() < 2)  {  return true;  }  else if(s[0] == s[s.length() - 1])  {  return isPalindrome(s.substr(1, s.length() - 2));  }  return false;  }   * OR -   bool isPalindrome(string s)  {  if(s.size() <= 1)  {  return true;  }  if(s.at(0) != s.at(s.size()-1))  {  return false;  }  return isPalindrome(s.substr(1,s.size()-2));  }   * OR -   bool isPalindrome(string s)  {  if (s.size() <= 1) return true;  if (s.at(0) != s.at(s.size()-1)) return false;  return isPalindrome(s.substr(1,s.size()-2));  } |  |

Recursive nth power function

Given non-negative integers x and n, x taken to the nth power can be defined as:

* x to the 0th power is 1
* x to the nth power can be obtained by multiplying x to the n-1'th power with x

Write a long-valued function named power that accepts two int parameters x and n (in that order) and recursively calculates and returns the value of x taken to the n'th power.

|  |  |
| --- | --- |
| **Answer Key**  int power(int x, int n) {return n == 0 ? 1 : x \* power(x, n-1);}  Other CORRECT solutions  int power(int x,int n)  {  if(n==1)  return x;  if(n==0)  return 1;  return power(x, n-1) \* x;  }   * or –   long power(int x, int n)  {  if(n == 0)  {  return 1;  }  else  {  return x \* power(x, n -1);  }  }   * or –   long power(int x, int n)  {  if(n==0)  return 1;  else  return x \* power(x, n-1);  } |  |

Recursive factorial (fact) function

Write a function called fact that recursively calculates the factorial value of its single int parameter. The value returned by fact is a long.

|  |  |
| --- | --- |
| **Answer Key**  long fact(int n) {return n == 0 ? 1 : n \* fact(n-1);}  Other CORRECT solutions  long fact(int n) {  if (n <= 1) {  return 1;  } else {  return (fact(n-1) \* (long) n);  }  }   * or -   long fact(int x)  {  if (x <= 1)  {  return 1;  }  else  {  return (fact(x - 1) \* (long) x);  }  } |  |

PreOrder member function

The following the class template for a binary tree that can hold values of any data type. Your task is to complete the coding for the function that performs an inorder, postorder or preorder display for this tree class. Use the program's given constructs and variables as appropriate.

#ifndef BINARYTREE\_H

#define BINARYTREE\_H

#include <iostream>

using namespace std;

template <class T>

class BinaryTree

{

public:

struct TreeNode

{

T value;

TreeNode \*left;

TreeNode \*right;

};

TreeNode \*root;

void insert(TreeNode \*&, TreeNode \*&);

void destroySubTree(TreeNode \*);

void deleteNode(T, TreeNode \*&);

void makeDeletion(TreeNode \*&);

void displayInOrder(TreeNode \*);

void displayPreOrder(TreeNode \*);

void displayPostOrder(TreeNode \*);

public:

BinaryTree() // Constructor

{ root = nullptr; }

~BinaryTree() // Destructor

{ destroySubTree(root); }

void insertNode(T);

bool searchNode(T);

void remove(T);

void displayInOrder()

{ displayInOrder(root); }

void displayPreOrder()

{ displayPreOrder(root); }

void displayPostOrder()

{ displayPostOrder(root); }

};

// Write the displayPreOrder function

// YOUR CODE HERE

Question options:

The following the class template for a binary tree that can hold values of any data type. Your task is to complete the coding for the function that performs an inorder, postorder or preorder display for this tree class. Use the program's given constructs and variables as appropriate.

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**// The displayPreOrder member function displays the values \***

**//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

**Answer Key**

**template <class T>**

**void BinaryTree<T>::displayPreOrder(TreeNode \*nodePtr)**

**{**

**if (nodePtr)**

**{**

**cout << nodePtr->value << endl;**

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

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

**}**

**}**

## Void Verify

Assume the existence of a class RangeException, with a constructor that accepts minimum, maximum and violating integer values (in that order).

Write a function, void verify(int min, int max) that reads in integers from the standard input and compares them against its two parameters . As long as the numbers are between min and max (inclusively), the function continues to read in values . If an input value is encountered that is less than min or greater than max, the function throws a RangeException with the min and max values , and the violating (i.e. out of range) input.

**void verify(int min, int max)**

**{**

**int i;**

**cin >> i;**

**while (cin)**

**{**

**if (i < min || i > max) throw RangeException(min, max, i);**

**cin >> i;**

**}**

**}**