cs380su21-meta.sg

Dashboard / My courses / cs380su21-meta.sg / 10 May - 16 May / Assignment 1 (Tree)

Description Submission </> Edit Submission view

Grade

Reviewed on Thursday, 20 May 2021, 3:44 AM by Automatic grade
grade: 100.00 / 100.00

Assessment report [-]
[+]**Summary of tests**

Submitted on Thursday, 20 May 2021, 3:44 AM (Download)

functions.cpp

```
/*!*****************************************************************************
\file functions.cpp
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: A
\par Programming Assignment 1
\date 05-19-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/

#include "functions.h"

namespace AI
{
    /*!*************************************************************************
    \brief
    Function to convert a string to integer.

    \param str
    The string passed in to be converted into integer.

    \return
    Return the string as integer.
    ***************************************************************************/
    int stringToInt(std::string str)
    {
        const char* stringToInt = &str[1];
        int numChild = std::atoi(stringToInt);

        return numChild;
    }
}

```

functions.h

```
/*!*****************************************************************************
\file functions.h
\author Vadim Surov, Goh Wei Zhe
\par DP email: vsurov\@digipen.edu, weizhe.goh\@digipen.edu
\par Course: CS380
\par Section: A
\par Programming Assignment 1
\date 05-19-2021
\brief
This file has declarations and definitions that are required for submission
*******************************************************************************/

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

#include <iostream>
#include <sstream>
#include <string>
#include <list>
#include <vector>
#include <queue>
#include <stack>
#include <algorithm>

#include "data.h"

namespace AI
{
    #define UNUSED(expr) (void)expr;

    // A simple graph node definition with serialization functions

    //Helper function
    int stringToInt(std::string str);

    template<typename T>
    struct Node
    {
        // Member data

        T value;
        Node* parent;
        std::list<Node*> children;

        Node(T value = {}, Node* parent = nullptr,
                const std::list<Node*>& children = {})
            : value{ value }, parent{ parent }, children{ children }{}

        ~Node()
        {
            for (auto child : children)
                delete child;
        }

        /*!*********************************************************************
        \brief
        Serialization. An overloading insertion operator function that takes
        and return a stream object.

        \param os
        Output stream to perform output.

        \param rhs
        Right hand side object.

        \return
        Returns the output through ostream.
        ***********************************************************************/
        friend std::ostream& operator<<(std::ostream& os, Node const& rhs)
        {
            //Recursive function
            PrintOutput(os, &rhs);

            return os;
        }

        /*!*********************************************************************
        \brief
        Recursive Function to print output.

        \param os
        Output stream to perform output.

        \param rhs
        Right hand side object.

        \return
        Returns the output through ostream.
        ***********************************************************************/
        static void PrintOutput(std::ostream& os, const Node* rhs)
        {
            //std::cout << "os << rhs->value: " << rhs->value << std::endl;

            os << rhs->value + " {"+std::to_string(rhs->children.size()) + " ";

            //loop through each node in children's list
            for (Node* n : rhs->children)
            {
                PrintOutput(os, n);
            }

            os << "} ";
        }

        /*!*********************************************************************
        \brief
        Deserialization function to handle input streams and return an istream
        object.

        \param is
        Input stream to read inputs.

        \param rhs
        Right hand side object.

        \return
        Returns the input through istream.
        *************************************************************************/
        friend std::istream& operator>>(std::istream& is, Node& rhs)
        {
            is >> rhs.value;
            //std::cout << "is >> rhs.value: " << rhs.value << std::endl;

            //Recursive function
            ReadInput(is, &rhs);
            return is;
        }

        /*!************************************************************************
        \brief
        Recursive Function to read input.

        \param is
        Input stream to read inputs.

        \param rhs
        Right hand side object.

        \return
        None.
        *************************************************************************/
        static void ReadInput(std::istream& is, Node* rhs)
        {
            std::string s;

            while (is >> s)
            {
                //std::cout << "is >> str: " << s << std::endl;

                //If found  {
                if (s.find("{") != std::string::npos)
                {
                    //convert str[1] to integer and store as no. of child
                    int numChild = stringToInt(s);

                    //std::cout << "no. of child: " << numChild << std::endl;

                    //For each children, check if children has a child
                    for (int i = 0; i < numChild; ++i)
                    {
                        Node* child = new Node;

                        is >> s;

                        child->parent = rhs;
                        child->value = s;

                        // std::cout << "child value: " << child->value
                         //<< std::endl << std::endl;

                        rhs->children.push_back(child);

                        ReadInput(is, child);
                    }
                }
                else if (s.find("}") != std::string::npos)
                {
                    return;
                }
            }
        }


        /*!************************************************************************
        \brief
        Function to get path from tree root to current node

        \return
        Returns values from root to this node as an array.
        *************************************************************************/
        std::vector<T> getPath() const
        {
           std::vector<T> r;

           r.push_back(this->value);
           Node* node = this->parent;

           while (node)
           {
               r.push_back(node->value);
                node = node->parent;
           }

           std::reverse(r.begin(), r.end());

           return r;
        }
    };


    /*!****************************************************************************
    \brief
    Implementation of the Breadth-First Search algorithm

    \param node
    The node to search from.
```

```
    \param lookingfor
    The value of node we looking for.

    \return
    Returns the node found.
    ****************************************************************************/
    template<typename T>
    Node<T>* BFS(Node<T> & node, const T & lookingfor)
    {
        std::queue<Node<T>*> Q;
        Q.push(&node);

        while (!Q.empty())
        {
            Node<T>* current = Q.front();
            Q.pop();

            if (current->value == lookingfor)
                return current;

            //loop through each node in children's list
            for (Node<T>* n : current->children)
            {
                Q.push(n);
            }
        }

        return nullptr;
    }

    /*!***************************************************************************
    \brief
    Implementation of the Depth-First Search algorithm

    \param node
    The node to search for.

    \param lookingfor
    The value of the node we looking for.

    \return
    Returns the node found.
    ****************************************************************************/
    template<typename T>
    Node<T>* DFS(Node<T> & node, const T & lookingfor)
    {
        std::stack <Node<T>*> Stack;
        Stack.push(&node);

        while (!Stack.empty())
        {
           Node<T>* current = Stack.top();
           Stack.pop();

           if (current->value == lookingfor)
               return current;

           //loop through each node in children's list
           for (Node<T>* n : current->children)
               Stack.push(n);
        }

        return nullptr;
    }

} // end namespace

#endif
```

VPL

◄ Slides

Jump to...

Slides ►

You are logged in as Wei Zhe GOH (Log out)
cs380su21-meta.sg
Data retention summary
Get the mobile app