HBreithaupt / DigiPenCode Archived

Code Issues Pull requests Actions Projects Security Insights

master

DigiPenCode / CS280 / Assignment4 / BSTree.h

HBreithaupt CS280

0 contributors

Raw Blame

155 lines (115 sloc) 4.36 KB

```
/******************************************************************************/
/*!
 \file BSTree.h
 \author      Haven Breithaupt
 \par DP email: h.breithaupt\@digipen.edu
 \par Course: CS280
 \par Assignment 4
 \date 10/31/15

 \brief
  Prototypes of Binary Search Tree.

 */
/******************************************************************************/


//----------------------------------------------------------------------------
#ifndef BSTREE_H
#define BSTREE_H
//----------------------------------------------------------------------------
#ifdef _MSC_VER
#pragma warning( disable : 4290 ) // suppress warning: C++ Exception Specification ignored
#endif

#include <string>    // std::string
#include <stdexcept> // std::exception
#include <algorithm> // Max

```

```
#include "ObjectAllocator.h"

/// exception class
class BSTException : public std::exception
{
  public:
      /// constructo for exception class
    BSTException(int ErrCode, const std::string& Message) :
      error_code_(ErrCode), message_(Message) {
    };

      /// getter function to determine type of exception
    virtual int code(void) const {
      return error_code_;
    }

      /// getter function to read message in exception
    virtual const char *what(void) const throw() {
      return message_.c_str();
    }
    virtual ~BSTException() {}

      /// error type for exceptions
    enum BST_EXCEPTION{E_DUPLICATE, E_NO_MEMORY};

  private:
    int error_code_;        ///< type of exception
    std::string message_;   ///< message for the exception
};

  /// binary search tree class
template <typename T>
class BSTree
{
  public:

      /// node class used in the binary tree
    struct BinTreeNode
    {
      BinTreeNode *left;    ///< pointer to left child
      BinTreeNode *right;   ///< pointer to right child


      T data; ///< information in the ndoe

      int balance_factor; ///< optional(not implemeneted)
      unsigned count;    ///< number of nodes in subtree(not used)

      BinTreeNode(void) : left(0), right(0), data(0), balance_factor(0), count(0) {};

        /// constructor for the nodes
      BinTreeNode(const T& value) : left(0), right(0), data(value), balance_factor(0), count(0
    };

      /// simplification for ease of use
    typedef BinTreeNode* BinTree;

    BSTree(ObjectAllocator *OA = 0, bool ShareOA = false);
    BSTree(const BSTree& rhs);
    virtual ~BSTree();
    BSTree<T>& operator=(const BSTree& rhs);
    const BinTreeNode* operator[](int index) const;

      // change value back to T& when templating
    virtual void insert(const T& value);
    virtual void remove(const T& value);

    void clear(void);
      // change value back to T& when templating
    bool find(const T& value, unsigned &compares) const;

    bool empty(void) const;
    unsigned int size(void) const;
    int height(void) const;
    BinTree root(void) const;

    static bool ImplementedIndexing(void);

  protected:
      // change value back to T& when templating
    BinTree make_node(const T& value);
    void FreeNode(BinTree node);
    int tree_height(BinTree tree) const;
    void FindPredecessor(BinTree tree, BinTree &predecessor) const;

      //! the head of the tree
    BinTree Root;

      //! pointer to object allocator
    ObjectAllocator *allocator;

      //! height of the tree
    int Height;

      //! number of nodes in the tree
    unsigned int NumNodes;

      //! bool to indicate whether the object or client owns
      //! the object allocator for thi object
    bool OwnOA;

      //! bool to indicate wheter or not copies of this object
      //! wil be sharing the same allocator
    bool ShareAlloc;
```

```

      //! removes all nodes in the tree
    void ClearRec(BinTree tree);

    virtual void InsertItem(BinTree &tree, const T& value, int depth);

  private:
    // private stuff

    // helper function for use with the copy constructor
  void CopyHelper(BinTree &destination, const BinTree &source);

      // change Data to T7 when templating
  virtual void DeleteItem(BinTree& tree, const T& Data);

  bool FindItem(BinTree tree, const T& Data, unsigned &compares) const;

};

#include "BSTree.cpp"

#endif
//----------------------------------------------------------------------------
```