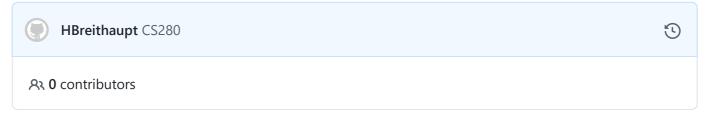
This repository has been archived by the owner. It is now read-only.

HBreithaupt / DigiPenCode Archived

Code Issues Pull requests Actions Projects Security Insights

° master → ···

DigiPenCode / CS280 / Assignment4 / BSTree.h



```
\square
 Raw
      Blame
155 lines (115 sloc) 4.36 KB
    1
 2
    /*!
 3
     \file BSTree.h
     \author Haven Breithaupt
 4
     \par DP email: h.breithaupt\@digipen.edu
 5
 6
     \par Course: CS280
     \par Assignment 4
 7
     \date 10/31/15
 8
 9
     \brief
 10
      Prototypes of Binary Search Tree.
 12
     */
 13
     15
 16
 17
    #ifndef BSTREE_H
18
 19
    #define BSTREE_H
    //-----
    #ifdef _MSC_VER
 21
    #pragma warning( disable : 4290 ) // suppress warning: C++ Exception Specification ignored
    #endif
 23
 24
    #include <string> // std::string
 25
    #include <stdexcept> // std::exception
 27
    #include <algorithm> // Max
 28
```

```
29
     #include "ObjectAllocator.h"
30
31
     /// exception class
     class BSTException : public std::exception
     {
34
       public:
           /// constructo for exception class
         BSTException(int ErrCode, const std::string& Message) :
           error_code_(ErrCode), message_(Message) {
38
         };
40
           /// getter function to determine type of exception
41
         virtual int code(void) const {
           return error_code_;
42
44
45
           /// getter function to read message in exception
         virtual const char *what(void) const throw() {
47
           return message_.c_str();
         }
         virtual ~BSTException() {}
           /// error type for exceptions
         enum BST_EXCEPTION{E_DUPLICATE, E_NO_MEMORY};
53
       private:
         int error_code_;
                                 ///< type of exception
         std::string message_;
                                 ///< message for the exception
57
     };
59
       /// binary search tree class
60
     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
72
           T data; ///< information in the ndoe
73
           int balance_factor; ///< optional(not implemeneted)</pre>
75
           unsigned count;
                             ///< number of nodes in subtree(not used)</pre>
           BinTreeNode(void) : left(0), right(0), data(0), balance_factor(0), count(0) {};
78
             /// constructor for the nodes
           BinTreeNode(const T& value) : left(0), right(0), data(value), balance_factor(0), count(0)
```

```
81
          };
 82
 83
            /// simplification for ease of use
          typedef BinTreeNode* BinTree;
 84
 85
 86
          BSTree(ObjectAllocator *OA = 0, bool ShareOA = false);
          BSTree(const BSTree& rhs);
 87
 22
          virtual ~BSTree();
 89
          BSTree<T>& operator=(const BSTree& rhs);
          const BinTreeNode* operator[](int index) const;
91
            // change value back to T& when templating
93
          virtual void insert(const T& value);
          virtual void remove(const T& value);
96
          void clear(void);
            // change value back to T& when templating
          bool find(const T& value, unsigned &compares) const;
100
          bool empty(void) const;
          unsigned int size(void) const;
102
          int height(void) const;
103
          BinTree root(void) const;
104
105
          static bool ImplementedIndexing(void);
106
        protected:
            // change value back to T& when templating
          BinTree make_node(const T& value);
110
          void FreeNode(BinTree node);
          int tree_height(BinTree tree) const;
111
112
          void FindPredecessor(BinTree tree, BinTree &predecessor) const;
113
114
            //! the head of the tree
115
          BinTree Root;
116
117
            //! pointer to object allocator
118
          ObjectAllocator *allocator;
119
120
            //! height of the tree
121
          int Height;
122
            //! number of nodes in the tree
124
          unsigned int NumNodes;
125
            //! bool to indicate whether the object or client owns
127
            //! the object allocator for thi object
128
          bool OwnOA;
            //! bool to indicate wheter or not copies of this object
130
            //! wil be sharing the same allocator
131
132
          bool ShareAlloc;
```

```
133
134
            //! removes all nodes in the tree
          void ClearRec(BinTree tree);
135
136
137
          virtual void InsertItem(BinTree &tree, const T& value, int depth);
138
139
        private:
          // private stuff
140
141
142
          // helper function for use with the copy constructor
        void CopyHelper(BinTree &destination, const BinTree &source);
143
144
145
            // change Data to T7 when templating
        virtual void DeleteItem(BinTree& tree, const T& Data);
146
147
148
        bool FindItem(BinTree tree, const T& Data, unsigned &compares) const;
149
150
      };
151
152
      #include "BSTree.cpp"
153
154
      #endif
      //----
155
```