FProjectLBandDP

0.3.0

Generated by Doxygen 1.8.17

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

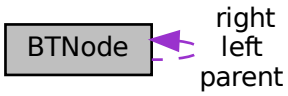
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

- BTNode (Products dataVal)
- char nodeName ()
- Products nodeData ()

Public Attributes

- BTNode * left
- BTNode * right
- BTNode * parent

3.1.1 Detailed Description

Definition at line 48 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BTNode()

```
BTNode::BTNode (
            Products dataVal )  [inline]
```

BTNode constructor

Parameters

dataVal	This is the product that is put into the binary tree.

Definition at line 59 of file main.cpp.

```
59                              {
60        //cout « "name = " « name « endl;
61        left = NULL;
62        right = NULL;
63        parent = NULL;
64        objName = name++;
65        data = dataVal;
66    }
```

3.1.3 Member Function Documentation

3.1.3.1 nodeData()

```
Products BTNode::nodeData ( )  [inline]
```

This reports the node's data

Definition at line 78 of file main.cpp.

```
78                              {
79          return(data);
80      }
```

3.1.3.2 nodeName()

```
char BTNode::nodeName ( )  [inline]
```

This reports the node's name

Definition at line 71 of file main.cpp.

```
71                          {
72          return(objName);
73      }
```

3.1.4 Member Data Documentation

3.1.4.1 left

```
BTNode* BTNode::left
```

Definition at line 50 of file main.cpp.

3.1.4.2 parent

```
BTNode* BTNode::parent
```

Definition at line 52 of file main.cpp.

3.1.4.3 right

```
BTNode* BTNode::right
```

Definition at line 51 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp

3.2 Products Class Reference

Public Member Functions

- Products ()
- Products (double p, double w)

Public Attributes

- double price
- double weight
- double ratio

3.2.1 Detailed Description

This is class has 2 different parameters used to make this object

Definition at line 22 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Products() [1/2]

```
Products::Products ( )  [inline]
```

Definition at line 31 of file main.cpp.

```
31                {
32
33     }
```

3.2.2.2 Products() [2/2]

```
Products::Products (
              double p,
              double w )  [inline]
```

This is the constructor for this class

Parameters

p	The price for the product.
w	The weight for the product.

Definition at line 41 of file main.cpp.

```
41                                  {
42          price = p;
43          weight = w;
44          ratio = w/p;
45     }
```

3.2.3 Member Data Documentation

3.2.3.1 price

```
double Products::price
```

Definition at line 27 of file main.cpp.

3.2.3.2 ratio

```
double Products::ratio
```

Definition at line 29 of file main.cpp.

3.2.3.3 weight

```
double Products::weight
```

Definition at line 28 of file main.cpp.

The documentation for this class was generated from the following file:

- /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp

Chapter 4

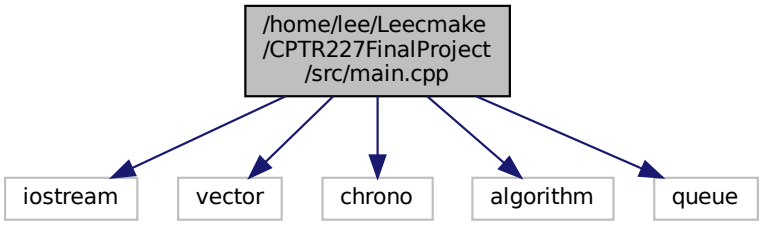
File Documentation

4.1 /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp File Reference

This is the final project made with code from HW11.

```
#include <iostream>
#include <vector>
#include <chrono>
#include <algorithm>
#include <queue>
```

Include dependency graph for main.cpp:



Classes

- class Products
- class BTNode

Functions

- BTNode * addNode (BTNode *rootNode, BTNode *n)
- BTNode * addNode (BTNode *rootNode, Products dataval)
- int randomGen (int min, int max)
- std::vector< Products > genProducts (int n)
- void printTree (BTNode *rootNode)
- void printBT (const string &prefix, BTNode *node, bool isLeft)
- void printBT (BTNode *node)
- void createTree (vector< Products > &tree, int index)
- bool comparator (const Products &a, const Products &b)
- int main (int, char **)

4.1.1 Detailed Description

This is the final project made with code from HW11.

This program is based on the knapsack problem and uses a binary tree to store the data.

Author

Daniel Pervis and Lee Beckermeyer

Date

4/21/2021

4.1.2 Function Documentation

4.1.2.1 addNode() [1/2]

```
BTNode* addNode (
            BTNode * rootNode,
            BTNode * n )
```

This function adds a node to a binary search tree.

Parameters

rootNode	is the pointer to the tree's root node
n	is the node to add

Returns

pointer to rootNode if successful, NULL otherwise

Definition at line 98 of file main.cpp.

```
98                                              {
99      BTNode* prev = NULL;
100      BTNode* w = rootNode;
101      if(rootNode == NULL) { // starting an empty tree
102          rootNode = n;
103      } else {
104          // Find the node n belongs under, prev, n's new parent
105          while(w != NULL) {
106              prev = w;
107              if(n->nodeData().ratio < w->nodeData().ratio){
108                  w = w->left;
109              } else if(n->nodeData().ratio > w->nodeData().ratio) {
110                  w = w->right;
111              } else { // data already in the tree
112                  return(NULL);
113              }
114          }
115          // now prev should contain the node that should be n's parent
116          // Add n to prev
117          if(n->nodeData().ratio < prev->nodeData().ratio) {
118              prev->left = n;
119          } else {
120              prev->right = n;
121          }
122      }
123      return(rootNode);
124  }
```

4.1.2.2 addNode() [2/2]

```
BTNode* addNode (
              BTNode * rootNode,
              Products dataval )
```

Adds a new node with the passed data value

Parameters

rootNode	pointer to root node
dataval	an integer for the new node's data

Returns

pointer to root node or NULL if not successful

Definition at line 134 of file main.cpp.

```
134                                                     {
135      BTNode* newNode = new BTNode(dataval);
136      if(addNode(rootNode, newNode) == NULL) {
137          cout « dataval.ratio « " already in tree" « endl;
138      } else {
139          cout « dataval.ratio « " succesfully added" « endl;
140      }
141      return(rootNode);
142  }
```

4.1.2.3 comparator()

```
bool comparator (
              const Products & a,
              const Products & b )
```

compares 2 products, currently not used.

Parameters

a	product a
b	product b

Definition at line 260 of file main.cpp.

```
260                                                 {
261     return a.ratio < b.ratio;
262 }
```

4.1.2.4 createTree()

```
void createTree (
            vector< Products > & tree,
            int index )
```

creates a binary tree

Parameters

tree	unknown
index	unknown

Definition at line 246 of file main.cpp.

```
246                                                 {
247     BTNode* root = new BTNode(tree[index]);
248     for (int i = 0; i < tree.size(); i++) {
249         addNode(root, tree[i]);
250     }
251     printBT(root);
252 }
```

4.1.2.5 genProducts()

```
std::vector<Products> genProducts (
            int n )
```

generates the products.

Parameters

n	The amount of products you want generated.

Definition at line 162 of file main.cpp.

```
162                                         {
163     vector<Products> output;
164     for (int i = 0; i < n; i++) {
165         output.push_back(Products(randomGen(1,1000), randomGen(5, 200)));
166     }
167     return output;
168 }
```

4.1.2.6 main()

```
int main (
              int ,
              char **  )
```

Definition at line 264 of file main.cpp.

```
264                       {
265     srand(time(NULL));
266     vector<Products> products = genProducts(50);
267     auto max = std::max_element(products.begin(), products.end(), [](const Products& a, const Products&
        b){
268         return a.ratio < b.ratio;
269     });
270     int index = distance(products.begin(), max);
271     cout « max->ratio « endl;
272     //sort(products.begin(), products.end(), &comparator);
273     for (int i = 0; i < products.size(); i++) {
274         cout « i « " : " « products[i].ratio « endl;
275     }
276     createTree(products, index);
277 }
```

4.1.2.7 printBT() [1/2]

```
void printBT (
              BTNode * node )
```

An overload to simplify calling printBT

Parameters

node	is the root node of the tree to be printed

Definition at line 235 of file main.cpp.

```
236 {
237     printBT("", node, false);
238 }
```

4.1.2.8 printBT() [2/2]

```
void printBT (
              const string & prefix,
              BTNode * node,
              bool isLeft )
```

Print a binary tree

This example is modified from: https://stackoverflow.com/a/51730733

Parameters

prefix	is a string of characters to start the line with
node	is the current node being printed
isLeft	bool true if the node is a left node

Definition at line 211 of file main.cpp.

```
212 {
213     if( node != NULL )
214     {
215         cout « prefix;
216 
217         cout « (isLeft ? "|--" : "--" );
218 
219         // print the value of the node
220         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
221         cout « node->nodeData().ratio « std::endl;
222 
223         // enter the next tree level - left and right branch
224         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
225         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
226     }
227 }
```

4.1.2.9 printTree()

```
void printTree (
            BTNode * rootNode )
```

prints a binary tree

Parameters

rootNode	The binary tree you want printed.

Definition at line 175 of file main.cpp.

```
175                                 {
176     queue<BTNode*> todo; // the queue of nodes left to visit
177     BTNode* cur; // current node
178     BTNode* prev; // The previous node
179 
180     todo.push(rootNode);
181 
182     while(!todo.empty()) {
183         cur = todo.front();
184         // Print current node
185         cout « cur->nodeName() « ':' « cur->nodeData().ratio « '\t';
186         // add cur->left to queue
187         if(cur->left != NULL) {
188             todo.push(cur->left);
189         }
190         // add cur->right to queue
191         if(cur->right != NULL) {
192             todo.push(cur->right);
193         }
194         // remove cur from queue
195         todo.pop();
196     }
197     cout « endl;
198 }
```

4.1.2.10 randomGen()

```
int randomGen (
            int min,
            int max )
```

Randomly generates a "double"(float in C++) number

Parameters

min	The minimum number that can be generated.
max	The maximum number that can be generated.

Definition at line 150 of file main.cpp.

```
150                                     {
151
152     double random = rand() % max + min;
153     //cout « rand() % max « endl;
154     return random;
155 }
```

Index