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

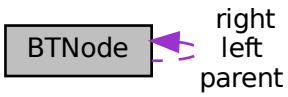
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

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

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     }
```

3.1.3.3 nodeRatio()

```
int BTNode::nodeRatio ( )  [inline]
```

This reports the node's ratio

Definition at line 85 of file main.cpp.

```
85                      {
86          return(data.ratio);
87      }
```

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

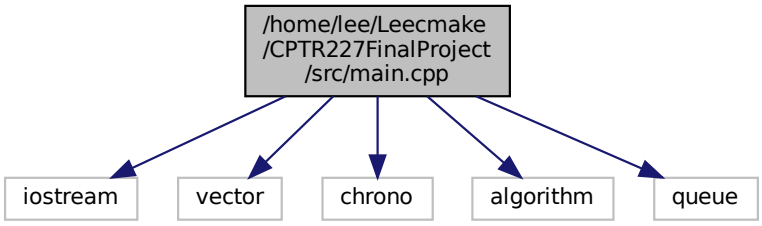
File Documentation

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 * addNodeTree (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 LeeStorage (vector< Products > &tree, int index, int weight, int n, BTNode *root)
- 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 105 of file main.cpp.

```
105                                                 {
106     BTNode* prev = NULL;
107     BTNode* w = rootNode;
108     if(rootNode == NULL) { // starting an empty tree
109         rootNode = n;
110     } else {
111         // Find the node n belongs under, prev, n's new parent
112         while(w != NULL) {
113             prev = w;
114             if(n->nodeData().ratio < w->nodeData().ratio){
115                 w = w->left;
116             } else if(n->nodeData().ratio > w->nodeData().ratio) {
117                 w = w->right;
118             } else { // data already in the tree
119                 return(NULL);
120             }
121         }
122         // now prev should contain the node that should be n's parent
123         // Add n to prev
124         if(n->nodeData().ratio < prev->nodeData().ratio) {
125             prev->left = n;
126         } else {
127             prev->right = n;
128         }
129     }
130     return(rootNode);
131 }
```

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 176 of file main.cpp.

```
176                                                 {
177     BTNode* newNode = new BTNode(dataval);
178     if(addNode(rootNode, newNode) == NULL) {
179         //cout « dataval « " already in tree" « endl;
180     } else {
181         //cout « dataval « " succesfully added" « endl;
182     }
183     return(rootNode);
184 }
```

4.1.2.3 addNodeTree()

```
BTNode* addNodeTree (
            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 141 of file main.cpp.

```
141                                                 {
142     BTNode* prev = NULL;
143     BTNode* w = rootNode;
144     if(rootNode == NULL) { // starting an empty tree
145         rootNode = n;
146     } else {
147         // Find the node n belongs under, prev, n's new parent
148         while(w != NULL) {
149             prev = w;
150             if(n->nodeRatio() < w->nodeRatio()){
151                 w = w->left;
152             } else if(n->nodeRatio() > w->nodeRatio()) {
153                 w = w->right;
154             } else { // data already in the tree
155                 return(NULL);
156             }
157         }
158         // now prev should contain the node that should be n's parent
159         // Add n to prev
160         if(n->nodeRatio() < prev->nodeRatio()) {
161             prev->left = n;
162         } else {
163             prev->right = n;
164         }
165     }
166     return(rootNode);
167 }
```

4.1.2.4 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 327 of file main.cpp.

```
327                                                 {
328     return a.ratio < b.ratio;
329 }
```

4.1.2.5 createTree()

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

creates a binary tree, also checks if the knapsack is full, if the knapsack isn't full it continues

Parameters

tree	a vector of products you want to turn into a tree.
index	the size of the vector, needed with the current implementation.

Definition at line 309 of file main.cpp.

```
309                                                 {
310     BTNode* root = new BTNode(tree[index]);
311     int weight = 0;
312     int n = 0;
313     for (Products x : tree){
314         addNode(root, x);
315         //LeeStorage(tree, index, weight, n ,root);//experiment, not any kind of official implementation
316     };
317     cout « "Weight of Knapsack: " « weight « endl;
318     printBT(root);
319 }
```

4.1.2.6 genProducts()

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

generates the products.

Parameters

n	The amount of products you want generated.

Definition at line 205 of file main.cpp.

```
205                                         {
206     vector<Products> output;
207     for (int i = 0; i < n; i++) {
208         output.push_back(Products(randomGen(1,1000), randomGen(5, 200)));
209     }
210     return output;
211 }
```

4.1.2.7 LeeStorage()

```
void LeeStorage (
            vector< Products > & tree,
            int index,
            int weight,
            int n,
            BTNode * root )
```

Lee's Crazy storage for random stuff, currently full of an experiment for the knapsack

Definition at line 286 of file main.cpp.

```
286                                                                                 {
287     for (Products x : tree){
288         //addNode(root, x);
289             n++;
290             int newweight = x.weight + weight;
291             if(newweight>=500 or n == index){
292                 continue;
293             }
294             else{
295                 weight = newweight;
296                 addNode(root, x);
297             }
298 
299 
300         };
301 };
```

4.1.2.8 main()

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

Definition at line 331 of file main.cpp.

```
331                       {
332     srand(time(NULL));
333     vector<Products> products = genProducts(50);
334     auto max = std::max_element(products.begin(), products.end(), [](const Products& a, const Products&
        b){
335         return a.ratio < b.ratio;
336     });
337     int index = distance(products.begin(), max);
338     cout « max->ratio « endl;
339     //sort(products.begin(), products.end(), &comparator);
340     for (int i = 1; i < products.size(); i++) {
341         cout « i « " : " « products[i].ratio « endl;
342     }
343     createTree(products, index);
344 }
```

4.1.2.9 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 278 of file main.cpp.

```
279 {
280     printBT("", node, false);
281 }
```

4.1.2.10 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 254 of file main.cpp.

```
255 {
256     if( node != NULL )
257     {
258         cout « prefix;
259 
260         cout « (isLeft ? "L--" : "R--" );
261 
262         // print the value of the node
263         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
264         cout « node->nodeData().ratio « std::endl;
265 
266         // enter the next tree level - left and right branch
267         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
268         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
269     }
270 }
```

4.1.2.11 printTree()

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

prints a binary tree

Parameters

rootNode	The binary tree you want printed.

Definition at line 218 of file main.cpp.

```
218                                 {
219     queue<BTNode*> todo; // the queue of nodes left to visit
220     BTNode* cur; // current node
221     BTNode* prev; // The previous node
222 
223     todo.push(rootNode);
224 
225     while(!todo.empty()) {
226         cur = todo.front();
227         // Print current node
228         cout « cur->nodeName() « ':' « cur->nodeData().ratio « '\t';
229         // add cur->left to queue
230         if(cur->left != NULL) {
231             todo.push(cur->left);
232         }
233         // add cur->right to queue
234         if(cur->right != NULL) {
235             todo.push(cur->right);
236         }
237         // remove cur from queue
238         todo.pop();
239     }
240     cout « endl;
241 }
```

4.1.2.12 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 193 of file main.cpp.

```
193                                 {
194 
195     double random = rand() % max + min;
196     //cout « rand() % max « endl;
197     return random;
198 }
```

Index