CS302

Assignment 3: The Crane

Description

The mighty crane. It can be used to lift heavy objects. This crane, however, does not want to lift each object individually but rather a set of objects connected in a structure that looks similar to a binary tree. Each node of this binary tree is either a joint (which connects to two nodes, which would be an interior node) or an object (a bike, car, or goped, which would be a leaf node). This structure needs to be perfectly balanced, thus the torque on each arm of this joint needs to be equal. Remember, torque is the weight multiplied by the length of the arm. Below is the class structure you will use (implemented for you)

```
class binTreeNode
{
public:
  virtual double getWeight() { return 0.0; }
  void setLeft(binTreeNode * r) { left = r; }
  void setRight(binTreeNode * r) { right = r; }
  binTreeNode* getLeft() { return left; }
  binTreeNode* getRight() { return right; }
  void setArmLength(int arm) { armLength = arm; }
  int getArmLength()  { return armLength; }

private:
  binTreeNode * left, * right;
  int armLength;
};

class goped : public binTreeNode
{
public:
  goped(double w, double e, double t) :
  wheelWeight(w), engineWeight(e), tankWeight(t)
  { }

  double getWeight()
  {
    return 2 * wheelWeight + engineWeight + tankWeight;
  }
```

```
private:
  double wheelWeight;
  double engineWeight;
  double tankWeight;
};

class bike : public binTreeNode
{
public:
  bike(double w, double f) :
  wheelWeight(w), frameWeight(f) { }

  double getWeight() { return 2 * wheelWeight + frameWeight; }
private:
  double wheelWeight;
  double frameWeight;
};

class car : public binTreeNode
{
public:
  car(double w, double e, double t, double b) :
  wheelWeight(w), engineWeight(e), transmissionWeight(t),
  bodyWeight(b) { }

  double getWeight()
  {
    return 4 * wheelWeight + engineWeight
    + transmissionWeight + bodyWeight;
  }
private:
  double wheelWeight;
  double engineWeight;
  double transmissionWeight;
  double bodyWeight;
};
```

Each object returns its weight using the `getWeight()` functions. The base class contains `virtual double getWeight()` so this class is polymorphic, which means that each leaf of this tree can be of a different type, but since the `getWeight()` function is `virtual`, the correct function is called. For example if you have

```
binTreeNode * ptr;

ptr = new bike(2,0, 2.0);

ptr->getWeight(); //calls bike's getWeight()

ptr = new goped(2.5, 5.6, 3.3);

ptr->getWeight(); //calls goped's getWeight()
```

So you will only need to use a `binTreeNode*` to traverse the binary tree and get the weight of each node, dynamic casting will not be needed.

Input

You will read the data from a file using linux redirection. Each line of the input will contain either

```
Joint LeftArmLength RightArmLength
```

or will contain

```
ObjectName Field1 Field2 ...
```

Each line represents a node in the binary tree, a joint would be an interior node and an object (goped, bike, or car) is a leaf node. The input is given in preorder fashion. You need to construct a binary tree from the input. The supplemental video might be helpful.

Output

For the binary tree read in from the file, if every interior node (joint) has equal torque on both sides, then your program outputs `"Crane can lift the items"` otherwise your program outputs `"Well this is awkward"`

Contents of Main

In your main file that you will submit, it is recommended to write the following functions

- `binTreeNode* buildTree()`
  - A recursive function that builds the tree in preorder fashion using the input that is given in preorder fashion
  - Each function call reads one line of the input, if the string read on the line contains `"Joint"` then you allocate a `binTreeNode()` object to a `binTreeNode * r` pointer, call `buildTree()` twice (for each side of the node) and set the result of the recursive function call into r's left and right side (then set the left and right child nodes' arm length) and then return r
  - If the string on an input line does not contain `"Joint"` then determine the object, read in the appropriate content (since bike, goped, and car object have different amount of fields), allocate the correct object to the pointer, and then return this pointer
- `double getObjectsWeight(binTreeNode * r)`
  - Recursive function that obtains the combined weight of objects on each side of a joint
  - If r is not a joint node, then return its weight, otherwise you need to recursively obtain the entire weight of a joints left side and right side
  - Once the weights of r's left and right side are obtained, multiply r's left arm length with the left side weight (same for the right side), compare the results, if they are the same return the weight of the left and right side up the tree, otherwise return some value that tells the parent node that something went wrong...
- `void deallocateTree(binTreeNode * r)`
  - Deallocates each node of the binary tree in postorder fashion

Using the functions you implemented, your int main would only contain this

```
int main()
{
  binTreeNode * root = buildTree();

  if (getObjectsWeight(root) != -1.0)
```

```
    cout << "Crane can lift the items\n";
  else
    cout << "Well this is awkward\n";

  deallocateObjects(root);

  return 0;
}
```

Specifications

- Make sure your code is memory leak free
- Document your functions (only the main file)
- Traverse and process the tree using recursive functions, you can modify main any way you want and have different functions (I only provided those functions to help you if in case you don't know where to start)

Sample Run

```
% g++ main.cpp
% ./a.out < crane01.txt
Crane can lift the items

% ./a.out < crane02.txt
Crane can lift the items

% ./a.out < crane03.txt
Well this is awkward

% ./a.out < crane04.txt
Crane can lift the items

% ./a.out < crane05.txt
Well this is awkward

% ./a.out < crane06.txt
Crane can lift the items

% ./a.out < crane07.txt
Well this is awkward
```

Submission

Submit your source code main.cpp to codegrade by the deadline

References

- Link to image(s) can be found at https://pngimg.com/image/20211
- Supplemental Video Link https://youtu.be/j4uprnz2a7s