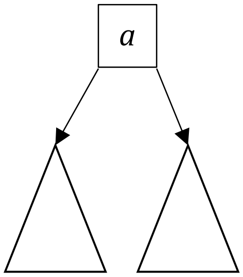
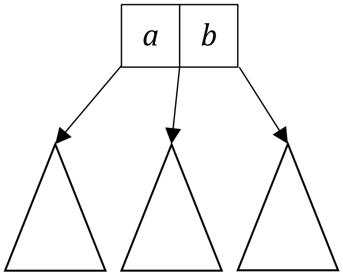
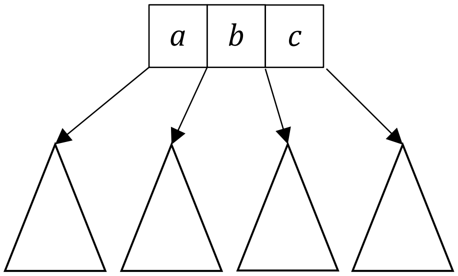
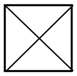
CS 352 HW 3: Polymorphism

Due Friday, February 28, 10pm

The questions on this assignment are all based around the *2–4 tree* data structure. This isn't a data-structures class, so you won't be required to implement any complex algorithms on 2–4 trees, but collection types are a good way to explore the polymorphism features of a language. You can get a good overview of the data structure on Wikipedia[1], but here is a summary of the relevant aspects:

2–4 trees are a balanced tree data structure with variable-sized internal nodes and (non-data-carrying) leaf nodes which are all at the same level. The details of the insertion and balancing algorithms are beyond the scope of this assignment, but what is relevant is that there are four kinds of 2–4 tree nodes:

 *2-node* 1 data element, 2 child trees (less-than and greater-than subtrees relative to the data element)	 *3-node* 2 data elements, 3 child trees (less-than, between, and greater-than subtrees relative to the data elements)
 *4-node* 3 data elements, 4 child trees (less-than, between low and middle, between middle and high, and greater-than relative to the data elements)	 *Nil* Placeholder for non-data-carrying leaf nodes.

[1] https://en.wikipedia.org/wiki/2%E2%80%933%E2%80%934_tree

Question 1:

[12] Implement the data type for a 2–4 tree in Haskell, subject to the following design constraints:

- The type should be named TwoFourTree.
- You should use a data declaration for algebraic types which includes all four node types as variants (the names of the variants are up to you).
- TwoFourTree should be polymorphic over its element type (don't worry about constraints on the element type).
- TwoFourTree should be deriving Show for easier interactive debugging.

The code for TwoFourTree should be placed in twofour.hs in your submission; start with the provided starter file.

[2] Also include a sample TwoFourTree variable in twofour.hs. The variable should be named sampleTree and include at least three non-nil tree nodes with a total of at least six data elements between them. Your sample tree will not be graded on whether it obeys the balancing invariants of the 2–4 tree data structure.

Question 2:

Include the following Haskell code in sized.hs:

[4] Define a Haskell typeclass named Sized t. It should have one function, size, which has type t -> Int.

[4] Implement Sized for [a], where the size of a list is its length. You may use the standard library or define your own list length function.

[8] Implement Sized for TwoFourTree a; the size of a 2–4 tree should be the number of data elements in the tree (in general, not equal to the number of tree nodes).

Question 3:

Implement a 2–4 tree in C++ in twofour.hpp. For those of you who already know some C++, we are going to ignore the memory-management aspects of C++ polymorphism until next unit, so this design will have a number of memory leaks. Fixing these leaks will be a problem on the next assignment, but you are not required to do so for this assignment.

[4] Implement an abstract base class called TwoFourTree.

- TwoFourTree should be generic over its element type; use a template parameter.
- TwoFourTree should also have an abstract virtual method called size, which takes no arguments, returns an int, and is const in its implicit this parameter.

[18] Make a subclass of TwoFourTree for each of the three node types, TwoNode, ThreeNode, and FourNode. Nil will be represented with a null pointer; the C++ null pointer constant is nullptr, but otherwise acts like a C null pointer.

- Each class should be templated over its element type T and inherit from TwoFourTree<T>.
- Each class should have private fields of type T for as many data elements as it contains, as well as private fields of type TwoFourTree<T>* for each subtree. You can name the fields whatever you want.
- Each class should have two public constructors:
  - one that takes each of its data elements by const reference and copies them, setting the subtree pointers to nullptr,
  - and another with const T & values for the data elements and TwoFourTree<T>* values for the subtree pointers.
- Each class should have a public implementation of the superclass size method which counts the number of data elements in the tree rooted at that node (again, in general not equal to the number of tree nodes). Be sure not to call size on null subtrees.

Style:

[6] Your Haskell and C++ code should not be significantly more complex than necessary, and should be structured in a way that facilitates understanding. Make use of whitespace and comments as appropriate to make your code clearly readable. Function and variable names should clearly represent their purpose or be sufficiently limited in scope to be clear from context.

Testing:

You can test your Haskell code by compiling and running twofour_test.hs:

```
ghc -make -o twofour_hs twofour_test.hs
./twofour_hs
```

The output should be your sample tree, a newline, 2, a space, and the number of data elements in your sample tree.

You can test your C++ code by compiling and running twofour_test.cpp:

```
g++ -std=c++14 -o twofour_cpp twofour_test.cpp
./twofour_cpp
```

The output should be this:

```
2 8
```

Question 4:

[6] For two different aspects of your code from questions 1-3, indicate the portion of code, state which of the three kinds of polymorphism (ad-hoc, parametric, or subtype) it uses, and describe how it is that kind of polymorphism.

Submission Instructions

Your code and testing files for Questions 1, 2 & 3 should be put into a zip file; include all code that was provided in the cs352_hw3.zip starter file.

Your answer to Question 3 should be put in a PDF file. Consult the instructor if you have trouble generating a PDF from your word processing software.

Both the zip and PDF files should be submitted on Moodle by the assignment due date.