The output from driver:

```
The item in the box: Basketball
The item in the box: Volleyball

Let's nest a single box!  A PlainBox in a BoxContainer!
The original PlainBox holds Basketball and the PlainBox in a BoxContainer (1
level of nesting) holds a Basketball (should both be "Basketball")

Let's try setting the item of the PlainBox in the BoxContainer!
Calling setItem("Football") on the nested boxes!
Now the nested box holds a Football (should be "Football")

Let's try to nest our level 1 nested box into a couple of other BoxContainers!
Pass our box into the constructor of the BoxContainer to nest.
Let's try to display it's item!  It should still be "Football"
The PlainBox nested into three BoxContainers has a Football inside (should be
"Football")
Does setItem() still work?  Let's change it to "Tennis Ball"!
The triple nested box holds a Tennis Ball (should be "Tennis Ball")

Final test!!!  Ten levels of nesting with a MagicBox!  The item should not
change when set.
Initial item is Volleyball
The box still holds Volleyball (should be "Volleyball") and calling
setItem("This shouldn't work!") on the box gives us Volleyball (should still be
"Volleyball")
```

A stack could be used instead of a recursive function in order to get the innermost box's item. We could keep opening boxes until it was no longer possible, pushing each box onto a stack. Once all boxes have been opened, the top of the stack would contain the innermost box. We can then simply call its getItem() or setItem() command as we need. I don't know how we would deal with when to stop opening boxes, but this is just pseudocode.

```
template<template<typename> class BoxType, typename ItemType>
void display(BoxContainer<BoxType<ItemType>> box)
{
	// empty stack for appending boxes to
	aStack = an empty stack

	// current box we are appending to stack
	aStack.push(box)

	// loop until there are no more boxes to open, i.e., a Magic or Plain
	// box has been reached
	while (there are boxes to open)
	{
		// get the next box
		box = box.getBox()

		// add box to stack
		aStack.add(box)
	}

	// innermost box on top of stack, get it, and get its item to display
	std::cout << aStack.peek().getItem()
}
```