Lab 11: Polymorphism

1. Operators on Base Classes

From lab 10 question 3, lets expand our main function to also include the following lines at the end (circle with radius 2 and triangle with base 3 height 4):

```
  std::cout << "c1 < t1? " << (c1 < t1) << std::endl;
  std::cout << "t1 < c1? " << (t1 < c1) << std::endl;
```

The shapes should be compared on their `area()`, with the smaller area shape being less than the other one. This should output

```
c1 < t1? 0
t1 < c1? 1
```

You will need to overload `operator<` for the shapes. Try doing it with `templates` and directly comparing two `Shape2D` objects. Do these two approaches work? Why?

Reminder that c1 and t1 are the Circle and Triangle from lab 10, deriving publicly from

```
#include <iostream>
#include <cmath> // can find math constants here, like for pi
#include <string>
#include <iomanip>

class Shape2D{
public:
    Shape2D(){}
    double area() const {return 0;}

    const std::string& getType() const {return type;}
    void setType(const std::string& type){this->type = type;}

private:
    std::string type;
};
```

2. Virtual Operators

Try the above, but modify `Shape2D`, `Circle` , and `Triangle` using polymorphism and virtual functions to make it work. (Since you are now more competent, I won't provide any hints for this exercise).

3. Pure Virtual Method

Modify `Shape2D` so that `area()` is abstract instead of outputting the current nonsensical 0.

4. Containers of Abstract Classes

Use the previous modified classes to solve this problem stated in words. Suppose you have some Circles and Triangles. You need to print them out in increasing order of their area. Your task is to sort the objects, then print them out in sorted order.

You are given:

```
Circle c1 {1.652};
Circle c2 {1.9123};
Triangle t1 {3.72, 5.432};
Triangle t2 {4.12, 4.552};
Triangle t3 {2.623, 6.52};
```

Hint: you may need to supply a comparator function to std::sort

5. Cost of Virtual Functions

You may wonder why unlike most other languages, class methods are not virtual by default.

The `sizeof(type or expression)` operator allows you to evaluate the size in bytes of a type or object. What is the size of `Circle`? What is the size of its members (`Circle` is a `Shape2D` so you have to include `Shape2D` members)? What could be accounting for its extra size? What is its size when `area` is not virtual?