CS 237	Discrete Mathematics I

Coolness

Predicates and Quantifiers — *Exploration*

Goal

The purpose of this exploration is for you to *creatively* explore the cool meaning of *binary* predicates and *nested* quantifiers.

Requirements

Using the supplied sample code as a guide and a starting point, create at least one binary predicate and implement a template-version of these four nested quantifier functions, shown side-by-side with their symbolic logic equivalents:

1. `forAllForAll` $\forall x \forall y\ P(x, y)$
2. `forAllForSome` $\forall x \exists y\ P(x, y)$
3. `forSomeForAll` $\exists x \forall y\ P(x, y)$
4. `forSomeForSome` $\exists x \exists y\ P(x, y)$

As Rosen describes on pages 51-52 (THINKING OF QUANTIFICATION AS LOOPS):

> In working with quantifications of more than one variable, it is sometimes helpful to think in terms of nested loops. (Of course, if there are infinitely many elements in the domain of some variable, we cannot actually loop through all values. Nevertheless, this way of thinking is helpful in understanding nested quantifiers.) For example, to see whether $\forall x \forall y\ P(x, y)$ is true, we loop through the values for x, and for each x we loop through the values for y. If we find that $P(x, y)$ is true for all values for x and y, we have determined that $\forall x \forall y\ P(x, y)$ is true. If we ever hit a value x for which we hit a value y for which $P(x, y)$ is false, we have shown that $\forall x \forall y\ P(x, y)$ is false.
>
> Similarly, to determine whether $\forall x \exists y\ P(x, y)$ is true, we loop through the values for x. For each x we loop through the values for y until we find a y for which $P(x, y)$ is true. If for every x we hit such a y, then $\forall x \exists y\ P(x, y)$ is true; if for some x we never hit such a y, then $\forall x \exists y\ P(x, y)$ is false.
>
> To see whether $\exists x \forall y\ P(x, y)$ is true, we loop through the values for x until we find an x for which $P(x, y)$ is always true when we loop through all values for y. Once we find such an x, we know that $\exists x \forall y\ P(x, y)$ is true. If we never hit such an x, then we know that $\exists x \forall y\ P(x, y)$ is false.
>
> Finally, to see whether $\exists x \exists y\ P(x, y)$ is true, we loop through the values for x, where for each x we loop through the values for y until we hit an x for which we hit a y for which $P(x, y)$ is true. The statement $\exists x \exists y\ P(x, y)$ is false only if we never hit an x for which we hit a y such that $P(x, y)$ is true.

Test your implementation of these four functions using a suitable binary (two-argument, or 2-ary) **Predicate** implementation instance. An example of such a **Predicate** is **GreaterThan**, whose `isTrue` function wrapper, specialized to the domain of integers, looks like this:

```
bool isTrue(int x, int y)
{
   return (x > y);
}
```

Note that this function has **two** arguments, `x` and `y`. That's why it's **bi**nary. Here's the entire class definition:

```
template <typename T1, typename T2>
class GreaterThan : public Predicate<int, int>
{
public:
   bool isTrue(int x, int y)
   {
      return (x > y);
   }

   bool isFalse(int x, int y)
   {
      return ! isTrue(x, y);
   }
};
```

Your creative task is to invent your own binary predicate, and get it to work with the generic **Predicate** superclass functions described above.