```
void flipStack2 (StackOfT& s)
//! updates s
//! requires |s| = 2
//! ensures s = #s[1,2) * #s[0,1)
```

Name: _____

Name: _____

One CM: _____

Reasoning Table for *flipStack2*

S	Code	Assume	Confirm
0			
	`T y;` `StackOfT t;`	/ / / / / / / / / / /	/ / / / / /
1			
	`s.pop(y)`	/ / / / / / / / / / /	/ / / / / /
2			
	`t.push(y);`	/ / / / / / / / / / /	/ / / / / /
3			
	`s.pop(y)`	/ / / / / / / / / / /	/ / / / / /
4			
	`t.push(y)`	/ / / / / / / / / / /	/ / / / / /
5			
	`s.transferFrom(t);`	/ / / / / / / / / / /	/ / / / / /
6			

To do:

1. Fill in the reasoning table for *flipStack2*
   - Use requires/ensures clauses from *flipStack2*
   - Use requires/ensures clauses from Stack's *pop*, *push*, and *transferFrom* (see backside for Stack's spec) Perform substitutions for *self* and parameter names and also attached state numbers
   - Reference the *goToRear* operation on the CSSE373 Moodle site

On a second piece of paper:

2. Write down VCs for each state (with the exception below), name them VC0, VC1, VC2, VC3, VC4, VC5, VC6
   Exception: If the proof of a VC would be trivial, i.e., if q is true in $p \rightarrow q$, then do not write down that VC
3. Prove the VC6

Stack Template

```
template <class T>
class Stack1
   //! is modeled by string of T
   //! exemplar self
{
public: // Standard Operations
   Stack1 ();
      //! replaces self
      //! ensures: self = <>
   ~Stack1 ();
   void clear (void);
      //! clears self
   void transferFrom (Stack1& source);
      //! replaces self
      //! clears source
      //! ensures: self = #source
   Stack1& operator = (Stack1& rhs);
      //! replaces self
      //! restores rhs
      //! ensures: self = rhs

   // Stack1 Specific Operations
   void push (T& x);
      //! updates self
      //! clears x
      //! ensures: self = <#x> * #self
   void pop (T& x);
      //! updates self
      //! replaces x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self  and  self = #self[1, |#self|)
   void replaceTop (T& x);
      //! updates self, x
      //! requires: self /= <>
      //! ensures: <x> is prefix of #self  and  self = <#x> * #self[1, |#self|)
   T& top (void);
      //! restores self
      //! requires: self /= <>
      //! ensures: <top> is prefix of self
   Integer length (void);
      //! restores self
      //! ensures: length = |self|

private: // Representation
};
```