Fun Facts of Assignment Semantics in C and Java

- C treats assignments as expression, while Java doesn't

- What are the output of Exp2.c? [a = 2, b = 2, c = 2; a = 0, b = 0, c = 3 ]
  - In assignments a = b = c evaluates c, assigns that to b, then assigns that to a.
  - In expressions c = 3 + (a = b) assigns the value of b to a and then adds 3 to the value and assigns it to c.

```c
#include <stdio.h>

int main () {
    int a = 1;
    int b = 0;
    int c = 2;
    a = b = c;
    printf("a = %d, b = %d, c = %d \n", a, b, c);
    a = 1;
    b = 0;
    c = 2;
    c = 3 + (a = b);
    printf("a = %d, b = %d, c = %d \n", a, b, c);
    return 0;
}
```

```java
public class Exp {
    public static void main (String[] args) {
        int a = 1;
        int b = 2;

        if (a = b) { //invalid in Java (a == b)
            System.out.printf("%d == %d \n", a, b);
        } else {
            System.out.printf("%d != %d \n", a, b);
        }
    }
}
```
Conditional Semantics

- In addition to the assignment statement, conditional statement (if statement) is another type of statement.
- Its meaning function also takes in a state and generates another state.

\[ M : \text{Statement} \times \text{State} \rightarrow \text{State} \]

- A conditional statement has three elements: an expression, then branch statement, and else branch statement.

\[ \text{Conditional} = \text{Expression test}; \text{Statement thenbranch, elsebranch} \]

- The meaning of a conditional depends on the meaning of the expression.
  - If the expression is true, the meaning of the conditional is the meaning of the then branch.
  - If the expression is false, the meaning of the conditional is the meaning of the else branch.

\[
M(\text{Conditional } c, \text{State } state) = \begin{cases} 
M(c.\text{thenbranch}, state), & \text{if } M(c.\text{test}, state) \text{ is true} \\
M(c.\text{elsebranch}, state), & \text{otherwise} 
\end{cases}
\]

- To extend our simple interpreter for the conditional statement, we need to add conditional class, the meaning function to statements including assignments, conditionals, and extendible to other statement types, the meaning function of conditional, and a main function to test them. Remember to extend the meaning function for binary expression to support relational operators: >, <, ==, and !=. (Part of HW5)

```python
# Conditional has three attribute: condition, thenbranch, and elsebranch
class Conditional:
    def __init__(self, condition, thenbranch, elsebranch):
        self.condition = condition
        self.thenbranch = thenbranch
        self.elsebranch = elsebranch

# Meaning function of Statement
# M(Statement statement, State state)
def M_Statement(statement, state):
    if isinstance(statement, Assignment):
        return M_Assignment(statement, state)
    elif isinstance(statement, Conditional):
        return M_Conditional(statement, state)
    else:
        print("ERROR: wrong type of statement")

# Meaning function of Conditional
# M(Conditional statement, State state)
def M_Conditional(statement, state):
    if M_Expression(statement.condition, state).value:
        state = M_Statement(statement.thenbranch, state)
    else:
        state = M_Statement(statement.elsebranch, state)
    return state
```
# expr should be a BinaryExpression
# state should be a State
# returns a value

def M_BinaryExpression(expr, state):
    if expr.operator == '+':
        val = M_Expression(expr.left, state).value + M_Expression(expr.right, state).value
    elif expr.operator == '-':
        val = M_Expression(expr.left, state).value - M_Expression(expr.right, state).value
    elif expr.operator == '*':
        val = M_Expression(expr.left, state).value * M_Expression(expr.right, state).value
    elif expr.operator == '/':
        val = M_Expression(expr.left, state).value / M_Expression(expr.right, state).value
    elif expr.operator == '>':
        val = M_Expression(expr.left, state).value > M_Expression(expr.right, state).value
    elif expr.operator == '<':
        val = M_Expression(expr.left, state).value < M_Expression(expr.right, state).value
    elif expr.operator == '==':
        val = M_Expression(expr.left, state).value == M_Expression(expr.right, state).value
    elif expr.operator == '!=':
        val = M_Expression(expr.left, state).value != M_Expression(expr.right, state).value
    return Value(val)

def main3():
    # if (a < b) {
    #   min = a;
    # } else {
    #   min = b;
    # }
    # {<min, 0>, <a, 3>, <b, 5>}
    varmin = Variable('min')
    vara = Variable('a')
    varb = Variable('b')
    state = State()
    state.setValue(varmin, 0)
    state.setValue(vara, 3)
    state.setValue(varb, 5)
    expr = BinaryExpression(vara, '<', varb)
    thenbranch = Assignment(varmin, M_Expression(vara, state))
    elsebranch = Assignment(varmin, M_Expression(varb, state))
    cond = Conditional(expr, thenbranch, elsebranch)
    newState = M_Statement(cond, state)
    print(newState)

if __name__ == "__main__":
    #main1()
    #main2()
    main3()