DIT636/DAT560 - Mutation Testing Activity

The following code iterates over an array and makes all negative values positive.

```
1. public int[] makePositive(int[] a){
2.     int threshold = 0;
3.     for(int i=0; i < a.length; i++){
4.         if(a[i] < threshold){
5.             a[i]= -a[i];
6.         }
7.     }
8.     return a;
9. }
```

1: How many mutations are possible for the following operators:

- **Relational Operator Replacement**
    - **Swap one of (<,<=,>,>=,==,!=) for one of the others**
- **Arithmetic Operator Replacement**
    - **Swap one of (+, -, *, /, %) for one of the others.**
    - **Swap one unary (-x, +x) for another**
    - **Swap one shortcut (--x,x--,++x,x++) for another**
    - **Can also swap one unary for one shortcut (e.g., -x to --x)**

2: Apply the relational operator replacement operation to statement 4 of the method, and identify test input that would lead to a different outcome from the unmutated method. You do not need to create a full unit test.

3: Design an equivalent mutant that no test case can detect. You may use any mutation operator discussed in class.

4: Design a valid (compiles), but useless (almost all tests will lead to different results than the unmutated method) mutant. You may use any mutation operator discussed in class.

ID	Operator	Description	Constraint
Operand Modifications			
crp	constant for constant replacement	replace constant $C1$ with constant $C2$	$C1 \neq C2$
scr	scalar for constant replacement	replace constant C with scalar variable X	$C \neq X$
acr	array for constant replacement	replace constant C with array reference $A[I]$	$C \neq A[I]$
scr	struct for constant replacement	replace constant C with struct field S	$C \neq S$
svr	scalar variable replacement	replace scalar variable X with a scalar variable Y	$X \neq Y$
csr	constant for scalar variable replacement	replace scalar variable X with a constant C	$X \neq C$
asr	array for scalar variable replacement	replace scalar variable X with an array reference $A[I]$	$X \neq A[I]$
ssr	struct for scalar replacement	replace scalar variable X with struct field S	$X \neq S$
vie	scalar variable initialization elimination	remove initialization of a scalar variable	
car	constant for array replacement	replace array reference $A[I]$ with constant C	$A[I] \neq C$
sar	scalar for array replacement	replace array reference $A[I]$ with scalar variable X	$A[I] \neq X$
cnr	comparable array replacement	replace array reference with a comparable array reference	
sar	struct for array reference replacement	replace array reference $A[I]$ with a struct field S	$A[I] \neq S$
Expression Modifications			
abs	absolute value insertion	replace e by abs(e)	$e < 0$
aor	arithmetic operator replacement	replace arithmetic operator ψ with arithmetic operator ϕ	$e_1 \psi e_2 \neq e_1 \phi e_2$
lcr	logical connector replacement	replace logical connector ψ with logical connector ϕ	$e_1 \psi e_2 \neq e_1 \phi e_2$
ror	relational operator replacement	replace relational operator ψ with relational operator ϕ	$e_1 \psi e_2 \neq e_1 \phi e_2$
uoi	unary operator insertion	insert unary operator	
cpr	constant for predicate replacement	replace predicate with a constant value	
Statement Modifications			
sdl	statement deletion	delete a statement	
sca	switch case replacement	replace the label of one case with another	
ses	end block shift	move } one statement earlier and later	

Figure 16.2: A sample set of mutation operators for the C language, with associated constraints to select test cases that distinguish generated mutants from the original program.