Expression			Type	Evaluation
0	==	0U	Unsigned	1
-1	<	0	Signed	1
-1	<	0U	Unsigned	0 *
2147483647	>	-2147483647-1	Signed	1
2147483647U	>	-2147483647-1	Unsigned	0 *
2147483647	>	(int) 2147483648U	Signed	1 *
-1	>	-2	Signed	1
(unsigned) -1	>	-2	Unsigned	1

Some possibly nonintuitive behavior arises due to C's handling of expressions containing combinations of signed and unsigned quantities. When an operation is performed where one operand is signed and the other is unsigned, C implicitly casts the signed argument to unsigned and performs the operations assuming the numbers are nonnegative. As we will see, this convention makes little difference for standard arithmetic operations, but it leads to nonintuitive results for relational operators such as < and >. Figure shows some sample relational expressions and their resulting evaluations, when data type int has a 32-bit two's-complement representation. Consider the comparison -1 < 0U. Since the second operand is unsigned, the first one is implicitly cast to unsigned, and hence the expression is equivalent to the comparison 4294967295U < 0U (recall that $T2Uw(-1) = UMaxw$), which of course is false. The other cases can be understood by similar analyses.

Effects of C promotion rules. Nonintuitive cases are marked by '*'. When either operand of a comparison is unsigned, the other operand is implicitly cast to unsigned.

we write TMin32 as -2,147,483,647-1.