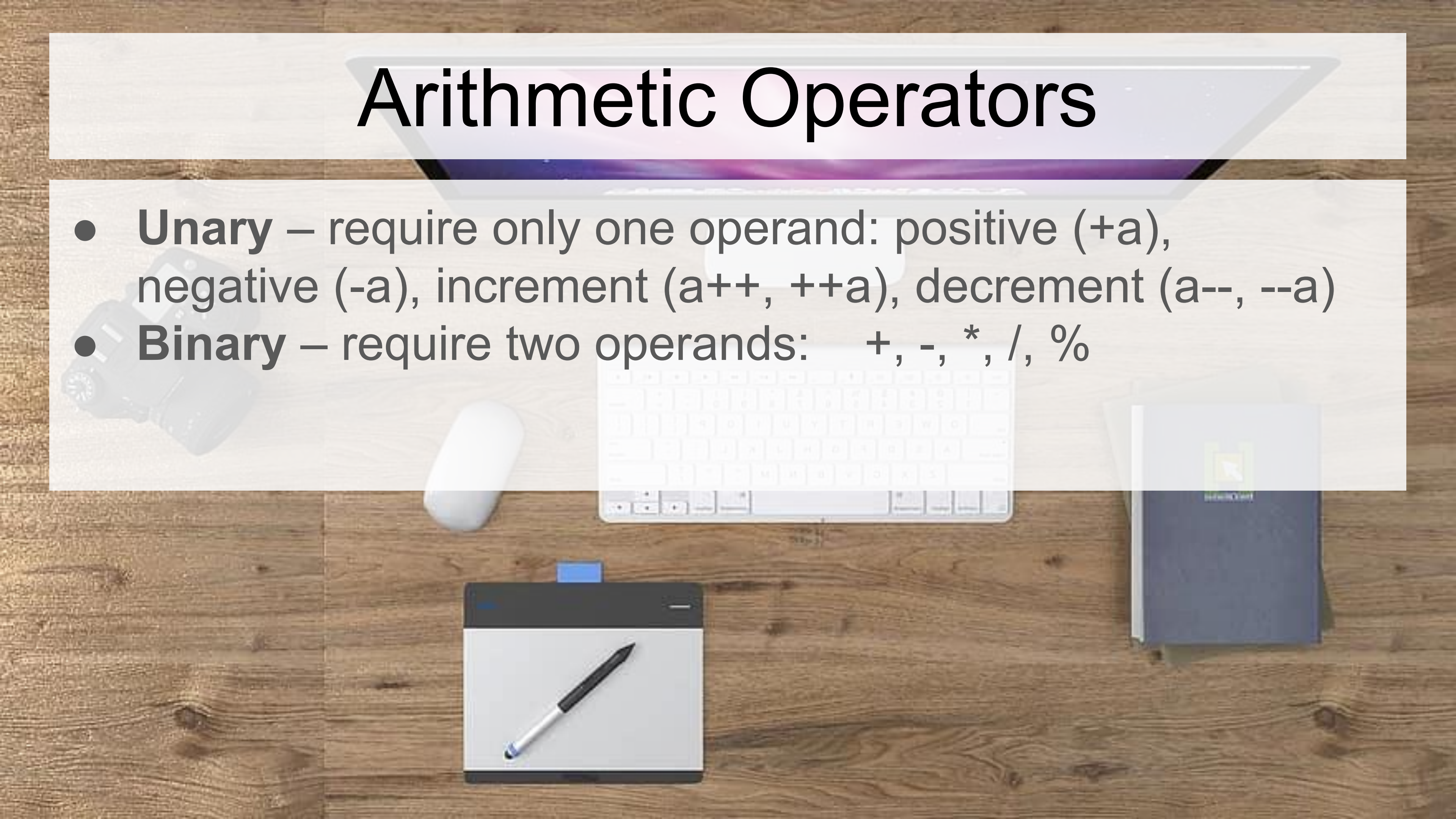
Operators

- Arithmetic operators
- Relational operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Special operators

Arithmetic Operators

- **Unary** – require only one operand: positive (+a), negative (-a), increment (a++, ++a), decrement (a--, --a)
- **Binary** – require two operands: +, -, *, /, %



Pre/Post Increments & Decrements

Order of increment and decrement operations:

Statement	Order of operations	**result** value	**num** value

result = num++;	result = num;
	num = num + 1;
		5	6

result = ++num;	num = num + 1;
	result = num
		6	6

result = num--;	result = num;
	num = num - 1;
		5	4

result = --num;	num = num - 1;
	result = num
		4	4

Assignment Operators

- Assignment operators assign values to the variable on the left side of the operand
- Assignment operators can be used to manipulate the value of (non-constant) variables throughout your program
- Example:

```
int A = 2, B = 4, C;
C = A + B;
```

In this example we are using the assignment operator to assign the value of `A + B` to `C` in line 2

Relational Operators

- Relational operators are used to check the relationship between two operands (two values)
- Results in either **TRUE (1)** or **FALSE(0)**
- Used in decision making and loops
- Arithmetic operators have a higher priority than relational operators
- Examples

```
number1 >= number2
number1 == number2
```

Logical Operators

- Used when more than one condition needs to be tested
- Similarly to relational operators, logical operators also result in either **TRUE (1)** or **FALSE(0)**

  Examples:

```
    varA > varB && varA > 0
    varA || varB < varA
    varA && varB
```

Expressions

- An expression is a combination of variables, constants, and operators
- Expressions are written according to the syntax of C language
- In a statement: `variable = expression;` the expression is evaluated first, then the previous value of the variable is replaced with the result of the expression