Operator Overloading

Terms

Binary operators
Friends of classes
Inline functions
Operator overloading
Spaceship operator
Subscript operator
Three-way comparison operator
Unary operators

Summary

- By overloading the built-in operators, we can make them work with our custom types (classes and structures). Most operators can be overloaded.
- If we overload == for a class, a C++ 20 compiler makes sure `!=` works as well.
- The *spaceship operator* <=> (also called *the three-way comparison operator*) determines, in a single expression, if X is less than, equal to, or greater than Y.
- If we overload <=> for a class, a C++ 20 compiler will generate all four relational operators (`<, <=, >, >=`) for us. It does not, however, generate == and `!=` operators as this can lead to suboptimal performance. So, the only two operators we need to overload are == and <=>.
- *Friend* functions have access to private members of a class despite not being members of the class. Friend functions undermine data hiding and should be used only when absolutely necessary.
- When overloading arithmetic operators (`+, -, *, /`), it's often best to overload their corresponding compound assignment operators (`+=, -=, *=, /=`).

- The copy constructor is called when creating a new object as a copy of an existing one. The assignment operator is used when assigning to an existing object.

- If we overload the assignment operator for a class, we often need to define a copy constructor for that class as well.

- The subscript operator `[ ]` is used to get access to individual elements in an object that behaves like an array or a collection.

- Functions defined in a class header file are known as *inline*. We can explicitly make functions defined in an implementation file inline using the **inline** keyword. Inline functions hint the compiler to optimize the executable by replacing each function call with the code in the function itself. Whether this happens or not is up to the compiler.


```
class Length {
public:
    // Comparison operators
    bool operator==(const Length& other) const;
    std::strong_ordering operator<=>(const Length& other) const;

    // Arithmetic operators
    Length operator+(const Length& other) const;
    Length& operator+=(const Length& other);

    // Assignment operator
    Length& operator=(const Length& other);

    // Increment operator
    Length& operator++(); // prefix
    Length operator++(int); // postfix

    // Type conversion
    operator int() const;
};
```

