Seminar 04
Operator overloading, friends, initializer list, const methods, constructor with default parameters

1. Const methods.

- Every method that does **NOT** change the object should be declared as **const**.
- Example in the header file:
  ```
  void print() const;
  ```
- Example in the source file:
  ```
  void ClassName::print() const {
      cout << <member>;
  }
  ```

2. Initializer list.

- Used for initializing const members or any member that needs to be initialized before the body of the constructor.
- Separated by commas, used like a constructor for the members.
- Other constructors can be called using the initializer list.
- Example:
  ```
  Person::Person() : age(0), name(nullptr)
  { … }
  ```
- Example2:
  ```
  Person::Person(int age, const char* name) : Person()
  { … }
  ```

3. Parameterized constructor with default parameters.

- Sometimes instead of defining a default constructor and a parameterized constructor, we can just define a parameterized constructor and use default values for the parameters.
- Example:
  ```
  Person::Person(int age = 0, const char* name = "N/A")
  {
      setAge(age);
      setName(name);
  }
  ```

4. Friend classes and friend functions.

- Classes and functions can be declared as friends to a given class
- Friends of the class can access all private data members and methods.
- Example: *friend-example.h*

5. Operator overloading.

- Changing how the operators work with objects from our classes.
- In C++ *almost* every operator can be overloaded.
- <return type> operator<operator>([parameters]);
- Example:

```cpp
class Complex {
public:
    Complex(int real, int imaginary)
        : real(real)
        , imag(imaginary)
    {}

    bool operator==(const Complex& other) const {
        return real == other.real && imag == other.imag;
    }

    Complex& operator+=(const Complex& other) {
        real += other.real;
        imag += other.imag;
        return *this;
    }

    Complex operator+(const Complex& other) const {
        return Complex(*this) += other;
    }

    friend std::ostream& operator<<(std::ostream& out,
                                    const Complex& obj) {
        return out << obj.real << " + " << obj.imag << "i";
    }

private:
    int real;
    int imag;
}
```

Source.cpp

```cpp
Complex c1(5, 6);
Complex c2(3, 1);
Complex c3(8, 7);
cout << (c1 == c2);      // false
cout << endl;
cout << (c1 + c2 == c3); // true
cout << endl;
cout << c1 << endl;      // 5 + 6i
```