Object Oriented programming and software engineering – Lab 12

Adam Korytowski – 2025

1. Operator overloading

Operator overloading in C++ allows you to redefine how operators (like +, -, *, ==, etc.) work with user-defined types (like classes). It lets your custom types behave more like built-in types, making your code cleaner and more intuitive. Example:

```
class Animal
{
public:
    int age = 0;
    Animal(int in_age): age(in_age) {}
    bool operator==(const Animal& a)
    {
        if (this->age == a.age)
        {
            return true;
        }
        return false;
    }
};
```

```
Animal* dog = new Animal(3);
Animal* cat = new Animal(3);
if (*dog == *cat)
{
    cout << "Equal ages";
}
```

Without overloading, *dog + *cat wouldn't work because C++ doesn't know how to add two Animal objects unless we tell it how.

2. Tasks (2 pts each)

- overload ‘+’ operator allowing to add two objects of your class (should work for objects of all of your deriving classes. This operator overloading should result in adding numerical values of one of the objects’ properties. Example:

```
Character* warrior = new Warrior(100);
Character* warrior2 = new Warrior(80);
```

```
Character c = *warrior + *warrior2;
```

- overload “==” operator allowing to compare two objects of your existing class, checking if one of their properties are equal
- overload “&&” operator for one of your existing classes. The overloaded && operator should return true if both objects’ properties satisfy logical comparisons (e. g. for *int* the property is >1). Test this functionality by creating two objects of your class and using the overloaded && operator to check compatibility between different objects based on their properties.