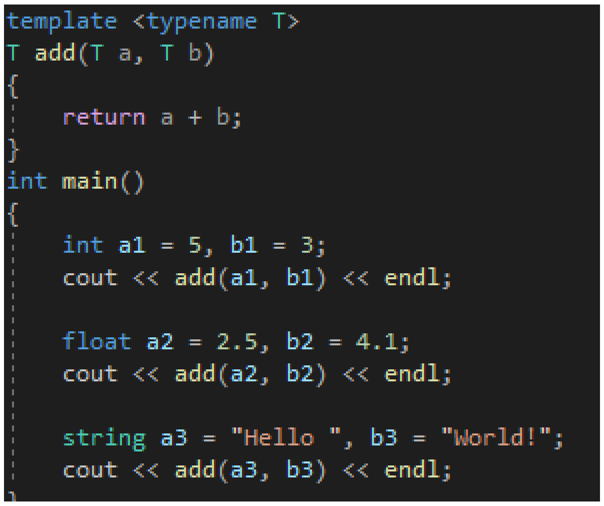
Object Oriented programming and software engineering – Lab 11

Adam Korytowski – 2025

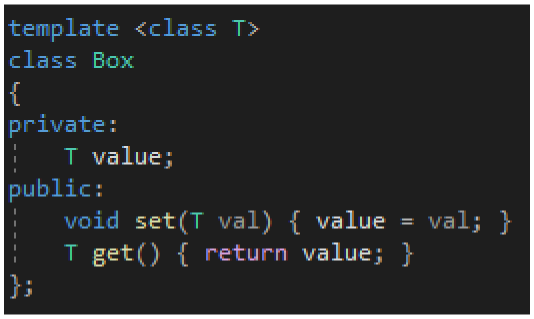
1. Function templates

A function template allows you to create a single function to work with different data types. Instead of writing multiple overloaded functions for different types, you write one template function.

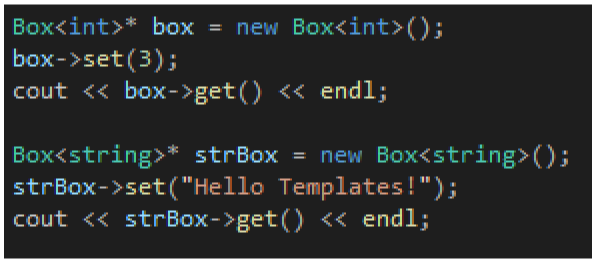


```
template <typename T>
T add(T a, T b)
{
    return a + b;
}
int main()
{
    int a1 = 5, b1 = 3;
    cout << add(a1, b1) << endl;

    float a2 = 2.5, b2 = 4.1;
    cout << add(a2, b2) << endl;

    string a3 = "Hello ", b3 = "World!";
    cout << add(a3, b3) << endl;
```

A class template lets you create a class to work with any data type. It's like creating a blueprint for a class that can adapt to various data types.



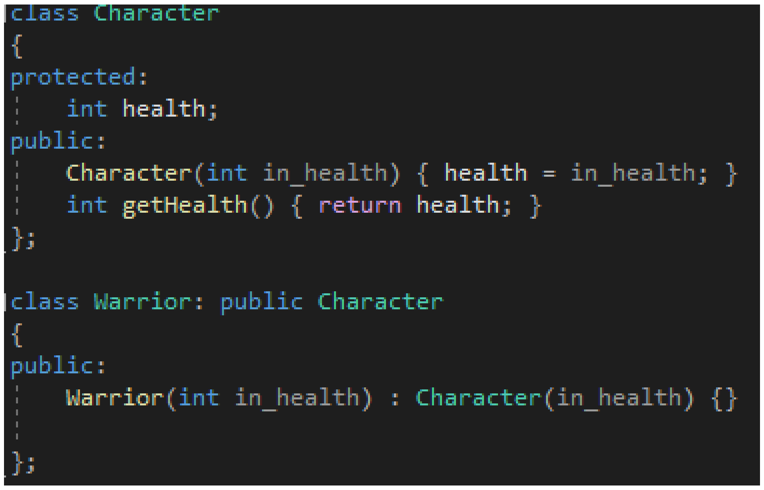
```
template <class T>
class Box
{
private:
    T value;
public:
    void set(T val) { value = val; }
    T get() { return value; }
};
```



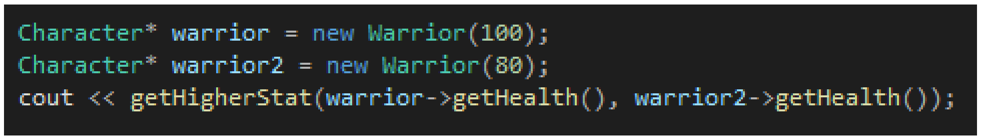
```
Box<int>* box = new Box<int>();
box->set(3);
cout << box->get() << endl;

Box<string>* strBox = new Box<string>();
strBox->set("Hello Templates!");
cout << strBox->get() << endl;
```

2. Using your existing code:

- Write a function template getHigherStat() that compares values of a specific field of two objects of any class having the field we want to compare (2 pts). Example:



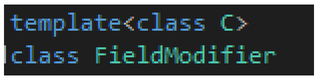
```
class Character
{
protected:
    int health;
public:
    Character(int in_health) { health = in_health; }
    int getHealth() { return health; }
};

class Warrior: public Character
{
public:
    Warrior(int in_health) : Character(in_health) {}

};
```



```
Character* warrior = new Warrior(100);
Character* warrior2 = new Warrior(80);
cout << getHigherStat(warrior->getHealth(), warrior2->getHealth());
```

In this example, the getHigherStat() function should return 100.

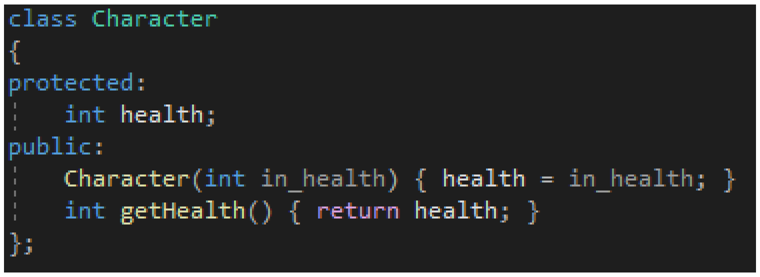
- Create a template class:



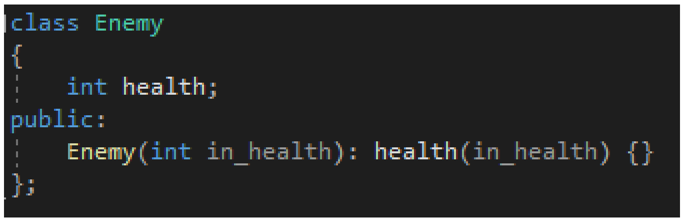
```
template<class C>
class FieldModifier
```

that will act as a manager for modifying values of fields of your existing classes (2 pts each). The class should contain functions for:

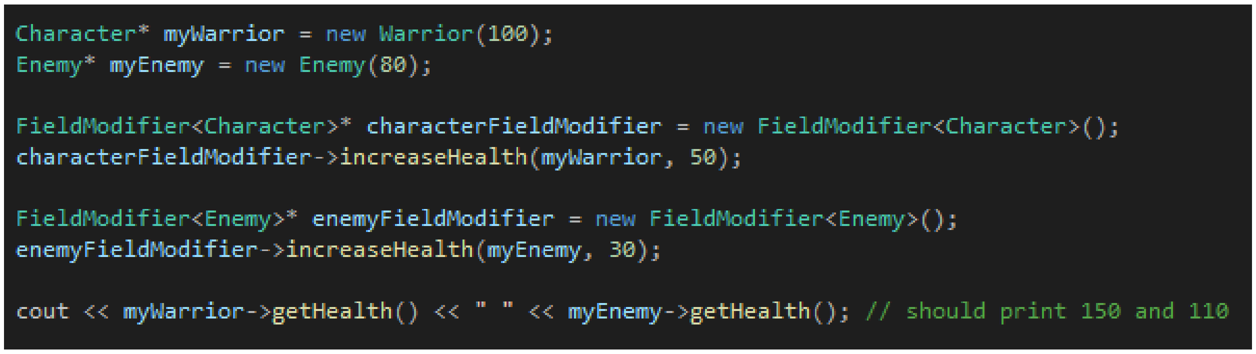
  - increasing value of a field of a class (create a field of the same name in two of your classes (not connected by inheritance). Example:



```cpp
class Character
{
protected:
    int health;
public:
    Character(int in_health) { health = in_health; }
    int getHealth() { return health; }
};
```

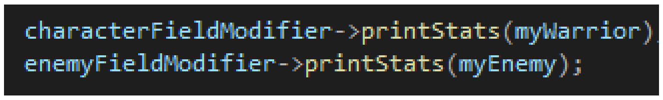


```cpp
class Enemy
{
    int health;
public:
    Enemy(int in_health): health(in_health) {}
};
```



```cpp
Character* myWarrior = new Warrior(100);
Enemy* myEnemy = new Enemy(80);

FieldModifier<Character>* characterFieldModifier = new FieldModifier<Character>();
characterFieldModifier->increaseHealth(myWarrior, 50);

FieldModifier<Enemy>* enemyFieldModifier = new FieldModifier<Enemy>();
enemyFieldModifier->increaseHealth(myEnemy, 30);

cout << myWarrior->getHealth() << " " << myEnemy->getHealth(); // should print 150 and 110
```

- printing values of fields of your classes, example of use:



```cpp
characterFieldModifier->printStats(myWarrior)
enemyFieldModifier->printStats(myEnemy);
```

The function should print values the class's member variables.