LAB #9 – More Classes & Polymorphism

1. First, finish Lab #8, if you haven’t done so already!!!

2. Continue to Practice Classes:

The following lists a Dice class that simulates rolling a die with a different number of sides. The default is a standard die with six sides. The rollTwoDice function simulates rolling two dice objects and returns the sum of their values. The srand function requires including cstdlib.

```cpp
class Dice {
    public:
        Dice();
        Dice(int numSides);
        virtual int rollDice() const;
    protected:
        int numSides;
};
Dice::Dice() {
    numSides = 6;
    srand(time(NULL)); // Seeds random number generator
}
Dice::Dice(int numSides) {
    this->numSides = numSides;
    srand(time(NULL)); // Seeds random number generator
}
int Dice::rollDice() const {
    return (rand() % numSides) + 1;
}

// Take two dice objects, roll them, and return the sum
int rollTwoDice(const Dice& die1, const Dice& die2) {
    return die1.rollDice() + die2.rollDice();
}
```

Write a main function that creates two Dice objects with a number of sides of your choosing. Invoke the rollTwoDice function in a loop that iterates ten times and verify that the functions are working as expected.

Next, create your own class, LoadedDice, that is derived from Dice. Add a default constructor and a constructor that takes the number of sides as input. Override the rollDice function in LoadedDice so that with a 50% chance the function returns the largest number possible (i.e. numSides), otherwise it returns what Dice’s rollDice function returns.
Test your class by replacing the `Dice` objects in main with `LoadedDice` objects. You should not need to change anything else. There should be many more dice rolls with the highest possible value. Polymorphism results in `LoadedDice's rollDice` function to be invoked instead of `Dice's rollDice` function inside `rollTwoDice`.

**Show your program to a lab TA for credit, and explain to a TA how you tested your code for correctness.**