1.

```cpp
#include <iostream>

// Base class
class Wizard {
public:
    virtual void castSpell() {
        std::cout << "Casting a generic spell" << std::endl;
    }
};

// Derived class
class Sorcerer : public Wizard {
public:
    void castSpell() override {
        std::cout << "Casting a fireball" << std::endl;
    }
};

// Another derived class
class Mage : public Sorcerer {
public:
    void conjure() {
        std::cout << "Conjuring magical creatures" << std::endl;
    }
};

int main() {
    // Wizard pointer to Wizard object
    Wizard* wizard = new Wizard();
    wizard->castSpell();

    // Wizard pointer to Sorcerer object
    Wizard* sorcerer = new Sorcerer();
    sorcerer->castSpell();

    // Wizard pointer to Mage object
    Wizard* mage = new Mage();
    mage->castSpell();

    // Mage pointer to Mage object
    Mage* realMage = new Mage();
    realMage->castSpell();
    realMage->conjure();

    delete wizard;
    delete sorcerer;
    delete mage;
    delete realMage;

    return 0;
}
```

```
Casting a generic spell
Casting a fireball
Casting a fireball
Casting a fireball
Conjuring magical creatures
```

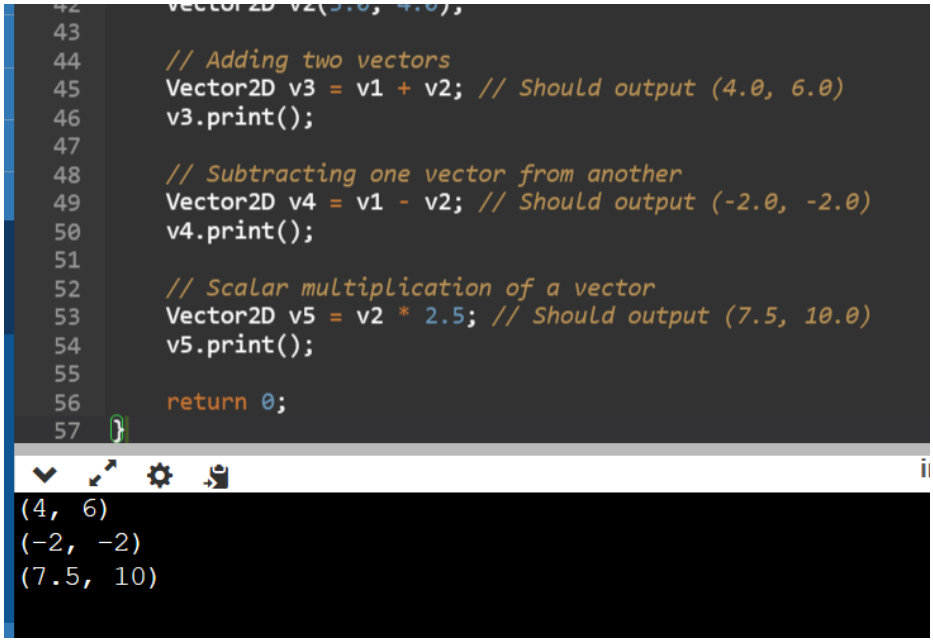
2.

```cpp
#include <iostream>
#include <string>

template <typename T>
class Stack {
private:
    static const int MAX_SIZE = 100;
    T arr[MAX_SIZE];
    int top;

public:
    Stack() : top(-1) {}

    void push(const T& item) {
        if (top == MAX_SIZE - 1) {
            std::cout << "Stack Overflow!" << std::endl;
            return;
        }
        arr[++top] = item;
    }

    T pop() {
        if (isEmpty()) {
            std::cout << "Stack Underflow!" << std::endl;
            return T();
        }
        return arr[top--];
    }

    T peek() const {
        if (isEmpty()) {
            std::cout << "Stack is empty!" << std::endl;
            return T();
        }
        return arr[top];
    }

    bool isEmpty() const {
        return top == -1;
    }
};

int main() {
    // Stack of integers
    Stack<int> intStack;
    intStack.push(10);
    intStack.push(20);
    intStack.push(30);

    std::cout << "Top element of intStack: " << intStack.peek() << std::endl;
    std::cout << "Popped element from intStack: " << intStack.pop() << std::endl;
    std::cout << "Is intStack empty? " << (intStack.isEmpty() ? "Yes" : "No") << std::endl;

    // Stack of strings
    Stack<std::string> stringStack;
    stringStack.push("Hello");
    stringStack.push("World");

    std::cout << "Top element of stringStack: " << stringStack.peek() << std::endl;
    std::cout << "Popped element from stringStack: " << stringStack.pop() << std::endl;
    std::cout << "Is stringStack empty? " << (stringStack.isEmpty() ? "Yes" : "No") << std::endl;

    return 0;
}
```

```
Top element of intStack: 30
Popped element from intStack: 30
Is intStack empty? No
Top element of stringStack: World
Popped element from stringStack: World
Is stringStack empty? No
```

3.

```cpp
#include <iostream>

class Vector2D {
private:
    float x, y;

public:
    // Constructor to initialize x and y
    Vector2D(float x_val, float y_val) : x(x_val), y(y_val) {}

    // Overload the + operator to add two Vector2D objects
    Vector2D operator+(const Vector2D &other) const {
        return Vector2D(x + other.x, y + other.y);
    }

    // Overload the - operator to subtract one Vector2D object from another
    Vector2D operator-(const Vector2D &other) const {
        return Vector2D(x - other.x, y - other.y);
    }

    // Overload the * operator to perform scalar multiplication
    Vector2D operator*(float scalar) const {
        return Vector2D(x * scalar, y * scalar);
    }

    // Method to print the vector in the format "(x, y)"
    void print() const {
        std::cout << "(" << x << ", " << y << ")" << std::endl;
    }
};

int main() {
    Vector2D v1(1.0, 2.0);
    Vector2D v2(3.0, 4.0);
```



```
    // Adding two vectors
    Vector2D v3 = v1 + v2; // Should output (4.0, 6.0)
    v3.print();

    // Subtracting one vector from another
    Vector2D v4 = v1 - v2; // Should output (-2.0, -2.0)
    v4.print();

    // Scalar multiplication of a vector
    Vector2D v5 = v2 * 2.5; // Should output (7.5, 10.0)
    v5.print();

    return 0;
}
```

```
(4, 6)
(-2, -2)
(7.5, 10)
```