Georgios Markou HWP2 Test2

a) Das Argument wird als const& deklariert, weil es nicht vom Copy Constructor modifiziert werden darf. Würde wir das Argument als pass-by-value übergeben, müssten wir erstmal eine Kopie davon erstellen, wofür der Copy Constructor zuständig ist.

b)

```
class Car
{
public:
	Car() : m_mileage(0) {}
	Car(int mileage) : m_mileage(mileage) {}

	int getMileage() const { return m_mileage; }

private:
	int m_mileage;
};
```

c)

```
struct Car
{
public:
	Car() : m_mileage(0) {}
	Car(int mileage) : m_mileage(mileage) {}

	int getMileage() const { return m_mileage; }

private:
	int m_mileage;
};
```

d)

1 ist ein Fehler, weil kein Default-Constructor existiert.
2 ist ebenfalls ein Fehler, weil beide Konstruktoren passen (mehrdeutig).
3 ist richtig, B wird verwendet.

e)

```
class Bird
{
public:
	Bird() : m_weight(0) {}
	Bird(float initWeight) : m_weight(initWeight) {}
	Bird(float initWeight, int dummyInt) : m_weight(initWeight) {}

private:
	float m_weight;
};
```

f bis i)

```
class Runway
{
public:
	Runway() : m_direction(0) {}
	Runway(int dir) : m_direction(dir) {}
	~Runway() {}

	Runway(const Runway& runway) : m_direction(runway.m_direction) {}
	Runway& operator = (const Runway& runway)
	{
		if (this == &runway)
			return *this;

		m_direction = runway.m_direction;

		return *this;
	}

	static float getAirTemperature() { return m_airTemperature; }

private:
	int m_direction;
	static float m_airTemperature;
};

float Runway::m_airTemperature = 20;

//Aufgabe h
int main()
{
	Runway r(10);

	r.getAirTemperature(); //1
	Runway::getAirTemperature(); //2
}
```