Task 7 Spike: Operator Overloading

EXTENSION

Context

Games often make use of a range of data types that are more complicated than primitives, but can still make use (conceptually, at least) of operators used by primitive data types. Operator overloading in C++ allows us to manipulate complex data types using standard operators.

Knowledge/Skill Gap:

The developer is not familiar with the concept of overloading operators to allow complex data types to be acted upon by C++ operators

Goals

Create either a vector (i.e. mathematical vector, not STL vector) or a matrix class object, then implement operator overloading to allow something similar to the following operations (this table only deals with a vector - if you're implementing a matrix, you'll have to do some research to determine what comparable operations are):

Overloaded Method	**Description**	**Example**
Constructor	Create a new vector with the passed in parameters & dimensions.	new Vector (1,2) // a 2D vector, x=1, y=2 new Vector (1,2,3) // a 3D vector, x=1, y=2, z=3

= (equals)	Assign the values of one vector to another	a = Vector(1,2,3)
		b = a //b == Vector(1,2,3)

+; -; +=; -=	Add/subtract the values of one vector to/from another	a = new Vector(1,2,3)
		c = a + new Vector(4,5,6) //c == new Vector(5,7,9)

*; /; *=; /=	Multiply/divide the values of one vector by/with another
	Multiply/Divide a vector by a scalar
		a = new Vector(1,2,3)
		b = a*new Vector(4,5,6) //b == new Vector(1,10,18)
		c = new Vector(7,8,9)
		d = c/2 //d == new Vector(3.5,4,4.5)

++; --	Extend (or reduce) the length of a vector by 1	a = new Vector(1,1,1) a++ //a == new Vector(2,2,2)

==
>; >=
<; <=
	Check to see if all the values of a vector are equal to/greater than/greater than or equal to all the values of another.	a = Vector(1,1,1)
		b = Vector(1,1,1)
		a == b //true

[]	Access the values of a vector	a = new Vector(1,2,3) a[1] //2 a['x'] //1

Expected Output

Repository
1. Code
2. Spike Report

Canvas
1. Spike Report

Notes

Consider doing this later
You don't have to do this now! You can come back to it later if you want to do it and as your skills improve.