Operator Overloading

Giving operators a prettier look

Operators

- C++ provides a rich set of operators and defines what these operators do when applied to operands of built in types.
- There are various types of operators
  - Arithmetical operators
  - Logical and relational operators
  - Others (see http://www.cplusplus.com/doc/tutorial/operators/)

Operator Overloading

- These operations are limited to basic data types
- What can be applied for user-defined data classes?
- A technique known as operator overloading allows us to customize the behaviour of operators within out classes
- This allows for very concise syntax and helps make code which uses custom classes, much more readable.

A first look...

- Consider this class
- How can we compare if a equals b?

```
class Position2D
{
public:
    float x, y;
};
```

```
int main()
{
    Position2D a, b;

    //One way to check if two positions are equal
    if(a.x == b.x && a.y == b.y)
    {
        //Do something
    }

    system("pause");
}
```

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

A first look...

- Would be nice if we can do this:

```
int main()
{
    Position2D a, b;

    //Error: no operator "==" matches these operands
    if(a == b)
    {
        //Do something
    }

    system("pause");
}
```

Operator Overloading

- We can do this by overloading the *operator==()* function.
- The syntax is
  - <return type> operator<operator-symbol>(<parameters>);
- The parameters depend on the type of operator
  - Binary operators require two parameters
  - Unary operators require one parameter

Unary operators

!	Logical NOT
&	address-of
~	one’s complement
*****	pointer dereference
+	unary plus
-	unary negative
++	increment
--	decrement

Common Binary operators

==	Equality
!=	Inequality
*****	Multiplication
***=**	Multiplication / Assignment
+	Addition
+=	Addition / Assignment
-	Subtraction
-=	Subtraction / Assignment
/	Division
/=	Division / Assignment
<	Less Than
Etc…	

Operator Overloading

- Example: Overloading the == operator in the Position class

```
class Point2D
{
public:
    float x, y;

    //Overload the == operator
    bool operator==(Point2D &other)
    {
        return x == other.x && y == other.y;
    }
};
```

- Note the use of the reference (&) this is not essential in this case but it is good practice

Arithmetic Operators

- Arithmetic Operators (+, -, /, *) can be overloaded
  - Allows arithmetic operations to be performed on user defined data types
  - If operator+() is overloaded, then it is good practice to over the operator+=() as well.

Arithmetic Operators

- Example of overloading the + operator

```
Point2D operator+(Point2D &other)
{
    Point2D temp;
    temp.x = x + other.x;
    temp.y = y + other.y;

    return temp;
}
```

- Note the use of a temporary value when performing the addition inside the function
  - What would happen if we modified the parameter which was passed into the function or the values of the object being referenced by the function?

Operator Parameters

- As with functions, any complex struct or class should be passed as a reference (or pointer)
- If the parameter is not to be modified by the operator, pass it as a const reference
- For binary operators (such as =)
  - The lhs (left hand side) of the operator is the first parameter
  - The rhs (right hand side) is the second parameter

Summary

- Overloading operators can make life more convenient
- Overload related sets of operands if you decide to overload
- Don’t overload an operator and make it do something unintuitive!