Operator Overloading

Programming Fundamentals

Dr Kevin Chalmers

School of Computing
Edinburgh Napier University
Edinburgh

k.chalmers@napier.ac.uk

Edinburgh Napier UNIVERSITY

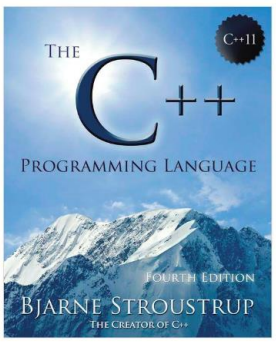
Notes

Outline

Notes

References

- Stroustrup (2011), *The C++ Programming Language, 4th Edition*
  - The inventor of C++
- Also look at the official C++ documentation coming with your development environment



Notes

Outline

Notes

Operator Overloading

- C++ implements a powerful feature, *operator overloading*.
- Operator overloading allows developers to redefine the behaviour of classes when they are being used with standard operators (e.g., +, -, *, etc.).
  - This is also known as ad-hoc polymorphism.
- Some other languages implement this feature (e.g. C#, Python etc) whilst some others don't (e.g. Java), making it an often overlooked feature.

Notes

Using Operators

- You have already been using operator overloading, e.g.
  - When using a vector type you can access values by using a function (e.g. `vec.at(4)`) or an operator overload (`vec[4]`).
  - Similarly, `cout << "Hello"` is also using an operator. This can be overloaded for different classes.
- Operator overloading uses the `operator` keyword, e.g.
  - `type operator+(const type &lhs, const type &rhs)`

Notes

Examples of Operators

Type	Function	Example
Arithmetic	Based on arithmetic operations	`+, -, *, /`
Assignment	What happens when you assign a value to an object	`=`
Increment, decrement	Both pre and post-fix	`variable++`, `++variable`
Logical	And, or, not, equals	`&&, ||, !`
Comparison	Greater than, less than	`<, >, ++, !=`
Member access	As used in `vector`	`[ ]`
Other	Casting, `new`, `delete`, etc.	

Notes

Examples of Applications

Arithmetics you can create classes for vector or matrices and the regular operators to perform

Object descriptions you can print an object description in the form of a `string` to `std::cout` or another file by using the << syntax.

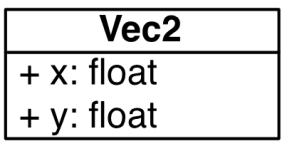
Object comparisons Easily compare two objects to see if they are "equal" (under the programmer's definition)

- Other definitions uses also exist

Notes

Case Study - vec2

- This lecture the practical will use the same case study: vec2
- vec2 is a 2-dimensional vector, in the mathematical definition, it can be a 2D point in space.



Notes

Questions?

Notes

Outline

Notes

Comparison : Equality

- We can implement the comparison == operator in a class as a *member function* in the following way:

Equality Operator

```
bool operator==(const vec2 &rhs)
{
  return (this->x == rhs.x) && (this->y == rhs.y);
}
```

- If we compare two vec2 objects with identical x and y values, it will return true, other it will return false.

Notes

Comparison : Inequality

- Similarly, we can implement the negative comparison != operator in the following way:

Inequality Operator

```
bool operator!=(const vec2 &rhs)
{
  return !(*this == rhs);
}
```

- We can use the syntax above because we already redefined the == operator, which is a handy shortcut *in this situation*.

Notes

Comparison : Greater / Less Than

- For our example, we will define less based on Pythagoras (there is no such thing as a vec2 being more or less than another):

$$length^2 = x^2 + y^2$$

Less Than Operator

```
bool operator<(const vec2 &rhs)
{
  float my_len = (this->x * this->x) + (this->y * this
      ->y);
  float rhs_len = (rhs.x * rhs.x) + (rhs.y * rhs.y);
  return my_len < rhs_len;
}
```

- In this example, the operator will return true or false based on the squared length of our vector.

Notes

Comparison : Caution!

- You need to be consistent. If you decided to apply Pythagoras to the *less than* operator, the same needs to apply to the *more than* operator. It is technically possible to implement anything, however you need to keep it logical, especially if others will use your class.
- Greater than or equal, and less than or equal need to be defined separately. You will need to implement all the logical cases individually.

Notes

Questions?

Notes

Outline

Notes

Arithmetic : Addition

- Intuitively the + operator can be used for arithmetic additions.
  - This is how we have been conditioned to use it from years and years of math classes.
- However, in programming, it is often used to *abstractly* add two things together. For example stings:
  - `string3 = string1 + string2;`
- The example above would conventionally perform an operation similar to the C `strcat` function, which is not an arithmetic operation.

Notes

Artihmetic : Addition

- In our example, we can add two vectors by adding their components. For example if `w` were the addition of two vectors `v` and `u`:

$$w_x = v_x + u_x$$
$$w_y = v_y + u_y$$

Adding Two Vectors

```
vec2 operator+(const vec2 &rhs)
{
  // Assuming the correct constructor exists
  return vec2(this->x + rhs.x, this->y + rhs.y);
}
```

Notes

Arithmetic : Multiplication

- Note in the last example, we actually create a new vector and return it to the parent function.
- C++ allows you to define different types of arguments for operators, e.g. you could multiply two vectors together:
  - `vec2 operator*(const vec2 &rhs) {...}`
- Or perform a scalar multiplication:
  - `vec2 operator*(float scale) {...}`

Notes

Questions?

Notes

Outline

Notes

Assignment

- You can assign a vector to another using the = operator:

Assignment Operator

```
vec2& operator=(const vec2 &rhs)
{
  this->x = rhs.x;
  this->y = rhs.y;
  return *this;
}
```

- Note that we return a *reference* to vec2. This is to ensure we return the original object and not a copy. In this case, we need to use the dereference operator.

Notes

Assignment

- Other assignment operators (+=, -=, *=, /=) work in a similar manner:

Addition-Assignment Operator

```
vec2& operator+=(const vec2 &rhs)
{
  this->x = this->x + rhs.x;
  this->y = this->y + rhs.y;
  return *this;
}
```

- These operators are shortcuts for assignment and arithmetic operations. They are widely used in C-like languages.

Notes

Member Access

- Our vec2 class contains two member floats. We can use the [] operator to access both members.
  - e.g. vec[0] or vec[1].

Member Access Operator

```
float& operator[](int index)
{
  assert(index >= 0 && index <= 1);
  if (index == 0)
    return x;
  else
    return y;
}
```

Notes

Input and Output

- The >> and << operators for input and output can also be overloaded.
- This is useful if we want to control the way an object is printed or read from an I/O stream.
- Other languages provide similar features for their object types, like `toString()` in Java or `description` in Objective-C.

Notes

Input and Output

- To redefine the input to a stream, we proceed this way:

Input Operator

```
friend istream& operator>>(istream &in, vec2 &value)
{
  in >> value.x >> value.y;
  return in;
}
```

- Note the `friend` keyword, which alters the public/private visibility of class members. This function is defined outside the class scope. `friend` is outside the scope of this module but is required in this scenario.

Notes

Example - Input

- The last example would allow the manual creation of a vec2 using the console:

Reading in a vec2

```
// Create object
vec2 v;
// Read from console
cin >> v;
```

- In the example above, a user would be prompted to enter two float values on the keyboard, thus creating a 2D vector with those values as x and y.

Notes

Example - Output

- The output to a stream is similar. Note that you can format the output to be easily understandable in logs and debugging consoles:

Output Operator

```
friend ostream& operator<<(ostream &out, const vec2 &
    value)
{
  out << "{" << value.x << ", " << value.y << "}";
  return out;
}
```

- This overload would allow us to output the x and y values of a vec2 by passing an object to a stream, like std::cout.

Notes

Questions?

Notes

Outline

Notes

Summary

- Operator overloading is useful to redefine the behaviours of primitive C++ operators.
- This enables a higher level abstraction, where we can manipulate objects with operators as if they were primitive data types.
- This can make classes "feel" easier and more natural to uase than only using these methods:
  - `vec3 = vec1 + vec2`
  - `vec3 = vec1.add(vec2)`

Notes

To do...

- In the lab - lots of work with operator overloading. This is useful for coursework 2.
  - Speaking of which, you should all be making progress through this now.
- Coursework 2 - any queries contact Kevin ASAP.
- Next time - data structures using pointers. Helps your thinking for coursework 2.

Notes