C++ Operator Overloading

Some content adapted from ‘Absolute C++’ by Walter Savitch

Outline

- Operator overloading

Operator Overloading Introduction

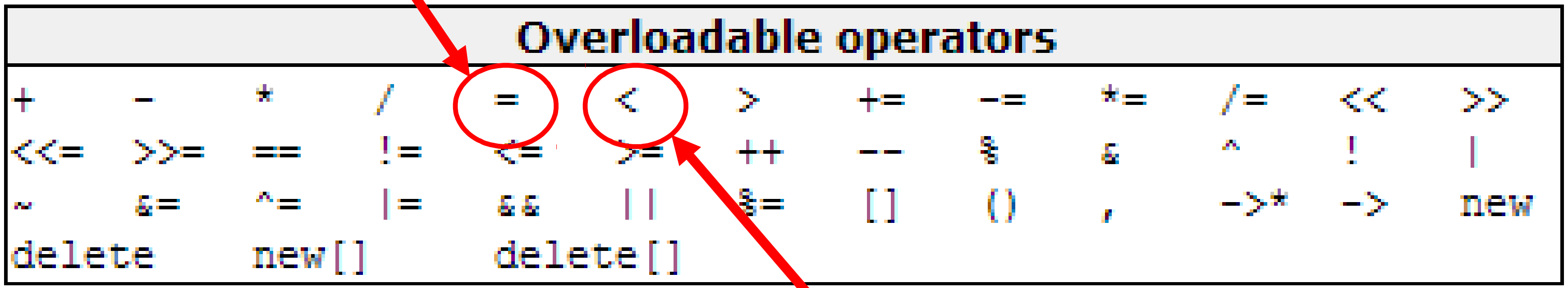
- Operators <,+, -, %, ==, etc.
  - Really are just functions!
- Simply "called" with different syntax:
  x < 7
  - “<" is binary operator with x & 7 as operands
  - We "like" this notation as humans
- Think of it as:
  <(x, 7)
  - “<" is the function name
  - x, 7 are the arguments
  - Function “<" returns bool of it’s arguments
- Can be done 2 ways
  - Overload as an object member function
  - Overload as a non member function

Operator Overloading Why

- Already work for C++ built-in types (int, double, etc.)
- Our types get same built in behavior. But we can (and usually need to) customize it programmatically.

Did this already for objects with dynamic data

```
//assignment operator
HoldsDynamicData & operator= (const HoldsDynamicData & other);
```

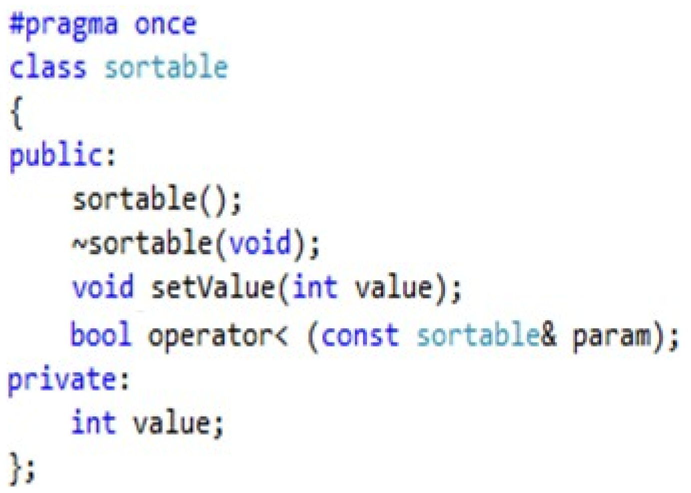


Overloadable operators												
+	-	*	/	=	<	>	+=	-=	*=	/=	<<	>>
<<=	>>=	==	!=	<=	>=	++	--	%	&	^	!	|
~	&=	^=	|=	&&	||	%=	[]	()	,	->*	->	new
delete		new[]		delete[]								

Implement this one to simplify sorting using std::sort

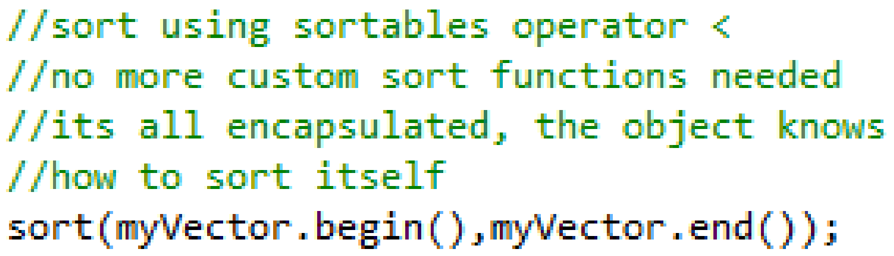
Sorting – what < overload buys you

- Remember using a get function to help with sorting?
- Instead implement < operator. Then the object knows how to sort itself



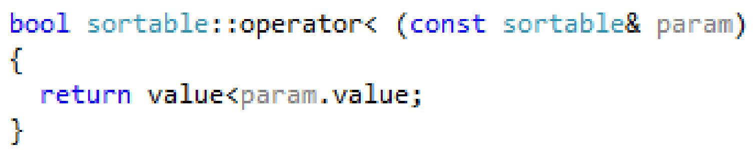
```
#pragma once
class sortable
{
public:
    sortable();
    ~sortable(void);
    void setValue(int value);
    bool operator< (const sortable& param);
private:
    int value;
};
```

```
vector<sortable> myVector;
```



```
//sort using sortables operator <
//no more custom sort functions needed
//its all encapsulated, the object knows
//how to sort itself
sort(myVector.begin(),myVector.end());
```

Just 2 arguments

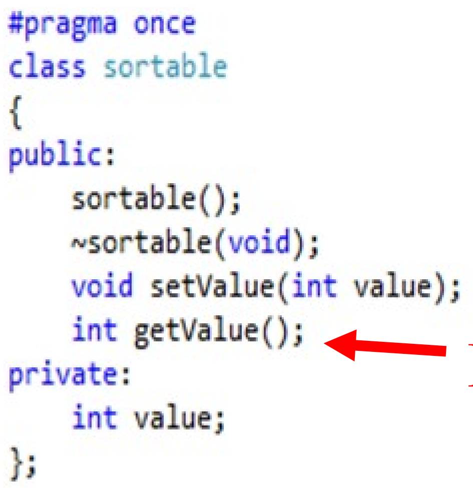


```
bool sortable::operator< (const sortable& param)
{
  return value<param.value;
}
```

implementation

Sorting– as opposed to using getters

- Remember using a get function or a friend function to help with sorting?



```
#pragma once
class sortable
{
public:
    sortable();
    ~sortable(void);
    void setValue(int value);
    int getValue();
private:
    int value;
};
```

Exposed data here

```
vector<sortable> myVector;
std::sort(myVector.begin(), myVector.end(), compareVal);

bool compareVal(const sortable &l,const sortable &r){
        return l.getValue() < r.getValue();
}
```

Summary

- Operators are really just functions