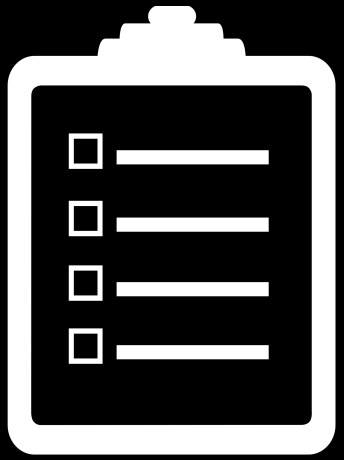
Functions

Ali Malik
malikali@stanford.edu

Game Plan



Recap

Operator Overloading

Functions

Lambdas

Announcements

Recap

Classes - Issues

C++ doesn’t know how to use operators on types defined by us

An algorithm needed a function that could capture a local variable

Operator Overloading

Operator Overloading

Allows you to define functionality for operators on any types.

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

Operator Overloading

Use only when overloading has an intuitive meaning:

```
Set<int> numSet;
numSet += 2;
numSet += 3;
// numSet is now {2, 3}
```

Good overload of `+=` operator!

Operator Overloading

Use only when overloading has an intuitive meaning:

```
Set<int> numSet;
numSet += 2;
numSet += 3;
// numSet is now {2, 3}
```

Operator Overloading

Use only when overloading has an intuitive meaning:

```
Set<int> numSet;
numSet += 2;
numSet += 3;
numSet, 4, 5;
// numSet is now {2, 3}
```

Operator Overloading

Use only when overloading has an intuitive meaning:

```
Set<int> numSet;
numSet += 2;
numSet += 3;
numSet, 4, 5;
// numSet is now ???
```

No intuitive understanding of what this does.

Operator Overloading

```
struct Point {
    int x, y;
    bool operator==(const Point& rhs) {
        return x == rhs.x && y == rhs.y;
    }
};
```

Operator Overloading

Class member function. LHS is implicit this object

```
struct Point {
    int x, y;
    bool operator==(const Point& rhs) {
        return x == rhs.x && y == rhs.y;
    }
};
```

Operator Overloading

```
struct Point {
    int x, y;
    bool operator==(const Point& rhs) {
        return x == rhs.x && y == rhs.y;
    }
};
```

Operator Overloading

```
struct Point {
    int x, y;
};

bool operator==(const Point& rhs) {
    return x == rhs.x && y == rhs.y;
}
```

Operator Overloading

```
struct Point {
    int x, y;
};

bool operator==(const Point& lhs, const Point& rhs) {
    return lhs.x == rhs.x && lhs.y == rhs.y;
}
```

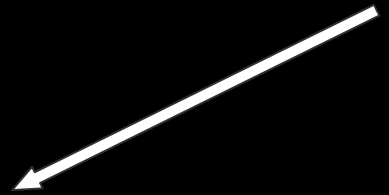
Operator Overloading

```
struct Point {
    int x, y;
};

bool operator==(const Point& lhs, const Point& rhs) {
    return lhs.x == rhs.x && lhs.y == rhs.y;
}
```

Operator Overloading

Non member function. LHS is explicit first parameter.



```
struct Point {
    int x, y;
};

bool operator==(const Point& lhs, const Point& rhs) {
    return lhs.x == rhs.x && lhs.y == rhs.y;
}
```

Operator Overloading

```
struct Point {
    int x, y;
};

bool operator==(const Point& lhs, const Point& rhs) {
    return lhs.x == rhs.x && lhs.y == rhs.y;
}
```

Operator Overloading

Two ways to overload operators:

- Member functions
- Non-member functions

Member Functions

Just add a function named operator@ to your class

```
bool operator==(const HashSet& rhs) const;
Set operator+(const Set& rhs) const;
Set& operator+=(const ValueType& value);
```

For binary operators, accept the right hand side as an argument.

I usually name mine rhs.

Non-member Functions

Add a function named operator@ outside your class.

Have it take all its operands.

```
bool operator==(const Point& lhs, const Point& rhs) {
    return lhs.x == rhs.x && lhs.y == rhs.y;
}
```

Operator Overloading

Some examples:

OperatorOverload
(OpOverload.`pro`)

Non-member Operators

The standard library tends to prefer this way

Allows for the lhs to be a non class type

If it needs access to internal private members, declare it in the class with the friend keyword!

Operator Overloading

Let’s go back for a second…

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

Operator Overloading

Let’s go back for a second…

Anything curious here?

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

Operator Overloading

Let’s go back for a second…

Anything curious here?

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

Operator Overloading

Let’s go back for a second…

Anything curious here?

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

Operator Overloading

Some experimentation:

FunctionOperator
(FuncitonOp.`pro`)

Functors

Functors

Classes which define the `()` operator.

Why is this useful?

- Can have state
- Customizable through constructor

Very useful for algorithms!

Functors

Remember this problem?

```
std::vector<Student> StudentDatabase::studentsInYear(std::string yearToFind) {
    vector<Student> ret;

    // Can't use pred function because we need to somehow give it yearToFind...
    // std::copy_if(db.begin(), db.end(),  std::back_inserter(ret), pred);

    // We'll settle for a for-loop
    for(auto student : db) {
        if(student.classLevel() == yearToFind) {
            ret.push_back(student);
        }
    }
    return ret;
```

Operator Overloading

Using functors:

StudentClass

(StudentClass.`pro`)

Functors

Functors let us make customizable functions!

We can pass useful information to their constructor that was not known at compile time.

But...

Kind of a Pain™

Functors

Functors let us make customizable functions!

We can pass useful information to their constructor that was not known at compile time.

But...

Kind of a Pain™

C++ has a solution!

Functors

Functors let us make customizable functions!

We can pass useful information to their constructor that was not known at compile time.

But...

Kind of a Pain™

C++11 has a solution!

Lambdas

Lambdas

A C++11 feature that lets you make functions on the fly.

```
[capture-list](params) -> ReturnType {
    // code
};
```

Lambdas

Best learnt by example:

```
auto print_int =  [](int x) {
                      cout << x << endl;
                  };


print_int(5)// outputs 5 to console
```

Lambdas

Best learnt by example:

```
vector<int> v{3, 1, 4, 1, 5};

std::sort(v.begin(), v.end(),

        [](int i, int j) -> bool { return i > j;});


// sorts vector in decreasing order
```

Lambdas

Most modern languages have lambdas in some form!

```
lessThanPy = lambda x, y: x < y

const lessThanJs = (x, y) => {return x < y;};

Comparator lessThanJava = (x, y) -> return x < y;

auto lessThanCpp = [](int i, int j) -> bool { return i < j;});
```

Questions

Lambda Captures

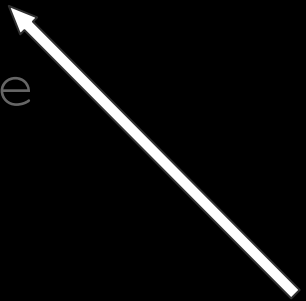
A C++11 feature that lets you make functions on the fly.

```
[capture-list](params) -> ReturnType {
    // code
};
```

Lambda Captures

A C++11 feature that lets you make functions on the fly.

```
[capture-list](params) -> ReturnType {
    // code
};
```



What is this for?

Lambda Captures

Remember this problem?

```
std::vector<Student> StudentDatabase::studentsInYear(std::string yearToFind) {
    vector<Student> ret;

    // Can't use pred function because we need to somehow give it yearToFind...
    // std::copy_if(db.begin(), db.end(),  std::back_inserter(ret), pred);

    // We'll settle for a for-loop
    for(auto student : db) {
        if(student.classLevel() == yearToFind) {
            ret.push_back(student);
        }
    }
    return ret;
```

Lambda Captures

You can capture available variables to use in the lambda

```
[byValue, &byReference]
```

You can also capture all currently available variables:

```
[=] // By value

[&] // By reference
```

This will only capture the ones used inside the function.

How Does This Work?

How Lambdas Work?

```
[capture-list](params) ->
ReturnType {

    // code

};
```

```
class SomeName {
public:
    SomeName(capture-list) {
        // set each private member to
        // thing in capture list
    }

    ReturnType operator()(params) {
        // code
    }

private:
    // create private member for each
    // thing in capture-list
};
```

Next Time

ParticleSimulator