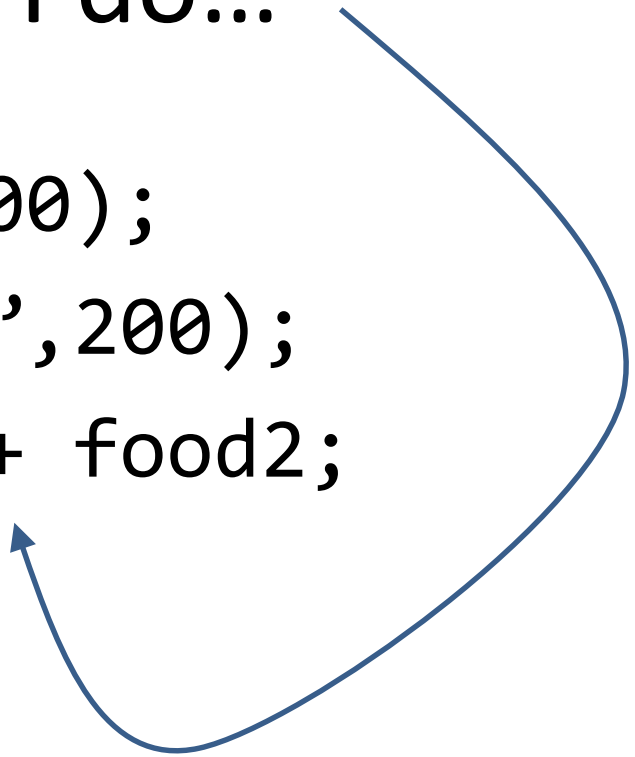
Overloading Operators

How can I do…

```
Food food1(“Rice”,100);
Food food2(“Chicken”,200);
Food food3 = food1 + food2;
```



Before Explaining

```
class Food {
private:
  string _name;
  int _cal;
public:
  Food() { _name = ""; _cal = 0; };
  Food(string, int);

  Food mix_food(Food& f2) {
    return Food(_name + f2._name, _cal +f2._cal);
  }

}
```

When We Use mixFood()

```
Food Food::mix_food(Food& f2) {
   return Food(_name + f2._name, _cal +f2._cal);
 }
```

```
Food food1("Fish",200);
Food food2("Icecream", 300);

food1.mix_food(food2);
```

- Will food1 be changed?
- Will food2 be changed?

When We Use mixFood()

```
Food Food::mix_food(Food& f2) {
   return Food(_name + f2._name, _cal +f2._cal);
 }
```

```
Food food1("Fish",200);
Food food2("Icecream", 300);

Food food3 = food1.mix_food(food2);
```

- food1 and food2 will NOT be changed
- The new combined value is copied to food3

Our mix_food Method

```
class Food {
private:
  string _name;
  int _cal;
public:
  Food() { _name = ""; _cal = 0; };
  Food(string, int);

  Food mix_food (Food& f2) {
    return Food(_name + f2._name, _cal +f2._cal);
  }

}
```

Replace the Function Name Only

```
class Food {
private:
  string _name;
  int _cal;
public:
  Food() { _name = ""; _cal = 0; };
  Food(string, int);

  Food operator+(Food& f2) {
    return Food(_name + f2._name, _cal +f2._cal);
  }

}
```

When We Use mixFood()

```
Food Food::operator+(Food& f2) {
   return Food(_name + f2._name, _cal +f2._cal);
 }
```

```
Food food1("Fish",200);
Food food2("Icecream", 300);

Food food3 = food1.operator+(food2);
```

- Everything the same if we just change "mix_food" to "operator+"
- So "operator+" is a function/method

When We Use mixFood()

```
Food Food::operator+(Food& f2) {
   return Food(_name + f2._name, _cal +f2._cal);
 }
```

```
Food food1("Fish",200);
Food food2("Icecream", 300);

Food food3 = food1.operator+(food2);
```

- But it's very clumsy, if I want to combine a few food:

```
food4 = food3.operator+(food1.operator+(food2));
```

When We Use mixFood()

```
Food Food::operator+(Food& f2) {
   return Food(_name + f2._name, _cal +f2._cal);
 }
```

```
Food food1("Fish",200);
Food food2("Icecream", 300);

Food food3 = food1.operator+(food2);
Food food3 = food1 + food2;
```

- They are equivalent!!
- So I can write (the one in the prev. slide)

```
food4 = food3 + food1 + food2;
```

```
Food food3 = food1.operator+(food2);
```

```
Food food3 = food1 + food2;
```

Operator +

```
Food food3 = food1.operator+(food2);
Food food3 = food1 + food2;
```

- The two lines are equivalent
  - The first one is clumsy
- Back to our normal integer addition, we can view it as:

```
int i,j,k;
i = j.operator+(k); // same as i = j + k;
```

Back to Our Discrete Math

- if we say 1 + 2
- “+” is a function that takes in two arguments
  - 1 and 2
  - And MAP it to 3
- In the “cheem” language of math:

$$\text{“+”} : R \times R \rightarrow R$$

Back to Our Discrete Math

- if we say 1 + 2
- “+” is a function that takes in two arguments
  - 1 and 2
  - And MAP it to 3 (The return value)
- In which, you can redefine the “MAP” to anything else, e.g.

  $+ : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$

Back to Our Discrete Math

- In which, you can redefine the “MAP”

$$+ : \{0,1\} \times \{0,1\} \rightarrow \{0,1\}$$

- Such that

$$0 + 0 = 0$$

$$0 + 1 = 1$$

$$1 + 0 = 1$$

$$1 + 1 = 0$$

- Group/Ring in Algebraic Structure

In Alan's PhD Thesis N Years Ago

- I defined arithmetic operations for spheres in any dimension

3.1.1 Sphere Arithmetic

Given two spheres, b_i and b_j, and a real number c, addition , scalar multiplication, and power of spheres are defined as follows:

$$\begin{aligned}(z_i, w_i) + (z_j, w_j) &= (z_i + z_j, w_i + w_j + 2\langle z_i, z_j\rangle)\\ c \cdot (z_i, w_i) &= (c \cdot z_i, c \cdot (w_i - (1-c)\|z_i\|^2))\\ (z_i, w_i)^c &= (z_i, c \cdot w_i).\end{aligned}$$

The first two equations are the standard operations on vectors in $\mathbb{R}^{d+1}$ under the paraboloid lifting map $(z_i, w_i) \rightarrow (z_i, \|z_i\|^2 - w_i)$.

- Meaning, what is c = a + b, if a and b are spheres?

We can Overload Other Operators

- We can print “1” by:

```
cout << 1;
```

- In fact
  - cout is an instance of a class ostream (stands for “output stream”)
- So the line above is equivalent to

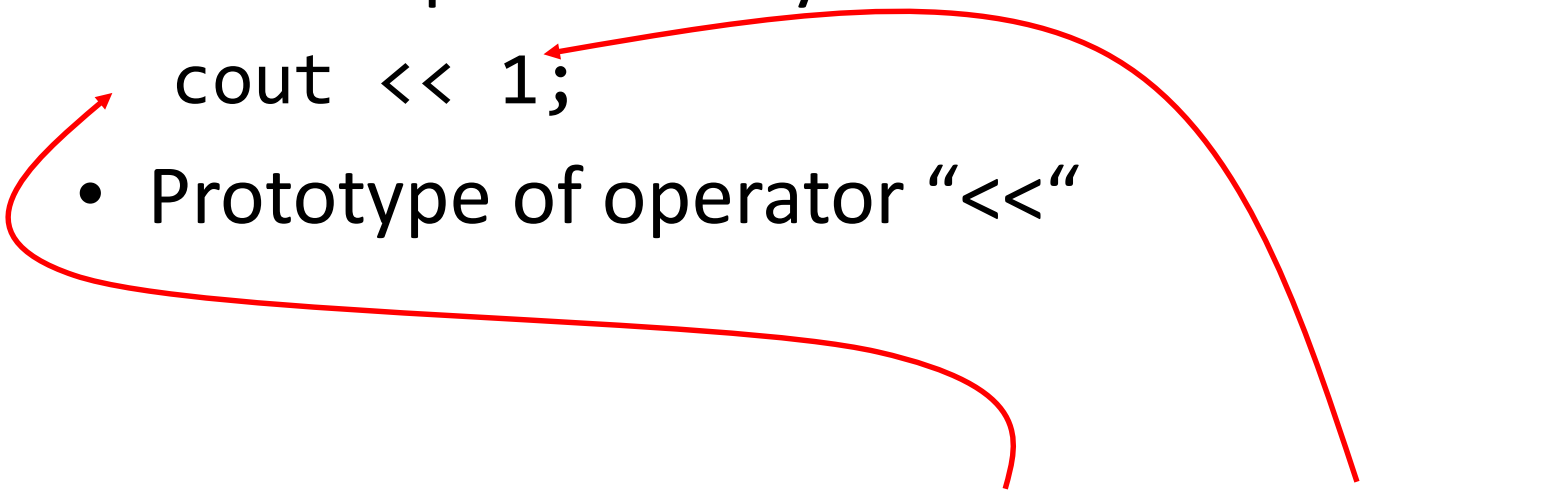
```
cout.operator<<(1);
```

We can Overload Other Operators

- We can print “1” by:

```
cout << 1;
```

- Prototype of operator “<<“



```
ostream& operator<<(ostream& os, const int& i)
{
    // the code that will print out i in the console
}
```

So, What if …

- If we want to print food1 (an instance of the class Food) by cout like this:

```
cout << food1;
```

```
ostream& operator<<(ostream& os, const Food& f)
{
    // the code that will print out f in the console
}
```

"cout <<" for the class Food

- Compare this to

```
ostream& operator<<(ostream& os, const int& i)
{
    // the code that will print out i in the console
}
```

"cout <<" for integers