Operator Overloading

Adding Objects?

```
class Money
{
  public:
    Money(int amount);
  private:
    int amt;
};

Money m1(500);
Money m2(100);
Money total = m1 + m2;
```

- It makes since to be able to add two Money objects
- But how do you tell the compiler how to do it?

Adding Objects?

```
class Money
{
  public:
    Money(int amount);
    Money Add()(const Money &money) const;
  private:
    int amt;
};

Money Money::Add(const Money &money) const
{
  Money ret(amt + money.amt);
  return ret;
}

Money m1(500);
Money m2(100);
Money total = m1.Add(m2);
```

- You could just add an Add() function....

Operator Overloading

```
class Money
{
  public:
    Money(int amount);
    Money operator+(const Money &money) const;
  private:
    int amt;
};

Money Money::operator+(const Money &money) const
{
  Money ret(amt + money.amt);
  return ret;
}

Money m1(500);
Money m2(100);
Money total = m1 + m2;
```

- Or you could add a + operator

Operator Overloading

```
class Money
{
  public:
    Money(int amount);
    Money operator+=(const Money &money);
  private:
    int amt;
};

Money Money::operator+=(const Money &money)
{
  amt += money.Amount();
  return amt;
}

Money m1(500);
Money m2(100);
Money m2 += m1;
```

- We can also make mutable operators like +=

Operator Overloading

```
class Money
{
  public:
    Money(int amount);
    int operator[](const Money &money) const;
  private:
    int amt;
};

int Money::operator[](int digit) const
{
  int ret = amt;

  for(int i = 0; i < digit; i++)
    ret /= 10;
  ret %= 10;

  return ret;
}

Money m1(500);
int digit = m1[2];
```

- We can even make things that are unintuitive
  - Not recommended in most cases
- This extracts the ith digit

Classes of Operators

- You can redefine any operator in any class
- Operators keep the same precedence and number of parameters
  - This means if you do something unexpected, like perform
    - Addition with *
    - Multiplication with +
  - The result will be wrong (in a mathematical sense) when nesting multiple operators in an expression
- You can go here for a list of valid operators
  - http://www.cplusplus.com/doc/tutorial/operators/