Some concepts …

- Constructors returning objects
- The `this` pointer
- References and returning references
- Destructor and copy constructor
- Overloading assignment operator

Constructors Returning Objects

- Constructor seems to be a "void" function …
  - It CAN return a value …

- Recall return statement in overloaded operator "+" for `Fraction` type:

```
return Fraction(numer, denom);
```

  - So constructor actually "returns" an object
  - Called an "anonymous object"

Recall `new` Operator …

- Creates new dynamic variable
- Returns pointer to the new variable
- For simple type variable:

```
int* p;
p = new int(17); // *p is 17 now
```

- For class type:
  - Constructor is called for new object
  - Can invoke different constructor with different arguments:

```
MyClass* mcPtr;
mcPtr = new MyClass(32.0, 17);
```

The -> operator

- Shorthand notation -- combines *dereferencing operator* and *dot operator*
- Specifies member of class "pointed to“ by given pointer
- Example:

```
Fraction* p = new Fraction;
p->simplify();
        // Equivalent to: (*p).simplify();
```

The `this` Pointer

- Member function definitions might need to refer to calling object
- Use predefined `this` pointer
  - Automatically points to calling object
- Examples:

```
Point::Point(int x, int y){
    this -> x = x;
    this -> y = y;
}
```

References

- Reference:
  - Name of a storage location
  - Similar to "pointer"
- Example of *stand alone* reference:

```
int robert;

int& bob = robert;
```

  - *bob* is reference to storage location for *robert*
  - Changes made to *bob* will affect *robert*

References Usage

- Seemingly dangerous
- Often useful in:
  - Pass-by-reference
    - as discussed in CMPT130
  - Returning a reference
    - Allows operator overload implementations to be written more naturally
    - Think of as returning an "alias" to a variable

Returning Reference

- Syntax:

```
int& foo( … );
```

  - `int&` and `int` are different return types
  - Must match in function declaration and header

- Returned item must "have" a reference
  - Like a variable of that type
  - Cannot be expression like "x+5"
    - Has no place in memory to "refer to”
- Cannot return a local variable by reference! Why?
- Major use:
  - Certain overloaded operators (such as `operator++`)

Destructor

- Opposite of constructor
- a member function
- automatically called when object is out-of-scope
- performs delete clean-up duties
- default version only removes ordinary variables, not dynamic variables
- A class has only one destructor with no arguments
- The name of destructor is distinguished from the default constructor by the tilde symbol **~**
- Example

```
~MyClass();
```

Copy Constructor

- Used for making a copy of an object
- A copy constructor is a constructor with one parameter of the same type as the class
  - The parameter is a pass-by-reference parameter
  - The parameter is usually a constant parameter
  - The constructor should create a complete, independent copy of its argument
- For example, how to define the copy constructor for the `Fraction` class?

Copy Constructor

- Automatically called when:
  1. Class object *declared and initialized* (at the same time) to other object
  2. When function returns class type object by value
  3. When argument of class type is "plugged in" as actual argument to call-by-value parameter
- Default copy constructor
  - performs member-wise copy
- For classes having pointer type member variables, write own copy constructor! (more in detail later…)

Assignment Operator =

- Must be overloaded as member operator
- Default assignment operator:
  - **Member-wise copy**
  - Member variables of the given object --> corresponding member variables of this object
- Example?
- Default version OK for simple classes, but:
  For classes having pointer type member variables, write own assignment operator! (more in detail later…)

Exercise – when are these functions invoked ? …