Day 32

- exercise 31 review
- day 32 recap questions
- exercise 32

Announcements/reminders

- HW6 due this evening by 11pm
  - written assignments, no late submissions
- HW7 due Wednesday 4/20 by 11pm
- Final project
  - If possible, please register your team by 11am on Friday 4/15
  - See Piazza post 575 for link to registration form: also use this post if you are looking for team member(s)
  - Final project will be posted on Friday: full intro on Monday 4/18

Exercise 31

template<typename T>
class my_set {
⋮
my_set(const my_set<T> &o);

Converting int_set to (generic) my_set<T>

This is essentially just a syntactic change.

Put template<typename T> before class definition (in my_set.h) and member function implementations (in my_set.inc).

Substitutions:
int_node --> my_node<T>
int_node --> my_node (names of constructor and destructor functions)
int_set --> my_set<T>
int_set --> my_set (names of constructor and destructor functions)
int --> T (except for size field and get_size() member function)

Exercise 31

The output stream insertion operator needs to use its own type parameter (since it's not a member of my_set<T>)

```
// in my_set.h
  template<typename U>
  friend std::ostream& operator<<(std::ostream& os, const my_set<U>& s);

// in my_set.inc
template<typename U>
std::ostream& operator<<(std::ostream& os, const my_set<U>& s){
  my_node<U> *n = s.head;
  // ...code to print member values omitted...
  return os;
}
```

Exercise 31

Assignment operator

```
// in my_set.h
my_set<T>& operator=(const my_set<T>& other);

// in my_set.cpp
template<typename T>
my_set<T>& my_set<T>::operator=(const my_set<T>& other) {
  if (this != &other) {
    my_node<T> *n = other.head;
    while (n != nullptr) {
      add(n->get_data());
      n = n->get_next();
    }
  }
  return *this;
}
```

Day 32 recap questions

1. Do derived classes inherit constructors?
2. What does protected imply for a class field?
3. What is polymorphism?
4. What is the purpose of the virtual keyword?
5. Can a child class have multiple parents?

1. Do derived classes inherit constructors?

No. Each derived class must define its own constructors. These will call one of the base class's constructors in its initializer list.

```
// base class
class Point2D {
private:
  double x, y;

public:
  Point2D() : x(0.0), y(0.0) { }
  Point2D(double x, double y)
    : x(x), y(y) { }

  double get_x() const { return x; }
  double get_y() const { return y; }
};
```

```
// derived class
class Point3D : public Point2D {
private:
  double z;

public:
  Point3D() : Point2D(), z(0.0) { }
  Point3D(double x, double y, double z)
    : Point2D(x, y)
    , z(z) { }

  double get_z() const { return z; }
};
```

2. What does protected imply for a class field?

A protected member may be directly accessed by code in derived classes, but may note be accessed by code in "unrelated" classes or functions.

OPINION

Note: it is never really necessary to make a member protected. Derived classes can (and should) use getter and setter functions to access private data values in the base class.

3. What is polymorphism?

Polymorphism is the phenomenon that anywhere in a program that there is either a pointer to a base class type or a reference to a base class type, that pointer or reference could really refer to an object that is an instance of a class derived from the base class type.

E.g.:

```
public class Dog : public Animal { ... }
public class Cat : public Animal { ... }
public class Owl : public Animal { ... }

void do_stuff(Animal &a) {
  // The reference a could refer to a Dog object, Cat object, Owl object,
  // or an instance of any class deriving from Animal
}
```

4. What is the purpose of the virtual keyword?

The virtual keyword marks a member function that can be overridden by a derived class with different behavior. A base class will *usually* have at least one virtual member function. The idea is that virtual member functions in the base class define *common operations* which can be implemented by derived classes with *varying behavior*.

```
class Animal {
public:
  virtual void vocalize() { cout << "?"; }
  // ...
};
```

```
class Dog : public Animal{
public:
  virtual void vocalize() {
     cout << "woof\n"; }
  // ...
};

class Cat : public Animal {
  virtual void vocalize() {
    cout << "meow\n"; }
  // ...
};
```

```
void stuff(Animal &a) {
  a.vocalize();
}

int main() {
  Dog leo;
  Cat ingo;

  stuff(leo);   // prints "woof"
  stuff(ingo); // prints "meow"
}
```

dynamic dispatch

5. Can a child class have multiple parents?

Yes. However, this is a feature that is not used very widely.

One example: the iostream type inherits from both istream and ostream.