601.220 Intermediate Programming

Spring 2023, Day 32 (April 12th)

Today's agenda

- Review exercise 31
- Day 32 recap questions
- Exercise 32

Reminders/Announcements

- HW7 is due Friday, April 14th
- Final project team registrations should have been submitted by now
    - Will be released on Friday, and we'll go over it in class on Monday

Exercise 31 review

Converting `int_set` to (generic) `my_set<T>`: this is essentially just a syntactic change.

Put `template<typename T>` before the 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 review

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

Exercise 31 review

Assignment operator

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

// in my_set.inc
  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.

```
// example 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, constructors

```
// 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; }
};
```

Picture of `Point2D` and `Point3D` objects

2. What does `protected` imply for a class field?

A `protected` member may be directly accessed by code in derived classes, but may not be accessed by code in "unrelated" classes or functions.

Opinion: It is never really necessary to make a member function protected. Derived classes can (and should) use public 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 could refer to a Dog, Cat, or 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. This allows the derived class to provide its own behavior for that member function.

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*.

Example base class with a virtual member function

```
// base class with a virtual member function representing
// a common operation
class Animal {
public:
  virtual void vocalize() { cout << "?\n"; }
  // ...
};
```

Example derived classes overriding a virtual member function

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

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

Polymorphism!

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

int main() {
  Dog leo;
  Cat ingo;

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

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`.

Since `stringstream` inherits from `iostream`, this explains why you can both read data from a `stringstream` and also write data to a `stringstream`.

Exercise 32

- Practice with examples of classes using inheritance and `virtual` member functions
- Talk to us if you have questions!

Notes

Notes

Notes

Notes

Notes