CS 162
Intro to CS II

Classes: Finish Polymorphism
Chap. 15.2
Quiz #7

• What is the difference in a friend function and inherited function?
  – Inheritance allows you to use public and protected members directly
  – Friends allow you to use private members directly

• What is the difference between redefining a function and polymorphism?
  – Redefine usually has different parameters
  – Polymorphism allows inherited functions to morph itself depending on object (using same function call)
Redefine vs. Polymorphism

```cpp
#include <iostream>
#include <cstring>
#include <cstdlib>
using namespace std;

class employee {
    public:
        employee() { }
        employee(int y) { years = y; }
        int get_vacation_days() { //This always calls employee w/o virtual
            return 10 + get_seniority_bonus();
        }
    }

    //When the virtual is missing it is redefined if in a child too
    int get_seniority_bonus() { return 2 * years; }

friend void test(employee &s);

private:
    int years;
};
class secretary : public employee{
    public:
        secretary(int y) : employee(y) { }
        int get_seniority_bonus() { return 0; } //Secretary doesn't get bonus
        void take_dictation(string txt) {
            cout << "Taking Dictation: " + txt << endl;
        }
};
```
Extending Types/Polymorphism

• Can upcast, but not down
  Parent p; Child c;
  p = c; //what will the polymorph function call now?

• What if we made pointers?
  Parent *p; Child *c = new Child;
  p = c;
```cpp
19 };
20 class secretary : public employee{
21     public:
22         secretary(int y) : employee(y) {}  
23         int get_seniority_bonus() { return 0; } //Secretary doesn't get bonus
24         void take_dictation(string txt) {
25             cout << "Taking Dictation: " + txt << endl;
26         }
27 }
28 //We can send a secretary or employee and get the polymorphism we need
29 void test(employee *s){
30     cout << s->get_vacation_days();
31 }
32 int main() {
33     employee e(5); //Employee and Secretary have same # years...
34     secretary s(5);
35     //We copy secretary into an employee, but it is an employee
36     e = s; //Breaks polymorphism
37     test(&s); //We can uphold polymorphism as long as we point to secretary
38     cout << "Employee vacation days: " << e.get_vacation_days() << endl;
39     cout << s.get_vacation_days() << endl; //Poor secretary, no love:
40     return 0;
41 }```

Breaking Polymorphism with Upcast

```
Upholding Polymorphism with Upcast

```cpp
19  }
20 class secretary : public employee{
21     public:
22     secretary(int y) : employee(y) {} 
23     int get_seniority_bonus() { return 0; } //Secretary doesn't get bonus
24     void take_dictation(string txt) {
25         cout << "Taking Dictation: " + txt << endl;
26     }
27 }
28 //We can send a secretary or employee and get the polymorphism we need
29 void test(employee *s){
30         cout << s->get_vacation_days();
31 }
32
33 int main() {
34     employee *e; //Employee and Secretary have same # years...
35     secretary *s = new secretary(3);
36
37     //We make e and s point to the same place which is a secretary
38     e = s;
39     test(s);   //Why do you care about this?
40     cout << "Employee vacation days: " << e->get_vacation_days() << endl;
41     cout << s->get_vacation_days() << endl; //Poor secretary, no love:(
42     return 0;
43 }
44,1
```
Make Destructors Virtual

• What does this do if destructor isn’t virtual?
  Parent *p = new Child;
  ...
  delete p;

• Example:
  class parent {
    public:
      parent() { //Have a constructor
        shared_ptr = new int;
      }
      virtual ~parent() { //Have a destructor
        delete shared_ptr;
      }
    private:
      int *shared_ptr;
  };
Pure Virtual

• Definition
  – Don’t need to define function in base/parent class
  – Why?
• Abstract class
  – One or more pure virtual functions in class
class figure {
public:
  figure();
  ~figure();
  virtual void draw() = 0;
  center() { ... draw(); ...}
};

class circle : public figure {
public:
  circle();
  ~circle();
  void draw() { ... }
};

class rectangle : public figure {
public:
  rectangle();
  ~rectangle();
  void draw() { ... }
};