CS 162
Intro to CS II
Classes: Finish Inheritance
Chap. 14.2
```cpp
int parent::get_shared_var() {
    return shared_var;
}

void parent::print_mssg() {
    cout << "I'm parent!" << endl;
}

//child class implementation
child::child() : parent() { //Need to call inherited constructor first
    unique_var = 0;
}

void child::print_mssg() {
    cout << "I'm child!" << shared_var << endl; //This takes precedence over parent
    shared_var = 0; //You can directly access protected inside child class
}

int main() {
    parent p; //Create a parent obj
    child c; //Create a child obj that inherits parent

    p.print_mssg(); //Parent's print_mssg called
    ((parent)c).print_mssg(); //We can call child or parent print_mssg

    //c.shared_var=0; //You cannot access protected outside child class
    return 0;
}
```

When do we create classes?

• “Is A” Relationship
• “Has A” Relationship
What is not inherited?

- Constructors
- Private Members
- Destructors
- Copy Constructor (by default content)
- Overloaded Assignment Ops
Destructors Example

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

class child : public parent {
    public:
        child();  //This constructor needs to call parent() constructor
        void print_mssg();  //Redefine or Override inherited function
    private:
        int unique_var;
};

• What is wrong with this?
Destructors

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

class parent {
  public:
    parent();  // Default constructor
    // Copy constructor
    parent(const parent &p) {
      shared_var = new int;
      *shared_var = *(p.shared_var);
    }
    // Destructor
    ~parent() {
      delete shared_var;
    }
    void print_mssg();
    int *get_shared_var();  // Return pointer to integer, not value
  protected:  // This makes shared_var directly available to child
    int *shared_var;
};

class child : public parent {
  public:
    child();  // This constructor needs to call parent() constructor
    ~child(){}  // Having a destructor assures you call the parent
    void print_mssg();  // Redefine or Override inherited funtion
  private:
    int unique_var;
};
```

parent::parent() {
"inheritance2.cpp" [dos] 62L, 17:1C written 21,0-1 Top
Destructors

```cpp
30 parent::parent() {
31     shared_var = new int;
32     *shared_var = 4;
33 }
34 int * parent::get_shared_var() {
35     return shared_var;
36 }
37 void parent::print_msg() {
38     cout << "I'm parent!" << endl;
39 }
40 //child class implementation
41 child::child() : parent() { //Need to call inherited constructor first
42     unique_var = 0;
43 }
44 void child::print_msg() {
45     child c;
46     cout << "I'm child!" << *shared_var << endl; //This takes precedence over parent
47     *shared_var = 0; //You can directly access protected inside child class
48     cout << c.shared_var << endl; //Print address of where int is stored on heap
49 }
50 void print_m() {
51     child c;
52     cout << c.get_shared_var() << endl;
53 }
54 int main() {
55     child c; //Create a child obj that inherits parent
56     c.print_msg(); //We can call child or parent print_msg
57     c.print_msg(); //We can call child or parent print_msg
58     print_m();
59     print_m();
```

Copy Constructor Example

class parent {
  public:
    parent() { //Have a constructor
      shared_ptr = new int;
      *shared_ptr = 4;
    }
    parent(const parent &p) { //Copy Constructor
      shared_ptr = new int;  //Why do this???
      *shared_ptr = *(p.shared_ptr);
    }
    ~parent() { //Have a destructor
      delete shared_ptr;
    }
  private:
    int *shared_ptr;
};
Protected vs. Private Inheritance

• Public, protected, and private members
  – Who has access to these?

• Public, protected, and private inheritance
  – What does this mean?
    class child : public parent { ... }
    class child : protected parent { ... }
    class child : private parent { ... }
Protected vs. Private Inheritance

- Protected
  - Public members are protected in child

- Private
  - All members are private to child

- Either case:
  - Cannot be used as parent, i.e. breaks “is a”