More Inheritance

- Book: 9.9, 10.4
- More Inheritance
- Polymorphism
- Virtual Functions

Destructors

- The opposite of constructors

Constructor/Destructor Example

```
class Employee
{
  public:
    Employee (string name = "",
              string ssn = ""); // cout << ctor: Name

    ~Employee() { cout << “dtor ” << mName << “\n”; }

  private:
    string mName;
    string mSSN;
};
```

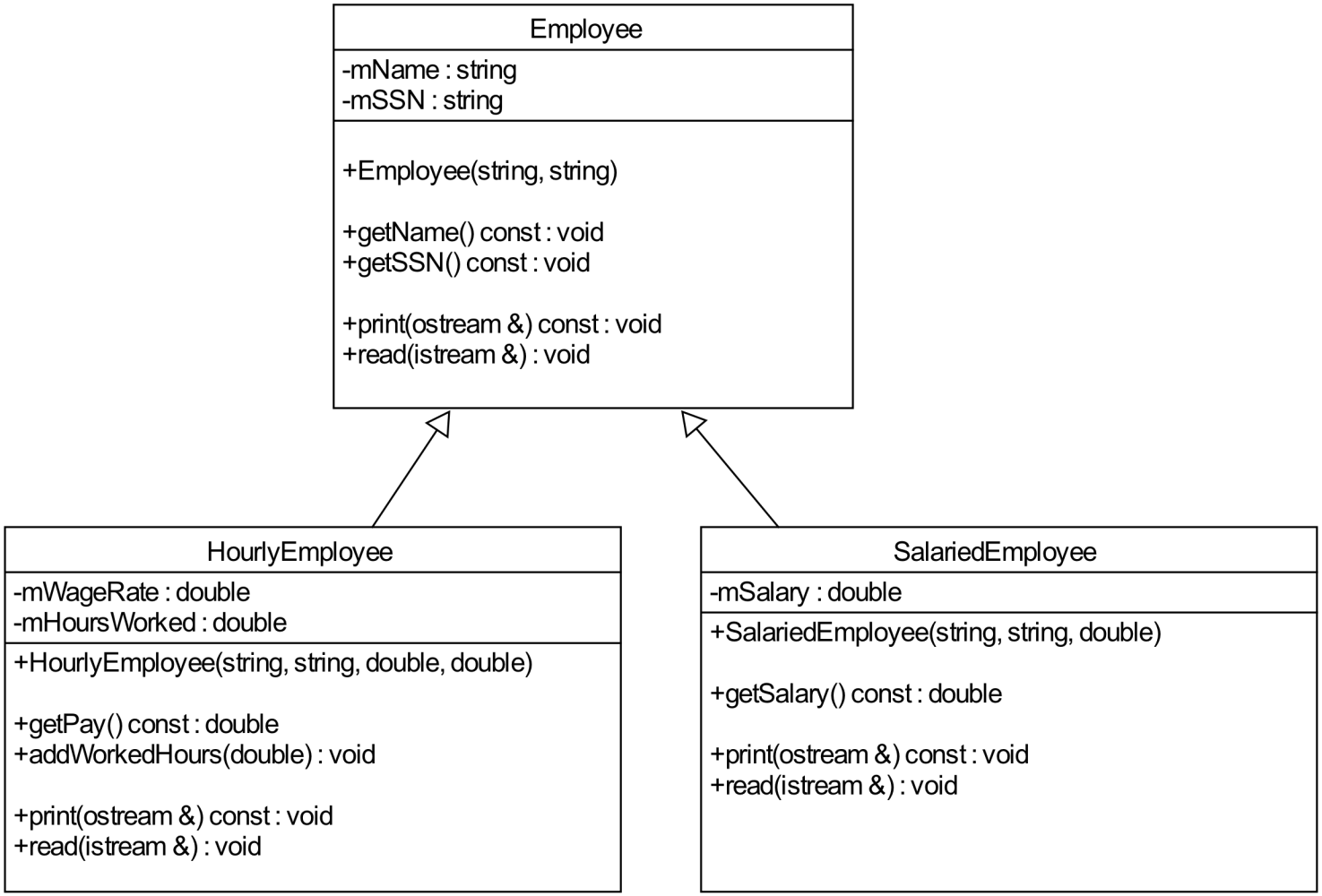
What is the output?

```
void funct ();

int main ()
{
  Employee cTest1 ("Doug");
  funct ();
  Employee cTest3 ("Shereen");

  return EXIT_SUCCESS;
}

void funct ()
{
 Employee cTest2 ("Chadd");
}
```

Polymorphism

- Code is said to be polymorphic if executing the code with different types of data (objects) produces different behavior
- Program in the general, rather than program in the specific
- *Virtual functions* make polymorphism possible

UML


Employee
-mName : string
-mSSN : string
+Employee(string, string)
+getName() const : void
+getSSN() const : void
+print(ostream &) const : void
+read(istream &) : void
HourlyEmployee
-mWageRate : double
-mHoursWorked : double
+HourlyEmployee(string, string, double, double)
+getPay() const : double
+addWorkedHours(double) : void
+print(ostream &) const : void
+read(istream &) : void
SalariedEmployee
-mSalary : double
+SalariedEmployee(string, string, double)
+getSalary() const : double
+print(ostream &) const : void
+read(istream &) : void

Motivation

```
// base class pointer
Employee *pcEmp = nullptr;

char choice;

cin>> choice;

switch(choice)
{
  case 'H':

    pcEmp = new HourlyEmployee();
    break;

  case 'S':
    pcEmp = new SalariedEmployee();
    break;
}

pcEmp->read(cin);  // which read() is called?
                   // what do we want to have happen?
```

Base Pointers

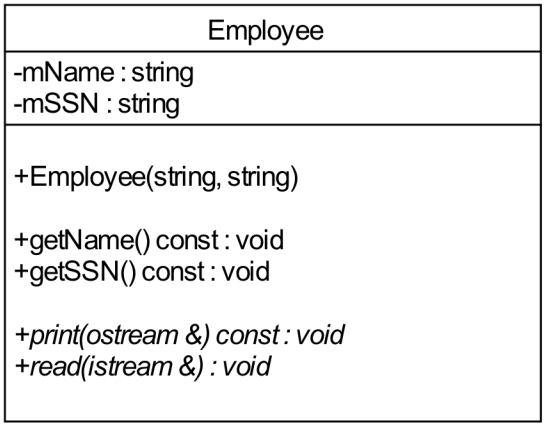
- A base class pointer can point at a child class object
- Calling a function on the base class pointer calls a function in the base class
- To cause a function in the child class to be called, mark the function as virtual

Virtual Functions

- You can tell the compiler to select the more specialized version of a member function by declaring the member function to be a virtual function
- Declare a virtual function by prefixing its declaration with the word virtual

Virtual Function Example

```
class Employee
{
  public:
    Employee (string name = "",
              string ssn = "");
    string getName () const;
    string getSSN () const;
    virtual void print (ostream &rcOut) const;

    virtual bool read(istream &rcIn);

  private:
    string mName;
    string mSSN;
};
```



Virtual Function Example

```
class HourlyEmployee : public Employee
{
  public:
    HourlyEmployee ();
    HourlyEmployee (string name, string ssn,
                    double hourlyRate, double hoursWorked);

    double getPay() const;
    void addWorkedHours(double hours);

    virtual void print(ostream &rcOut) const;

    virtual bool read(istream &rcIn);


private:
    double mWageRate;
    double mHoursWorked;
};
```

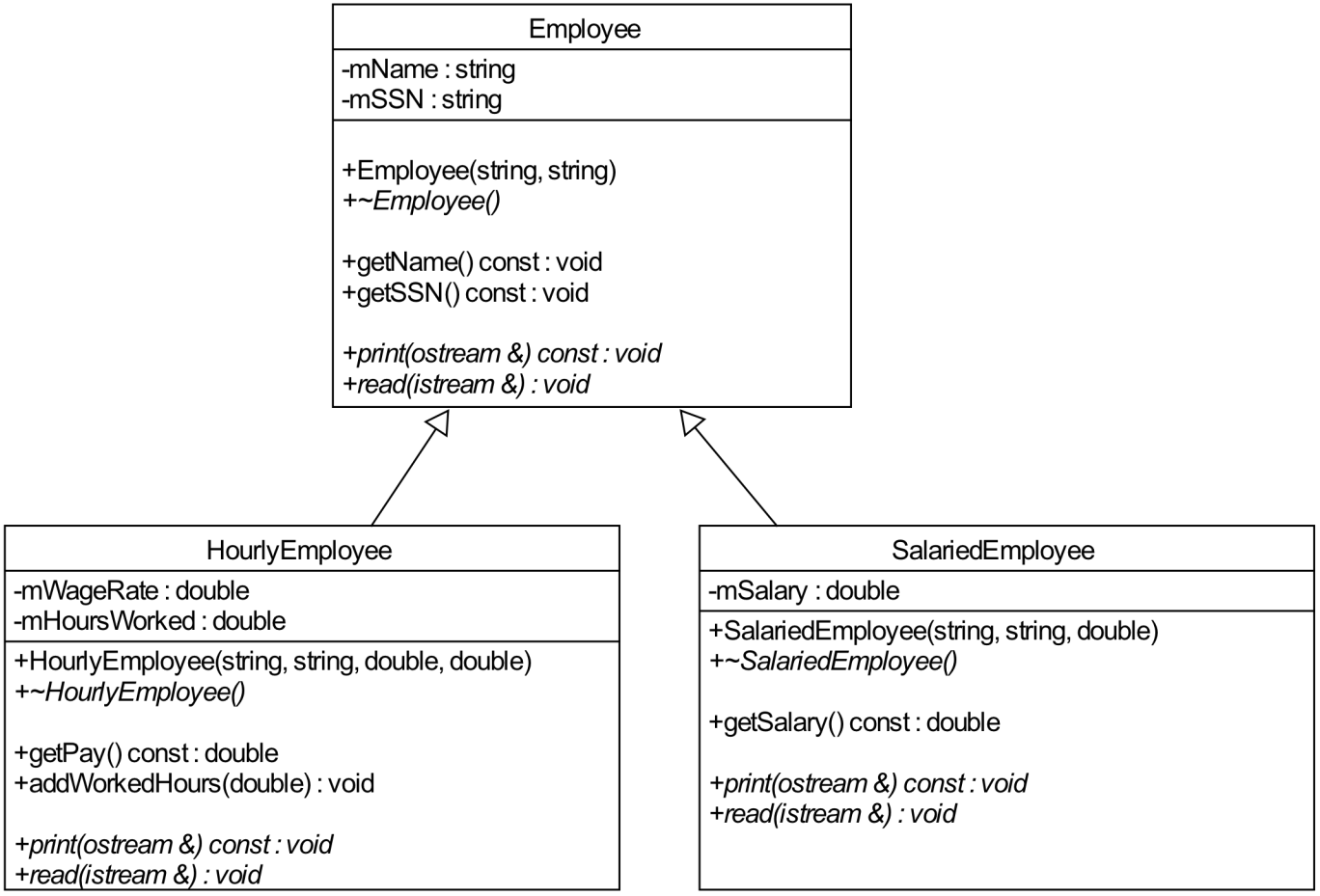
Virtual Function Example

```
bool HourlyEmployee::read(istream &rcIn) {

   if( Employee::read(rcIn) &&
       rcIn >> mWageRate >> mHoursWorked )
   {
     return true;
   }

   return false;
}
```

Virtual Function Example

```
void displayEmployee(const Employee &rcEmp) {

   rcEmp.print(cout);
}

// You can pass a child class object
// by reference as a parent object

displayEmployee(*pcEmp);
```

UML


Employee
-mName : string
-mSSN : string
+Employee(string, string)
+~Employee()
+getName() const : void
+getSSN() const : void
+print(ostream &) const : void
+read(istream &) : void
HourlyEmployee
-mWageRate : double
-mHoursWorked : double
+HourlyEmployee(string, string, double, double)
+~HourlyEmployee()
+getPay() const : double
+addWorkedHours(double) : void
+print(ostream &) const : void
+read(istream &) : void
SalariedEmployee
-mSalary : double
+SalariedEmployee(string, string, double)
+~SalariedEmployee()
+getSalary() const : double
+print(ostream &) const : void
+read(istream &) : void

Virtual Destructor

- Any potential base class should have a virtual destructor
- Why? The compiler performs static binding on any destructor not declared virtual

Which dtors get called?

```
Employee *pcEmp = new
HourlyEmployee();

delete pcEmp;
```

Without virtual dtor

With virtual dtor

STOP

What is the output? Why?

- If the following 2 changes are made to the previous program, what is the output? Why?

```
virtual void Def1::Foo () {out << "Def1->Foo" << endl;}

virtual void Def2::Foo () {cout << "Def2->Foo" << endl;}

int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2;
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

Virtual Destructor

```
virtual ~Def1 () {cout << "~Def1" << endl;}

int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2;
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```

Consider

```
class Def1
{
  public:
    Def1 (int id) : mID(id) {cout << "Def1" << mID << "\n";}
    ~Def1 () {cout << "~Def1 " << mID << "\n";}
    void Foo () {cout << "Def1->Foo\n";}
 private:
  int mID;
};

class Def2 : public Def1
{
  public:
    Def2 (int id) : Def1(id) {cout << "Def2" << mID << "\n";}
    ~Def1 () {cout << "~Def2 " << mID << "\n";}
    void Foo () {cout << "Def2->Foo\n";}
};
```

What is the output? Why?

```
int main ()
{
    Def1 *pcDef1 = new Def1;
    Def2 *pcDef2 = new Def2;
    pcDef1->Foo ();
    pcDef2->Foo ();
    delete pcDef1;
    delete pcDef2;
}
```

What is the output? Why?

```
int main ()
{
  Def1 *pcDef1 = new Def1;
  Def1 *pcDef2 = new Def2; // type Def2 to Def1
  pcDef1->Foo();
  pcDef2->Foo();
  delete pcDef1;
  delete pcDef2;
}
```