15.1 Inheritance Basics

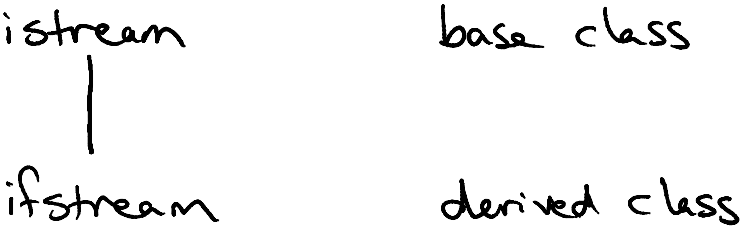
inheritance - one class is created from another class

derived class - new class made (child)

base class - original class (parent)

derived class gets all features of base class, plus adds more

Example:



ifstream is both istream & ifstream
both types work for param lists
eg operator >> (istream & i, ...
takes both istream & ifstream

istream is only istream
eg operator >> (ifstream & i, ...
takes only ifstream objects, not istream objects

Derived Classes

some features in common among several desired classes
eg Name, SSN for employees

some features unique to each class
eg hours & hourly rate for hourly, salary rate for salaried

make a base class that has all of the common features

make derived classes for each set of unique features

Features of base & derived classes
- all features of base class is in derived class
- derived class can redefine base feature(s)
- derived class can add new features

Example:

```
//DISPLAY 15.1 Interface for the Base Class Employee
//This is the header file employee.h.
//This is the interface for the class Employee.
//This is primarily intended to be used as a base class to derive
//classes for different kinds of employees.
#ifndef EMPLOYEE_H
#define EMPLOYEE_H

#include <string>
using namespace std;

namespace employeessavitch
{
        class Employee
        {
        public:
                Employee( );
                Employee(string the_name, string the_ssn);
                string get_name( ) const;
                string get_ssn( ) const;
                double get_net_pay( ) const;
                void set_name(string new_name);
                void set_ssn(string new_ssn);
                void set_net_pay(double new_net_pay);
                void print_check( ) const;          //func of interest
        private:
                string name;
                string ssn;
                double net_pay;
        };

}//employeessavitch

#endif //EMPLOYEE_H
//DISPLAY 15.2 Implementation for the Base Class Employee
//This is the file: employee.cpp.
//This is the implementation for the class Employee.
//The interface for the class Employee is in the header file employee.h.
#include <string>
#include <cstdlib>
#include <iostream>
#include "employee.h"
using namespace std;

namespace employeessavitch
{
        Employee::Employee( ) : name("No name yet"), ssn("No number yet"),
```

```
net_pay(0)
{
        //deliberately empty
}

Employee::Employee(string the_name, string the_number)
: name(the_name), ssn(the_number), net_pay(0)
{
        //deliberately empty
}

string Employee::get_name( ) const
{
        return name;
}

string Employee::get_ssn( ) const
{
        return ssn;
}

double Employee::get_net_pay( ) const
{
        return net_pay;
}

void Employee::set_name(string new_name)
{
        name = new_name;
}
void Employee::set_ssn(string new_ssn)
{
        ssn = new_ssn;
}

void Employee::set_net_pay (double new_net_pay)
{
        net_pay = new_net_pay;
}

void Employee::print_check( ) const
{
        cout << "\nERROR: print_check FUNCTION CALLED FOR
        AN \n"
             << "UNDIFFERENTIATED EMPLOYEE. Aborting the
             program.\n"
             << "Check with the author of the program about
             this bug.\n";
        exit(1);
}
```

of interest

//print error because no salary //info in base class

```
}//employeessavitch
//DISPLAY 15.3 Interface for the Derived Class HourlyEmployee
//This is the header file hourlyemployee.h.
//This is the interface for the class HourlyEmployee.
#ifndef HOURLYEMPLOYEE_H
#define HOURLYEMPLOYEE_H

#include <string>
#include "employee.h"

using namespace std;

namespace employeessavitch
{
```

//use to inherit from Employee

```
namespace employeessavitch
{                                                    //use to inherit from Employee
        class HourlyEmployee : public Employee
        {
        public:
                HourlyEmployee( );
                HourlyEmployee(string the_name, string the_ssn,
                        double the_wage_rate, double the_hours);
                void set_rate(double new_wage_rate);
                double get_rate( ) const;
                void set_hours(double hours_worked);
                double get_hours( ) const;
                void print_check( ) ;                //redefine this
                                                     //func from base
        private:
                double wage_rate;
                double hours;
        };

}//employeessavitch

#endif //HOURLYMPLOYEE_H
//DISPLAY 15.4 Interface for the Derived Class SalariedEmployee
//This is the header file salariedemployee.h.
//This is the interface for the class SalariedEmployee.
#ifndef SALARIEDEMPLOYEE_H
#define SALARIEDEMPLOYEE_H

#include <string>
#include "employee.h"

using namespace std;

namespace employeessavitch
{                                                    //note inheritance again
        class SalariedEmployee : public Employee
        {
        public:
                SalariedEmployee( );
                SalariedEmployee (string the_name, string the_ssn,
                        double the_weekly_salary);
                double get_salary( ) const;
                void set_salary(double new_salary);
                void print_check( );                 //again, redefine
        private:
                double salary;//weekly
        };

}//employeessavitch

#endif //SALARIEDEMPLOYEE_H

//DISPLAY 15.5 Implementation for the Derived Class HourlyEmployee
//This is the file: hourlyemployee.cpp
//This is the implementation for the class HourlyEmployee.
//The interface for the class HourlyEmployee is in
//the header file hourlyemployee.h.
#include <string>
#include <iostream>
#include "hourlyemployee.h"
using namespace std;

namespace employeessavitch
{                                                    //call base constructor
```

```
using namespace std;

namespace employeessavitch
{
        HourlyEmployee::HourlyEmployee( ) : Employee( ), wage_rate(0),
        hours(0)
        {
                //deliberately empty
        }

        HourlyEmployee::HourlyEmployee(string the_name, string the_number,
                double the_wage_rate, double the_hours)
        : Employee(the_name, the_number), wage_rate(the_wage_rate),
        hours(the_hours)
        {
                //deliberately empty
        }

        void HourlyEmployee::set_rate(double new_wage_rate)
        {
                wage_rate = new_wage_rate;
        }

        double HourlyEmployee::get_rate( ) const
        {
                return wage_rate;
        }

        void HourlyEmployee::set_hours(double hours_worked)
        {
                hours = hours_worked;
        }

        double HourlyEmployee::get_hours( ) const
        {
                return hours;
        }

        void HourlyEmployee::print_check( )
        {
        set_net_pay(hours * wage_rate);

        cout << "\n_____\n";
        cout << "Pay to the order of " << get_name( ) << endl;
        cout << "The sum of " << get_net_pay( ) << " Dollars\n";
        cout << "_____\n";
        cout << "Check Stub: NOT NEGOTIABLE\n";
        cout << "Employee Number: " << get_ssn( ) << endl;
        cout << "Hourly Employee. \nHours worked: " << hours
                << " Rate: " << wage_rate << " Pay: "
                << get_net_pay( ) << endl;
        cout << "_____\n";
        }

}//employeessavitch

//DISPLAY 15.6 Implementation for the Derived Class SalariedEmployee
//This is the file salariedemployee.cpp.
//This is the implementation for the class SalariedEmployee.
//The interface for the class SalariedEmployee is in
//the header file salariedemployee.h.
#include <iostream>
#include <string>
#include "salariedemployee.h"
```

//call base constructor

//call base constructor

//redefine print_check

```
#include "salariedemployee.h"
using namespace std;

namespace employeessavitch
{
	SalariedEmployee::SalariedEmployee( ) : Employee( ), salary(0)
	{
		//deliberately empty
	}
	SalariedEmployee::SalariedEmployee(string the_name, string the_number,
		double the_weekly_salary)
		: Employee(the_name, the_number),
		salary(the_weekly_salary)
	{
		//deliberately empty
	}

	double SalariedEmployee::get_salary( ) const
	{
		return salary;
	}

	void SalariedEmployee::set_salary(double new_salary)
	{
		salary = new_salary;
	}
	void SalariedEmployee::print_check( )
	{

	set_net_pay(salary);
	cout << "\n_____
	\n";
	cout << "Pay to the order of " << get_name( ) << endl;
	cout << "The sum of " << get_net_pay( ) << " Dollars\n";
	cout << "_____\n";
	cout << "Check Stub NOT NEGOTIABLE \n";
	cout << "Employee Number: " << get_ssn( ) << endl;
	cout << "Salaried Employee. Regular Pay: "
		<< salary << endl;
	cout << "_____\n";
	}

}//employeessavitch
```

Constructors in Derived Classes
must invoke base constructors
invoke using : syntax
if base class constructor NOT invoked, automatically calls the default constructor of base
- most ancestral base class is called first
eg A → B → C
C's constructors would automatically call A's default

then B's default

Private member var/func in Base class
  private vars/funcs in base class
    are still private to derived class
  derived class cannot access them
    via the dot/arrow operators
  derived class must use base class's
    interface to manipulate private
    data/funcs
  Security precaution
    prevents people from inheriting
      your class just to get direct
      access to private vars/funcs
    Can provide limited access via
      the protected section
The protected qualifier
  makes var/func private to outside
    world, but available to derived
  overcomes issues w/ private for
    derived class as described above
Redefinition of Member Functions
  in derived class, only list functions
    that should be changed from base
  prototype must match base prototype
    not the same as overloading because
      it is the exact same prototype
Invoking base func after redefinition
  use scope resolution operator
    HourlyEmployee sally;

    sally.print_check(); //use derived
    sally.Employee::print_check();

// use base

15.2 Inheritance Details

Functions that are NOT inherited
- constructors
- private member functions
- destructors
- copy constructor
- assignment operator (=)

Assignment operators & Copy Constructors
- can invoke base version w/scope operator
- Assignment:

```
Derived& Derived:: operator =(
        const Derived & right)
{
    Base:: operator = (right);
    :
}
```

- Copy constructor:

```
Derived:: Derived(const Derived &obj)
    : Base(obj)
{
    :
}
```

Destructors in Derived Classes
- base destructor automatically invoked
- Order is:
  - A → B → C
  - C's destructor first
  - then B's destructor
  - then A's destructor
- Order is reverse of constructor order