CSCI 1061U
Programming Workshop 2

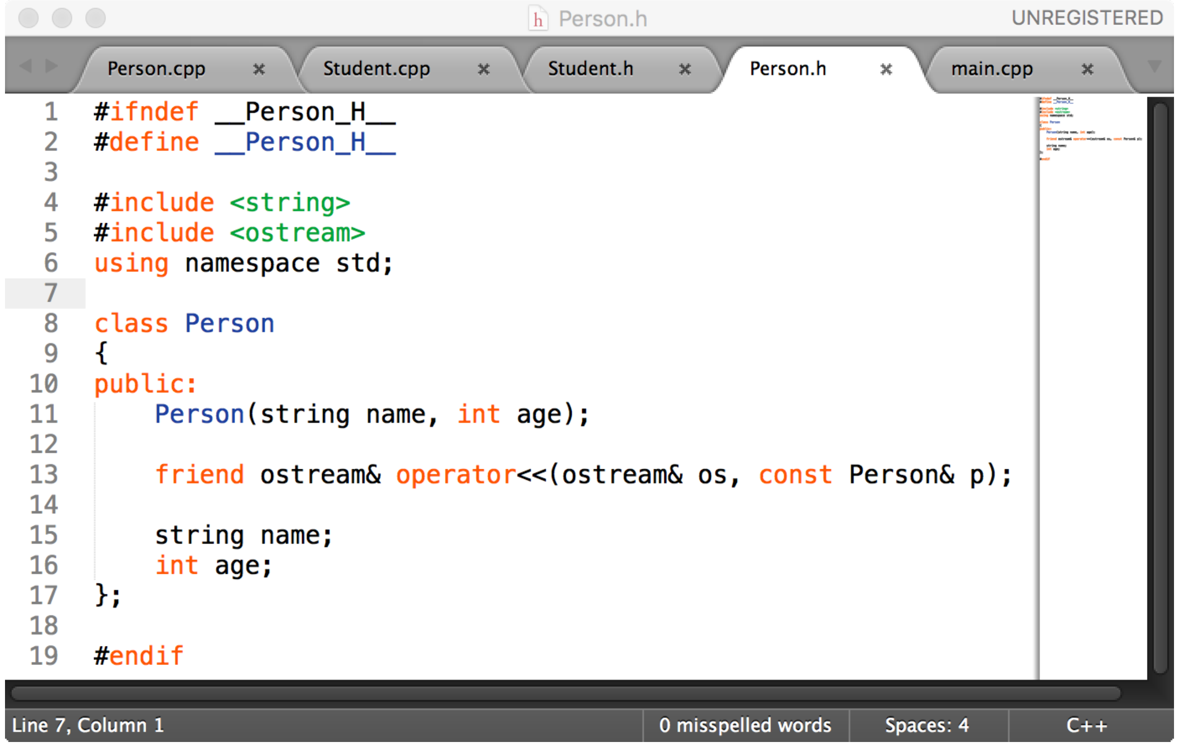
Inheritance in C++

Inheritance

- An important concept in Object Oriented Programming
- Facilitates abstraction

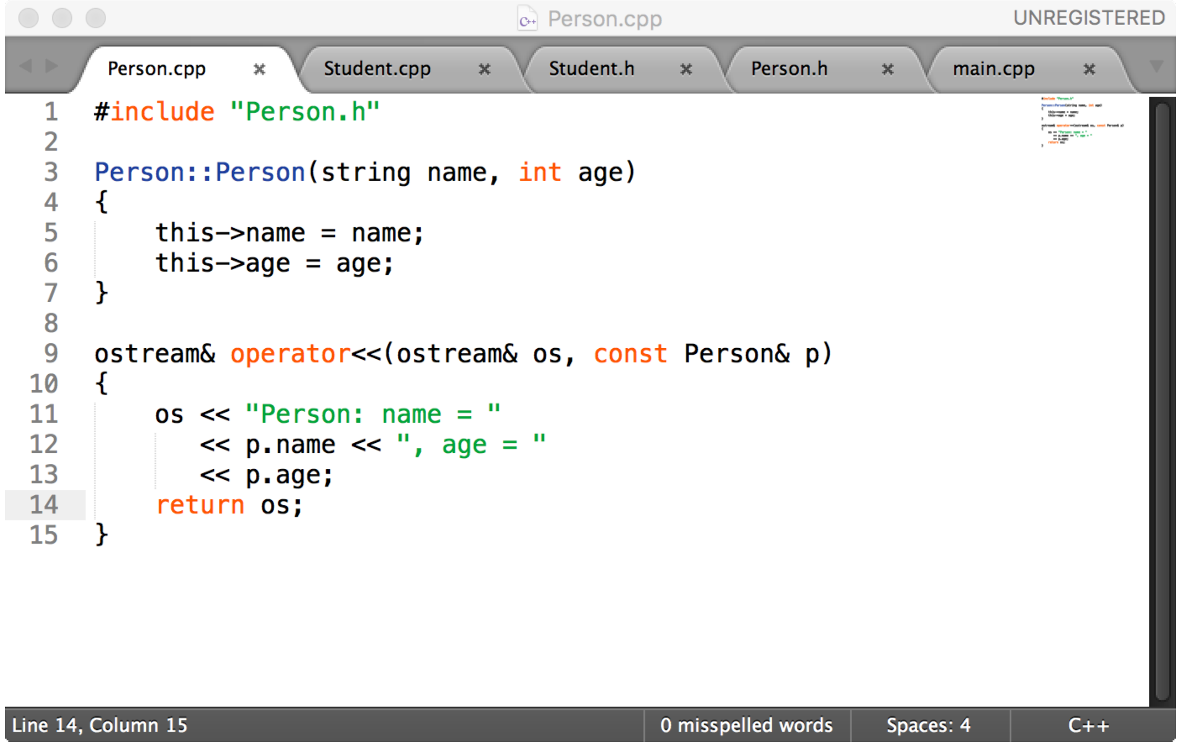
- Mechanism
  - General form of a class is defined
  - Specialized forms inherit from the general form and add functionality to it

Inheritance example: Person class



```cpp
#ifndef __Person_H__
#define __Person_H__

#include <string>
#include <ostream>
using namespace std;

class Person
{
public:
    Person(string name, int age);

    friend ostream& operator<<(ostream& os, const Person& p);

    string name;
    int age;
};

#endif
```

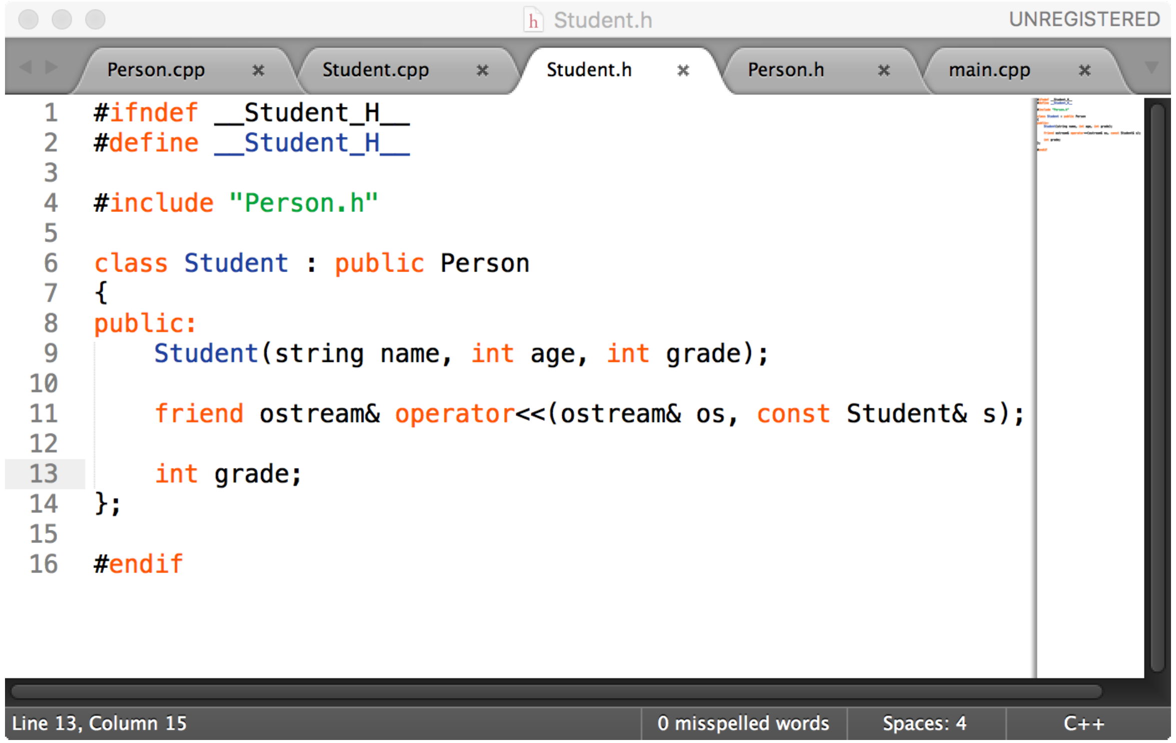
Person.h



```cpp
#include "Person.h"

Person::Person(string name, int age)
{
    this->name = name;
    this->age = age;
}

ostream& operator<<(ostream& os, const Person& p)
{
    os << "Person: name = "
       << p.name << ", age = "
       << p.age;
    return os;
}
```

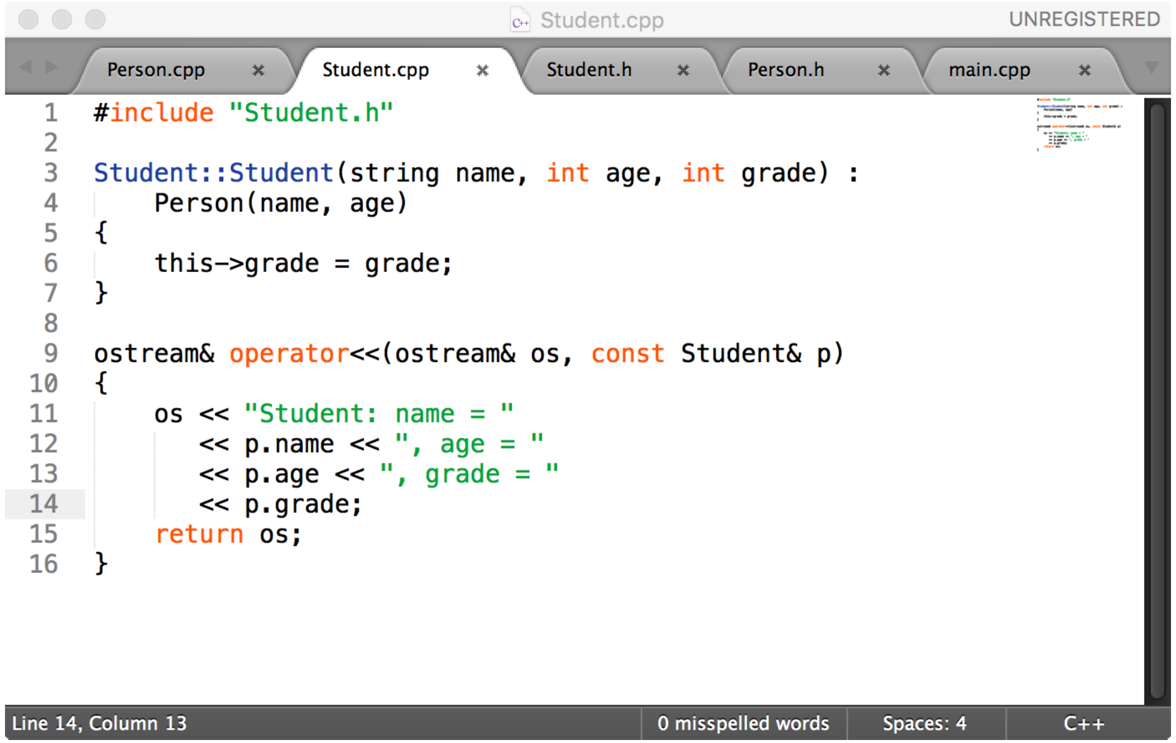
Person.cpp

Inheritance example: Student class



```cpp
#ifndef __Student_H__
#define __Student_H__

#include "Person.h"

class Student : public Person
{
public:
    Student(string name, int age, int grade);

    friend ostream& operator<<(ostream& os, const Student& s);

    int grade;
};

#endif
```

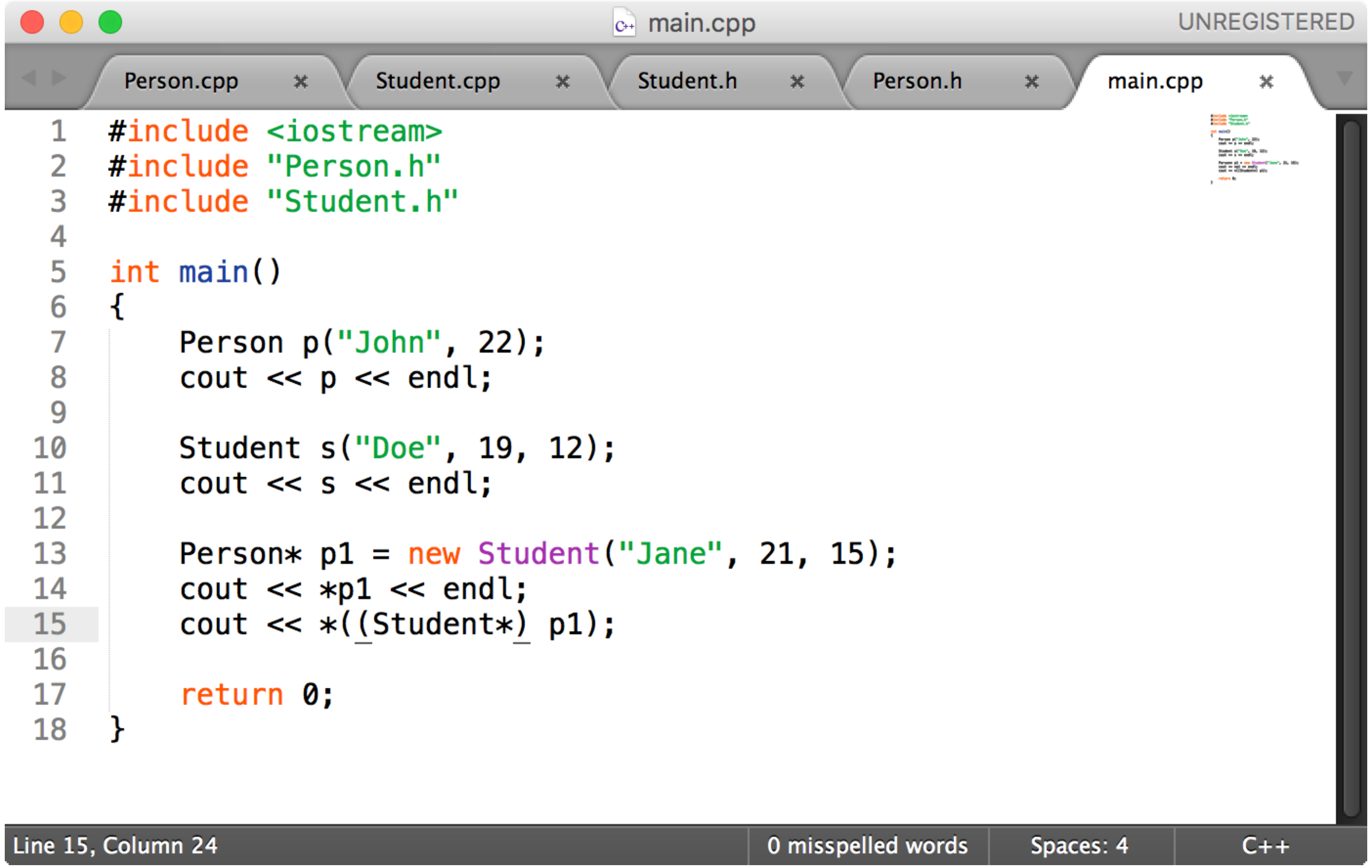
Student.h



```cpp
#include "Student.h"

Student::Student(string name, int age, int grade) :
    Person(name, age)
{
    this->grade = grade;
}

ostream& operator<<(ostream& os, const Student& p)
{
    os << "Student: name = "
        << p.name << ", age = "
        << p.age << ", grade = "
        << p.grade;
    return os;
}
```

Student.cpp

Inheritance example: All students are persons



```cpp
#include <iostream>
#include "Person.h"
#include "Student.h"

int main()
{
    Person p("John", 22);
    cout << p << endl;

    Student s("Doe", 19, 12);
    cout << s << endl;

    Person* p1 = new Student("Jane", 21, 15);
    cout << *p1 << endl;
    cout << *((Student*) p1);

    return 0;
}
```

Inheritance mechanics

- Base class (Person)
  - "General" class from which others derive
- Derived class (Student)
  - Automatically has base class’s:
    - Member variables
    - Member functions
  - Can then add additional member functions and variables

Derived classes

- Derived classes
  - Automatically have all member variables
  - Automatically have all member functions

- Derived class said to "inherit" members from the base class

- Can then redefine existing members and/or add new members

Inheritance: common terms

- Parent class
    - Refers to base class
- Child class
    - Refers to derived class
- Ancestor class
    - Class that’s a parent of a parent
- Descendant class
    - Opposite of ancestor

Inheritance: constructors

- Base class constructors are ***not*** inherited in derived classes
- Base class constructor must initialize all base class member variables
- The derived class constructor can use base class constructors to initialize base class member variables

Private member variables of base class

- Derived class "inherits" private member variables
  - But still cannot directly access them
- Private member variables can ONLY be accessed "by name" in member functions of the class they’re defined in

Private methods of base class

- Cannot be accessed outside the implementation of base class
- Cannot be called in derived class

Private members vs. private methods of base class

- Private member variables can be accessed indirectly via accessor or mutator member functions
- Private member functions simply cannot be accessed in derived class
    - These should be used only in class they’re defined

Protected members and methods of base class

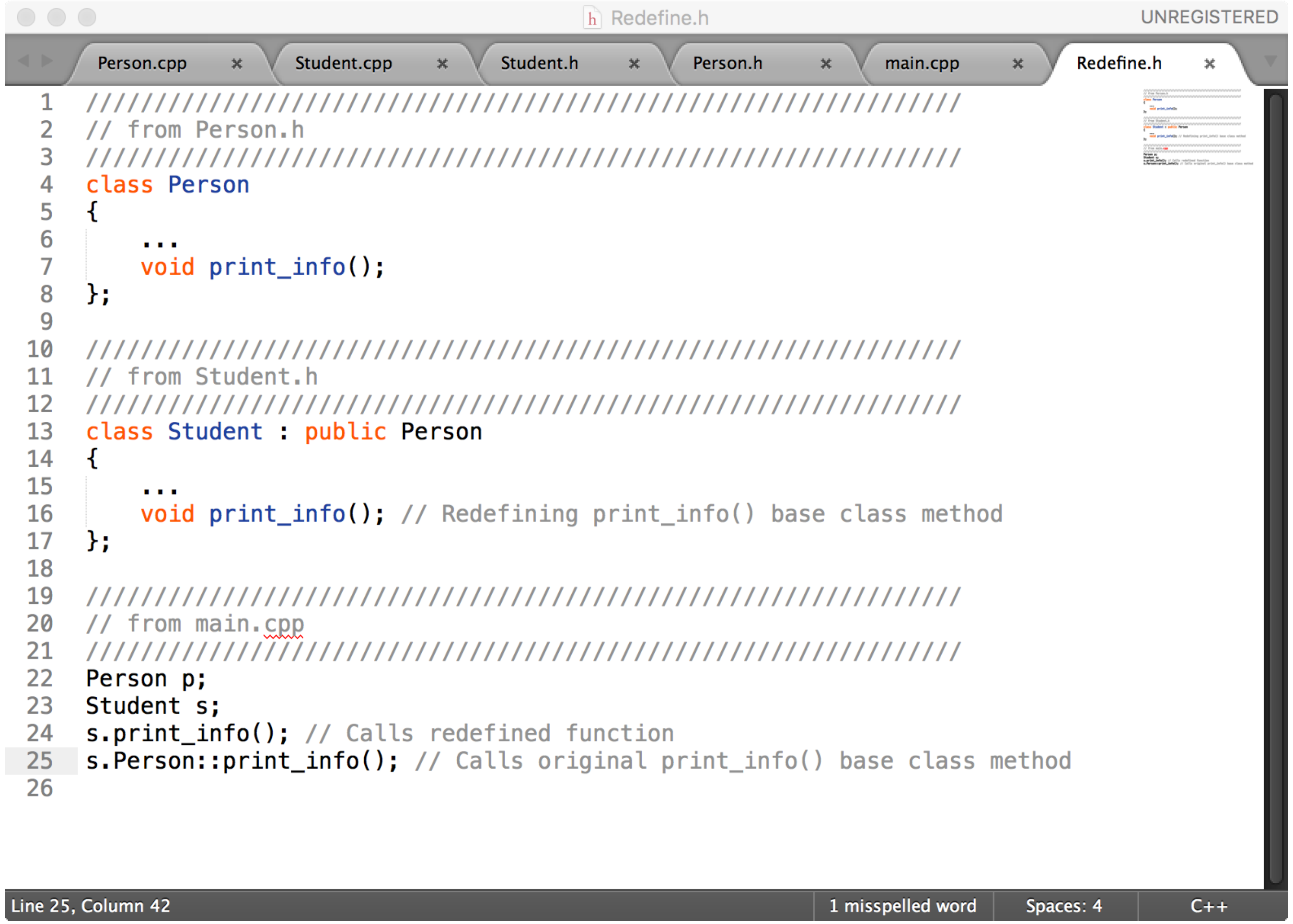
- Allows access "by name" in derived class
- Not visible in other classes
- Many feel this "violates" information hiding

Redefining and overloading functions in derived class

- Derived class can add new functions, redefine some functions, and overload other functions

- Redefining:
  - "re-writes" a base class function
  - Same parameter list
- Overloading:
  - Different parameter list
  - Defined "new" function that takes different parameters
  - Overloaded functions must have different signatures

Is it possible to access a redefined base class function?



```cpp
////////////////////////////////////////////////////////////////
// from Person.h
////////////////////////////////////////////////////////////////
class Person
{
    ...
    void print_info();
};

////////////////////////////////////////////////////////////////
// from Student.h
////////////////////////////////////////////////////////////////
class Student : public Person
{
    ...
    void print_info(); // Redefining print_info() base class method
};

////////////////////////////////////////////////////////////////
// from main.cpp
////////////////////////////////////////////////////////////////
Person p;
Student s;
s.print_info(); // Calls redefined function
s.Person::print_info(); // Calls original print_info() base class method

```

Base class methods that are ***not*** inherited

- Constructors
- Destructors
- Copy constructor
- Assignment operator

Destructor in derived class

- When derived class destructor is invoked, it automatically calls base class destructor!
- Derived class destructors need only be concerned with derived class variables

Credit: Copyright © 2016 Pearson Inc. All rights reserved.

Destructor calling order

- Consider:
  class B derives from class A
  class C derives from class B
  A ← B ← C

- When object of class C goes out of scope:
  - Class C destructor called 1st
  - Then class B destructor called
  - Finally class A destructor is called

Protected and Private Inheritance

- New inheritance "forms"
  - Both are rarely used
- Protected inheritance:
  class SalariedEmployee : protected Employee
  {…}
  - Public members in base class become protected in derived class
- Private inheritance:
  class SalariedEmployee : private Employee
  {…}
  - All members in base class become private in derived class

Credit: Copyright © 2016 Pearson Inc. All rights reserved.

Multiple inheritance

- Derived class can have more than one base class!
    - Syntax just includes all base classes separated by commas:
      class derivedMulti : public base1, base2
      {…}
- Possibilities for ambiguity are endless!
- Dangerous undertaking!
    - Some believe should never be used
    - Certainly should only be used be experienced programmers!

Summary

- Inheritance in C++

- Readings
  - Ch. 6