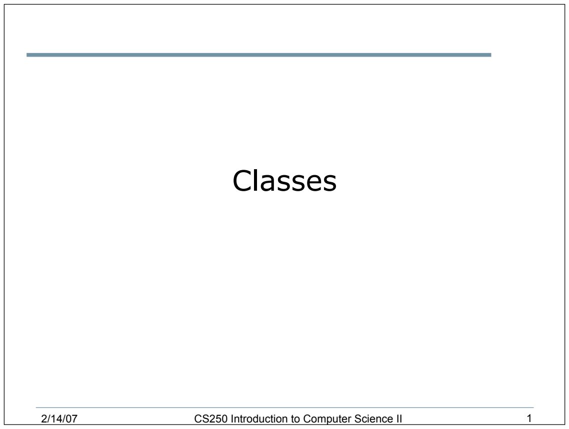
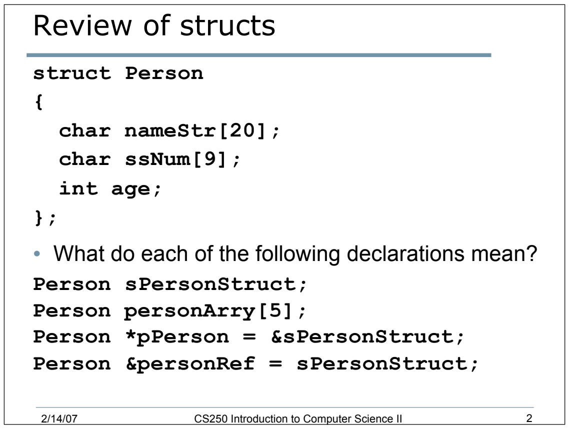


Classes

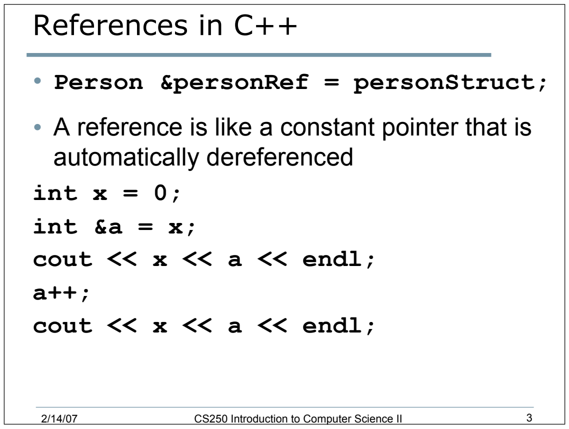


Review of structs

```
struct Person
{
  char nameStr[20];
  char ssNum[9];
  int age;
};
```

- What do each of the following declarations mean?

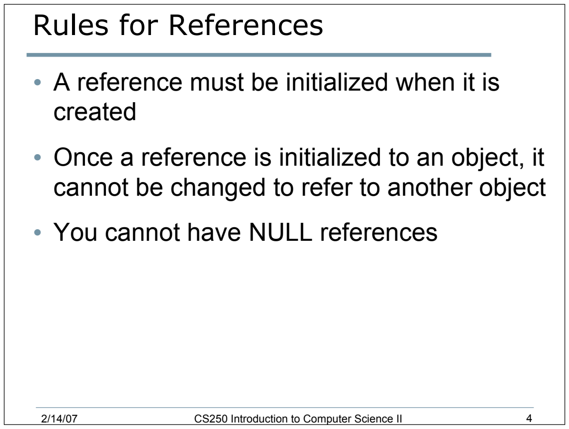
```
Person sPersonStruct;
Person personArry[5];
Person *pPerson = &sPersonStruct;
Person &personRef = sPersonStruct;
```



References in C++

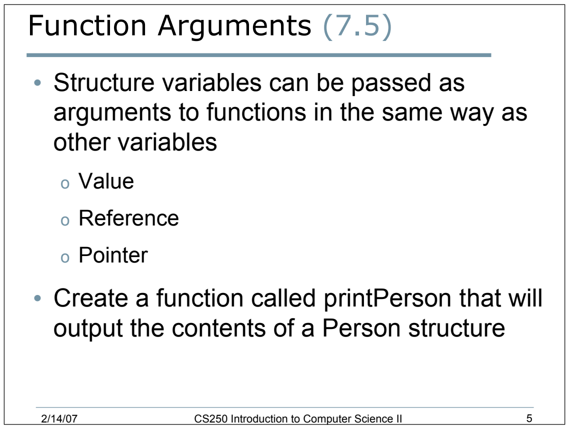
- **Person &personRef = personStruct;**
- A reference is like a constant pointer that is automatically dereferenced

```
int x = 0;
int &a = x;
cout << x << a << endl;
a++;
cout << x << a << endl;
```



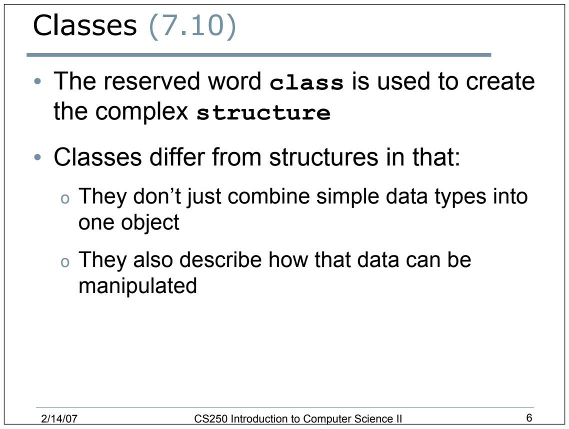
Rules for References

- A reference must be initialized when it is created
- Once a reference is initialized to an object, it cannot be changed to refer to another object
- You cannot have NULL references



Function Arguments (7.5)

- Structure variables can be passed as arguments to functions in the same way as other variables
  - Value
  - Reference
  - Pointer
- Create a function called printPerson that will output the contents of a Person structure



Classes (7.10)

- The reserved word **`class`** is used to create the complex **`structure`**
- Classes differ from structures in that:
  - They don't just combine simple data types into one object
  - They also describe how that data can be manipulated

More on Objects

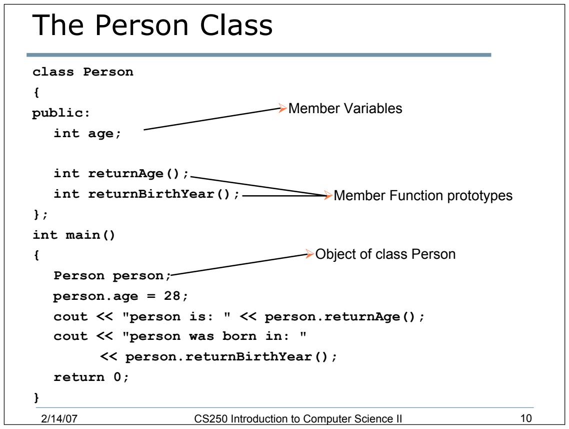
- Object-oriented programming hides the details of objects from objects of other types
- When an object needs information from another object or needs another object to perform a task, it sends a message to the object requesting what it needs
- As a result, object-oriented programs can be written more generically than structured programs
- Usually, making changes to the object-oriented programs is easier than changing structured programs

In Summary

- A `class` is like a `struct` but much more
- Whereas `structs` can contain simple data types, `classes` contain both *data types* and *functions* that manipulate the class data

A C++ Example

- Enough of theory!
- Let’s have a look at a real example.
- We will create a `class Person` that will:
  - Store information about person
  - Store functions to manipulate this information



The Person Class

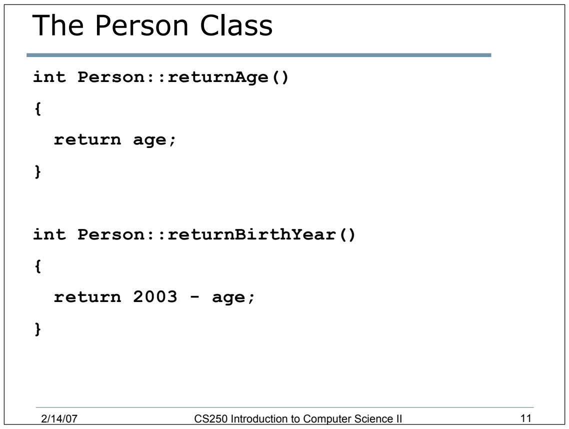
```
class Person
{
public:
   int age;

   int returnAge();
   int returnBirthYear();
};
int main()
{
   Person person;
   person.age = 28;
   cout << "person is: " << person.returnAge();
   cout << "person was born in: "
         << person.returnBirthYear();
   return 0;
}
```

Member Variables

Member Function prototypes

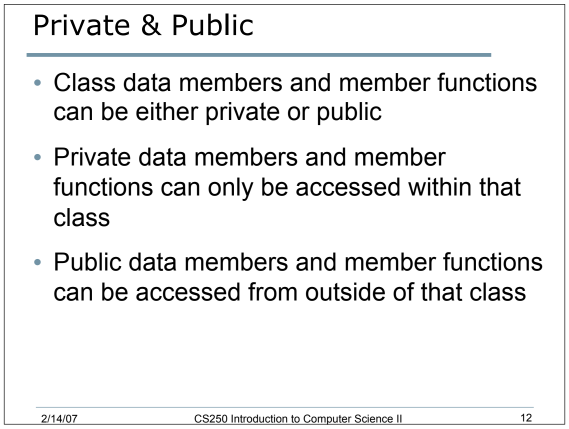
Object of class Person



The Person Class

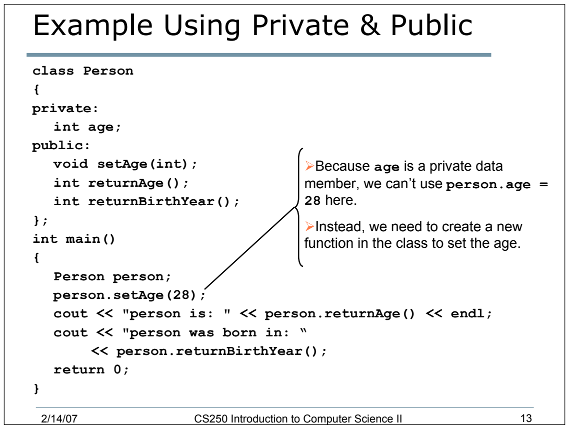
```
int Person::returnAge()
{
  return age;
}


int Person::returnBirthYear()
{
  return 2003 - age;
}
```



Private & Public

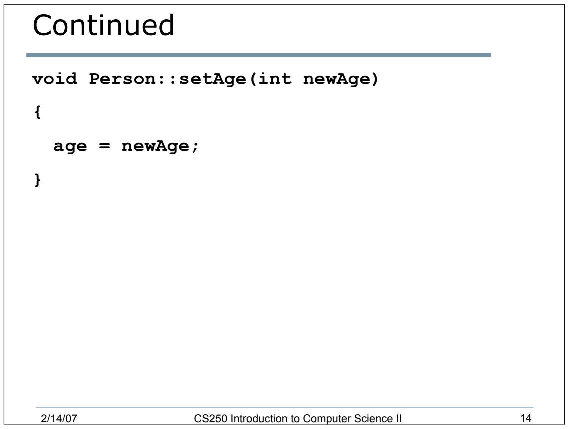
- Class data members and member functions can be either private or public
- Private data members and member functions can only be accessed within that class
- Public data members and member functions can be accessed from outside of that class



Example Using Private & Public

```
class Person
{
private:
  int age;
public:
  void setAge(int);
  int returnAge();
  int returnBirthYear();
};
int main()
{
  Person person;
  person.setAge(28);
  cout << "person is: " << person.returnAge() << endl;
  cout << "person was born in: “
      << person.returnBirthYear();
  return 0;
}
```

- Because `age` is a private data member, we can't use `person.age = 28` here.
- Instead, we need to create a new function in the class to set the age.



Continued

```
void Person::setAge(int newAge)
{
  age = newAge;
}
```