Classes

Nov 3, Ch 10.1 - 10.3

TALKS FOR 2 HOURS STRAIGHT

PAUSES FOR CLASS TO LAUGH AT JOKE
- **class / struct** (structure)

```cpp
struct position
{
    int x;
    int y;
};
```

```cpp
class blah2
{
    public:
        void foo();
    private:
        std::string word;
};
```

- **constructor**

```cpp
class myName
{
    public:
        myName();
        myName(int x);
        char takeThis();
    private:
        char itsASecretToEveryone;
};
```
Arrays group together similar data types (any amount you want)

Classes and structs group together dissimilar types that are logically similar
classes and structs are outlines/blue prints of an organization structure

Thus when you create a variable of your class's type, you create an instance
Suppose you wanted to write a function to find the maximum element in an array. How would you return both an index and the element?
Suppose you wanted to write a function to find the maximum element in an array. How would you return both an index and the element?

1. Use a global variable to share between functions
2. Use call-by-reference (See: findMax.cpp)
A struct (structure) is a grouping of similar objects.

```cpp
struct closet
{
    string belts[10];
    string shoes[20];
    string shirts[40];
    string pants[30];
    string dresses[20];
};
```

(See: findMaxV2.cpp)
struct

You just made your own data type (just like int/double/char/etc.)

You can make as many variables of this type as you want

The dot operator tells the computer to go inside the object/container

twoInts x = findMax(numbers);
cout << "Maximum is numbers["<<x.second<<"] = "<<x.first<<endl;
You need the dot to differentiate between two different variables

You can also think of the dot as possessive in English ( . → 's )

```c
struct date
{
    int day;
    int month;
    int year;
};
```
struct

You can initialize a struct using braces (much like arrays, goes in order declared)

```c
struct date {
    int day;
    int month;
    int year;
};

// Nov 3rd 2015
date today = {3, 11, 2015};
```

You can also use = to assign all elements

```c
date another = today;
```

... same as ...

```c
date another;
another.day = today.day;
another.month = today.month;
another.year = today.year;
```
Suppose we are planning to redo all the classroom name plates in Keller hall.

How would you store all the room information...
... without structs?
... with structs?

(See: room.cpp)
class

A class is functionally the same as a struct (creates a new data type)

However, the notation is slightly different (contains functions)

```c
struct date
{
    int day;
    int month;
    int year;
};
```

```c
class date
{
    public:
    int day;
    int month;
    int year;
    void print();
};
```
You can put `const` to the right of the function in a class to designate that it will not change any of the member variables.

```cpp
class date
{
public:
    int day;
    int month;
    int year;
    void print() const;
};
```

`const` means cannot change day, month or year.
To define a class functions, we need to specify the scope using :: (scope resolution)

```cpp
// class "date"'s version of print
void date::print() {
    cout << month << "/" << day << "/" << year;
}
```

... compared to ...

```cpp
// not related to "date" class
void print() {
    cout << "Hello!\n";
}
```

(See: date.cpp)
Scope resolution is actually what namespaces are for:

```cpp
using namespace std;
```

Using the aboves lets us write:

```cpp
cout << "Hi" << endl;
```

... instead of ...

```cpp
std::cout << "Hi" << endl;
```

annoying to rewrite every time
The `::` is very similar to the `.` operator

`::` is used to specify the location in a general sense (without a specific variable involved)
Example: Put socks on before shoes

`. ` is used to specify the ownership of a variable or function (owner is another variable)
Example: Tie my shoe laces
class / structs

classes and structs make code much easier to modify in addition to organize

Learning how to write code is practice, this will become natural if you do it a lot

Writing code that can easily be added to is much more difficult
class inside of another class? Sure, why not!
Suppose we wanted to change the HORRIBLE floor-to-number convention in Keller Hall (thus the re-plating of classrooms)

Break up the rooms by floor and have a floor label for each floor

We can keep the same number scheme: 2-130 is 2\textsuperscript{nd} floor room number 130
(See: building.cpp)
class date
{
private:
    int day;
    int month;
    int year;
public:
    void print();
    void setDate(int day, int month, int year);
};
public vs private

The **public** keyword allows anyone anywhere to access the variable/method

The **private** keyword only allows access in the class where the variable/method is defined
public vs private

All variables should be **private**

While this means you need methods to set variables, users do not need to know how the class works

This allows an easier interface for the user (also easier to modify/update code)

(See: datePrivate.cpp)
public vs private
public vs private

Creating interfaces with public allows users to not worry about the private implementation.

So... more work for you (programmer) less work for everyone else.
Constructors

The date class has two functions: setDate() and print()

As we need to run setDate() on a variable before it is useful anyways

In fact, such a thing exists and is called a constructor (run every time you create a variable)
Constructors

The class name and the constructor must be identical
(constructors also have no return type)

```cpp
class date
{
private:
int day;
int month;
int year;
public:
    date(int day, int month, int year);
    // ^ constructor has same name as class
    void print();
};
```

(See: dateConstructor.cpp)
Constructors

If you don't put a constructor, C++ will make a default constructor for you (no arguments)

```cpp
date();    // default constructor
```

To use the default constructor say this:

```cpp
date never;
```

... not this:

```cpp
date notWhatYouWant();
// ^ function declaration
```
Constructors

If you declared constructors you must use one of those

Only if you declare no constructors, does C++ make one for you (the default)

Note: our dateConstructor.cpp has no way to change the value of the date after it is created (thus gives control over how to use class)
Just as writing very long main() functions can start to get confusing...

... writing very long .cpp files can also get confusing

Classes are a good way to split up code among different files
#include

You can #include your class back in at the top or link to it at compile time.

You have to be careful as #include basically copies/pastes text for you.

Will not compile if class declared twice (used in two different classes you #include).
To get around this, you can use compiler commands in your file

```
#ifndef DATE
#define DATE

class date
{
public:
  int day;
  int month;
  int year;
  void print() const;
};
#endif
```

This ensures you only have declarations once
(See: dateClass.hpp, dateClass.cpp, runDate.cpp)