Recursion
Ch 14
Highlights

- recursion

```cpp
int main()
{
    cout << "HI\n!";
    main();
}
```

- classes

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

No fancy blue words or classes this chapter

Recursion is simply calling a method from inside itself

This copy will re-run the method on any new arguments or information

(See: badRecursion.cpp)
Click to exit presentation...
If you forget your stopping case, you will not get an infinite loop but crash the program. This is because every function call takes up more memory, so you constantly ask for more memory. Eventually the memory (stack) cannot store anymore.
Your mother is so fat,
the recursive function computing her mass causes a stack overflow.
Recursion basics

Good recursion must have 2 parts:
- A recursive call on a smaller problem
- An ending case

(see: https://www.youtube.com/watch?v=-xMYvVr9fd4)

In order to use recursion, you must be able to identify a subproblem that is very similar to the original problem

Each step must get you closer to the solution
Recursion basics

For recursion, you can basically assume your function works as you want it to (even though you have not written it)

If you have the ending case and reduction step correct, then it will!
Recursion: Family tree

Person

Descendant

So... Luke has a SISTER!
A child couldn't sleep, so her mother told a story about a little frog, who couldn't sleep, so the frog's mother told a story about a little bear, who couldn't sleep, so bear's mother told a story about a little weasel... who fell asleep. ...and the little bear fell asleep; ...and the little frog fell asleep; ...and the child fell asleep.
Recursion: Basic example

Remember, code starts in main and runs from top to bottom in sequence (normally)

When you call a function you go execute all the function's code is run before going back to the original code

Code order is important in recursion!

(See: stringRecursion.cpp)
Recursion: Fibonacci

The Fibonacci numbers are defined as:

\[ F(n) = F(n-1) + F(n-2) \]

In other words, you add the previous two to get the next.

This is recursion! Computing \( F(n) \) involves solving smaller problems of \( F(n-1) \)
(See: fibonacciRecursion.cpp)
Recursion

There are two important parts of recursion:
- A **stopping** case that ends the recursion
- A **reduction** case that reduces the problem

What are the base and stopping cases for the Fibonacci numbers?

\[ F_n = F_{n-1} + F_{n-2}, \]

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ...

(sum of the previous two numbers)
Recursion

What if I wanted to just count down to zero? `countdown(5)` would show:
5
4
3
2
1
0!

(see: `countdown.cpp`
Recursion

What if we defined tangent recursively as:

\[ \tan(x) = \frac{x}{1 - \frac{x^2}{3 - \frac{x^2}{5 - \ldots}}} \]

Assume we take an input for how many times to do this recursion
What is the pattern? What is the stopping case?
How do we move towards the stopping case

(see: tangent.cpp)
Recursion: Root finding

Find a root of:
(see: rootFind.cpp)

Method:
1. Find one positive y and 1 neg. y
2. Find midpoint (of x values)
3. update y-pos/neg
Recursion

How would you solve a sudoku problem?
Rules:
1. Every row has numbers 1-9
2. Every column has numbers 1-9
3. The nine 3x3 boxes have numbers 1-9

Reduce problem?
Stopping case?

(see: sudokuSolver.cpp)
Recursion

Recursion is simply when a function calls itself (we did this for the maze in week 5)

This is quite powerful, but also confusing

(see: towerHanoi.cpp)
Recursion

Do not try to solve chess in this manner!

You will segfault
(you will also not finish computing before the sun burns the earth to a crisp)
Miscellaneous notes

Try googling “recursion” and click on the spelling suggestion

Recursion is very powerful and used in many advanced algorithms

It will give you a headache for a while... =(
Structs/classes

Ch 10.1 - 10.3

HOW DOES COMPUTER PROGRAMMING WORK?

MAGIC.

You have no class.
struct/class vs array

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?
struct

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)
struct

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

```cpp
struct twoInts {
    int first;
    int second;
};

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;
};
```

```c
int main()
{
    date today;
    date midterm2;
    today.month = 11;
    midterm2.month = 11;
}
```
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
today = {3, 11, 2015};

date another = today;

... same as ...

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)
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;
};
```
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;
};
```

```cpp
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;
};
```
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

specific case
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

class inside of another class? Sure, why not!

(see: nestedClass.cpp)