Lecture 20
Wrapup

CS211 – Fundamentals of Computer Programming II

Branden Ghena – Fall 2021

Slides adapted from:
Jesse Tov

Northwestern

Administrivia

- Projects due tonight
    - Prioritize getting as many spec items completed as possible
    - Don’t forget to write model tests as well!

- Demos from volunteers on Thursday
    - No lecture, just games
    - I’ll post a sign-up form

Today’s Goals

- Review what you’ve learned and why it is useful

- Understand when to use or avoid C/C++ in future projects

- Consider what’s next after CS211

Outline

- **Course Goals**
- When should you use C and C++?
- Review of Class Topics
- What’s next?

So, why CS211?

- It’s going to make you a **much** better programmer

- It’s going to teach you a bunch of new skills

- It’s going to enable you to succeed in future classes

Formal goals

CS211:

- Teaches software design skills at a small-to-medium scale
  - Some smaller programs: Overlapped, Brickout
  - Some larger programs: Rank-choice Voting, Reversi

Formal goals

CS211:

- Teaches software design skills at a small-to-medium scale
  - Some smaller programs: Overlapped, Brickout
  - Some larger programs: Rank-choice Voting, Reversi

- Bridges students from *How to Design Programs* languages to industry-standard languages and tools
  - Unix shell: SSH, ls, cd,
  - C and C++ programming languages
  - CLion IDE
  - Make and CMake

Upsides to C and C++

- You are in charge of everything
  - You can do anything you want without constraints

- Capable of directly interacting with hardware (“systems language”)
  - Grab exactly as much memory as you need and manage it yourself
  - Makes it incredibly fast (~100x faster than Python)
  - Makes it incredibly efficient (no memory is wasted)

- These lead to the languages being very widely used
  - Top five programming languages for decades include C and C++

Downsides to C and C++

- You are in charge of everything
  - And nothing is taken care of for you

- Things you “can’t” do are **UNDEFINED BEHAVIOR**
  - To enable portability, the languages just straight-up don’t say what happens if you violate the rules
  - The computer could do *anything*

- Backwards compatibility means features are only ever added
  - You’ll see this especially in C++, C just has less features total
  - C++ feels like a bunch of things stapled together
    - And there’s an amazing programming language hiding in there

So why teach C and C++?

- You’ll learn a lot more about programming
  - Syntax and ideas from C inspired a lot of other languages
  - Feels very different from Racket or Python

- You’ll become a better programmer
  - You’re going to run into a lot of errors and problems in this class
  - Hopefully they teach you to better design and plan your code

- Prepare you to dig deeper into computer systems
  - A “systems language” is needed to interact directly with hardware
  - Major options: Pascal, C, C++, Ada, Rust

Outline

- Course Goals
- **When should you use C and C++?**
- Review of Class Topics
- What’s next?

When should you use C?

- You probably shouldn’t

When should you use C?

- You probably shouldn’t

- Stronger: Don’t use C.

When should you use C?

- You probably shouldn’t

- Stronger: Don’t use C.

- Stronger still (and what I actually believe):

  Using C when you could use a safer language is engineering malpractice.

  C and **UNDEFINED BEHAVIOR** are the root of many security vulnerabilities

What is C good for?

- Very particular things

- Need for extreme efficiency and speed
  - Often efficient services for *other* programs
  - Systems Programming

- Low-level memory or hardware manipulation
  - Interact with raw memory
  - Computer Systems

Slowly we are replacing the need for C

- Need for extreme efficiency and speed
    - Beware premature optimization
        - Often algorithm and library choice are more important than language
    - C++ (and others) are often good for this as well
- Low-level memory or hardware manipulation
    - New languages like Rust are starting to meet the needs here

The value of learning C

- The impact it has on every other language you might learn
    - Java, Objective-C, C#, Go, Javascript, Swift, PHP, Perl, Python

    - You’ll see lots of similar ideas
        - Structs
        - Curly braces and semicolons
        - if, while, for
        - Arrays and square bracket indexing

- You may use it for future systems courses: CS213, CS343, etc.
- Some experience helps you understand the danger

What about C++?

- More ambiguous than C

- Definitely don’t use *old* C++
    - We learned modern C++14
        - Includes many more standard libraries
        - Includes safer memory management (smart pointers)

- There are other languages with many of the benefits without the confusing parts
    - But really big, important software often eventually ends up in C++

Use the right programming language for the job

- Remember: there is no *best* programming language
  - Every tool is situational

- C and C++ are *not* good for simple programs and demonstrations
  - So use something simpler, like Python

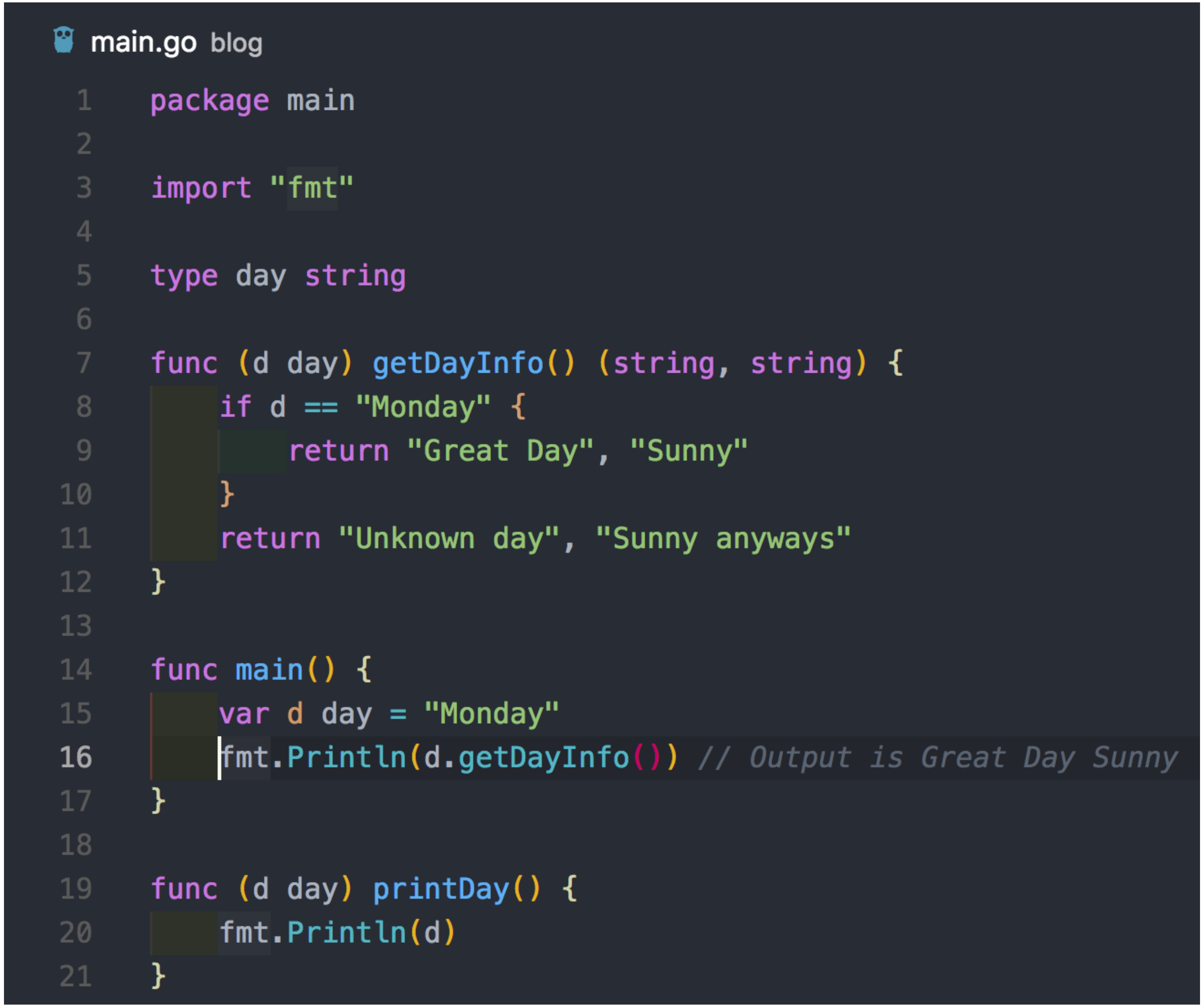
- But if we wrote all of our video game engines in Python, games would be very limited in what they could do
  - So use something more complex, like C++

Break + example Go code

- I'm guessing few of you have used Go
    - But do you understand it?

- Where does code start?

- What is the type of d?

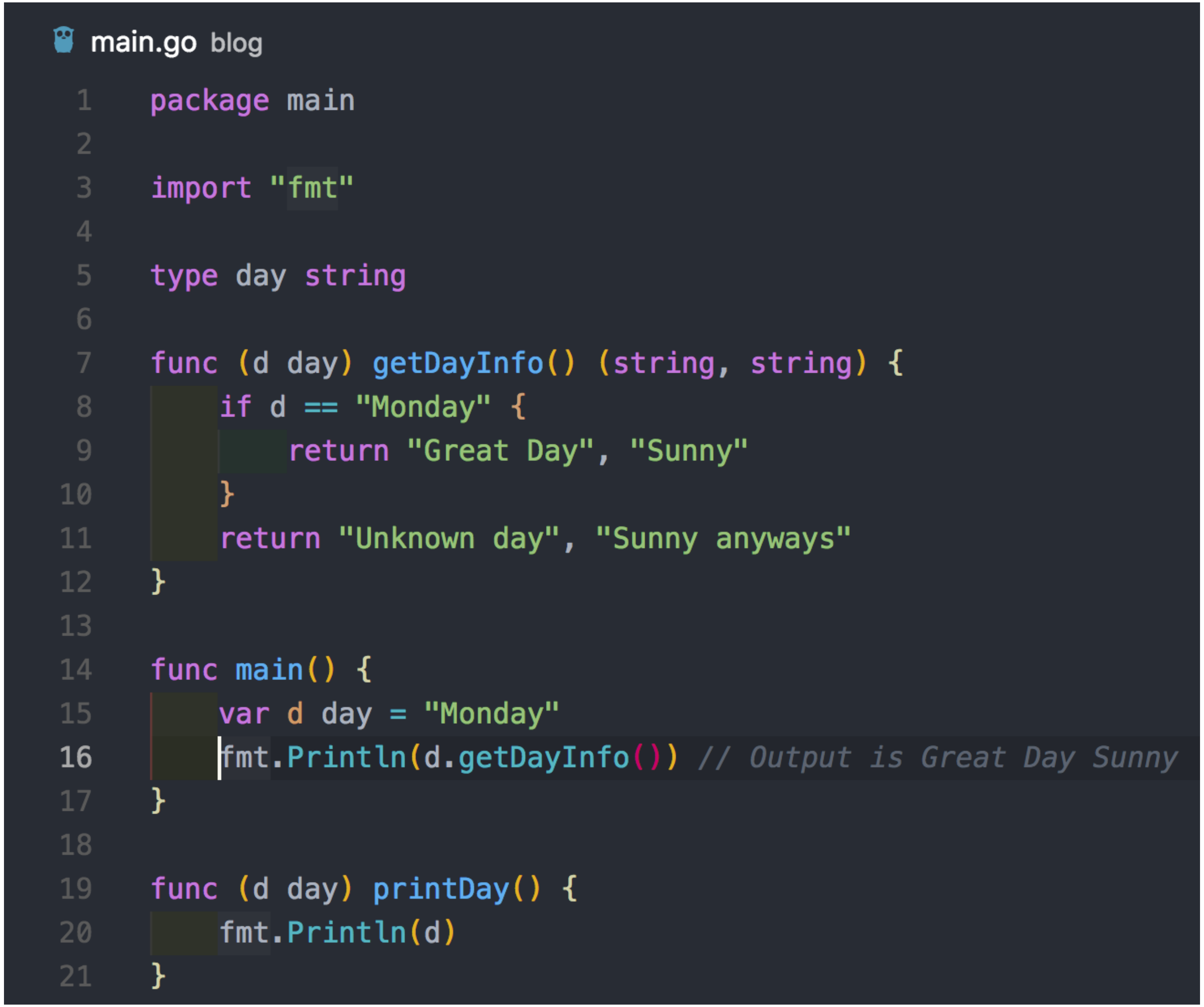


```go
package main

import "fmt"

type day string

func (d day) getDayInfo() (string, string) {
	if d == "Monday" {
		return "Great Day", "Sunny"
	}
	return "Unknown day", "Sunny anyways"
}

func main() {
	var d day = "Monday"
	fmt.Println(d.getDayInfo()) // Output is Great Day Sunny
}

func (d day) printDay() {
	fmt.Println(d)
}
```

Break + example Go code

- I’m guessing few of you have used Go
    - But do you understand it?

- Where does code start?
    - **main()**
- What is the type of d?
    - **day which is a string**



```go
package main

import "fmt"

type day string

func (d day) getDayInfo() (string, string) {
    if d == "Monday" {
        return "Great Day", "Sunny"
    }
    return "Unknown day", "Sunny anyways"
}

func main() {
    var d day = "Monday"
    fmt.Println(d.getDayInfo()) // Output is Great Day Sunny
}

func (d day) printDay() {
    fmt.Println(d)
}
```

Outline

- Course Goals
- When should you use C and C++?
- **Review of Class Topics**
- What’s next?

What did we learn in CS211?

- In reverse order:
  - Game Design
  - C++ Programming
  - C Programming
  - Unix Shell

Game Design

- Model, View, Controller concept
  - **Model** handles the program state
  - **View** displays information based on the state
  - **Controller** modifies the state based on user input

- Breaking a system up into these three parts enables more robust, testable code
  - Applicable to any interactive program, not just games

C++ Programming

- Object Oriented Programming
  - Using objects and methods
  - Creating our own Classes

- Encapsulation
  - Internal state should be private
  - Only expose operations that maintain validity of our internal state

- Resource Acquisition Is Initialization (RAII)
  - Wrap resources in an object
  - Allocate when constructed and deallocate when automatically destructed

C Programming

- C syntax and structure
  - If, while, for
  - Functions and return values
  - Headers and Source files

- Types and Variables
  - Name, Object, Value
  - Type determines the kind of value and size of object

z: 5

- Memory management
  - Stack, Data, and Heap segments
  - When to `malloc()` and `free()` and possible errors

Unix Shell (a.k.a. Linux terminal)

- SSH access to remote machines
  - This will be a recurring need in future classes

- Interacting with files and programs
  - cd, ls
  - Relative and absolute paths
  - Providing flags to programs and looking up documentation

Recommendation: don’t forget about Unix

- Keep playing around with Unix shell
  - Incredibly useful tool for software development and productivity

  - Several options
    - Native MacOS
    - Windows Subsystem for Linux (WSL)
    - Linux installed in a virtual machine

Outline

- Course Goals

- When should you use C and C++?

- Review of Class Topics

- **What’s next?**

More CS classes!

- CS211 is a pre-requisite for CS213
  - I’ll see a lot of you this January 😊
- CS111, CS150, and CS211 are the “programming classes”
  - Teach you how to program
  - Teach you programming languages
- Future classes in CS are “computer science classes”
  - Teach you how to understand computation and computers
  - How do we use computers to understand and effect our world
    - You’ll write programs along the way

New languages

- “Wait, but I only know like four programming languages?!!”
    - Learning others will be up to you

- The same ideas you’ve already learned will apply
    - Types and Imperative Programming
    - Functional Programming
    - Debugging and Testing

- Lots of great guides online for popular languages

Full-Stack Programming

- A benefit to being a “computer scientist” versus “knowing a programming language”
    - Our curriculum teaches you multiple different parts of the software stack

- You can understand front-end (user-facing) software
    - Probably something like Python or Javascript

- You can understand back-end (software-facing) software
    - Probably something like C++

Plenty More Testing and Debugging

- If you’re going to do a lot of programming, debugging is the most useful skill
  - You get better with lots of practice

- Learning to test your code will help you be more successful
  - Especially on big projects

Outline

- Course Goals
- When should you use C and C++?
- Review of Class Topics
- What’s next?