Interlude

Why Object Oriented Programming?

Announcements for This Lecture

This Week

- Today is an **Interlude**
  - **Nothing today is on exam**
  - "Big Picture" lecture; may still be very helpful to you
- Thursday is **Recursion**
  - Hardest topic in course
  - But will try to make it easy
  - Be there or be LOST
- Then the class gets easier…

Announcements

- Assignment 1 Resubmissions
  - 181 out of 195 have a 10
  - Today is absolute last day
- Assignment 2 is graded
  - Solution posted in CMS
  - Mean 12.8/15; Median 13
- Assignment 3 due today
  - Turn in by Midnight
  - Will be graded by weekend

A Short History of Programming: First Generation

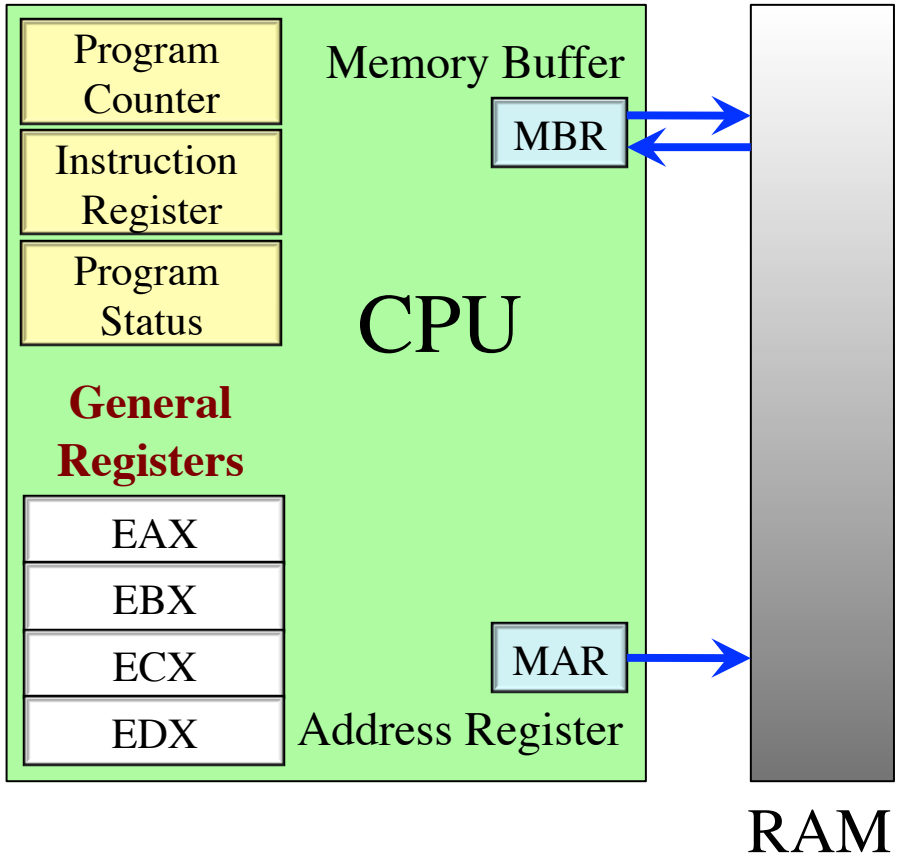
A Short History of Programming: Assembly Language

```
.data  # start of data

x:
  .long   1
  .long   5
  .long   2
  .long   18

sum:
  .long 0

.text  # start of code

.globl _start
```

```
_start:
   movl $4, %eax # EAX is counter
                 # for number of
                 # words to sum
   movl $0, %ebx # EBX stores sum
   movl $x, %ecx # ECX points to
                 # next item to sum
top:
   addl (%ecx), %ebx
   addl $4, %ecx # move to next
   decl %eax     # decr. counter
   jnz top       # if counter not
                 # 0, then loop
done:
   movl %ebx,sum # done, store
                 # result in "sum"
```

A Short History of Programming: Assembly Language

- Commands correspond to machine instructions
  - Close to machine language (no need to compile)
  - Programming "on the iron"
- Lot of load/store commands
  - `movl` from previous slide
  - **Registers** hold the data (like a variable)
  - Moving between RAM and registers is tedious
- Hard to write large programs



"High Level" Languages

- Language that is abstracted from the computer
  - Working with a **metaphor**, not **on the iron**
  - "Folders" are actually a metaphor of a metaphor
- Compiles (or translates) to assembly
  - And then down to pure machine language
  - One high-level line may be many lines of assembly
- Today there are many languages to choose from
  - We cannot agree on which metaphors are best
  - A lot of CS is **coming up with these metaphors**

A Short History of Programming: BASIC

```
10 INPUT "What is your name: ", U$
20 PRINT "Hello "; U$
30 INPUT "How many stars do you want
40 S$ = ""
50 FOR I = 1 TO N
60 S$ = S$ + "*"
70 NEXT I
80 PRINT S$
90 INPUT "Do you want more stars? ", A$
100 IF LEN(A$) = 0 THEN GOTO 90
110 A$ = LEFT$(A$, 1)
120 IF A$ = "Y" OR A$ = "y" THEN GOTO 30
130 PRINT "Goodbye "; U$
140 END
```

Very complex in assembly

- Draw String on monitor
- Wait for keyboard input
- Convert input to String
- Put String in variable U$

A Short History of Programming: BASIC

- **The Metaphor**
  - Commands have line numbers
  - Process the lines in order
  - GOTO, NEXT statements can jump forward or back
- Made programming easy!
  - On every computer in 80s
  - Primary hobbyist language
- But code was **monolithic**
  - Use a single number ordering
  - Code all fits in a single "file"
  - Large programs still hard

```
10 REM  Sample BASIC Program
20 REM  Counts To Ten
30 REM
40 PRINT "I am a BASIC program"
50 PRINT "that counts to ten."
60 PRINT
70 FOR I=1 TO 10
80 PRINT I
90 NEXT I
100 PRINT
110 PRINT "Thanks for running me."
120 END
```

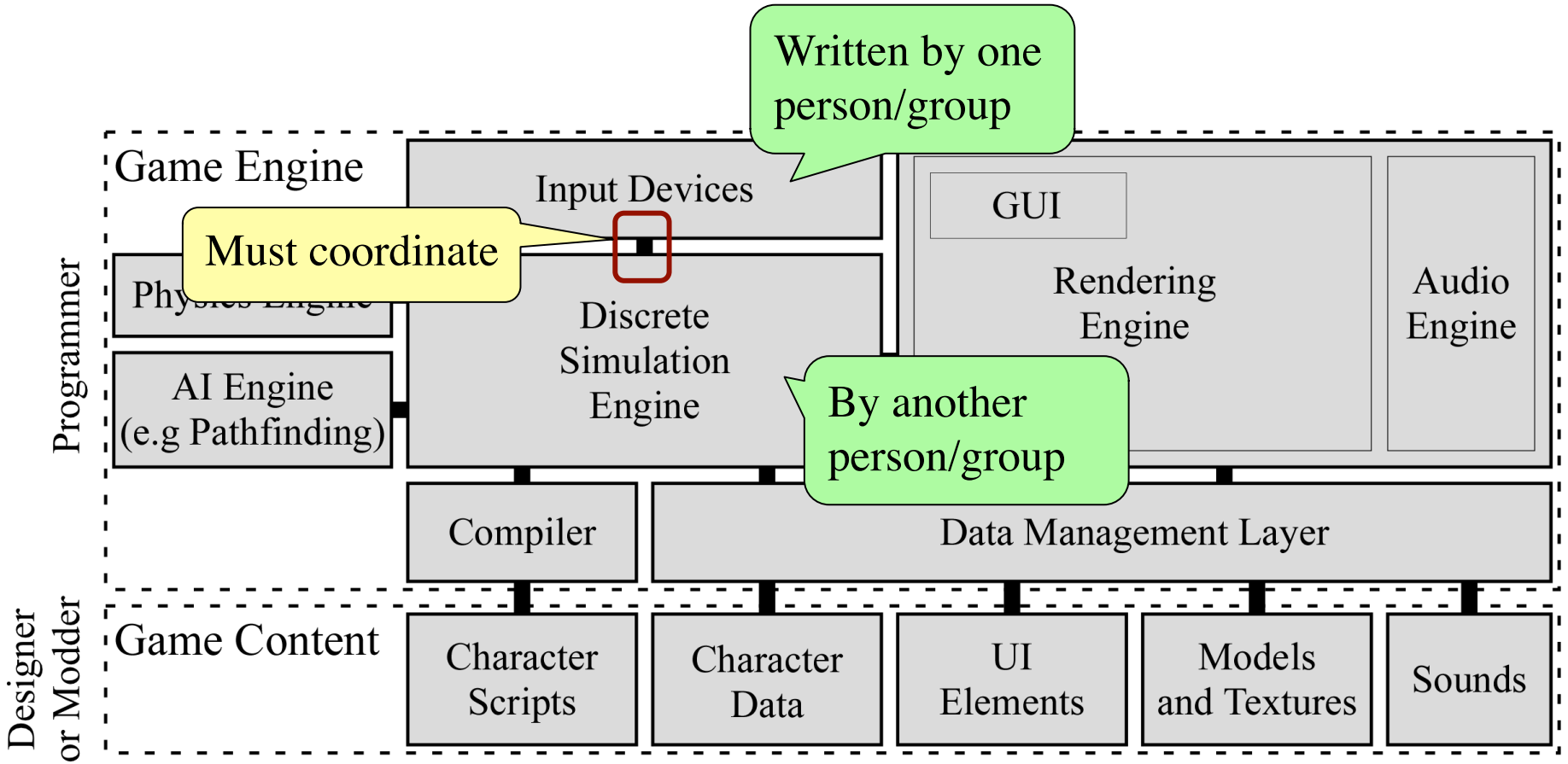
Computer Game Development

Credits: Planetfall (1983)

Steve Meretzky

Credits: Portal (2007)

Challenge: Breaking Up Software



A Short History of Programming: C (and its descendants)

```
int main() {
  int var1 = 1;
  int var2 = 2;
  printf("max(%d,%d)=%d",
         var1, var2,
         max(var1,var2));

  exit 0;
}
```

```
int max(int a, int b) {
   if (a > b) {
       return a;
   }
   return b;
}
```

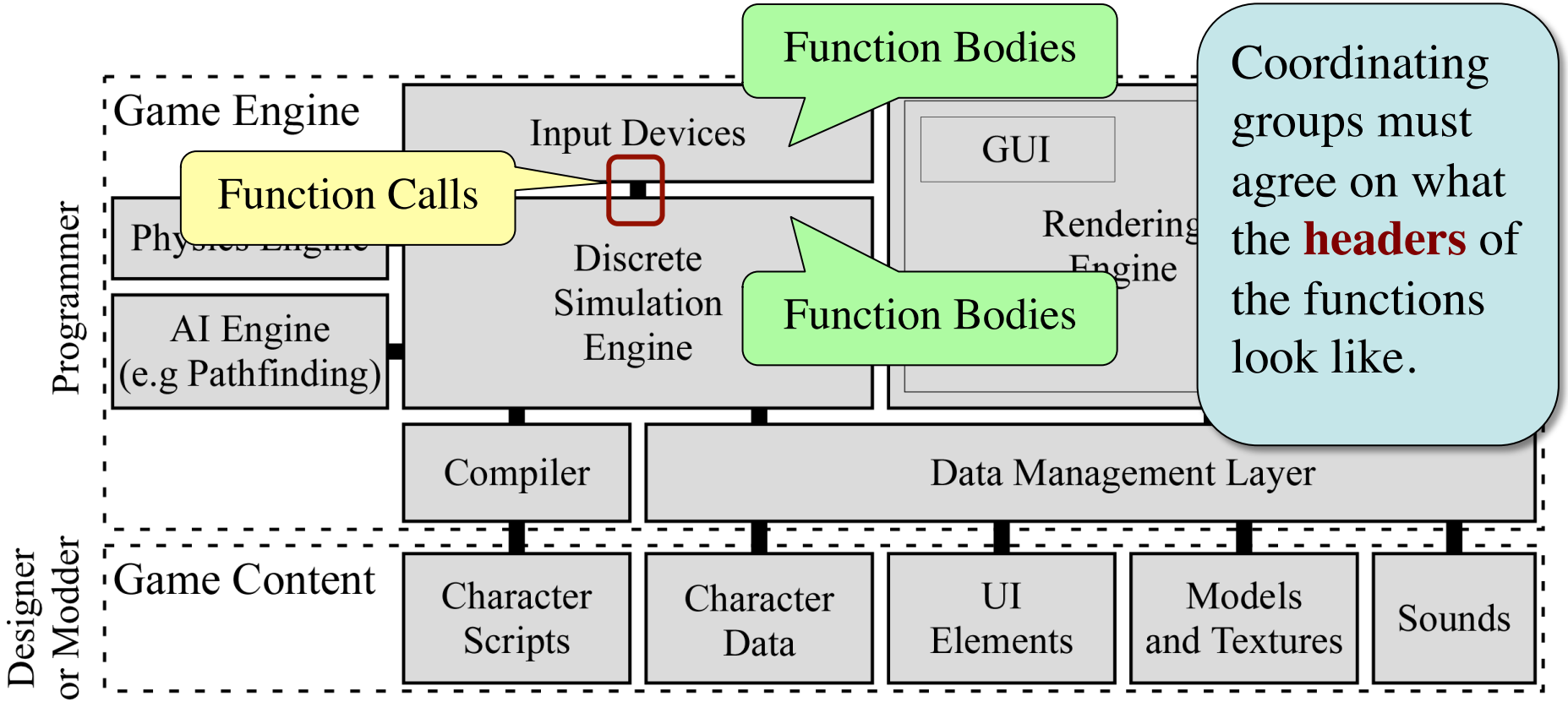
A Short History of Programming: C (and its descendants)

- **The Metaphor**
  - Organized via **functions** (with side-effects)
  - **Procedures**: functions that return a value of type void
  - Function declarations can spread over multiple files
- Supports **modularity**
  - Each programmer works to define a function (or many)
  - Can *call* a function made by another programmer

```
int max(int a, int b) {
   if (a > b) {
       return a;
   }
   return b;
}
```

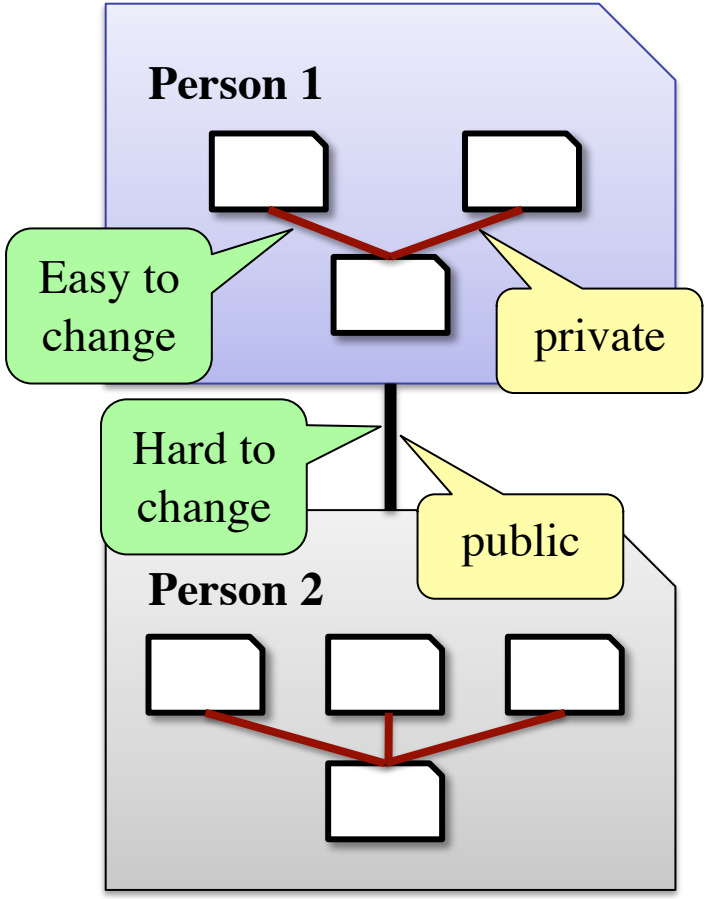
C is like programming in Java, but only using static methods

Challenge: Breaking Up Software



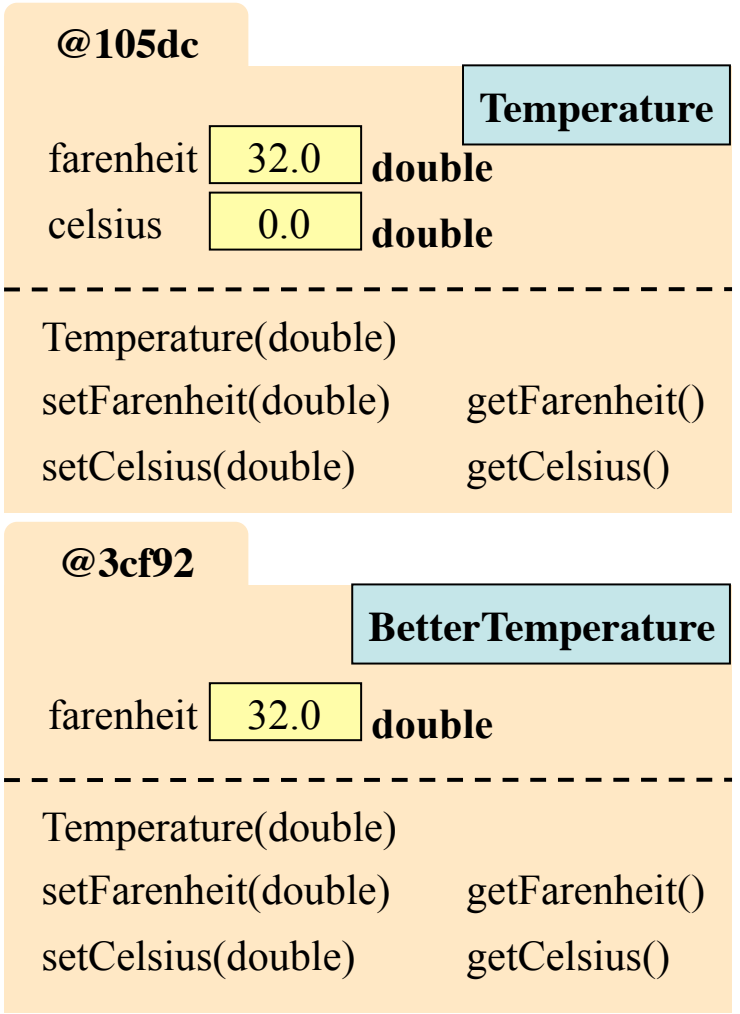
Encapsulation: Reducing Dependencies

- Development is iterative
  - You are always making changes (to improve your software)
- Coordination hurts iteration
  - Others are calling your functions
  - If you change your functions work, their code may no longer work
  - **Example**: Our test code in A1
- **Encapsulation**: limit what the other programmers can access in your code
  - If cannot access, changes are okay



Encapsulation is the Primary Purpose of Object Oriented Programming

- Encapsulation applies to both code and data!
  - Data in JFrame is hidden
  - Could you tell if it was changed in later versions?
- Encapsulation in Java
  - Make all data private
  - Force data access through the methods (getters/setters)
- Public methods: **the interface**
  - Not allowed to change the interface without permission



Object Oriented Descendants of C

Direct Descendants

- **C++**
  - Optional OO features for C; a superset of C language
  - High performance language
  - But kludgy and messy
- **Objective-C**
  - Also superset of C language
  - Trades performance for collaboration features
  - Used in OS X, iPhone

"Cousin" Languages

- **Java**
  - "Reimagining" of C++
    - Fixes a lot of the problems
    - A little easier for beginners
  - Compile one, run anywhere
    - Ideal language for the web
    - Others are platform specific
- **C#**
  - "Reimagining" of Java
  - Microsoft specific language

Java for Beginners: Challenges

Advantages

- Very useful language
  - Can make professional-quality programs
  - Standard language for web
- Works on any platform (OS X, Windows, mobile)
- Easier than many alternatives
  - C++ too hard for beginners
  - Others not as powerful

Disadvantages

- The C influence is showing
  - types, functions, void
  - Designed for people who learned the other languages
- Too much to learn at start
  - Cannot program without learning classes/objects first
  - Lots of mysterious keywords

Java for Beginners: Challenges

Advantages

- Very useful language
  - Can make professional-quality programs
  - Standar
- Works on
  (OS X, W
- Easier than many alternatives
  - C++ too hard for beginners
  - Others not as powerful

Disadvantages

- The C influence is showing
  - s, void
  - eople who
  - r languages
  - arn at start
  - Cannot program without learning classes/objects first
  - Lots of mysterious keywords

What We Really Want:
- Language with modern OO features
- As easy to pick up as BASIC was

A Short History of Programming: Is Python Best for Beginners?

- **The Metaphor**
  - Mainly functions (like C)
  - Classes and objects exist, but are all optional
  - No distinction between the "interactions pane" and file
- An **untyped** language
  - Language checks the types for you automatically
  - Both a plus and a minus
- CS 1110 starting this Fall

```
def greet(name):
  print 'Hello', name

greet('Jack')
greet('Jill')
greet('Bob')
```