

Welcome to CS 12

Wagner Truppel
Lecturer, Dept. of Computer
Science & Engineering
UC Riverside

wagner@cs.ucr.edu
<http://www.cs.ucr.edu/~wagner>

<http://www.cs.ucr.edu/cs12>

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 1

Today's Topics

- Course information
 - ◆ Structure is much like CS 10
 - ◆ No iLearn/BlackBoard, though
 - ◆ Linux, not Visual C++
 - ◆ Guest talks
 - ◆ Email & Mailing List
 - ◆ HWs and Quizzes
 - ◆ Lectures don't cover everything
 - ◆ Lectures also cover stuff not in the book
 - ◆ Regarding my slides
- A little puzzle to get you warmed up
 - ◆ How many people ahead of you in a movie theater line?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 2

This Week's Topics

- Review of some basic ideas in CS
 - ◆ Number Systems
 - ◆ Computer Organization
 - ◆ Computer Programming
- Structured Programming concepts
 - ◆ Top-down approach
 - ◆ Procedural Programming
 - ◆ Bottom-up approach
 - ◆ Brief overview of Object-Oriented Programming

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 3

Number Systems

- The difference between a **quantity** and its **numerical representation**
- Eleven dots:
 - 11 ?
 - I say 23...
 - Or, if you prefer, 21...
 - Also, 15.
 - And, of course, *also* 11.
 - How come ?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 4

Number Systems

- The difference between a **quantity** and its **numerical representation**
- Eleven dots:
 - 2 and 3 • ⇨ 23_4
 - 2 and 1 • ⇨ 21_5
 - 1 and 5 • ⇨ 15_6
 - 1 and 1 • ⇨ 11_{10}
- 23_4 , 21_5 , 15_6 , and 11_{10} are different numerical representations of the **same quantity**

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 5

Number Systems

- Eleven dots:
 - 2 and 1 • ⇨ 21_5
 - 1 and 1 • ⇨ 11_{10}
- The subscripted numbers are called the **base** of the number system in question
- = 21 in base 5 = 11 in base 10
- Base b uses **only** the digits 0, 1, 2, ..., (b-1)
- Examples:
 - Base 10 uses 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
 - Base 3 uses 0, 1, 2

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 6

Number Systems

■ Example: Base 3

	↪	0_3
.	↪	1_3
..	↪	2_3
...	↪	10_3
... .	↪	11_3
... ..	↪	12_3
... ...	↪	20_3
...	↪	21_3
...	↪	22_3
...	↪	??

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 7

Number Systems

■ Example: Base 3

... .. ↪ ?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 8

Number Systems

■ Example: Base 3

... .. ↪ 30_3 ?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 9

Number Systems

- Example: Base 3
... .. $\Rightarrow 30_3$? No !

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 10

Number Systems

- Example: Base 3
... .. $\Rightarrow 30_3$? No !
..... = 1 group of nine
 = 1 group of 3 squared
 = 100_3
- Why **3 squared**?
- Just like $100_{10} = 1$ group of **10 squared**
- Just like $1000_{10} = 1$ group of **10 cubed**

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 11

Number Systems

- $21843_{10} =$
 $3 \times 10^0 + 4 \times 10^1 + 8 \times 10^2 +$
 $1 \times 10^3 + 2 \times 10^4$
- $21201_3 =$
 $1 \times 3^0 + 0 \times 3^1 + 2 \times 3^2 +$
 $1 \times 3^3 + 2 \times 3^4 =$
 $1_{10} + 0_{10} + 18_{10} + 27_{10} + 162_{10}$
 $= 208_{10}$

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 12

Number Systems

- 2 1 8 4 3_b
 b⁰ units
 b¹ units
 b² units
 b³ units
 b⁴ units
- Remember to read from **right to left** !
- Remember that the **exponents start at 0** !

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 13

Number Systems

- **Important: Base 2 (Binary System)**

↔	0000 ₂	↔	0 ₁₀
•	↔ 0001 ₂	↔	1 ₁₀
••	↔ 0010 ₂	↔	2 ₁₀
•••	↔ 0011 ₂	↔	3 ₁₀
••••	↔ 0100 ₂	↔	4 ₁₀
•••••	↔ 0101 ₂	↔	5 ₁₀
••••••	↔ 0110 ₂	↔	6 ₁₀
•••••••	↔ 0111 ₂	↔	7 ₁₀
••••••••	↔ 1000 ₂	↔	8 ₁₀

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 14

Number Systems

- The digits 0₂ and 1₂ are called **Binary digits** or **bits**
- 8 bits form one **byte**
- Why should we care about **base 2** ?
- Because:
 - ◆ Electricity **not** flowing ↔ 0
 - ◆ Electricity **flowing** ↔ 1
- Thus, we can represent numbers with electrical devices
- And that's what an electronic computer is !

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 15

Computer Organization

- Ok, there's more to a computer than just that...
 - ◆ Memory
 - ◆ ALU (**A**rithmetic and **L**ogical **U**nit)
 - ◆ CPU (**C**entral **P**rocessing **U**nit)
 - ◆ Clock
 - ◆ I/O devices
 - ◆ And more...

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 16

Computer Programming

- What *really* happens when you write and compile a computer program ?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 17

Computer Programming

```
for (int i = 0; i < a.length; i++)  
{ a[i] = 2i + 1; }  
...
```

Source code written in some high-level language (C, C++, etc)

Compiler

Libraries (already compiled into machine code)

Compiled code for your program (object code)

Linker

Executable code for your program

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 18

Computer Programming

- Where does your program reside ?
 - ◆ When it's not running, it's stored in the hard disk
 - ◆ When it's running, it's stored in memory
 - ◆ Why ?
- What really happens when you run a computer program ?

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 19

Memory Organization

- Memory contents
- Addresses
- Program counter

00010110
00010101
00010100
00010011
00010010
00010001
00010000
00001111
00001110
00001101
00001100
00001011
00001010
00001001
00001000
00001111
00001110
00001101
00001100
00000111
00000110
00000011
00000010
00000001
00000000

extra memory reserved for your program

your program's machine code

pc

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 20

Computer Programming

- What really happens when you run a computer program ?
- We'll continue to look into that question in the next few lectures...
- And you'll learn the complete story when you take *CS 61: Machine Organization and Assembly Language Programming*

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 21

Structured Programming

- Split your problem into simpler parts then solve each part separately
- Recognize common parts and solve them only once
- Top-down approach
- Bottom-up approach
- Procedural programming
- Object-oriented programming

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 22

Top-down approach

- Break the problem down from the top into smaller and smaller parts
- More intuitive than bottom-up approach (more on this later)
- **Functions** are the natural tools to use (as opposed to *objects*)

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 23

Top-down approach

- Example: How to do well in CS 12
 - ◆ Must do well in the lecture part
 - * Must attend the lectures
 - * Must do the lecture readings ahead of time
 - * Must reserve some time to study
 - ◆ Must do well in the lab part
 - * Must attend the labs
 - * Must do the lab readings ahead of time
 - * Must reserve some time to practice how to write and compile code

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 24

- `To_do_well_in(course)`
 - ◆ `Must_do_well_in(lecture)`
 - ◆ `Must_do_well_in(lab)`
- `Must_do_well_in(component)`
 - ◆ `Must_attend(component)`
 - ◆ `Must_do_readings_ahead_of_time()`
 - ◆ `Must_reserve_time_to_study(component)`
- `Must_reserve_time_to_study(component)`
 - ◆ `If (component is lecture)`
 - ✦ `Reserve_time_to_study()`
 - ◆ `Else // (component is lab)`
 - ✦ `Reserve_time_to_practice()`
- **Pseudo-code**: code that is closer to a human language than to a programming language, but just as precise as a programming language

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 25

Bottom-up approach

- Figure out the smaller parts first, then put them together
- Less intuitive than top-down approach because it's not always easy to figure out what parts you need
- **Objects** are the natural tools to use (as opposed to *functions*)
- We'll get back to Bottom-up when we cover Object Oriented Programming

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 26

Bottom-up approach

- Example: drawing program
- You'll probably need triangles, squares, circles, etc, even if you're not sure how they will all fit together in the end
- All of those things are *shapes*
- **Every** shape has a position **and** needs to be drawn
- Base *class* **Shape**
 - ◆ `x_pos, y_pos`
 - ◆ `draw()`

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 27

Bottom-up approach

- class **Triangle** is a **Shape**
 - ◆ a, b, c // *sides*
 - ◆ functions specific to a triangle

- class **Square** is a **Shape**
 - ◆ len // *length of side*
 - ◆ functions specific to a square

- class **Circle** is a **Shape**
 - ◆ r // *radius*
 - ◆ functions specific to a circle

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 28

UML: Unified Modeling Language

```

classDiagram
    class Shape {
        int x_pos
        int y_pos
        void draw()
    }
    class Triangle {
        int a
        int b
        int c
        // triangle functions
    }
    class Square {
        int len
        // square functions
    }
    class Circle {
        int r
        // circle functions
    }
    Shape <|-- Triangle
    Shape <|-- Square
    Shape <|-- Circle
    
```

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 29

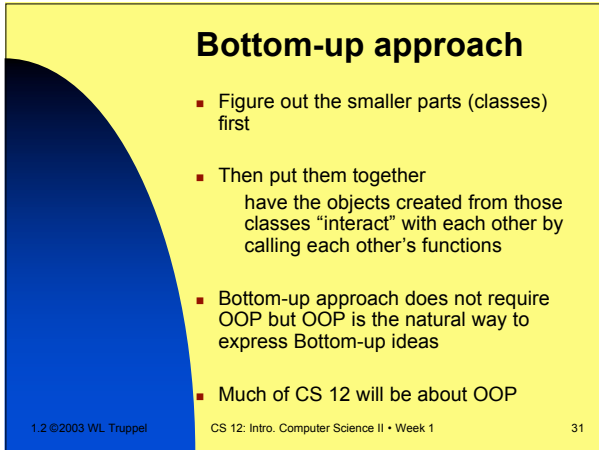
Objects and Classes

- Just as you can define an **int** variable
`int x, n, radius;`

- You can also define a more complicated variable, an *object* of a given *class*:
`Triangle t = Triangle(3, 4, 5);`
`Square sq = Square(10);`
`Circle c = Circle(7);`

- The cool thing is that because these objects are also **Shape** objects, they all know how to draw themselves since you defined a `draw()` function in the **Shape** class:
`t.draw(); // draws the triangle t`
`sq.draw(); // draws the square sq`
`c.draw(); // draws the circle c`

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 30



Bottom-up approach

- Figure out the smaller parts (classes) first
- Then put them together
have the objects created from those classes "interact" with each other by calling each other's functions
- Bottom-up approach does not require OOP but OOP is the natural way to express Bottom-up ideas
- Much of CS 12 will be about OOP

1.2 ©2003 WL Truppel CS 12: Intro. Computer Science II • Week 1 31
