

# CS 10 Review (Elementary C++)

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

[wagner@cs.ucr.edu](mailto:wagner@cs.ucr.edu)  
<http://www.cs.ucr.edu/~wagner>

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

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

---

---

---

---

---

---

---

---

## Announcements I

- Email us using **only** your UCR address
  - ◆ If you're not a CS or ENG major, use the account created for you in the lab
  - ◆ Example: `cs12ab@cs.ucr.edu`
  - ◆ TAs and I will **not** reply to messages with non-UCR addresses
- Email with missing information
  - ◆ Please make your messages easy for us to deal with so that you can get a prompt reply
  - ◆ Include your **complete name**, **student ID**, the **course you're taking**, your **lecture section**, and your **lab section**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 2

---

---

---

---

---

---

---

---

## Announcements II

- We're **not** using ilearn/Blackboard
- Class mailing list
  - ◆ See the course web page for directions
  - ◆ You **must** add yourself to the list
  - ◆ Read it frequently!
  - ◆ You can only sign in with a UCR address
  - ◆ Use the list to ask questions and interact with us and with other CS 12 students
  - ◆ **No** postings containing code

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 3

---

---

---

---

---

---

---

---

## Today's Topics

- Structured Programming
  - ◆ Top-down approach
  - ◆ Bottom-up approach
- Review of basic C++
  - ◆ Elementary data types
  - ◆ Variables, operators
  - ◆ Assignment op, automatic conversion
  - ◆ Flow of control
  - ◆ Functions
    - \* main()
    - \* Call by value
    - \* Call by reference
    - \* Overloading basics

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 4

---

---

---

---

---

---

---

---

## 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 5

---

---

---

---

---

---

---

---

## 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*)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 6

---

---

---

---

---

---

---

---

## 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 7

---

---

---

---

---

---

---

---

- 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(comp)
  - ◆ If (comp is lecture)
    - \* Reserve\_time\_to\_study()
  - ◆ Else // (comp 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 8

---

---

---

---

---

---

---

---

## 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 9

---

---

---

---

---

---

---

---

### 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()`

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 10

---

---

---

---

---

---

---

---

### 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 11

---

---

---

---

---

---

---

---

### 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
```

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 12

---

---

---

---

---

---

---

---

## 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 b/c 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*

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 13

---

---

---

---

---

---

---

---

## 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

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 14

---

---

---

---

---

---

---

---

## Review of basic C++

- Variables, operators
- Assignment op, automatic conversion
- Type casting
- Flow of control
- Functions
  - ◆ `main()`
  - ◆ Call by value
  - ◆ Call by reference
  - ◆ Overloading

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 15

---

---

---

---

---

---

---

---

## Variables

- Are “containers” for values
- Must be declared before they can be used
- Are type-specific: an **int** variable is not the same as a **float** variable, for example
- Values are stored in memory cells, occupying a machine dependent amount of bytes.
  - ♦ Eg: typically, an **int** takes **4 bytes**
- Examples:  
int n = 3;  
float radius = 0.54;

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 16

---

---

---

---

---

---

---

---

## Operators I

- Let you manipulate your variables
- Arithmetic operators
  - ♦ things like +, -, \*, /, ++, --
- ♦ Egs: **(a + 3)\*(b++)**
- ♦ **n++** ⇨ use **n**, then increment **n** by 1
- ♦ **++n** ⇨ increment **n** by 1, then use **n**
- ♦ Watch out for integer division !
  - \* 3 / 6 evaluates to **ZERO**
  - \* 3.0 / 6.0 evaluates to 0.5
  - \* 3 / 6.0 evaluates to 0.5 ⇨ auto conversion
  - \* 3.0 / 6 evaluates to 0.5 ⇨ auto conversion

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 17

---

---

---

---

---

---

---

---

## Operators II

- Logical operators
  - ♦ things like ==, !=, >, <, <=, >=, &&, ||
  - ♦ Evaluate to a **bool** (**true** or **false**)
  - ♦ **3 > 5** evaluates to **false**
  - ♦ **3 != 5** evaluates to **true**
  - ♦ **x == 5** result depends on the value of **x**
  - ♦ **&&** is the **AND** operator
    - \* **(3 > 5) && (3 != 5)** results in **false**
  - ♦ **||** is the **OR** operator
    - \* **(3 > 5) || (3 != 5)** results in **true**
  - ♦ Don't confuse **==** with **=** (*assignment op*)

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 18

---

---

---

---

---

---

---

---

### Operators III

- Assignment operator
  - ♦ `int z = 5;` declares `z`, sets its value to **5** and then returns that value
  - ♦ Thus, for eg, `(z = 7) == 2` returns **false**
  - ♦ Don't confuse `==` with `=`

```
if (z = 3)
{ /* do something */ }
```

is valid  
`(z = 3)` returns **3** which is interpreted as **true**  
only **0** and **0.0** are interpreted as **false**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 19

---

---

---

---

---

---

---

---

### Automatic conversion

- If necessary, **shorts** are converted to **ints**, **ints** to **longs**, **longs** to **floats**, and **floats** to **doubles**
- Example:
  - ♦ `char c = 'Z';`
  - ♦ `3.6 * (c + 5);`
  - ♦ `c` is converted to an **int**, added to **5**, the result is converted to a **float** and then multiplied by **3.6**. The result is a **float**.
  - ♦ This is an example of **bad** code, for at least **two** reasons. Can you guess them?
- **Any non-zero** numerical result appearing in a **boolean** expression is converted to **true**; **0** and **0.0** are converted to **false**

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 20

---

---

---

---

---

---

---

---

### Flow of control

- Things like **if**, **while**, **do**, **for**
- `if (boolean expression)`  
`{ /* do something */ }`
- `if (boolean expression)`  
`{ /* do something */ }`  
`else`  
`{ /* do something different */ }`
- `while (boolean expression)`  
`{ /* do something */ }`

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 21

---

---

---

---

---

---

---

---

### Flow of control

- **do**  
{ /\* something \*/  
**while** (boolean expression)
- **for** (init; condition; modifier)  
{ /\* do something \*/ }
- **for** (int i = 0; i < MAX; i++)  
{ cout << (2 \* i) << endl; }
- Watch out for ; and , inside the ()

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 22

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 23

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 24

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 25

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ? Yes
- `8 = x;` // and this ?

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 26

---

---

---

---

---

---

---

---

### L-values and R-values

- Suppose we have `int x = 3;`
- `x == 5;` // does this make sense ? Yes
- `x = 5;` // and this ? Yes
- `8 == x;` // how about this ? Yes
- `8 = x;` // and this ? No ! Why not ?
- Because **8** is a **literal** value

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 27

---

---

---

---

---

---

---

---

## L-values and R-values

- `8 = x; // illegal !`
- You can't assign values to a literal value
- Literal values are **not L-values** - you cannot have them on the **left** side of an assignment
- Values that **can** be on the **left** side are called **L-values**, those which can be on the **right** side are called **R-values**.
- The distinction is important when you consider functions and their return values.

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 28

---

---

---

---

---

---

---

---

## Functions I

- **main()**
  - ◆ Every program **must** have one
  - ◆ It's the entry point of execution of your program
  - ◆ **Must** return an integer
  - ◆ **int main()**
  - ◆ Should return **0** if no problems are encountered
  - ◆ Should return an error code if problems are encountered

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 29

---

---

---

---

---

---

---

---

## Functions II

- `// function declarations`
- `void drawMonster(int monster_id, int x, int y)`
- `void moveMonster(int mnstr_id, float dx, float dy)`
- `// function definition`
- `int max(int a, int b)`

```
{  
    if (a > b)  
        return a;  
    else  
        return b;  
}
```
- `// function call`
- `int z = max(r, s);`

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 30

---

---

---

---

---

---

---

---

### Functions III

- Call by value
  - The **value** of the argument is **copied** into a local variable inside the function
  - void f(int x, float v)**
- Call by reference
  - The **address in memory** of the argument is passed to the function
  - Lets you change the value of what's being passed to the function, from inside the function
  - void f(int & x, float & v)**
  - &** means "address of"

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 31

---

---

---

---

---

---

---

---

### Calls by value/reference

- Important example:

```
void swap(int x, int y)
{
    int temp = x;
    x = y;
    y = temp;
}
```
- Does **not** work!
- Inside the function body, **x** and **y** are **local** variables whose initial values are **copies** of the arguments.

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 32

---

---

---

---

---

---

---

---

### Calls by value/reference

- Important example:

```
void swap(int &x, int &y)
{
    int temp = x;
    x = y;
    y = temp;
}
```
- Now it **does** work!
- Inside the function body, **x** and **y** are **not** local variables; they represent the actual arguments
- When the function exits (returns), **x** and **y** have their values exchanged
- We'll see in detail how **call by reference** is accomplished when we study **activation frames** next week

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 33

---

---

---

---

---

---

---

---

## Function overloading

- **Signature** of a function
  - ◆ **Name** of the function
  - ◆ **Number** of arguments
  - ◆ **Types** of those arguments
  - ◆ **Order** of those arguments
- The signature of a function includes **neither** the function's return type **nor** the **names** of the params
- It's ok to define many functions with the same name, as long as they have different **signatures**:
- Examples:

```
void swap(int & x, int & y)
void swap(char & x, char & y)
void swap(float & x, float & y)
void swap(double & x, double & y)
```
- This gets boring, is error-prone, and duplicates code...
- Is there a way to define a single swap function that applies to *every* type ?
- Yes ! That's done with **Templates**.

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 34

---

---

---

---

---

---

---

---

## Reminder

- If you have not read them yet, make sure to read handouts 1 & 2:  
**intro\_comp\_org.pdf**  
**stack\_frames.pdf**
- For next lecture read **sections 10.1, 10.2, and 10.3** of Savitch's book

©2003 WL Truppel CS 12: Intro. Computer Science II • Lecture 2 35

---

---

---

---

---

---

---

---