CSCI/CMPE 1370
Engineering Computer Science I
(for CSCI/CMPE majors, minors)

Beiyu Lin

Zoom Meeting ID: 758 217 1203

Email: beiyu.lin@utrgv.edu

Computer Science I

- Goal #1:
  - Introduce you to the ideas behind computer science
- Goal #2: Learn two practical skills:
  - How to design solutions to different problems
  - How to implement those solutions writing computer programs in C++
- **Expect to spend at least as much time outside of class as you spend in class** (ideally at least **3 hours** per **each hour** spent in the classroom)

Computer Science I

- Course concepts and skills are broken up into 5 modules (they are highly cumulative!)
- For each module…
  - I will introduce *concepts* and show *examples*
  - You will have to read the *book*, review the material discussed in class, and practice with the examples provided
  - You will test yourself: are you prepared?
  - I will assess your knowledge (assignments, exercises, and exams)

Computer Science I

- **Learning** requires a little knowledge and a **lot of practice:**
  - *Read the textbook* and do the activities to get the knowledge
  - download the examples discussed in class and analyze them thoroughly until you understand them
  - Make small changes, try to predict the new results, and then test them to see if you get what you expected
  - *Discuss with your teammate* what you did for practice

If you do receive some help from classmates, please identify their names in the comment block at the top of your main source file and above the block of code that was shared. If you do not identify the students that helped you with your program, then you are at risk of being identified as copying or plagiarizing.

Course Information

- All course materials and announcements will be available on Blackboard
  - Syllabus
  - Instructor and TA contact information
  - Lecture materials
  - Assignments, labs, review questions
  - Due dates, exam schedule, announcements
- Course announcements and other updates will be handled through Announcements and Calendar
  - You are expected to check your Blackboard course every weekday and at least once on weekends

Lab Section

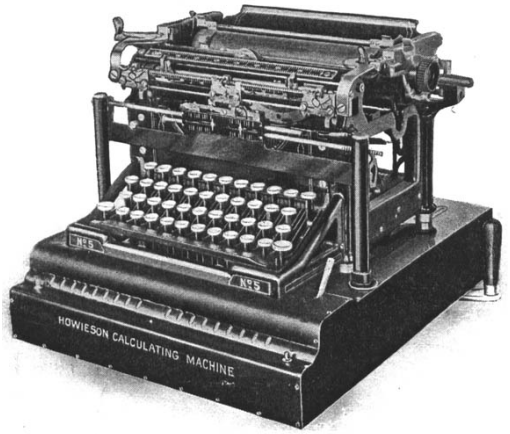
- You must be registered for the co-requisite lab course CSCI/CMPE 1170
- In lab you will:
  - Have to individually analyze code and then discuss the solution(s)
  - Have hands-on time to write code
  - Practice with the concepts presented in lecture
  - Ask questions, the TAs are there to help
  - Ask questions, you are encouraged to discuss amongst yourselves
    - (just make sure you're getting real practice, that is, you do write the programs not just watch somebody else doing it)

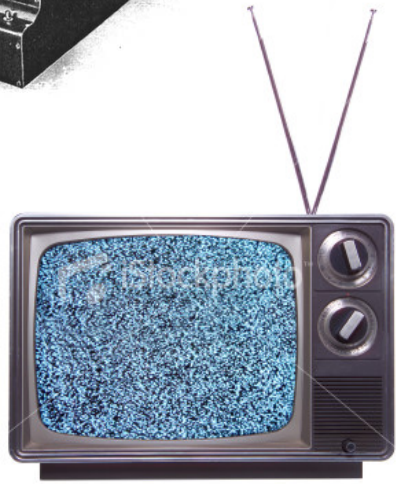
Textbook

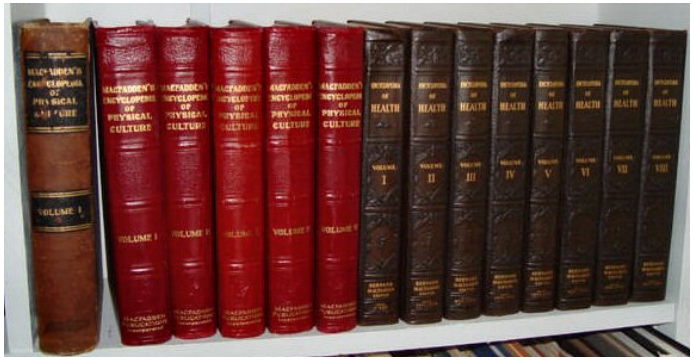
- Zyante Programming in C++ online interactive text
  - Contains interactive activities and practice
  - Allows me to follow your progress
  - $58, online access good through end of semester
    - Can be downloaded and saved
  - Limited internet access? Online payment issues? See me!
  - Information for subscribing is on zybook_subscription.pdf in Miscellaneous https://zybooks.zyante.com/#/zybooks
- Want a physical C++ textbook?
  - Go for it, any one will be fine
  - I'd recommend one with lots of examples

Tools

- The textbook has it's own web-based programming environment
  - Very convenient for exercises
  - We will also be using *Microsoft Visual C++*
  - Integrated Development Environment (IDE)
  - Available for free download
  - You may also use online C++ compilers (like cpp.sh and www.onlinegdb.com)







Programs

- Writing a computer program is simply giving instructions that a computer can follow
  - In this course, we'll be giving instructions in the C++ language
- A program is a sequence of *statements*
  - Each statement tells the computer to do one (or more) *operations*

Language, Syntax and Semantics

- C++ is an *artificial language*
  - Similar to a natural language, only the rules are a lot more strict
  - To use a language, you have to know the syntax (what goes where) and the semantics (what does it mean)
  - In natural language, you can bend things quite a bit
    - i are teach u good!!!1!!11!
  - In a programming language, you have to follow the rules
  - Unlike in natural language, statements in C++ have only one meaning: they tell the computer to do a specific thing

Language, Syntax and Semantics

- Every *statement* in C++ is like a sentence in English
  - Both are made up of words and symbols separated by spaces
  - English syntax rule: sentences end with punctuation
  - C++ syntax rule: statements end with a semi-colon (;)
    - By convention, each statement gets its own line
- Based on this rule, here is the simplest C++ statement:

```
;
```

  - It is a valid statement (follows the rules)
  - It tells the computer to do nothing

Operators

- To tell the computer to do something, a statement can use an *operator*
- Each operator specifies a particular operation
  - Insertion operator (<<): print something, somewhere
  - Addition operator (+): add two things
  - Subtraction operator (-): subtract one thing from another
  - Assignment operator (=): store something, somewhere
- Each of these operators is *binary*
  - They take two *arguments*
  - *Math: y = x + 2, x = 5, y = 7;*
  - *Computer: assume we defined a variable called y;*
    - *I want to assign value 7 to the variable y.*
    - *y = 7;*
    - *Print out the value of y:*
    - *cout << y; ⇔ English: print out the value of y.*

Operators

- Syntax rule: a binary operator needs a *left-hand side (LHS)* argument and a *right-hand side (RHS)* argument
  - Insertion operator: cout << “Hello world!”;
  - English: print out the “hello world!”.
    - LHS: where to print
    - RHS: what to print
  - Addition operator: 5 + 6;
    - LHS/RHS: the two numbers to add
  - Subtraction operator: 19 – 3;
    - LHS: the number to subtract from
    - RHS: the number to subtract
- In general, a statement with a binary operator looks like:

  LHS operator RHS;

The Print Statement

- How does the computer understand this statement?
  - The *insertion operator* (`<<`) instructs the computer to print something: e.g., `cout << "Hello World!";`
    - To do so, it has to be told what to print and where
  - To the right of the operator is the *value* that we want it to print
    - `4` is the literal number 4
    - `"Hello there"` is a literal *string* of characters
  - To the left of the operator is the place we want to print to
    - `cout` specifies that black box on the screen
  - All statements in C++ end with a semi-colon (`;`)

Data

- What are valid arguments?
  - Depends on the operator
  - Most of the time, a piece of *data*
- Four *primitive* data types
  - An integer (whole number) ⇔ integer
    ```
    12
    ```
  - A real number ⇔ data type is called double
    ```
    3.14
    ```
  - A character (surrounded by single quotes)
    ```
    'A'
    ```
  - A string (surrounded by double quotes)
    ```
    "Hello world." "today is Thursday."
    ```

Data

- Addition and subtraction only work with numbers (integers or real numbers s)
- Printing (insertion) works with numbers, characters or strings on the RHS
  - LHS is the place to print, we'll get back to that

Arithmetic Operators

- C++ arithmetic operators:

+	Addition	
-	Subtraction	
*	Multiplication	
/	Division	5/2 = 2 (programming: integer/integer = integer) 5/2.0 = 2.5 (integer / double = double) 5+2+4+5/2.0 => the result is in double type 5 + 2 + 4.0 + 5/2 => the result is in double type
%	Modulus	5%2 = 1

- +, -, *, and / can be used with integral and floating-point data types
- These are *binary operators*
  - They operate on 2 *operands*

Arithmetic Expressions

- Combinations of arithmetic operators and numbers

```
23 + 4 = 27
5 – 6 * 20 = -115
56.882 – (34 / 23) = 55.882
```

```
Order:() has the highest priority
        /, *, % has the second highest priority
        -, +
```

Order of Precedence

- All operations inside of () are evaluated first
- *, /, and % are at the same level of precedence and are evaluated next
- + and – have the same level of precedence and are evaluated last
- When operators are on the same level
  - Performed from left to right (associativity)
- `(3 * 7) - 6 + (2 * 5 / 4) + 6` means
- (21 – 6) + 2 + 6
- (15 + 2) + 6 = 23

`(((3 * 7) - 6) + ((2 * 5) / 4 )) + 6`

Integer vs. Floating Point (double)

- For all arithmetic operators
  - If both operands are integers, returns an integer
  - If either operand is floating-point (double), returns floating-point (double).
- For integer division, this means truncating the result

  7.0 / 2.0 = 3.5

  7.0 / 2 = 3.5

  7 / 2 = 3

Exercise 1

- Evaluate the following expressions:

A. 13 / 4 = 3

B. 3 – 7 % 5 = 1

C. 8 + 5 * 2.0 = 18.0

D. 17.0 / 4 = 4.25

Exercise 2

- After these statements are executed, what are the values of variables `a`, `b`, `c` and `d`?

```
a = 3;
b = 4;
c = (a % b) * 6;
   // (3%4) *6 = 3*6 = 18
d = c / b;
   // 18/4 = 4;
```

Exercise 3

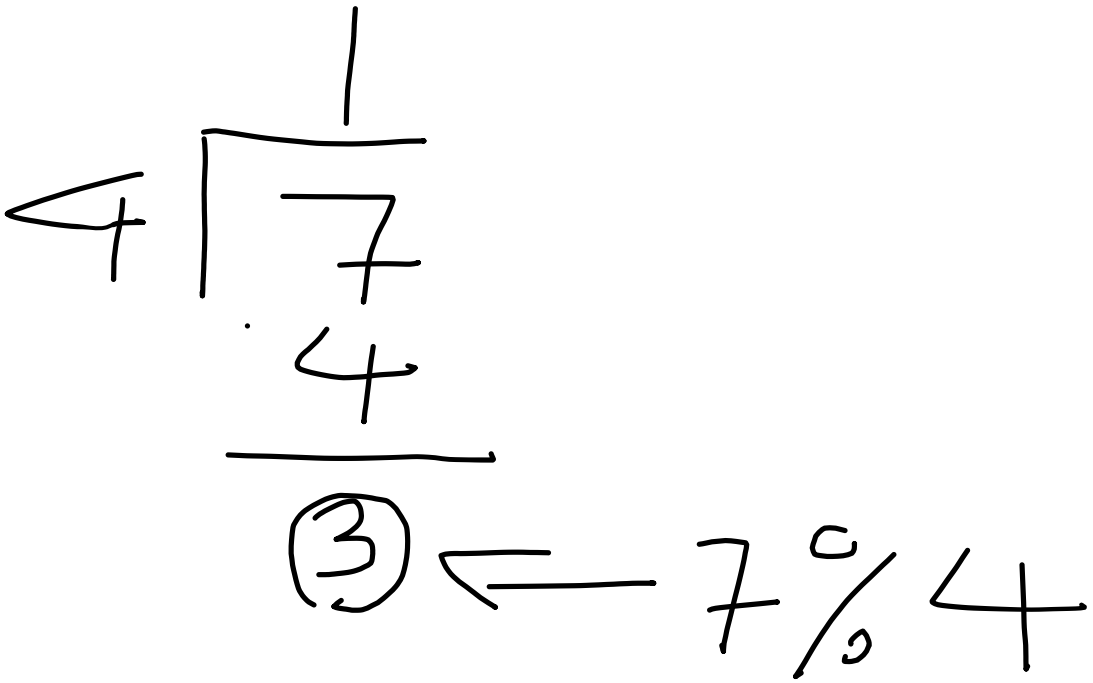
Write a function to compute an average and print the result

```
void avg()
{
   // statements go here
}
```

1. Write C++ statements that *declare* the following variables of type *int*: `num1`, `num2`, `num3` and `average`
2. Write C++ statements that *assign* `125` to `num1`, `28` to `num2` and `-25` to `num3`
3. Write a C++ statement that *assign* the average of `num1`, `num2` and `num3` into `average`
4. Write C++ statements that print the values of `num1`, `num2`, `num3` and `average`

modulus

- 7 % 4 = the remainder of 7 / 4.
- Think as 7 is divided by 4:



- 9 % 4 = 1