# CMPT 125: Introduction to Computing Science and Programming II

Spring 2023

Week 1: Course Intro, Introduction to C

Instructor: Victor Cheung, PhD

School of Computing Science, Simon Fraser University

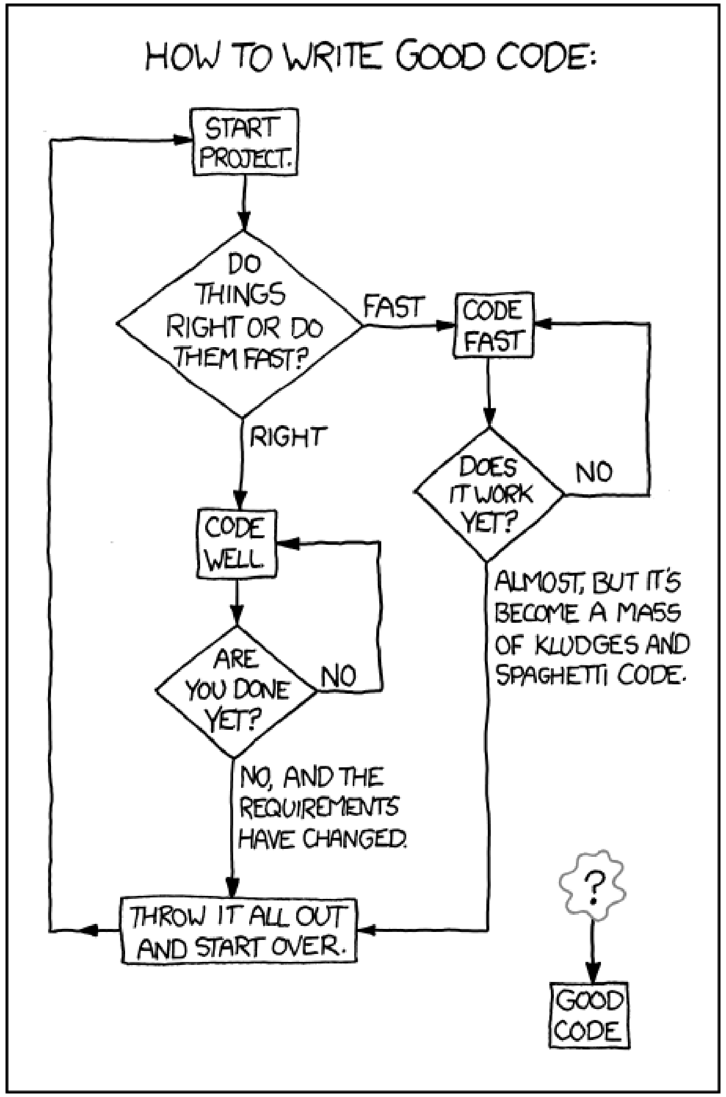
## The Pasta Theory of Programming – By Raymond J. Rubey

Spagetti
Complicated. Hard to understand.

Lasagna
Understandable. Layered.

Ravioli
Modular. Nourishing

Source: https://www.gnu.org/fun/jokes/pasta.code.html


HOW TO WRITE GOOD CODE:
START PROJECT.
DO THINGS RIGHT OR DO THEM FAST?
FAST
CODE FAST
DOES IT WORK YET?
NO
ALMOST, BUT IT'S BECOME A MASS OF KLUDGES AND SPAGHETTI CODE.
RIGHT
CODE WELL
ARE YOU DONE YET?
NO
NO, AND THE REQUIREMENTS HAVE CHANGED.
THROW IT ALL OUT AND START OVER.
?
GOOD CODE

Source: https://xkcd.com/844/

## It Is Important to Write Good Code

- **Easy to maintain**
  - If you need to change something you know where & how
- **Easy to recall**
  - Write working code effectively and efficiently
- **Easy to share**
  - Reusable, adds value, engaging, and empowering

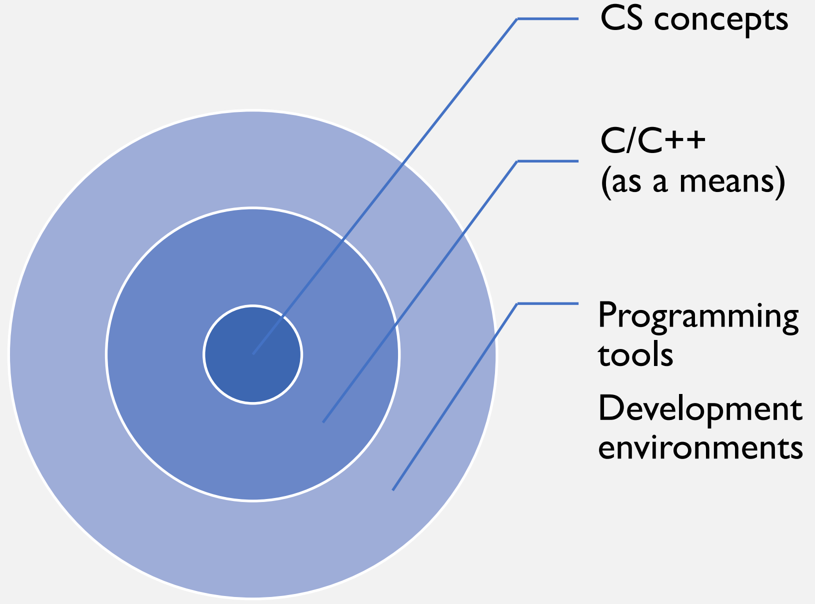
Takes time and practice, but worth it!

## What You Will Learn in This Course

- Design & implement solutions to problems using C/C++
- Explain, analyze, & compare algorithms in terms of performance
- Describe & utilize fundamental computing science concepts
- Maintain good coding practice and style

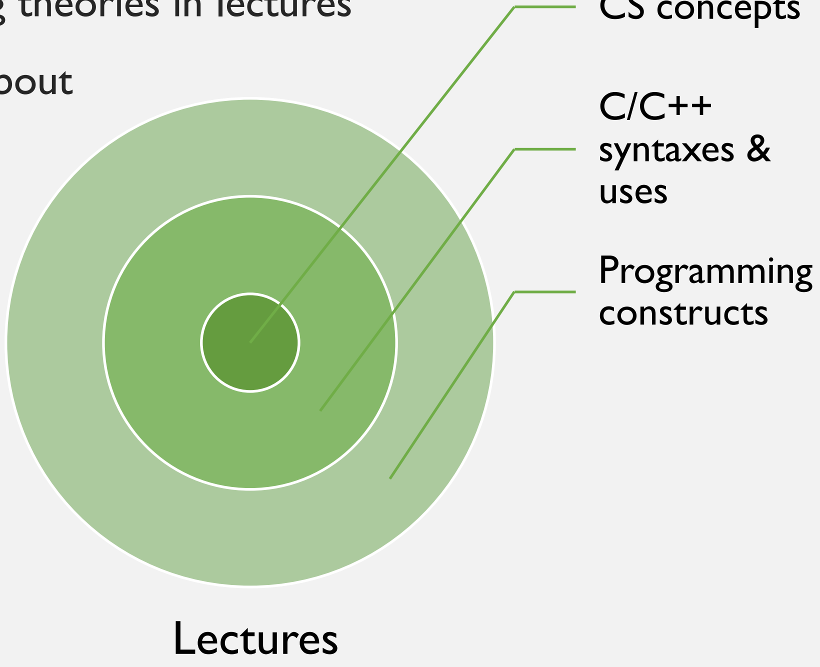
## What This Course Is Really About

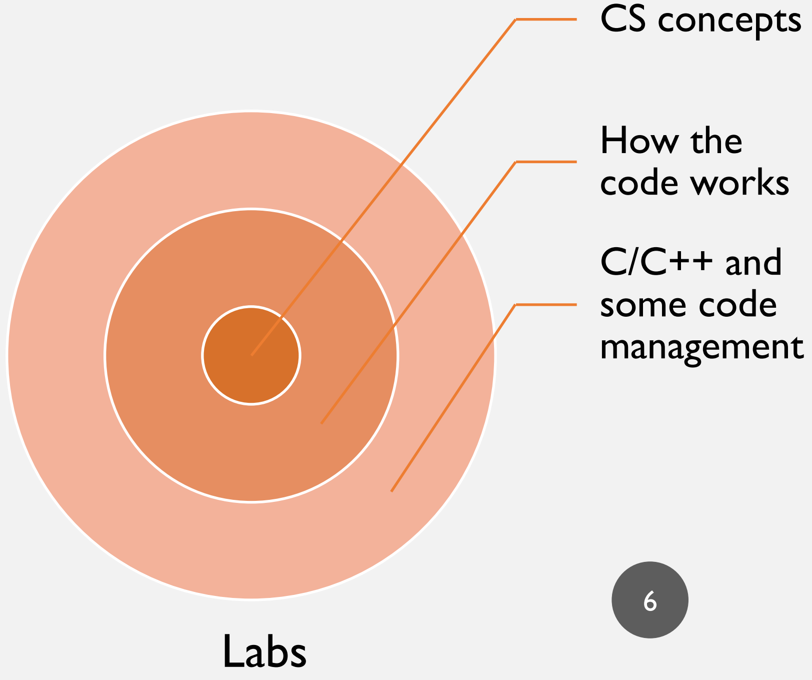
- Main objective is to learn about the concepts of Computing Science and Programming
  - And a little bit of good coding practice
- C is just a means to practice it
  - You might have learned other languages by yourself or through other courses
- Use tools to work with C source code
  - Makefiles and IDEs, also a bit of Unix/Linux



## How This Course Is Related to Your Labs

- CMPT 125 has **lectures** and **labs**
  - Lectures talk about theories informing practice in labs
  - Labs involves practice informing theories in lectures
    - You can also ask questions about assignments in labs



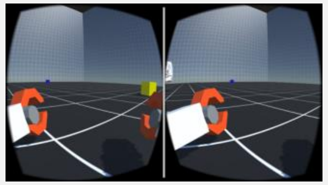


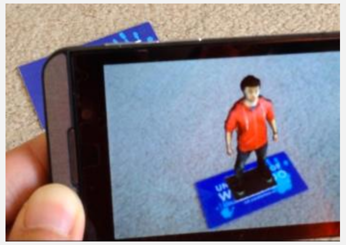
## Lecture Overview

- Logistics
- Expectations
- How to do well in this course
- Concepts in Computing Science
- Introduction to C
  - First program
  - Variables

## Logistics – Your Course Instructor

- Victor Cheung
  - PhD in System Design Engineering, University of Waterloo
  - Research in interaction & interface design, tangible systems, emerging technologies
  - v_cheung@sfu.ca





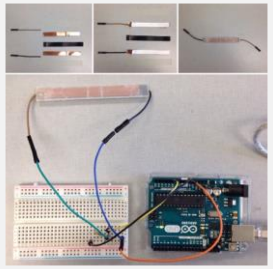
Coffee Please

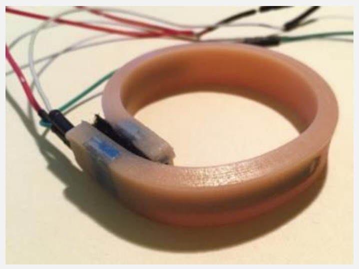
Instructions

Read the order at the top.
Pick the correct size on the sides.
Pick the correct ingredients at the bottom.

Serve when done.

You are the barista.





# Logistics – Course Info

- **Lectures**: Mondays 2:30p – 4:20p (SSCB 9200), Wednesdays 2:30p – 3:20p (DFA 300)
  - SFU Canvas (https://canvas.sfu.ca/courses/75467), CourSys (https://coursys.sfu.ca/2023sp-cmpt-125-d1/)
- **Labs**: Thursdays 8:30a – 9:20a / 9:30a – 10:20a / 10:30a – 11:20a / 11:30a – 12:20p (ASB 9898)
- Office Hours: TBA, check SFU Canvas
- Teaching Team
  - **Instructor**: Victor Cheung, PhD | Email: v_cheung@sfu.ca
  - **Teaching Assistants**:
    - Pedram Agand | Email: pagand@sfu.ca
    - Yifei Chen | Email: yca429@sfu.ca
    - Rohan Mathur | Email: rma135@sfu.ca
    - Brandon Miles | Email: bam13@sfu.ca
    - Bowen Shi | Email: bas96@sfu.ca
  - Help email that reaches all the teaching personnel: cmpt-125-d1-help@sfu.ca

When sending your email:
- Use your SFU email
- In the subject line begin with "CMPT 125 D100:" and then topic of your message. For example, CMPT 125 D100: Missed Assignment 1
- In the body of your message, include your full name, SFU ID, 9-digit student number and section number

## Logistics – Readings

- We'll use parts of this book and a collection of online readings
  - Seacord, R. C. (2020). Effective C. In Effective C. No Starch Press. Online access (requires SFU login): https://sfu-primo.hosted.exlibrisgroup.com/permalink/f/usv8m3/01SFUL_ALMA51348448870003611
    *Instructor's notes prevail when there is a conflict
- Good reference for C/C++:
  - http://cplusplus.com/
  - https://en.cppreference.com/
  - https://www.programiz.com/c-programming

## Logistics – Course Materials

- All materials (or links to them) are on Canvas (https://canvas.sfu.ca/courses/75467)
  - We will also use CourSys (https://coursys.sfu.ca/2023sp-cmpt-125-d1/) for submissions
  - No recordings will be available. IT service might record audio automatically, email me if you have a valid reason to miss a lecture and I'll see what I can do (but generally speaking if you miss a lecture you miss a lecture, decision is yours)
- Use the **Discussion** forums to ask questions or give suggestions
  - We will be monitoring them to give you answers in a timely manner
  - We might remove your post if it has solutions or anything that might lead to plagiarism or cheating, or if it is inappropriate
- Software tools
  - Connect or go to CSIL machines to develop & test your code (we'll use that to mark your code)
    - If your submitted code doesn't compile/run there, you get 0 marks
  - You can however use other tools to write your code if you want (e.g., Replit, online-ide, Atom, VS Code)

## Logistics – Assignments/Exams

- Tentative:
  - Assignment 1/2/3/4 (30%) | due on Jan 27/ Feb17/ Mar 17/ Apr 7
  - Lab Attendance/Demo (10%): best 10 out of 12
  - Midterm Exam (25%): cumulative and include all materials covered thus far | on Feb 27 during class
  - Final Exam (35%): cumulative and include all materials covered thus far | TBA
- Assignment **late penalty**: 10% per calendar day (each 0 to 24 hour period past due), max 2 days late
- The instructor will do his best to accommodate within reason, but will have the final say
  - Contact the instructor as soon as possible
- More details as the due dates approach

## Logistics – General Etiquette

- **Be on time** (if you are late, come in quietly)
- Ask if you have **questions** (raise your hand or ask after the lecture)
- Treat all messaging platforms (e.g., discussion forums, emails) **professionally**
- Email using your **SFU account**, put "**CMPT 125 D100**" in the title
  - Include your full name and student ID in your signature
  - Use full sentences and no textisms like "b4", "w8", "that's gr8", "thx 4 BN aQr8"
    - Fun read: An Analysis of Language in University Students' Text Messages by Lyddy et al. 2013 https://doi.org/10.1111/jcc4.12045
  - **Check it frequently** – if I need to contact you directly this will be where I do it

## Things That You Should Know (Doesn’t have to be in C)

- Variables
- Data types (integer, float, char, string, boolean)
- Lists/arrays
- Basic I/O
- Conditionals (if/if-else/if-else if-else)
- Loops (for, while)
- Functions, parameter passing
- The design → test → debug cycle

**Don’t worry if you only remember some, we’ll revisit them**

## Examples of Things You Should Know

- You should be comfortable solving the following problems (in Python, Java, …etc.)
- Write a function that gets a string and decide if it is a palindrome. For example:
  is_palindrome("ABCDCBA") needs to return True
  is_palindrome("ABCDB") needs to return False
- Write a function that gets a string and finds the longest substring that is a palindrome. For example:
  longest_substring_palindrome("ABCDBAABBA") needs to return "ABBA"
  longest_substring_palindrome("ABCDB") needs to return "A"
- Write a function that gets a list of digits, and outputs a string containing the largest number you can writing using these digits. For example:
  max_number([1,2,8,8,1,1, 0]) needs to return "8821110"

## Should You Take This Course?

### SHOULD

- If you want to learn about Computing Science and what it is about
- If you want to become better in designing and writing your code
- If you have some programming background and want to learn about C

### SHOULD NOT

- If you are here only to learn C
- If you do not want to write code
- If you are looking for an easy A

## Who Will Do Well in This Course?

- Attend classes and pay attention
- Participate in discussions
- Make progress on assignments/projects regularly
- Get advice from the instructor/TAs when needed
- Follow instructions carefully on assignments
- Regularly review readings / course notes (and find other reading materials)
- Put in the extra effort in maintaining your code

- For grading policies refer to the **Course Policies** page on Canvas

Looking for information online ⚠

- Check **course notes** first!
- Then check **readings & course materials**
- Google/StackOverflow/YouTube may seem useful but can also contain advanced code that will be more confusing than helpful… proceed with caution

## Academic Integrity

- No plagiarism
  - All referenced work must be appropriately cited, including code, websites, images, figures, and graphs
  - Never directly copy & paste and treat it as yours
  - You must write your own code (we will use software, e.g., MOSS, to check)
- No cheating
  - All assignments/exams must be your own work (team's work if it is a team project)
  - It is ok to discuss with others for assignments, but you have to write your own work. Mention whom you have talked to if necessary
- All incidents will be reported according to SFU policy
  - http://www.sfu.ca/policies/gazette/student/s10-01.html
  - Understand what is considered cheating and what is not at SFU's **Academic Integrity Tutorial** (https://canvas.sfu.ca/courses/56136)

## You Are An Adult Now

- You are not a child anymore and you are responsible to your own decisions & actions
  - **Manage your time**
    - Plan ahead for deadlines, get enough rests, do stuff that you enjoy
  - **Be proactive**
    - Look up information beyond what’s in the slides, ask questions when in doubt
    - We might not answer your questions if we have already provided answers to them
  - **Do honest work**
    - All the academic integrity information are available and you can’t claim you don’t know about them

# Questions?

If you have questions later in the course, feel free to ask them in the Discussion forum on Canvas

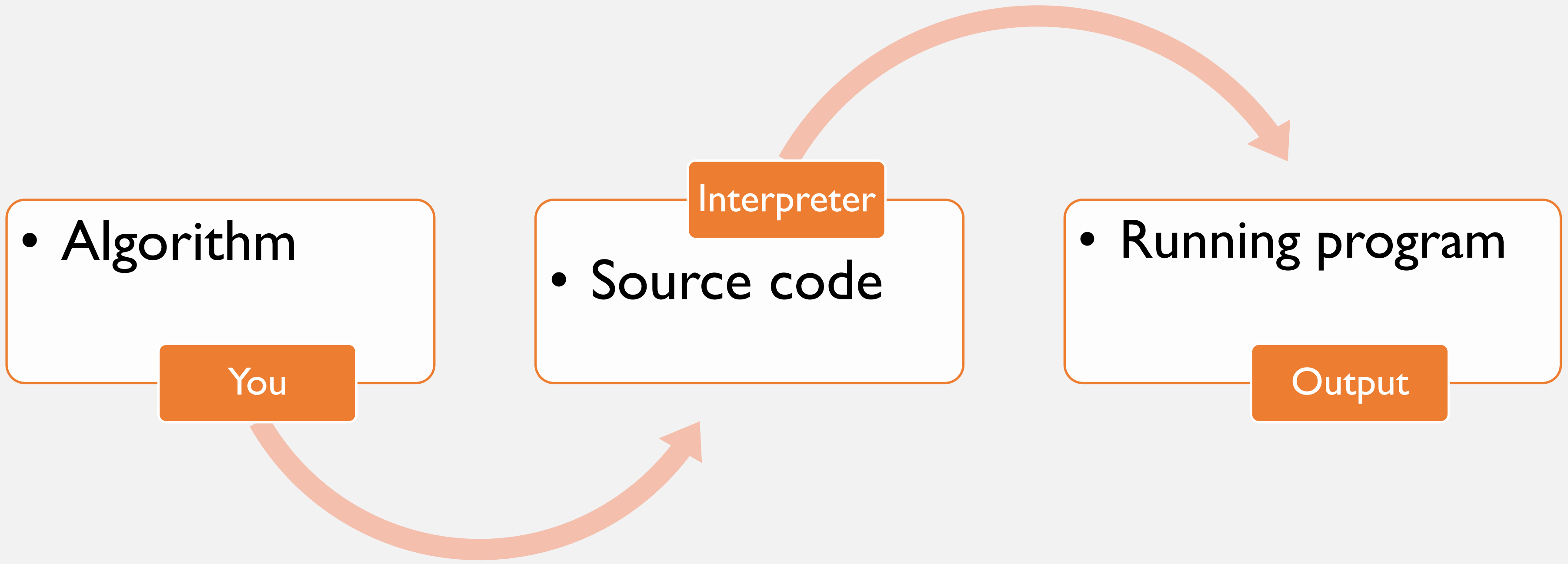
## Let’s start with Some Concepts

## Some Core Concepts in Computing Science

- Algorithms – a sequence of instructions to complete a task or solve a problem
  - There are usually **many different ways** to complete the same task or solve the same problem (hence different algorithms)
  - The clearer each step is the better (there are other ways to make an algorithm better, e.g., faster, requires less space)
- Programming – the action of communicating an algorithm to computers using a specific language
  - Here we use C (and some C++)
- Data structures – specific ways for computers to store a collection of data for better management (e.g., insert, remove, search)
  - Different structures have different properties and trade-offs (e.g., faster in some operations slower in others, takes more space)
  - Most are independent from programming languages (i.e., you can implement them using languages other than C/C++)

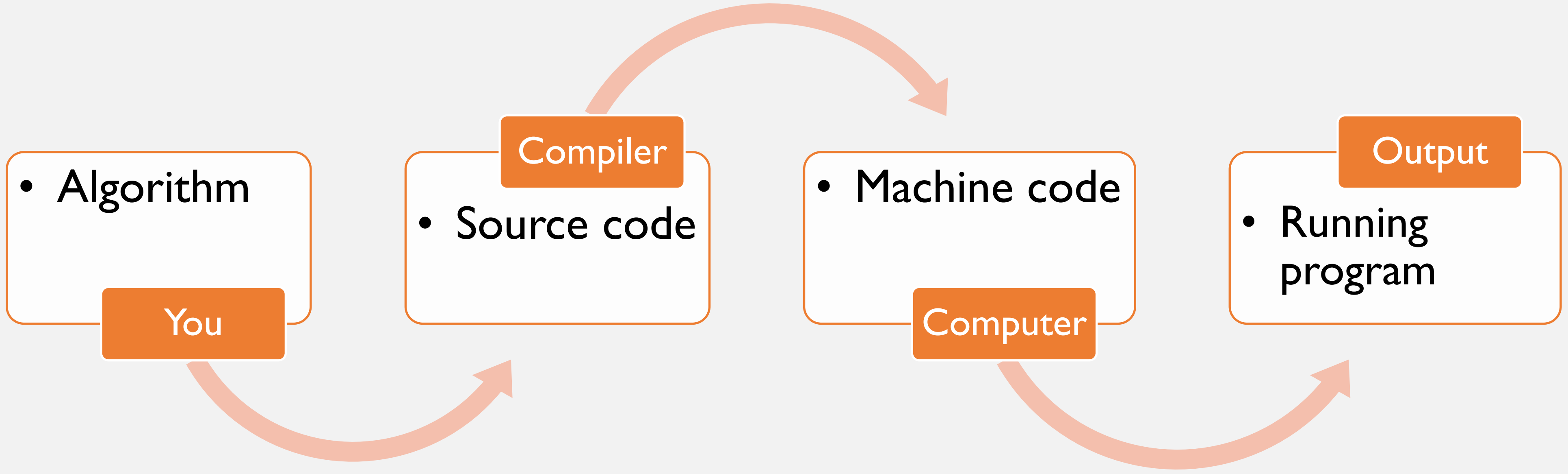
## The Compilation Process (I)

- Interpreted programming language (e.g., Python, Javascript)
  - You implement algorithms into code, interpreter runs one instruction at a time when executing the program



## The Compilation Process (2)

- Compiled programming language (e.g., C/C++, Java)
  - You implement algorithms into code, compiler converts code into machine code (instructions), computer runs one instruction at a time when executing the program



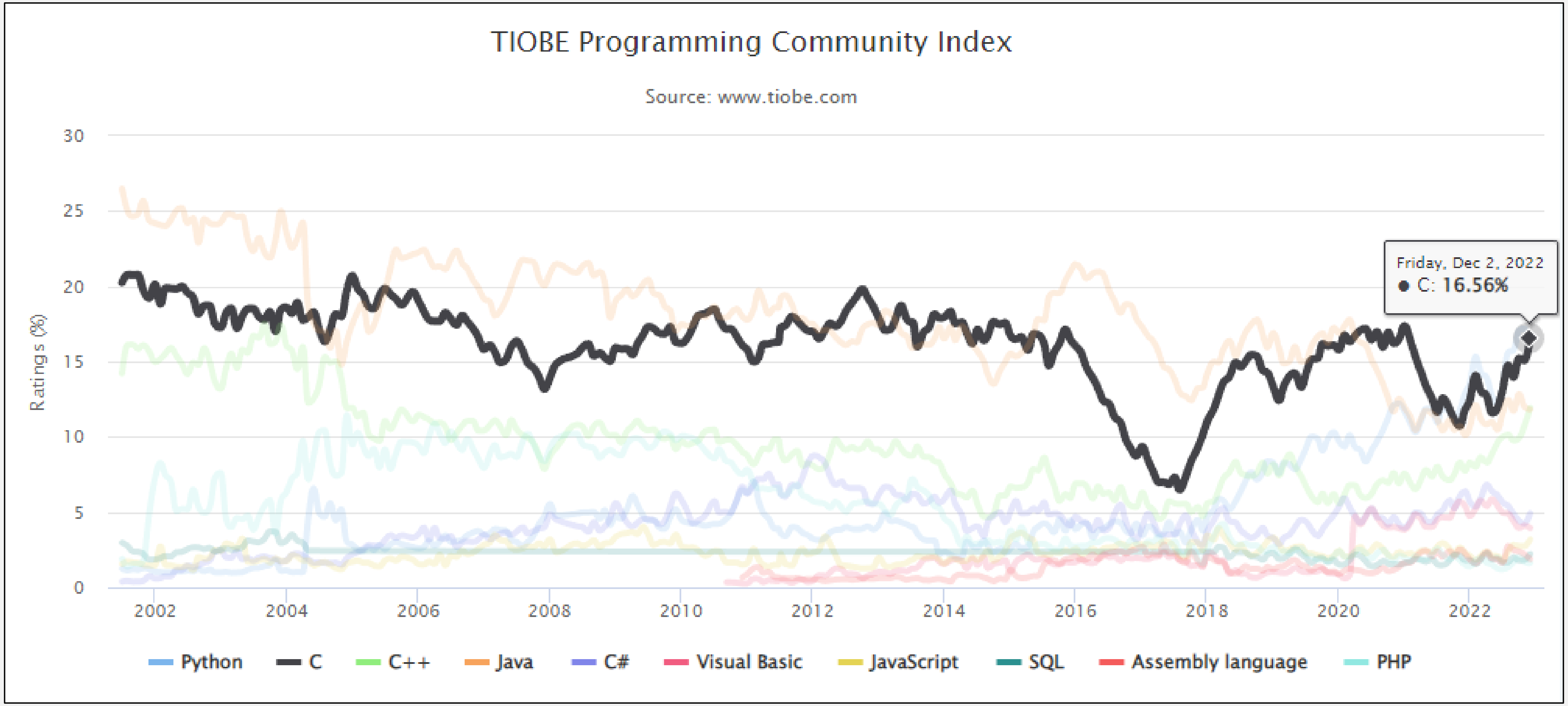
## Interpreted VS Compiled

### INTERPRETED

- Errors are caught only when the interpreter is trying to run them
- Can test the code quicker because the interpreter directly runs it
- Program is less optimized because the interpreter needs to convert the code into machine code internally every time

### COMPILED

- Some errors can be caught during compile time so they can be fixed earlier
- Takes a bit longer to get the code running because of the compilation process
- Program often performs better because some optimizations for the operating system and architecture can be done by the compiler



C is within the top 3 most popular (used/searched/taught) programming language

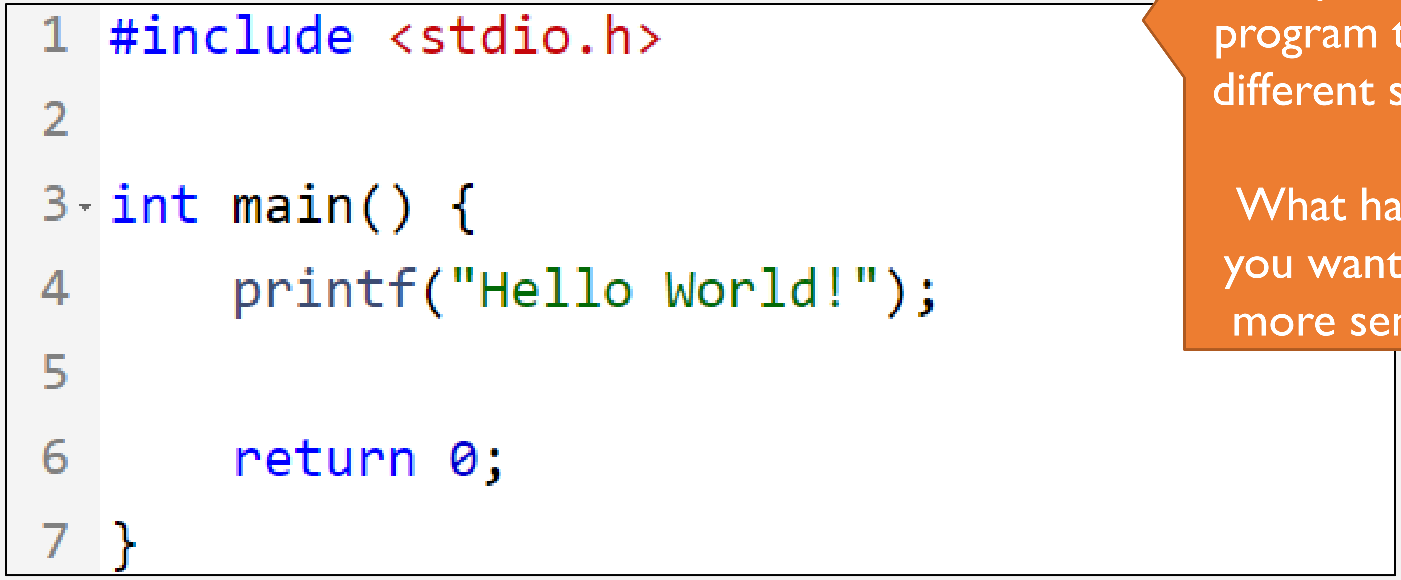
## Something about C

- C was developed by Dennis Ritchie between 1972 & 1973 at AT&T Bell Labs
- Designed to **map to typical machine instruction** – so the source code can easily be compiled & optimized, but also sometimes hard to understand
- **Widely used** to implement operating systems, computationally intensive programs, computer graphics
- Provides **low-level access to system memory** via pointers – highly flexible but can cause strange and sometimes harmful impacts to the system

# Let’s Write a C Program

This allows you to use external code (library)

Lines 3-7 is what we call the “main function”, which is the entry point of a C program



```
1 #include <stdio.h>
2
3 int main() {
4     printf("Hello World!");
5
6     return 0;
7 }
```

What would you do if you want the program to print a different sentence?

What happens if you want to print more sentences?

*Refer to the “Accessing CSIL machines remotely” page on Canvas to learn about different ways to connect to CSIL

## Homework!

- Read Ch. 1 of the Effective C book
  - Use gcc instead of cc to compile your source code in CSIL machines
  - Investigate the difference between puts and printf
- Do the "Tell Us about You" survey on Canvas
- Find out how to connect to a CSIL machine (CLI and RDP)
- Get yourself an access fob to CSIL (our labs will take place there starting next week)
- Write the "Hello World" program from scratch without referring to any notes
  - Try printing something other than "Hello World!", then try printing more than one sentence
  - For fun, watch this video I made: https://www.youtube.com/watch?v=sQa1HHTMmmQ