INTRODUCTION TO COMPUTER SCIENCE AND ENGINEERING

TABLE OF CONTENTS

What is Computer Science:

Contrast to popular believe, computer science is not all about programming, but about understanding how to computationally solve problems. It is about creating high-quality software in a systematic, controlled, and efficient manner – an overlapping aspect in engineering.

Computer Science "should be" primarily about how to solve a problem with computing technique. It should be better called "Software Engineering". This can include many aspects such as algorithms design, languages, hardware architecture, systems software, applications software, etc. In this 21st century, it has evolved to include web development.

It is a training in all aspects of the software life cycle, from specification through analysis and design, to testing maintenance and evaluation of the product.

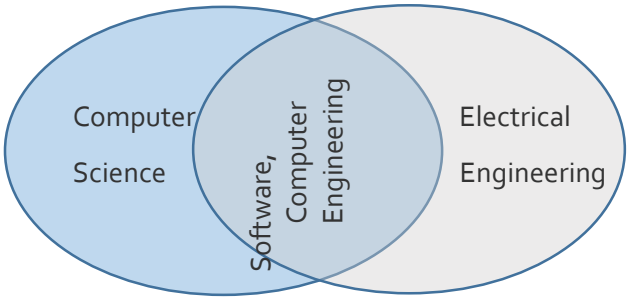
If I have to summarize Computer Science in less than 10 words – Problems Solving for Practical Applications using Computer Technology.

Problem in the current AP Computer Science: Computer Science is beyond Fundamental Java Computer programming. It is the "Computational Thinking" aspect which should be introduced as the fundamental computer science principle, not Java programming.

Science : Study of the structure and behavior of the physical and natural world through observation and experiment. It aims to learn the truth of something.

Engineering : Study of the making of practical application with the knowledge of pure sciences. It aims to learn how to apply with the truth of something.

Computer Science : Study of the scientific and practical approach to computation and its applications.



Computational Thinking – the core in Computer Science!

- **(by NSF) – CT** is defined comprehensively to encompass computational concepts, methods, models, algorithms, and tools.
- Computational thinking will be a fundamental skill used by in any profession, not just science and engineers.
- CT focuses on the process of abstraction to achieve automation with:
  - choosing the right model of abstraction
  - clearly identify the layers of abstraction, either takes place simultaneous and/or in sequence
  - clearly defining the relationships between layers and how they are linked

VERY MUCH LIKE IN ENGINEERING:

Efficiency :

- How fast?
- How much space?
- How much power?
- Trade off among speed, space, and power?

Correctness:

- Does it do the right thing?
- Tolerance level allowed?
- Trade off among this with time, storage, etc.

-ility:

- Simplicity (but and Elegant)
- Reusability
- Scalability
- Maintainability

Cost

- In memory
- In storage
- In $$$

CT has been used in our daily live..

Things that we do….	Algorithms
Looking up a name in an alphabetically sorted list	Linear: start at the top – Binary search: start in the middle
Standing in line at a bank, supermarket, customs & immigration	Performance analysis of task scheduling
Airline tickets booking	Database transaction synchronization and processing. 2-phase commit technique.
Trying to visit as many colleges as you can with minimal travelling time and distance	Traveling salesman - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city
Login to your google a/c	Encryption…Data Base lookup
Sort out your lego pieces	Using hashing algorithm (e g. ., by shape, by color,etc.)
Your browser infests your machine with advertising about a pair of jeans that you just looked up online.	Data Mining (ok.. perhaps the example I gave shred a bad light over data mining. Data mining enables professionals and

	scientists to learn from pattern in order to analyze and predict.)
Grade school arithmetic	GCD, Factorial

More.... in :

- Mathematics, Physics
- Biology, Medicine, NeuroSciences
- Astronomy, Aerospace
- Meteorology
- Machine learning, such as the Credit Cards system, Wall Street, Amazon, Google, Facial Recognition
- Economics, Law, Healthcare
- Other areas such as archaeology, Journalism, Music, English, Art, Design, Photography, ...

Trend in college

An introduction of computer science, based on the principles of computational thinking will be used as the entry point for any entering engineering student. Soon it will be an entry point for any student with limited programming experience. Many strong universities, such as CMU, MIT, Cornell, U Penn, Brown, Harvard, etc., have already made basic computer science concept and/or programming as an entry level requirement for engineering students.

From my personal professional experience, in the fiercely competitive software development industry, if you are judging by the title, it is called "Software Engineer", not programmer. It really depends on how and what you receive from your either academic training, and/or your personal endeavor. No matter which school you go to, it really boils down to your own self-motivation and initiative to stretch your own full potential.

STRATEGIC GOALS OF THIS WORKSHOP:

- Mainly to aim for the students who show possible interest in computer science major in College.
- Increase competence and confidence in computational problem solving.
- Explore the process and concepts needed to go from high-level descriptions of algorithms to correct imperative implementations.
- Learning a language for expressing computations in C (with some C++) using Microsoft Visual Studio Express.
- Learning about the process of writing and debugging a program
- Learning about the process of moving from a problem statement to a computational formulation of a method for solving the problem
- Learning a couple basic set of "recipes"/"proto-types" — in Search and Sorting Algorithms.
-

THIS WORKSHOP WILL COVER:

- Simple Data Structures - 1-D and 2-D data array, structure.
- Bit-wise operations.
- Simple State Machine
- Computational algorithms in Euclid, Newton's method, Sieve of Eratosthenes, etc.
- Decomposition, Abstractions, Encapsulation.
- Control Structures and iteration.
- Introduction in stack and recursion.
- Work on a few problems from USACO.
- use theoretical ideas to formulate and solve problems in computer science. It integrates mathematical material with general problem solving techniques and computer science applications.
- May explore concepts:
    - Parallel composition.
    - Polymorphism – in procedural vs object-oriented.
    - Memory pointers.
    - Algorithms in sort and search – e.g., quicksort, binary search, depth-first search, breath-first-search, shortest path algorithms.
    - Aware of worst-case asymptotic complexity concept.

EXPECTATION:

- If you do not have much programming experience before, this will be a good introduction level. You will work on many programs.
- If you have had good amount of experience in data structure, you will be expected to do some really hard work here. ☺
- By the end of the week, you should be able to :
    - Take on basic but non-trivial problems and resolved it with programming on your own.
    - Read and write code for some basic imperative algorithms and data structures. Algorithms and Data Structures.
    - Describe the implementation of a number of basic algorithms and data structure.
- Important key: You progress at your own pace. You are not expected to complete a full term computer science introduction course within a week. Many concepts talked about in the later days this week serves an exploratory level learning.

DAY 1 - UNDERSTANDING THE SCOPE AND WARM UP

Concepts today:

- ☐ Know about bytes vs bits. Bit-wise operation
- ☐ Binary Arithmetic
- ☐ Fundamentals in Debugger including break points, observation of variables, watch feature.

EXERCISES

Track A Scope	Track B Scope
Proper Development Habit - Coding Styles Stress in Simplicity, Elegance, and Efficiency	
Do the binary math (on paper first)	
Use: HTTP://LEARNING.STORMINGROBOTS.COM Warm up - Bare Basics. and Basic C++ IO Stream	Complete the Computer Arithmetic, i.e. perform +, -, *, / with binary; so no more "+, -, *,/" Base Conversion Challenge
By the end of the day, you should be familiar with: ☐ Programming Expressions in C. ☐ Basics in Standard I/O ☐ Rudimentary in using #define preprocessor directive ☐ Work on the all exercises in Warm Up packet.	
Fundamentals in Debugger including break points, observation of variables, watch feature.	

DAY 2:

Review how to do binary bit arithmetic using computing.

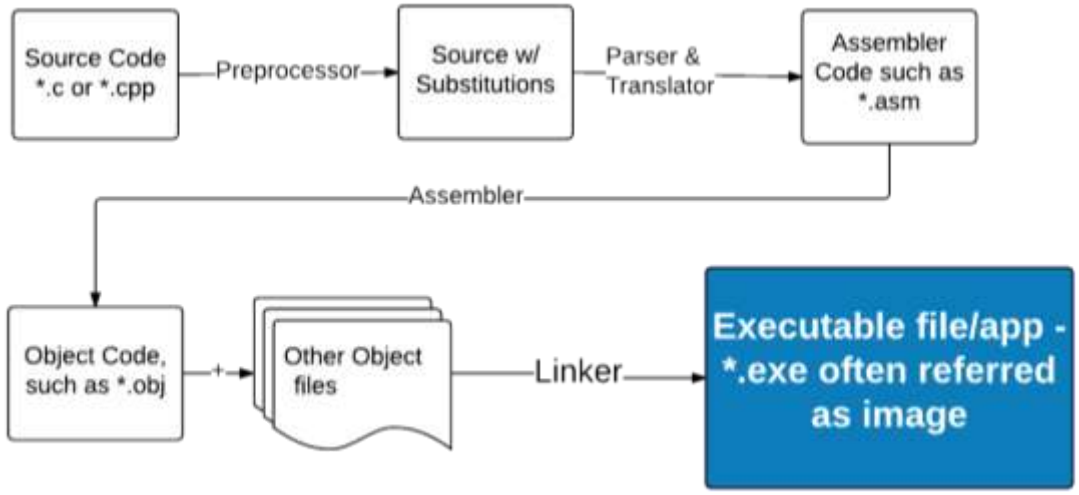
Concepts today:

- ☐ Explore the process from a text source file to executable file.
- ☐ Control structure.. simple to nested. Rules of thumb.
- ☐ break, continue, and NO goto. Proper and safe style.
- ☐ State machine – switch
- ☐ Enumeration and structure.
- ☐ Convert conditions into number.
- ☐ Re-emphasize : Fundamentals in Debugger including break points, observation of variables, watch feature.
- ☐ Review How to Write Robust and Readable Codes, such as:
    - Proper naming convention
    - Minimize global
    - All global starts with an upper case. Use camel style in variables
    - Local variables start with lower case.
    - Useful comment
    - Many more.. We will go thru these practice in our exercises.

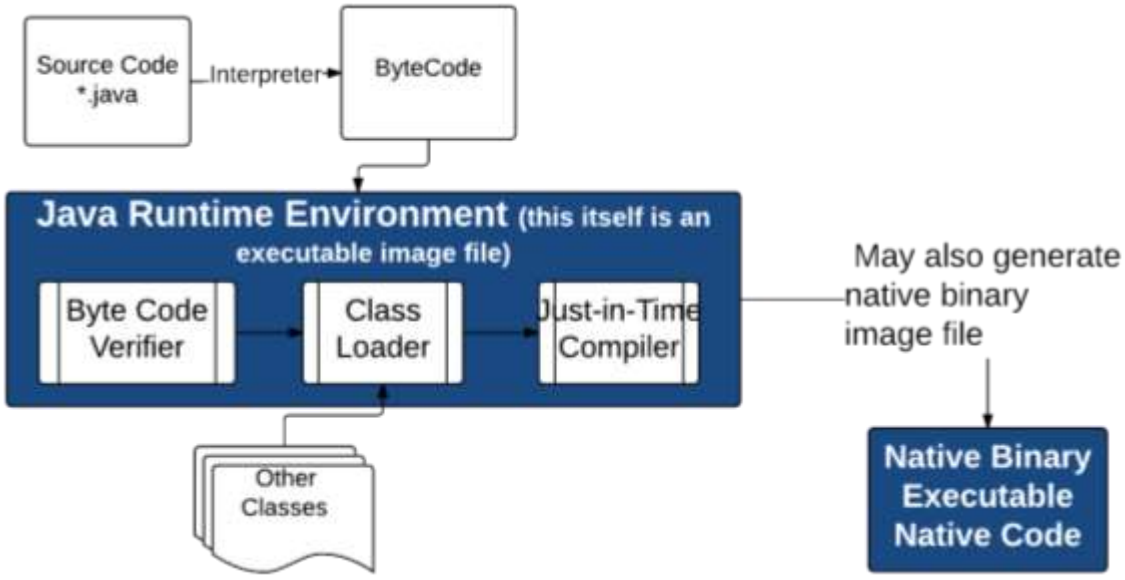
FROM TEXT FILE TO EXECUTABLE :

- Source *.cpp file to *.exe via a compiler.
- A compiler is a computer program that translates source code written in a certain programming language into binary image which can run by the computer, i.e. we often call it the executable image file.
- What makes your program to become an application which can execute? Let's take a look of Compiler process :

C COMPILER PROCESS:



JAVA COMPILING PROCESS



FINITE STATE MACHINE / TABLE / DIAGRAM

What is FSM?

- Model decision making
- Inputs, states, outputs
- Capturing behavior

Why?

- Impose well-thought out design and flow
- Increase the likelihood of establishing a more robust infrastructure
- Help in debugging

Steps:

1. Define States - Condition / Situation / Criteria
2. Describe States - Somewhat like documentation ... comment, etc.
3. Identify Input, Output through transitions from one to another State
4. Identify the behavior which cause the States Transitions - actions which cause the transitions from one state to another.

Samples:

- Simple bank a/c operations - deposit & withdraw money
- Operate a robot based on 3 touch sensors input.

Inputs:

- Any form of stimulus from the outside world
- In high level : bank a/c balance becomes zero, getting a good grade, etc.
- In low level : single bits, such as a digital signal high or low.

States:

- Describe current condition of the FSM
- Often within a limited number of States for a given FSM

Outputs:

- depend on the state and inputs.
- Can be an input for another State

Behaviors

- Should be absolutely clear. No ambiguity is allowed.
- Complete and consistent.

DAY 3 – ARRAYS/ SIMPLE ABSTRACT DATA TYPE / FUNCTIONS

- Review binary division math
- Review State Machine
- Fundamental Programming Design:
  - A lengthy topic. In order to stay within the scope of this workshop, we will just summarize it in only the following:
    - **Decomposition** : is a method to break a problem into **self-contained**, manageable parts.
    - **Abstraction** - it allows us to ignore the details of a piece of code, and use it as a black box – input x, get y.

FUNCTIONS:

Description from the Master Study in Machine Learning Handbook from CMU:

- "…an introduction to programming based on a "functional" model of computation. The functional model is a natural generalization of algebra in which programs are formulas that describe the output of a computation in terms of its inputs—-that is, as a function…"

By the end of the day, you should be familiar with:	Work on the Array/Struct/Functions Packet.
☐ 1-D, 2-D array. ☐ Simple Abstract Data Types - Simple struct and Union ☐ Enumeration Type ☐ Create Functions with and without parameters/arguments and return types. ☐ Program Design - multiple programs, include header file	Additional Exercises: - Apply state machine using RobotC / NXT /Sensors to activate motors.

MAY EXPLORE:

Multitasking programming synchronization among processes – use RobotC - as the multi-tasking environment. Learning about pointers for memory consumption.

DAY 4 – RECURSION – STACK

- Stack : LIFO
- Recursion vs iteration:
  - Factorial
  - Fibonacci
  - Insertion Sort
- Learn how to design recursive functions.
- May explore Depth First Search using Depth First Search method.

EXERCISES:

- Complete the exercises in the online Recursion and Iteration Packet.

Exploration Topics

What is meant by Asymptotic?

- O(1) constant
- O(log n) logarithmic
- O(n) linear
- O(n log n) "n log n"
- O(n2) quadratic
- O(n3) cubic

Why think Worst-case and Average-case?

Why choose insertion sort for data filtering. .. vs. with Quicksort.

DAY 5 - PUT FUNDAMENTALS IN APPLICATION

A robotic exercise using all the fundamentals used in the past 4 days –

- create a state machine with using 4 sensors to create all possible deterministic states
- using insertion sort to filter out noise.

EXPLORATION TOPICS:

Explore Queue : FIFO

May Explore if appropriate to the group:

- Dijkstra's Algorithm - when no edges have negative weight
- Maze design with Breath First Search Method
- Minimum Spanning Tree - Kruskal's algorithm and Prim's algorithm.
  - Find disjointed set. (because of we cannot have cyclic connections.)
  - Create pairs of vertices
  - sort the edges by the weight
  - go thru each pair to create disjointed sets (list)
  - Add to disjointed list and create spanning trees
  - Add to disjointed list
    - Add member (member)
      - setX = which set (member)
      - if member
      - if setX does not exist
      - create set (setX)
      - else
      - add member (SetX, member)
    - if both pair of members does not exist one of the Sets
      - add in the spanning tree list