CS601: Software Development for Scientific Computing

Autumn 2022

Week1: Overview

Let us listen to Jack Dongarra

- https://youtu.be/Oe9LRKoE6L0

Course Takeaways (intended)

- Non-CS majors:
  1. Write code and
  2. Develop software (not just write standalone code)
     - Numerical software
- CS-Majors:

  In addition to the above two:

  3. Learn to face mathematical equations and implement them with confidence

What is this course about?

Software Development

+

Scientific Computing

This course NOT about..

- Software Engineering
  - Systematic study of Techniques, Methodology, and Tools to build correct software within time and price budget (topics covered in CS305)
    - People, Software life cycle and management etc.
- Scientific Computing
  - Rigorous exploration of numerical methods, their analysis, and theories
  - Programming models (topics covered in CS410)

Who this course is for?

- You are interested in scientific computing
- You are interested in high-performance computing
- You want to build / add to a large software system

Feedback from Autumn'21

1. Everything was intresting. Only the FEM part I felt was too heavy.
2. Awesome!
3. Really passionate about what students learn and enthusiastic to clear the doubts.
4. The FEM classes were a bit of an overload and really confusing to be studied in four days. Other than that I think the course was wonderfully taken.
5. NO
6. NO
7. NO
8. NO
9. It was too difficult. 3rd year students should not be allowed to take this course.
10. This course had too much irrelevant things to learn. it was a CS course, but i studied more topics in mechanical engineering than CS Topics. Should have taken number theory instead. The programming assignments were out of my league to be honest.
11. Sir was good, but i think he was better when he was teaching just the software engineering course. Not much materials were available on the internet.
12. It was good.
13. It was fine.
14. I like FEM and some programming aspects most.
15. Professor helped a lot in the understanding of the course.
16. It was fine.
17. N/A
18. N/A
19. N/A
20. N/A
21.

GitHub Accounts

- https://forms.gle/GTb1vCWj8FpbzaLb8

Software Development

- *Software development is the process of conceiving, specifying, designing, programming, documenting, testing, and bug fixing involved in creating and maintaining applications, frameworks, or other software components.*

Software development is a process of writing and maintaining the source code, but in a broader sense, it includes all that is involved between the conception of the desired software through to the final manifestation of the software, …

- Wikipedia on “Software Development”

Scientific Computing

- Also called computational science
  - *Development of models to understand systems (biological, physical, chemical, engineering, humanities)*

Collection of tools, techniques, and theories required to solve on a computer mathematical models of problems in science and engineering

Why C++ ?

- C/C++/Fortran codes form the majority in scientific computing codes
- Catch a lot of errors early (e.g. at *compile-time* rather than at *run-time)*
- Has features for *object-oriented* software development
- Known to result in codes with better *performance*

Let us dive into an example….

Example - Factorial

- n! = n x (n-1) x (n-2) x . . . x 3 x 2 x 1

  (n–1)! = (n-1) x (n-2) x . . . x 3 x 2 x 1

  therefore,

  Definition1: n! = n x (n-1)!

  is this definition complete?

  - plug 0 to n and the equation breaks.

  Definition2:

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

Exercise 1

- Does this code implement the definition of factorial correctly?

```
int fact(int n){
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Example - Factorial

Definition2:

$$n! = \begin{cases} n \times (n-1)! & \text{when } n \geq 1 \\ 1 & \text{when } n = 0 \end{cases}$$

is this definition complete?

- `n!` is not defined for negative `n`

Solution - Factorial

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Exercise 2

- In how many `flops` does the code execute?

  assume 1 `flop` = 1 step executing ***any*** arithmetic operation

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Exercise 3

- Does the code yield correct results for any n?

```
int fact(int n){
    if(n<0)
        return ERROR;
    if(n==0)
       return 1;

    return n*fact(n-1);

}
```

Who this course is for?

- Anybody who wishes to develop "computational thinking"
  - A skill necessary for everyone, not just computer programmers
  - An approach to problem solving, designing systems, and understanding human behavior that draws on concepts fundamental to computer science.

Computational Thinking - Examples

- How difficult is the problem to solve? And what is the best way to solve?
- Modularizing something in anticipation of multiple users
- Prefetching and caching in anticipation of future use
- Thinking recursively
- Reformulating a seemingly difficult problem into one which we know how to solve by reduction, embedding, transformation, simulation
  - Are approximate solutions accepted?
  - False positives and False negatives allowed? etc.
- Using abstraction and decomposition in tackling large problem
- …

Computational Thinking – 2 As

- Abstractions
  - Our "mental" tools
  - Includes: choosing right abstractions, operating at multiple layers of abstractions, and defining relationships among layers
- Automation
  - Our "metal" tools that amplify the power of "mental" tools
  - Is mechanizing our abstractions, layers, and relationships
    - Need precise and exact notations / models for the "computer" below ("computer" can be human or machine)

Computing - 2 As Combined

- Computing is the automation of our abstractions
- Provides us the ability to scale
  - Make infeasible problems feasible
    - E.g. SHA-1 not safe anymore
  - Improve the answer's precision
    - E.g. capture the image of a black-hole

Summary: choose the right abstraction and computer

Recap

- Need to be precise
  - recall: `n! = 1` for `n=0`, not defined for negative `n`
- Choosing right abstractions
  - recall: use of recursion, correct data type
- Ability to define the complexity
  - recall: `flop` calculation
- Next?

Recap

- Need to be precise
  - recall: `n! = 1` for `n=0`, not defined for negative `n`
- Choosing right abstractions
  - recall: use of recursion, correct data type
- Ability to define the complexity
  - recall: `flop` calculation
- Choose the right “computer” for mechanizing the abstractions chosen