

CS601: Software Development for Scientific Computing

Autumn 2023

Week1: Overview

Course Takeaways (intended)

- Non-CS majors:
 1. Write code (mostly in C/C++) 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

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*

What is this course about?

Software Development

+

Scientific Computing

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*

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?

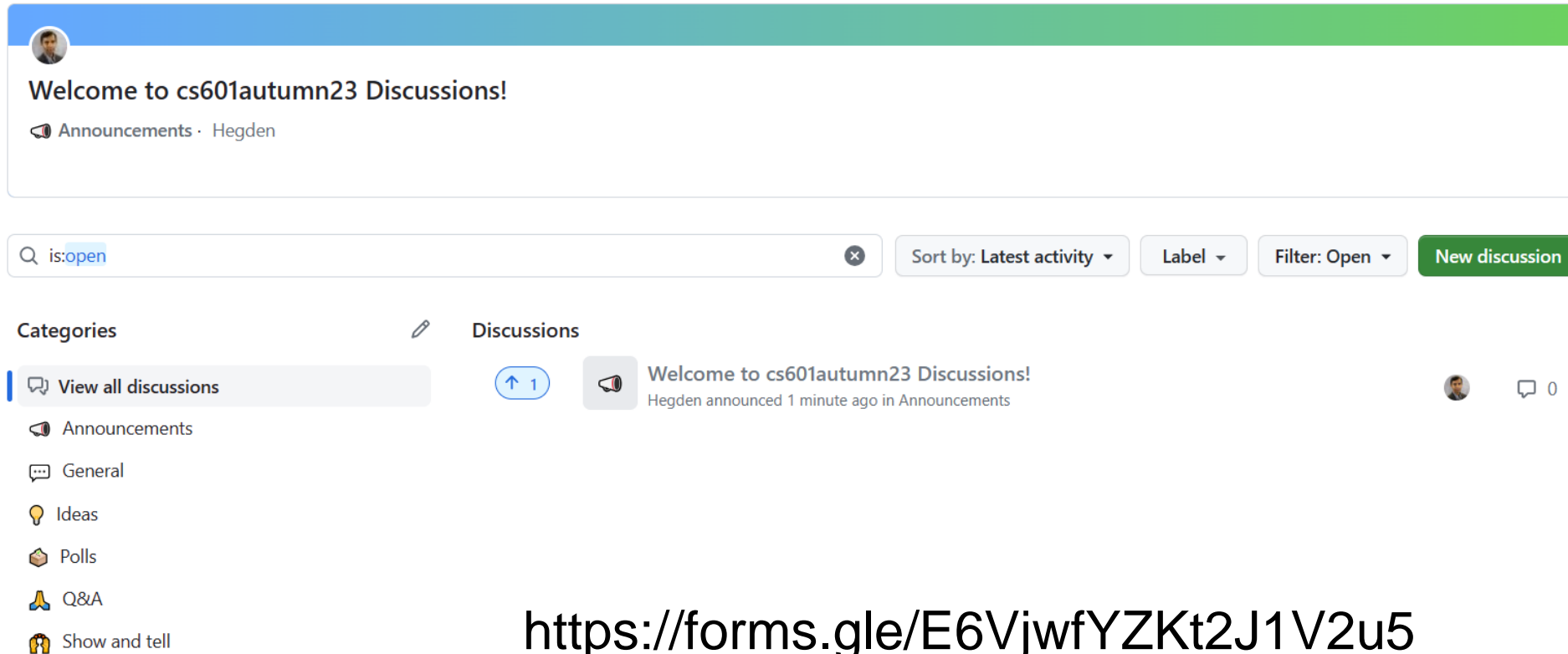
- You are interested in scientific computing
- You are interested in high-performance computing
- You want to build / add to a large software system

Course Webpage

- <https://hegden.github.io/cs601>

GitHub Discussions

- <https://github.com/IITDhCSE/cs601autumn23/discussions>



The screenshot shows the GitHub Discussions interface for the repository `cs601autumn23`. At the top, a welcome banner from user `Hegden` in the `Announcements` category is displayed. Below the banner is a search bar with the text `is:open`, a close button, and filters for `Sort by: Latest activity`, `Label`, and `Filter: Open`. A `New discussion` button is also present. On the left, the `Categories` sidebar lists `View all discussions` (selected), `Announcements`, `General`, `Ideas`, `Polls`, `Q&A`, and `Show and tell`. The main `Discussions` area shows a single announcement from `Hegden` titled `Welcome to cs601autumn23 Discussions!`, posted 1 minute ago. A small icon indicates 1 discussion is shown.

Welcome to cs601autumn23 Discussions!

Announcements · Hegden

Search: `is:open` [Close] [Sort by: Latest activity] [Label] [Filter: Open] [New discussion]

Categories

- View all discussions
- Announcements
- General
- Ideas
- Polls
- Q&A
- Show and tell

Discussions

1

Welcome to cs601autumn23 Discussions!
Hegden announced 1 minute ago in Announcements

<https://forms.gle/E6VjwfYZKt2J1V2u5>

Let us try an exam question (from this course) on ChatGPT

- Question:

Computing $\sqrt{x^2 + y^2}$ is a common problem. A common implementation strategy is as follows:

```
double ComputeHypotenuse(double x, double y) {  
    return sqrt(x*x + y*y);  
}
```

However, the above strategy is not robust. How would you implement a robust code?

- <https://chat.openai.com/share/bcf4d871-21cf-4799-9275-1486345ce6dd>

Takeaways:

- You still need to know the right questions to ask.
- Know if the answer provided makes sense.

Develop intuition

Let us dive into an example....

Example - Factorial

- $$n! = n \times (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$$
$$(n-1)! = (n-1) \times (n-2) \times \dots \times 3 \times 2 \times 1$$

therefore,

Definition1: $n! = n \times (n-1)!$

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);  
}
```