

slido.com  
jhuintprog

## 601.220 Intermediate Programming

Summer 2022, Meeting 2 (June 8th)

# Today's agenda

- Academic ethics for individual homeworks
- Exercise 2 review
- “Day 3” material
  - Git and Emacs
  - Exercises 3-A and 3-B
- “Day 4” material
  - C logical operators and control flow
  - Exercise 4

## Reminders

- CA office hours! (see website for schedule)
- HW0 is due tomorrow (Thursday June 9th) by 11pm
  - You will need to fully set up to use your personal Git repo!
  - You will also need to have access to Gradescope
  - Let me know ASAP if either of these isn't the case
- HW1 is due Wednesday, June 15th

# Academic ethics for individual homeworks

- The individual homeworks (HW0, HW1, HW3, HW5, HW7) must be completed individually
- Do not look at anyone else's code
- Do not allow anyone (other than course staff) to look at your code
- Do not use code from the internet, students who have taken the course previously, etc.
- We will run a similarity comparison on submissions
- Violations will be reported to the student conduct office

## Exercise 2 review

Reading two integers:

```
int a, b;  
printf("Enter two integers: ");  
scanf("%d", &a);  
scanf("%d", &b);
```

Another possibility:

```
int a, b;  
printf("Enter two integers: ");  
scanf("%d %d", &a, &b);
```

## Exercise 2 review

The user might not enter a valid integer. You can detect this by checking the return value of `scanf`, which returns a count of how many values were read successfully.

```
int a;
printf("Enter an integer: ");
if (scanf("%d", &a) != 1) {
    printf("invalid input\n");
}
```

This technique will be useful for HW1.

## Exercise 2 review

Computations on input values a and b:

```
int sum, diff, prod, quot, rem;
```

```
sum = a + b;
```

```
diff = a - b;
```

```
prod = a * b;
```

```
quot = a / b;
```

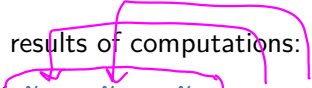
```
rem = a % b;
```

↑  
assignment

## Exercise 3 review

Printing results of computations:

```
printf("%d + %d = %d", a, b, sum);  
printf("%d - %d = %d", a, b, diff);  
printf("%d * %d = %d", a, b, prod);  
printf("%d / %d = %d", a, b, quot);  
printf("%d %% %d = %d", a, b, rem);
```



Note %% in a printf format string means “print a literal % character.”

↓  
\\n



## Day 3 recap questions

- ❶ Why do we use version control system like **git**?
- ❷ Name six common **git** commands?
- ❸ What are the files that must be included in your submission?
- ❹ How do you save and quit on *emacs* editor?
- ❺ How do you search and replace on *emacs*?

# 1. Why do we use version control system like **git**?

Version control systems record and archive a history of “snapshots” the files in a project. (Git calls these snapshots “commits”.)

Advantages:

- Know exactly how code changed, be able to revert to working code if a bug is introduced or if a file is lost/corrupted
- Serves as a backup when you push commits to a remote repository (i.e., on Github)
- Team members can share changes with each other. (This is essential for team projects.)
- Synchronize files between multiple computers

## 2. Name six common **git** commands?

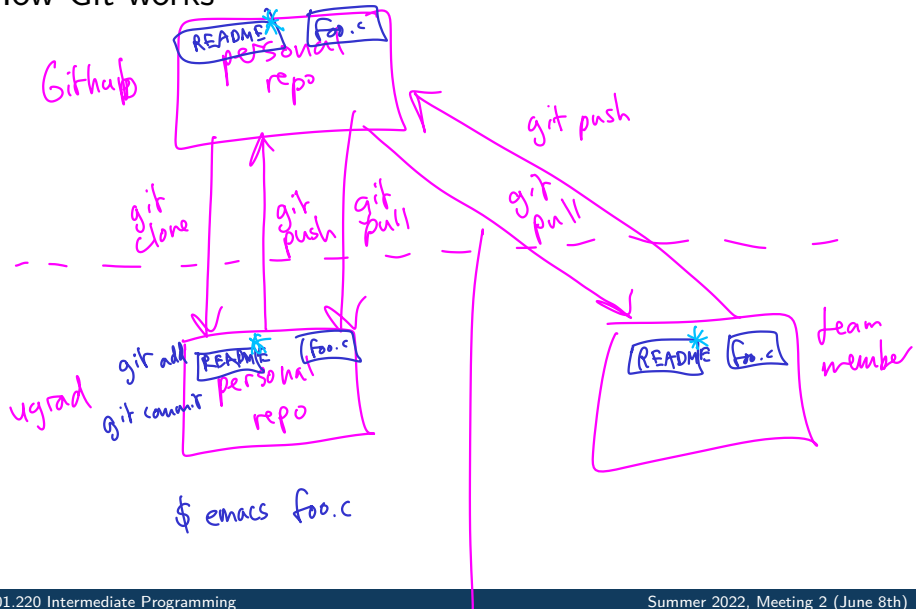
*commit = snapshot*

*git clone*

- ① `git init`: initialize the current directory as a new Git repository
- ② `git add`: specify file(s) that should be part of the next commit
- ③ `git commit`: create a new commit with all of the modifications staged using `git add`, with a meaningful log message
- ④ `git status`: show which files have been modified are haven't yet been added to the repository
- ⑤ `git diff`: show file modifications (relative to the previous commit)
- ⑥ `git push`: send commit(s) to the remote "origin" repository
- ⑦ `git pull`: receive commit(s) from the origin repository

*synch w/ remote repo*

# How Git works



### 3. What are the files that must be included in your submission?

- All source files (e.g., .c, .h files)
- gitlog.txt: summarizes your Git commits
- Possibly: README
- Possibly: a Makefile (once we've covered make)

Create a zip file with all of these files, copy it to your local computer, then upload the zip file to Gradescope. E.g.:

```
cd ~/my220repo/homework/hw0  
zip -9r my_hw0.zip *.c gitlog.txt
```

On your local machine

```
scp USERNAME@ugradx.cs.jhu.edu:my220repo/homework/hw0/my_hw0.zip .
```

pscp

## 4. How do you save and quit on *emacs* editor?

Save: Control-X followed by Control-S

Quit: Control-X followed by Control-C

Pro tip: use multiple terminal windows! Keep your editor (e.g., Emacs) open in one terminal, then use another terminal to run compiler commands, run the program, etc.

## 5. How do you search and replace on *emacs*?

Meta-% (a.k.a. ESC followed by %)

## Work on Exercises 3-A and 3-B

back <sup>to</sup> main meeting  
at 11:30

Exercise 3-A: clone your personal Github repo, practice creating a zipfile and transferring it to your local computer (assignment submission workflow)

Exercise 3-B: Clone the course public Git repo and copy starter code from it (workflow for starting an exercise or homework assignment)

Breakout rooms 1–10 are “social”, breakout rooms 11+ are for individual or small group work.



## Day 4 recap questions

- ❶ Which one is the logical “and” operator in C, `&&` or `&` or both?
- ❷ Which one is the logical “negation” operator in C, `~` or `!` or both?
- ❸ What is the result of evaluating `(34 + 2) / 40 || 80 > 'A' && 15 % 4` ?
- ❹ What does the keyword **break** do in a control structure?
- ❺ What does the keyword **continue** do in loops?
- ❻ How many times is the *initialize* statement in a *for loop* executed?

1. Which one is the logical “and” operator in C, && or & or both?

&& is the logical “and” operator.

& is the *bitwise* “and” operator.

If you're expressing a condition (for an if, if/else, or loop) then you want &&.

2. Which one is the logical “negation” operator in C, `~` or `!` or both?

`!` is the logical negation operator.

`~` is bitwise complement, a.k.a. bitwise negation.

3. What is the result of evaluating  $(34 + 2) / 40 \mid\mid$   
 $80 > 'A' \&\& 15 \% 4 ?$

Key ideas.

- 0 is false, and all other integer values are true
- all logical operators are guaranteed to produce either 0 or 1 as a result

1

#### 4. What does the keyword **break** do in a control structure?

`break` jumps to the code that follows the control structure.

So, it can be used to terminate a loop early.

**Important:** `break` (and `continue`) can make loops harder to understand. They should be used very sparingly, and only when they truly simplify the code.

## break and switch statements

break is also used to end a case (or group of cases) in a switch statement. This is a necessary use of break (as opposed to break in the body of a loop, which is never strictly necessary.)

```
char grade;  
;  
switch ( grade ) {  
case 'a':  
case 'A':  
    [ what to do for  
      grade of A ]  
    break;  
case 'b':  
case 'B':  
    ;  
}
```

## 5. What does the keyword **continue** do in loops?

`continue` jumps to the top of the loop.

Again, this can make the loop harder to understand, so use sparingly.

## 6. How many times is the *initialize* statement in a *for* loop executed?

Once, before the loop starts.

Assume *i* is an `int` variable. These loops are equivalent:

*// while loop*

```
i = 1;
while (i <= 10) {
    printf("%d\n", i);
    i++;
}
```

*// for loop*

```
for (i = 1; i <= 10; i++) {
    printf("%d\n", i);
}
```



## Exercise 4

GPA calculator program: practice

- starting an exercise using code in the public repository
- input using `scanf`
- output using `printf`
- loops
- `switch` statements

Breakout rooms 1–10 are “social”, breakout rooms 11+ are for individual or small group work.

# Notes

# Notes

# Notes

# Notes

# Notes