CSE306
SOFTWARE QUALITY IN PRACTICE

Dr. Carl Alphonce
alphonce@buffalo.edu
343 Davis Hall

www.cse.buffalo.edu/faculty/alphonce/SP24/CSE306
ROADMAP

• Announcements
• Course overview
• Syllabus highlights (full syllabus on website)
• Academic Integrity
• Tasks for this week
• Setting the stage
ANNOUNCEMENTS

Weekly schedule:

• Lecture on Monday, lab on Tuesday
• Lecture on Wednesday, lab on Thursday
• My OH: T/W 1:15-2:45

Labs begin THIS WEEK (i.e. tomorrow)

• Expectation: you should complete within lab period.
SENS ACCOUNT

- Labs are held in Bell 340 and SENS/CSE lab
- You need a SENS computing account
- If you previously had a course in Bell 340 you should have an account.
- If you were registered in this class before today you should have an account.
- If you do not have an account, request one TODAY: https://www.sens.buffalo.edu/accounts
COURSE OVERVIEW

Fix bad code → Write good code
COURSE OVERVIEW

**Fix Bad Code**
- valgrind
- git
- profiler
- debugger
- make
- compiler
- shell script
- teammates
- task board
- automated testing
- sound
- process

**Write Good Code**
and more!
(LEX) Twenty-two lab-based exercises, two per week, done in lab throughout the semester.

(PRE)/(PST) Team projects focused on process. PRE is a pre-assessment in weeks 1 - 3 of the semester. PST is a post-assessment in weeks 12 and 13. Students are required to document their development/debugging process.

(EXP) Two three-week team projects. These projects ask student teams to apply the tools and techniques they have been taught up to that point in the course to existing code. Students are required to document their use of the tools and the results they obtained.

(LPR) A two-part in-lab practical exam, in weeks 13 and 14.
<table>
<thead>
<tr>
<th>LEARNING OUTCOME</th>
<th>INSTRUCTIONAL METHODS</th>
<th>ACTIVITY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Employ static and dynamic analysis tools to detect faults in a given piece of software.</td>
<td>Lecture-based instruction</td>
<td>LEX LPR PRE/PST EXP</td>
</tr>
<tr>
<td>Employ profiling tools to identify performance issues (both time and memory) in a given piece of software.</td>
<td>Lab-based hands-on exercises.</td>
<td>LPR PRE/PST EXP</td>
</tr>
<tr>
<td>Employ testing frameworks to write tests that fail in the presence of software faults, and pass otherwise</td>
<td>Lab-based hands-on exercises.</td>
<td>PRE/PST EXP</td>
</tr>
<tr>
<td>Employ a structured, methodical approach to detecting, testing, identifying and correcting software faults.</td>
<td>Lab-based hands-on exercises.</td>
<td>PRE/PST EXP</td>
</tr>
<tr>
<td>Work productively as a member of a software development team.</td>
<td></td>
<td>PRE/PST EXP</td>
</tr>
</tbody>
</table>
Each piece of student work will be assessed using performance indicators, each of which has a rubric with four performance levels:

- “insufficient evidence”
- “developing”
- “secure”
- “exemplary”

Each performance indicator is assessed independently of the others.
The overall grade for a piece of work is determined by comparing actual performance relative to performance expectations, which increase throughout the course.

Early in a topic the expectation is that performance is close to the "insufficient evidence" level.

Towards the middle of the topic we expect students to be at or above "developing".

Towards the end of the course we expect students to be at or above the "secure" level.

Specific rubrics and performance expectations are available in UBLearns for each assignment.
# Syllabus Highlights

**Activities • Assessment • Grading**

Overall Grade Breakdown

<table>
<thead>
<tr>
<th>Individual (60%)</th>
<th>Team (30%)</th>
<th>Mixed</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEX 30%</td>
<td>PRE 2%</td>
<td>ACT 10%</td>
</tr>
<tr>
<td>LPR 30%</td>
<td>EXP 6% + 6%</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PST 16%</td>
<td></td>
</tr>
</tbody>
</table>

**Code Completion vs Process**
INDIVIDUAL WORK: You must write code, answer questions, and employ tools as an individual, and may not seek direct assistance or answers from classmates.

TEAM WORK: You and your teammates must write the code by together - everyone must contribute.

You may look up (but must cite!) resources for the necessary algorithms and data structures.
TASKS FOR THIS WEEK

Teams & Labs

• Form teams of size 3 or 4 this week, by tomorrow's first lab if possible (use Piazza's "Search for Teammates" post)

• I recommend all team members attend same lab, but not required.

• Labs start this week.

• **One member** of each team must make a private post in Piazza with the UBITs of each person on their team.
(PRE) PROCESS PROJECT

Instructions:
posted on website once teams are formed

Activity log:
keep track of how/when you work
(format given in instructions)

Warm-up on C programming:
Document your programming process at this point in the course
(PRE) PROCESS PROJECT

Compiler

• use gcc compiler with C11 standard

• You can work on any machine, but test on cerf.cse.buffalo.edu (this is our reference system) before submitting: that's where we'll grade!

• If you have trouble connecting to cerf, try turing.cse.buffalo.edu (same filesystem is mounted).
SETTING THE STAGE
Is this code buggy?

#include <stdio.h>

int main(void) {
    printf("%s\n","Hello, world.");
    return 0;
}
SETTING THE STAGE

Is this code buggy?

#include <stdio.h>

int main(void) {
    printf("\%s\n","Hello, world.");
    return 0;
}

It depends - what was the specification for the program?
SETTNG THE STAGE
Is this code buggy?

#include <stdio.h>

int main(void) {
    printf("%s\n","Hello, world.");
    return 0;
}

It depends - what was the specification for the program?

Should the program check the return value of printf?
SETTING THE STAGE

Is this code buggy?

```c
#include <stdio.h>

int main(void) {
    printf("%s\n","Hello, world.");
    return 0;
}
```

It depends - what was the specification for the program?

Should the program check the return value of `printf`?

What is the return value of `printf`?
<table>
<thead>
<tr>
<th>Rate</th>
<th>Taxable Income Bracket</th>
<th>Tax Owed</th>
</tr>
</thead>
<tbody>
<tr>
<td>10%</td>
<td>$0 to $9,325</td>
<td>10% of Taxable Income</td>
</tr>
<tr>
<td>15%</td>
<td>$9,325 to $37,950</td>
<td>$932.50 plus 15% of the excess over $9,325</td>
</tr>
<tr>
<td>25%</td>
<td>$37,950 to $91,900</td>
<td>$5,226.25 plus 25% of the excess over $37,950</td>
</tr>
<tr>
<td>28%</td>
<td>$91,900 to $191,650</td>
<td>$18,713.75 plus 28% of the excess over $91,900</td>
</tr>
<tr>
<td>33%</td>
<td>$191,650 to $416,700</td>
<td>$46,643.75 plus 33% of the excess over $191,650</td>
</tr>
<tr>
<td>35%</td>
<td>$416,700 to $418,400</td>
<td>$120,910.25 plus 35% of the excess over $416,700</td>
</tr>
<tr>
<td>39.60%</td>
<td>$418,400+</td>
<td>$121,505.25 plus 39.6% of the excess over $418,400</td>
</tr>
</tbody>
</table>

https://taxfoundation.org/2017-tax-brackets
IS THIS CODE BUGGY?

Does this implement the table on the previous slide?

double f(double x) {
    if (x < 9325) { return 0.1 * x; }
    if (x < 37950) { return 932.50 + 0.15 * (x - 9325); }
    if (x < 91900) { return 5225.25 + 0.25 * (x - 37950); }
    if (x < 191650) { return 18713.75 + 0.28 * (x - 91900); }
    if (x < 416700) { return 46643.75 + 0.33 * (x - 196150); }
    if (x < 418400) { return 120910.25 + 0.35 * (x - 416700); }
    return 131505.25 + 36.9 * (x - 418400);
}
Is This Code Buggy?

double f(double x) {
    if (x < 9325) { return 0.1 * x; }
    if (x < 37950) { return 932.50 + 0.15 * (x - 9325); }
    if (x < 91900) { return 5225.25 + 0.25 * (x - 37950); }
    if (x < 191650) { return 18713.75 + 0.28 * (x - 91900); }
    if (x < 416700) { return 46643.75 + 0.33 * (x - 191650); }
    if (x < 418400) { return 120910.25 + 0.35 * (x - 416700); }
    return 131505.25 + 36.9 * (x - 418400);
}

The code also doesn't protest if x < 0.

How can we develop software to minimize risk of errors and maximize chance bugs are discovered?
WHAT WE’LL BE DOING IN THIS COURSE

• Learn tools to explore program structure and behavior.

• Consider correctness relative to a specification and performance relative to a requirement.

• Employ a methodical approach to tracking down, identifying, documenting and fixing problems with code.

• Employ a methodical approach to developing code.
STATIC VS DYNAMIC PROGRAM ANALYSIS

• static analysis - done on program without executing it
• dynamic analysis - done on program by executing it