CSE306
SOFTWARE QUALITY IN PRACTICE

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

www.cse.buffalo.edu/faculty/alphonce/FA24/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 Tuesday morning, lab on Tuesday afternoon
- Lecture on Thursday morning, lab on Thursday afternoon
- My OH: M 10:15 - 11:45, T 9:15 - 10:45

Labs begin TODAY:

- Expectation: you should (mostly) complete within lab period.
    - exception: special consideration given for add/drop: you can make up work assigned before you were registered by the day after add-drop.
- Attendance taken in lab and factors into your grade.

LABS IN BALDY 19

- Labs are held in Baldy 19
- Bring your own laptop (or use a departmental laptop)
- If using own machine, ensure you can connect remotely to turing.cse.buffalo.edu or cerf.cse.buffalo.edu
- You can code locally, but do NOT use VSCode for remote development.

COURSE OVERVIEW

$$\begin{tikzcd}
\text{FIX BAD CODE} \arrow[r, bend left] & \text{WRITE GOOD CODE}
\end{tikzcd}$$

COURSE OVERVIEW

debugger

FIX

valgrind

git

profiler

BAD

make

CODE

compiler

WRITE

teammates

GOOD

shell script

CODE

sound

task board

process

automated testing

and more!

SYLLABUS HIGHLIGHTS
ACTIVITIES • ASSESSMENT • GRADING

- (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.

SYLLABUS HIGHLIGHTS
ACTIVITIES • ASSESSMENT • GRADING

LEARNING OUTCOME	INSTRUCTIONAL METHODS	ACTIVITY
Employ static and dynamic analysis tools to detect faults in a given piece of software.	Lecture-based instruction Lab-based hands-on exercises.	LEX LPR PRE/PST EXP
Employ profiling tools to identify performance issues (both time and memory) in a given piece of software.		
Employ testing frameworks to write tests that fail in the presence of software faults, and pass otherwise		
Employ a structured, methodical approach to detecting, testing, identifying and correcting software faults.		LPR PRE/PST EXP
Work productively as a member of a software development team.		PRE/PST EXP

SYLLABUS HIGHLIGHTS
ACTIVITIES • ASSESSMENT • GRADING

Assessment using Performance indicators (PIs)

- 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.

SYLLABUS HIGHLIGHTS

ACTIVITIES • ASSESSMENT • GRADING

Assessment -> Grades

- 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

INDIVIDUAL (60%)	TEAM (30%)	MIXED
LEX 30%	PRE 2%	ACT 10%
LPR 30%	EXP 6% + 6%	
	PST 16%	

CODE COMPLETION
VS
PROCESS

SYLLABUS HIGHLIGHTS
ACADEMIC INTEGRITY

- 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 Thursday's 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?

SETTING THE STAGE

Is this code buggy?

Should the program check the return value of printf?

```
#include <stdio.h>

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

It depends - what was the specification for the program?

SETTING THE STAGE

Is this code buggy?

```
#include <stdio.h>

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

Should the program check the return value of printf?

What is the return value of printf?

It depends - what was the specification for the program?

2017 TAX TABLE (INDIVIDUAL)

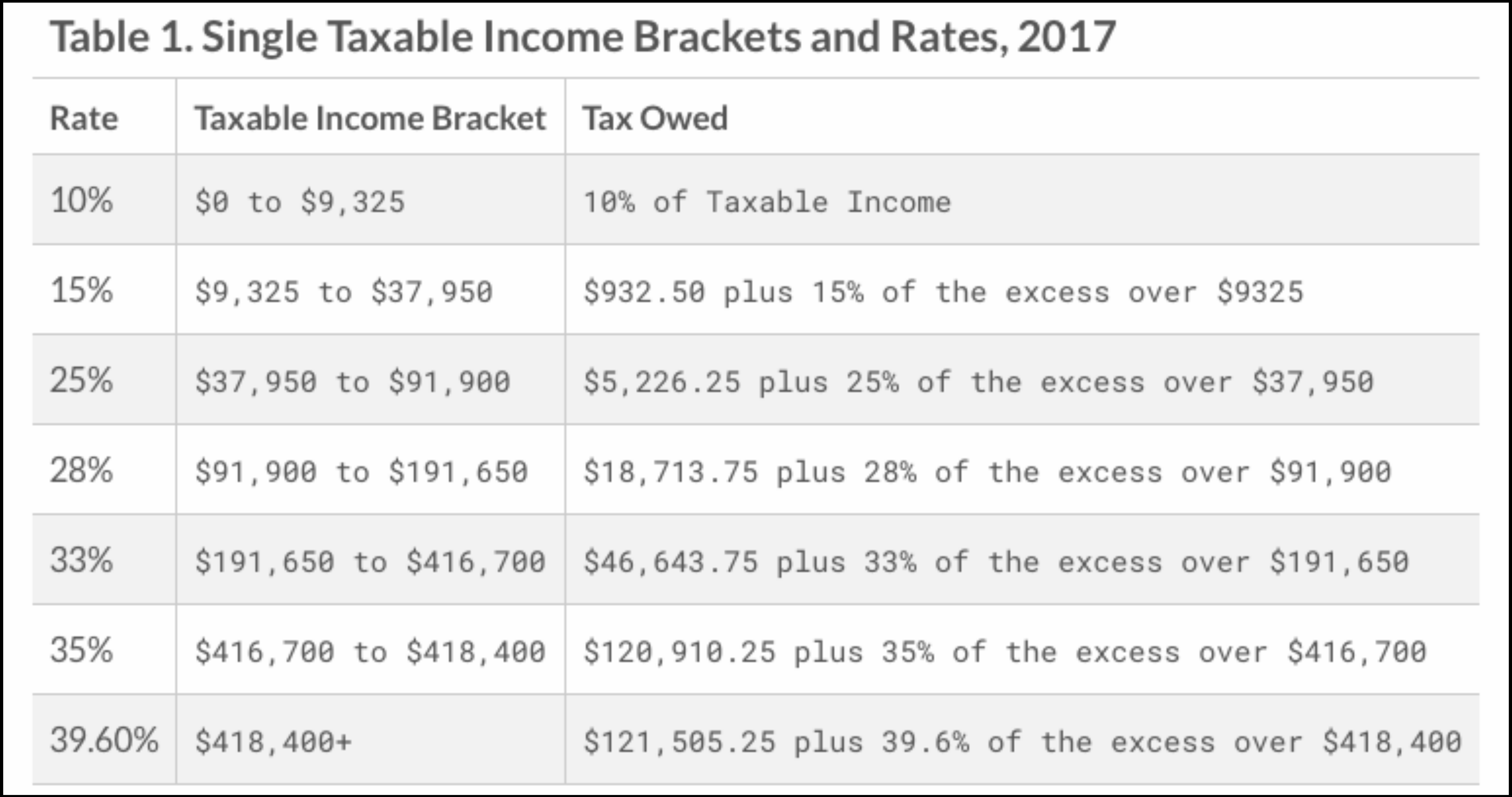


Table 1. Single Taxable Income Brackets and Rates, 2017

Rate	Taxable Income Bracket	Tax Owed
10%	$0 to $9,325	10% of Taxable Income
15%	$9,325 to $37,950	$932.50 plus 15% of the excess over $9325
25%	$37,950 to $91,900	$5,226.25 plus 25% of the excess over $37,950
28%	$91,900 to $191,650	$18,713.75 plus 28% of the excess over $91,900
33%	$191,650 to $416,700	$46,643.75 plus 33% of the excess over $191,650
35%	$416,700 to $418,400	$120,910.25 plus 35% of the excess over $416,700
39.60%	$418,400+	$121,505.25 plus 39.6% of the excess over $418,400

https://taxfoundation.org/2017-tax-brackets

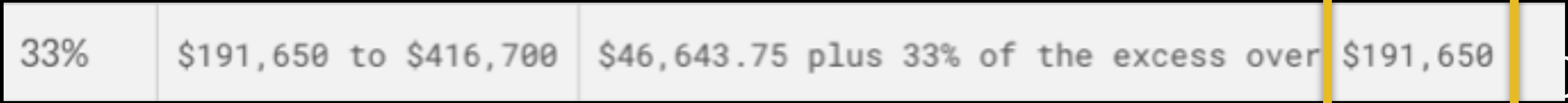
IS THIS CODE BUGGY?

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);
}
```

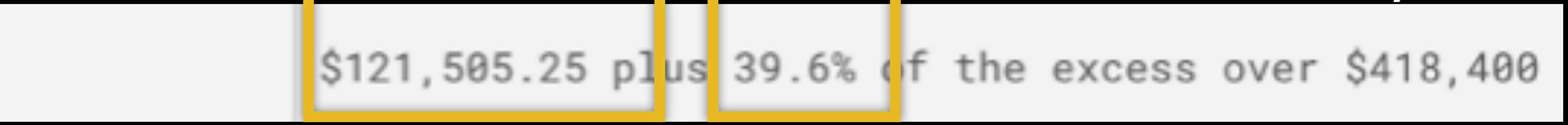
IS THIS CODE BUGGY?

```
double f(double x) {
  if (x < 9325) { return 0.1 * x; }
  if (x < 37950) { return 932.50 + 0.15 * (x - 9325); }
```



33%	$191,650 to $416,700	$46,643.75 plus 33% of the excess over $191,650

```
                                                            }
  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);
```



$121,505.25 plus 39.6% of the excess over $418,400

```
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