CSE410 aka CSE306
Software Quality

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

http://www.cse.buffalo.edu/faculty/alphonce/SP17/CSE410
https://piazza.com/class/iybn33z3aro2p
Syllabus

- Posted on website
- Covered most last class
- Academic Integrity
Departmental Policy on Violations of Academic Integrity (AI)

The CSE Department has a zero-tolerance policy regarding academic integrity (AI) violations.

When there is a potential violation of academic integrity in a course, the course director shall first notify the concerned students. This notification begins the review and appeals process defined in the University’s Academic Integrity statement: http://catalog.buffalo.edu/policies/course/integrity.html

Upon conclusion of the review and appeals process, if the department, school, and university have determined that the student has committed a violation, the following sanctions will be imposed upon the student:

§ 1. Documentation. The department, school, and university will record the student's name in departmental, decanal, and university-level academic integrity violations databases.

§ 2. Penalty Assessment. The standing policy of the Department is that all students involved in an academic integrity violation will receive an F grade for the course. The course director may recommend a lesser penalty for the first instance of academic integrity violation, and the adjudication committees that hear the appeal at the department, decanal and provost level may recommend a lesser or greater penalty.
Reminders

- Piazza

- BitBucket private repositories
  - CSE410Solo-<username>
  - CSE410Team-<username>

- Add PreProcessProject to team repo
Compiler

- use /util/bin/gcc compiler
- use -std=C11 (you can use other options too)
- test on timberlake.cse.buffalo.edu (that's our reference system)
The 13 Golden Rules of Debugging

1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate facts from interpretation
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didn’t fix it, it ain’t fixed
13. Cover your bugfix with a regression test
1. Understand the requirements
2. Make it fail
3. Simplify the test case
4. Read the right error message
5. Check the plug
6. Separate fact from fiction
7. Divide and conquer
8. Match the tool to the bug
9. One change at a time
10. Keep an audit trail
11. Get a fresh view
12. If you didn't fix it, it ain't fixed
13. Cover your bug fix with a regression test
Essential tools

- compiler (e.g. gcc)
- debugger (e.g. gdb)
- memory checker (e.g. memcheck)
- runtime profiler (e.g. gprof)
- automated testing framework (e.g. cunit)
- build tool (e.g. make, gradle)
- code repository (e.g. git)
Classification of bugs

- Common bug (source code, predictable)
- Sporadic bug (intermittent)
- Heisenbugs (averse to observation)
  - race conditions
  - memory access violations
  - (programmer) optimizations
- Multiple bugs - several must be fixed before program behavior changes - consider violating rule #9 “one change at a time”
uncertainty principle

...the uncertainty principle, also known as Heisenberg's uncertainty principle, is any of a variety of mathematical inequalities[1] asserting a fundamental limit to the precision with which certain pairs of physical properties of a particle, known as complementary variables, such as position $x$ and momentum $p$, can be known.

observer effect

...the term observer effect refers to changes that the act of observation will make on a phenomenon being observed. This is often the result of instruments that, by necessity, alter the state of what they measure in some manner.

debugging tools

- instrument code during compilation

- instrumented code may behave differently than uninstrumented code

- in other words: the act of using a debugger may cause a bug to “disappear”, only to reappear when compiled without instrumentation
compiling and running

without debugger

- compile using gcc, with `-o` flag if you want to specify a name for the resulting executable (other than "a.out")
  - `gcc -o factorial factorial.c`
- launch program using by running executable:
  - `factorial 5`

with debugger

- compile using gcc, with `-g` flag to include debugging information in executable
  - `gcc -g -o factorial factorial.c`
- launch program using gdb
  - `gdb factorial`

NB: no program argument supplied in gdb invocation
basic commands

- quit - get out of gdb
- help - on-line help system
- run (with program arguments)
short demo

- bt (backtrace)
- up / down / frame N
- info frame / info args / info locals
- break
- run / step / continue / next

https://www.recurse.com/blog/7-understanding-c-by-learning-assembly