

20240416 Chapter 3 details

Prerequisites to begin working on a project:

(0) - Starts with the Problem Definition (what details are desired)

(a) List of requirements (gathered via user stories)

(b) Architecture of the Problem

CONCEPT: The earlier a defect occurs in the process and later it is detected, the more costly the problem

UML Usage (4/16 and 4/18)

→ Only class diagrams will be used for project

- Organize class hierarchy
- - sign is private. + sign is public.
- Generalization relationship - inheritance - triangle to base class
- Association relationship - aggregation - stored as a variable in another class - solid line
 - Full or empty diamond included for composition versus aggregation
 - Composition is most typical; aggregation is different (more like working together)
- Object type is not relevant in UML (pointer or not) - only shows up within class card
- Book+Pages are composition; not a book without pages/cover
- Dependency (third relationship)
 - Example: function in class A uses class B in a function
- Place Abstract Classes with italic class name in UML
- Entity versus Boundary versus Control
 - MVC design - Model View Controller
 - Model: Entity
 - View: Boundary (user interface)
 - Controller: Control that manipulates interaction between model and view
 - Separation of UI and model code allows for simple classes
- Pure Virtual Functions have no additional distinction, can be bold/italics
-

20240417 Discussion : GDB and Valgrind

g++ filename.ext -g -o newfile.exe

(gdb) break line-number

(gdb) print variablename

(gdb) step - goes into the function code

(gdb) next - runs function but does not enter function code

(gdb) continue - runs to end

(gdb) info breakpoints

(gdb) del break 1

(gdb) quit

exiting the debugger also removes breakpoints

Valgrind:

Memory debugging via memcheck g++ -g -O0 *.cpp -o newfile.exe

valgrind --leak-check=full filename.exe

-track-origins=yes gives locations of memory leaks

additional valgrind details:

valgrind ./filename.exe (runs valgrind and gives list of issues)

Commands show up in the output for further commands

Unit Testing: 20240423

- Manufacturer to Quality relationships
- Unit Tests should be on github pull requests
- Protects your code from others' mistakes
- Write Failing Test - Make Code Work - Eliminate Redundancy
- Unit Testing Versus Integration Testing
 - Unit Testing: SUT (System Under Testing)
 - Arrange: Open part of app to test
 - Act: apply stimulus to part of app
 - Assert: observe resulting behavior and verify results
 - Google Test: gtest primer - assertions
 - Assert Versus Expect true/equal/etc.
 - Assert fails mid-function if incorrect state
 - Expect continues to end of function even if state fails
 - Test cases should not throw, but do more expect/assert cases against values like nullptr
 - Assert is best used when a test after may seg-fault
 - EXPECT_NEAR will take error margin as third argument
 - Can use stringstream to store integer values to compare to specific decimal places
 - Can use output streams as an argument for the location of an output

Stubs and Drivers

- Top-Down = Stub
- Bottom-Up = Driver
- Driver: Module that calls your program
- Stub: Being called by the program

Function and Non-Function Testing

- Function: The actual output value of the test
- Non-Function: Formatting issues, etc.
- Stress testing, like having tons of users log in at once, is non-function

Project Testing

- Code Coverage in Testing:
- Coverage = Lines Executed by Tests / Total Lines
- 80 percent is required coverage

Continuous Integration

main.yml

Change actions/checkout@v4

”Makefile” for github, testing, etc. as a report
Can run several ”steps” (or programs, tests, etc.)

Interfaces

- Interface of a class with 3 functions is those three functions
- The ”Set of Actions” that can be done with this class
- Use of different variable naming conventions for private vs prot.
- Go Interface (slide)
- Dynamic Binding and parent-class pointers (elf family from 10B)
- Push cat and dog objects into an animals vector
- In Essence, writing an Interface in C++ is writing a class
- Creates a duality between interface and implementation without affecting each other

SOLID

- Single-Responsibility Principle (SRP)
 - Splitting up large functions into their individual parts
 - Reduction down to a single task per function
 - Makes the code more testable
-
- Open-Closed Principle (OCP)
 - Making everything into classes?
-
-
- Liskov Substitution Principle (LSP)
 -
 -
 -
- Interface Segregation Principle (ISP)
 -
 -
 -
 -
- Dependency Inversion Principle (DIP)
 -
 -
 -

Milestone 2 Checklist

Exam1

Testing is last topic in order
Labs 1-4
Mostly Multiple Choice, Some Open-Ended questions
Not memorization-heavy

Cheat-Sheet: One page, double sided - can be printed super tiny
Know meanings of function calls but not necessarily memorize the calls
Discussion Review Session 5/8