Intro

Sunday, September 09, 2007
1:21 PM

- Ch1 - Software Development
  - just coding does not work for large projects
  - must analyze & design, then code
  - Phases
    - Problem Analysis & Specification
    - Design
    - Coding
    - Testing
    - Maintenance
  - Waterfall Model
    - classic model
    - phases done sequentially
    - not realistic as real projects may loop back to earlier phases
    - also called software lifecycle
  - Problem Analysis & Specification
    - homework assignments typically do this stage for you, but not the real world
    - Ex: HW1 is a specification
      - Real world may have asked how to code a Line & Circle class such that one could use one var to reference both
    - specification says what the program should do
      - preconditions are requirements for the program
      - postconditions are the consequences of running the program
    - have to take user request (sometimes vague) and formulate specification
  - Design
    - take specifications & plan how to code
    - modularization
      - divide problem into parts that can be tackled separately
    - Top-Down Design
      - start w/ whole specification & subdivide into separate parts
      - each part may be further divided
      - continue until have small, manageable parts that can be added together to solve the problem
      - solution typically has two parts
        - storage structures - how to store the data/ input
        - algorithms - actions, eg processing data
    - Object Oriented Design
      - top-down focuses on tasks
      - OOD focuses on objects that contain data & operations
      - object is an instance of a class
    - Pseudocode can be used for either type of design to express how the design should be coded
    - Try to handle as many conditions as possible in the design
      - eg mem allocation failure, bad user input, no input
    - use design phase to optimize code
      - chose the storage structure (eg data structures) that are

        suited for the problem
        evaluate different algorithms to find which uses less time and for memory

- Coding
    - translate design into actual program
    - many languages can be used
        - if using OOD, need object oriented language
    - code should be readable & documented
        - makes easier for others to read
        - "self commenting code" is not always true
        - put comments for each class and/or function that states your name, the date, its purpose, preconditions & post conditions
        - note any special segments of code
        - use readable class & var names
- Testing
    - errors are to be expected
    - test code w/ as many types of input as possible (verification)
    - make sure code matches specifications (validation)
    - fix one error at a time
        - sometimes errors compound
        - fix one, see what it affects
        - if fix has unintended consequences, only have to check one change
    - types of errors
        - syntax
        - run-time
        - logic
    - syntax errors
        - detected at compilation
        - long page could be caused by one error
    - run-time errors
        - program compiled but does not run as expected (crashes)
        - divide-by-zero, index-out-of-range
    - logic errors
        - program doesn't crash, but doesn't behave correctly
        - coding error (eg < instead of >)
        - unexpected user input
        - flaw in design
    - testing should try to find these errors
        - use a variety of input
        - test boundary values
        - test all execution paths possible
        - does NOT prove code is error-free, just that worked for what was tested
- Maintenance
    - fix any bugs missed in testing
    - add new features
    - update for new hardware, OS, etc
    - update for policy changes
    - eventually should result in obsolescence
        - code is retired
        - new program created
        - often most neglected part of lifecycle

Ch 2.1 - First Look at ADTs

- What is an abstract data type?
  - function of object is defined without considering implementation
  - focus on what object does
- Implementation
  - provides data storage & algorithms
  - says how to do object's tasks
  - don't need to know implementation details to be able to use object

Ch 3.1 - Data Structures, ADTs & Implementations

- data structures used to store data
  - can work w/ algorithm
    - make code easier (eg array vs many single value vars)
    - access data faster
    - use less memory
  - book refers to a data structure as storage
    - misleading since many data structures imply both storage & accessing
    - eg a Stack stores data a certain way and has push() & pop()

Procedural & Object-Oriented Programming (Ch 3.6 & 4.1)

- Procedural programming
  - design implemented a series of functions & variables
  - not encapsulated
  - action oriented (verb)
- Object-Oriented programming
  - create objects that have vars & functions for a specific purpose
  - encapsulated
  - focuses on subject, not action (noun)
- data structures can be implemented either way

Ch 9.1 Reusability & Genericity

- don't want to reinvent the wheel
- good data structure can store many kinds of data
- overloading & templates can be used to "Write once, use many"
- libraries can provide commonly used algorithms & data structures
  - eg math library provides square root
- language has to support overloading and/or templates
  - some languages only support "aliases"
  - still have to recompile program to change stored data (eg int to double)

Ch 14.1 Overview of OOP

- Properties of OOP
  - Encapsulation
  - Inheritance
  - Polymorphism
- Encapsulation
  - data storage & related functions bundled together

- separate definition (header file) from implementation (cpp file)
  - user's code should not rely on implementation details
  - implementation could be changed w/o affecting user's code

Inheritance
- derive new classes from existing
  - add new features
- create base classes
  - has general features
- when doing OOD, have to consider structure of inheritance trees (class hierarchies in book)

Polymorphism
- meaning depends on context
- compiler waits until run-time to associate function call w/ function body