Software Engineering

CS305, Autumn 2020

Week 10

Class Progress…

- Last Week
  - RUP phases,
  - Software Construction
    - Inspections/Reviews
- This week
  - Software Construction
    - Coding
    - Refactoring
    - Introduction to testing and unit testing (if time permits)

Coding

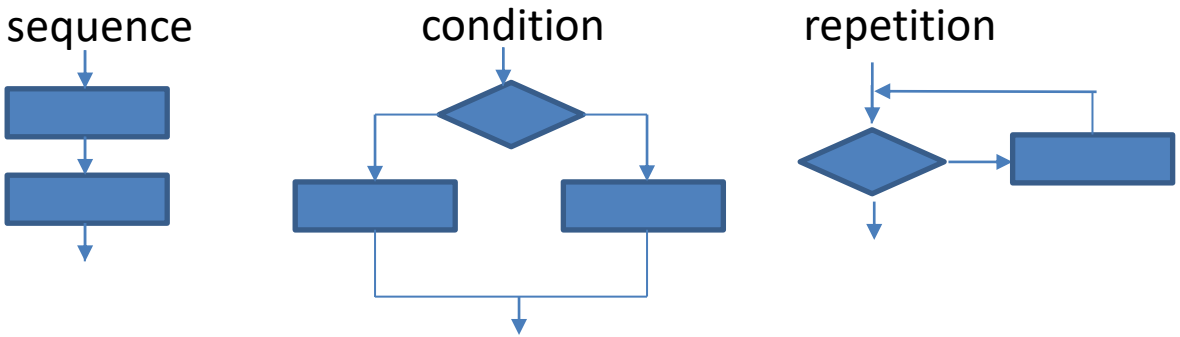
Coding

- Could involve:
  - Writing source code / programming in a chosen language
  - Automatic generation of source code using a design representation of the component to be constructed
  - Automatic generation of executable code using a *fourth-generation* language – program generating language

Human understanding is facilitated by linear sequence of logical statements

Programming Paradigms

- Unstructured Programming
  - Writing a sequence of commands or statements that access 'Global' data. E.g. Assembly lang. programming.
- Structured Programming (sometimes used interchangeably with procedural programming)
  - Dijkstra's advice on using simple logical constructs of:



  - Focus on writing 'modular' programs. Have single-entry and single-exit for a procedure / function (control construct). E.g. C, Assembly lang. programming

Programming Paradigms

- Object Oriented Programming
  - Modeling real-world objects. Data is the centerpiece. Combine data and functions, allow code reuse, incremental dev. maintainability, modularity. *(more in Week3 lectures).* E.g. C++, Java
- Functional Programming
  - Focus on what to do and not how to do. Don't create state that is changeable. E.g. Lisp, Racket
- Concurrent Programming
  - Focus on concurrent execution of a sequence of statements.
  - Parallel programming is a type.
  - E.g. Threads programming (Java threads), Open MP, MPI, CUDA-C.

Coding Principles

- Ensure that the problem is well-understood before coding (i.e. design is clear, programming language is clear)
- Follow Dijkstra's advice and create modular code that is highly cohesive and loosely coupled
- Select data structures that meet the design objectives
- Create readable code (have indentation, blank lines, and comments)
- Select meaningful names for variables, functions, and follow coding standards and best practices
  - `tmp, temp, data` are "symptoms of programmer laziness".
  - (for GCC) https://gcc.gnu.org/wiki/CppConventions
- Get code reviewed by peers

Code Review – class exercise

- Review the following Fortran code

```
 1 DOUBLE PRECISION FUNCTION SIN(X, E)
 2 C     THIS DECLARATION COMPUTES SIN(X)TO ACCURACY E
 3         DOUBLE PRECISION E,TERM,SUM
 4         REAL X
 5         TERM=X
 6         DO 20 I=3,100,2
 7         TERM=TERM*X**2/(I*(I-1))
 8         IF(TERM.LT.E)GO TO 30
 9         SUM=SUM+(-1**(I/2))*TERM
10      20 CONTINUE
11      30 SIN=SUM
12         RETURN
13         END
```

Code Review – class exercise

- Review the following Fortran code

- `C` is comment to end of line
- The `CONTINUE` statement is often used as a place to hang a statement label, usually it is the end of a `DO` loop. If the `CONTINUE` statement is used as the terminal statement of a `DO` loop, the next statement executed depends on the `DO` loop exit condition.
- `.LT.` is less than
- `**` is exponentiation (has higher priority than *)

- `DO` *`label var = expr1, expr2, expr3`*
  `statements`
  `Label CONTINUE`

`var` is the loop variable (often called the *loop index*) which must be integer. *`expr1`* specifies the initial value of *`var`*, *`expr2`* is the terminating bound, and *`expr3`* is the increment (step).

```
1  DOUBLE PRECISION FUNCTION SIN(X, E)
2  C     THIS DECLARATION COMPUTES SIN(X)TO ACCURACY E
3        DOUBLE PRECISION E,TERM,SUM
4        REAL X
5        TERM=X
6        DO 20 I=3,100,2
7        TERM=TERM*X**2/(I*(I-1))
8        IF(TERM.LT.E)GO TO 30
9        SUM=SUM+(-1**(I/2))*TERM
10    20 CONTINUE
11    30 SIN=SUM
12       RETURN
13       END
```

Code Inspection Checklist (excerpt)

1. Data (DA)

- Is each variable correctly typed?
- Is each variable initialized before use?
- Is the initialization appropriate for the type?
- Can global variables be made local?
- Are buffer overflows checked?
- Is dynamically allocated memory freed?

2. Interface (IF)

- Are appropriate values returned from functions?
- Do function calls have correct parameter types/values?
- Are return values tested?

3. Functionality (FN)

- Do loops terminate?
- Do all loops iterate the correct number of times (no off-by-one errors)?
- Is behavior correct if a loop is never entered?
- Is there dead (unreachable) code?
- Do all switch statements have a default case?
- Do all switch arms have break statements? If not, is the ``fall through'' correct?

4. Input/Output (IO)

- Are files opened before use?
- Are files closed after use?
- Are error conditions checked?

5. Other (OT)

- Any defect discovered that does not fall into one of the above categories.

Slide courtesy: Alex Orso, CS3300

Further Reading

- Code Reviews:

http://web.mit.edu/6.005/www/fa16/classes/04-code-review/

Misc: "The Mess We're In" – Joe Armstrong
https://youtu.be/lKXe3HUG2l4

Pay special attention to the slide on "7 deadly sins" at around 8:00