Software Design Methods

These lecture notes created by Mr. James B. (Jim) Carlson, NCSU

Overview

- Software Goals
- Why *design* software before *coding* it?
- How should software be designed?
  - Pseudo-code
  - Flow charts
  - State machines
- How should software be coded (written)?

- Useful books
  - **The Practice of Programming**, Brian W. Kernighan & Rob Pike, Addison Wesley, 1999
  - **Real-Time Systems Development**, Rob Williams, Butterworth-Heinemann/Elsevier, 2006

Software Goals

- Simplicity – program is short and simple

- Clarity – program is easy for humans and machines to understand

- Generality – program can be used for a broad range of situations

Software Design

How do you think companies create software?

- Just dive in and start writing code, or
- Plan the architecture and structure of the software?

Software is like any engineering project - you need to identify WHAT you want to do and HOW you want to get there.

- WHAT = requirements
- HOW = development process

How do you know you developed the software successfully?

- Compare the finished product to the requirements (compliance)

Why Bother?

"He who fails to plan, plans to fail"
"Poor planning produces pathetically poor performance"

- Most companies have an established process for developing hardware and software.

- Software development processes can differ between companies, or even between projects in the same company.

- Software development can occur at the same time that hardware is designed (co-development), especially with embedded products. A delay in either affects the timing of the other.

What is an Algorithm?

A formula? A solution? A sequence of steps? A recipe?

An algorithm is created in the design phase

How is an algorithm represented?

- Typically represented as pseudo code
- Historically represented as flowcharts

Do yourself a favor – write algorithms before code – always!

Software Design Representations

- Pseudocode
  - Easy to write but vague
- Flow Chart
  - Good for describing an algorithm: steps in processing, with conditional (if-else) and repeated (loop) execution
- State machine
  - Good for describing system with multiple states (operating modes) and transition rules

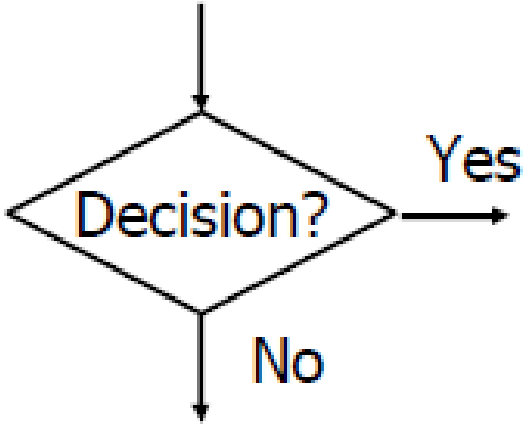
Pseudo Code

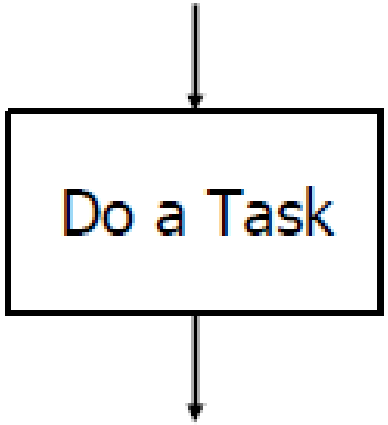
- Pseudo code is written in English to describe the functionality of a particular software module (subroutine)
- Include name of module/subroutine, author, date, description of functionality of module, and actual steps
- Often you can take the pseudo code and use them lines in your program as comments
- Avoid a very fine level of detail (although this may sometimes be difficult to do)
- Avoid writing code – use English, not assembly language (or higher-level language) instructions

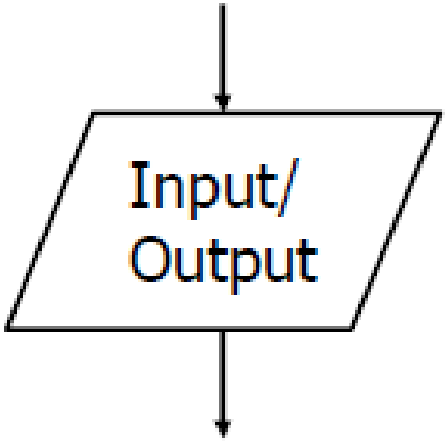
Software Design Process

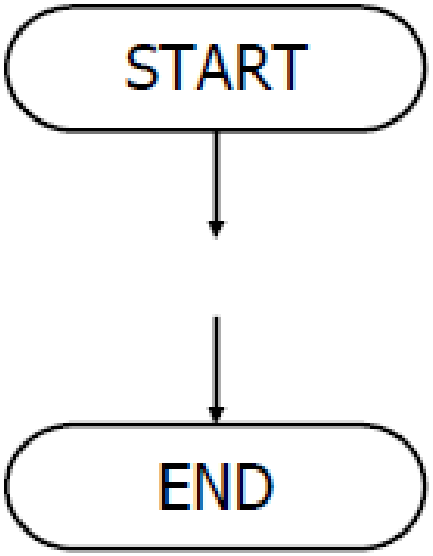
1. Study the problem FIRST (THINK!!).
2. Write the important parts of your problem down.
   Define the **requirements of** the system.
3. Break the problem into manageable pieces.
   Solve each of the pieces individually.
4. Write an algorithm of the solution of your problem, or for each piece you identified.
5. Define test cases for your program
   1. Every requirement should be covered by one or more test.
      Otherwise, how do you know the code meets the requirements?
   2. More details in software testing lecture.
6. Develop the code incrementally *(function-by-function).*
   1. Write function code, including comments.
   2. Test the function.
   3. Fix bugs in this function before starting a new one!
7. Assemble program from modules, testing as you go

Flowchart Symbols (Control Flow)









How Should Software be Coded?

- Code has two requirements
  - To work
  - To communicate how it works to the author and others
    - After the code's author wins the lottery and quits the company, how hard do you want it to be to pick up the pieces?
- Variations in coding styles confuse the reader, so define two aspects of coding style to avoid variation
  - Syntax
  - Semantics
- So use a *Coding Standard or Style Guide* to define correct practices
  - Naming conventions
  - Memory allocation
  - Portability
  - ISRs
  - Comments
  - File locations
  - Eliminates arguments over minor issues

Example Coding Style Guidelines

1. Names
   1. Use descriptive names for global variables, short names for locals
   2. Use active names for functions (use verbs): Initialize_UART
   3. Be clear what a boolean return value means! Check_Battery vs. Battery_Is_Fully_Charged
2. Consistency and idioms
   1. Use consistent indentation and brace styles
   2. Use idioms (standard method of using a control structure): e.g. for loop
   3. Use else-if chains for multi-way branches
3. Expressions and statements
   1. Indent to show structure
   2. Make expressions easy to understand, avoid negative tests
   3. Parenthesize to avoid ambiguity
   4. Break up complex expressions
   5. Be clear: child = (!LC&&!RC)?0:(!LC?RC:LC); is not clear
   6. Be careful with side effects: array[i++] = i++;

Example Coding Style Guidelines

4. Macros
   1. Parenthesize the macro body **and** arguments

      ```
      #define square(x)    ((x) * (x))
      ```

5. Magic numbers
   1. Give names to magic numbers with either #define or enum

      ```
      #define MAX_TEMP (551)
      enum{ MAX_TEMP = 551, /* maximum allowed temperature */
            MIN_TEMP =  38,   /* minimum allowed temperature */ };
      ```

   2. Use character constants rather than integers: if ch==65 ???? if ch ==‘A’
   3. Use language to calculate the size of an object: sizeof(mystruct)
6. Comments
   1. Clarify, don’t confuse
   2. Don’t belabor the obvious
   3. Don’t comment bad code – rewrite it instead
   4. Don’t contradict the code

Example Coding Style Guidelines

7. Use a standard comment block at the entry of each function
    1. Function Name
    2. Author Name
    3. Date of each modification
    4. Description of what function does
    5. Description of arguments
    6. Pre-conditions
    7. Description of return value
    8. Post-conditions
8. Defensive programming
    1. Upon entering a function, verify that the arguments are valid
    2. Verify intermediate results are valid
    3. Is the computed value which is about to be returned valid?
    4. Check the value returned by any function which can return an invalid value
9. No function should be more than 60 lines long, including comments.