# Arduino Programming Part 7: Flow charts and Top-down design

EAS 199B, Winter 2013

Gerald Recktenwald
Portland State University
gerry@me.pdx.edu

## Goals

Introduce flow charts

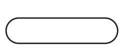
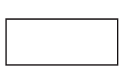
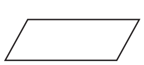
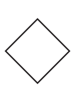
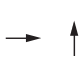
- A tool for developing algorithms
- A tool for documenting algorithms
- A visual method of communicating about any sequential or iterative process
- Great for visual learners!

Top-down design

- One technique for creating a plan for large, multi-step problems
- Not tied to flow charts, but can be used effectively with flow charts



## Flow chart symbols

| | | |
|---|---|---|
|  | Terminator | Start or stop a sequence. May contain module name. |
|  | Process | A step in the process or computational algorithm |
|  | Data input | Information from outside of the algorithm or process |
|  | Decision | Choose a flow path for continuing the algorithm or process |
|  | Flow indicators | Connect other elements |
|  | Connector or Junction | Optional joint where flow indicators merge |

## Exercise 1

Draw the flow chart to read and display the salinity value on the LCD monitor

Keep it simple

- 5 or so symbols (not counting arrows)
- Describe only the high level actions

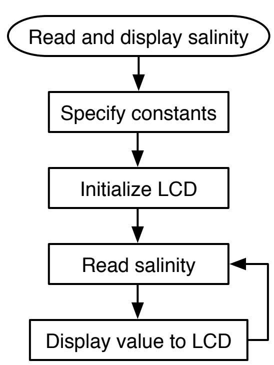


## Exercise 1

Your answer goes here.



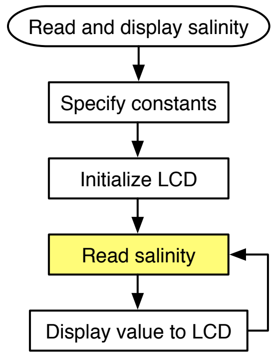
## Exercise 1



# Exercise 2

Expand the "Read salinity" step in another flow chart

- Keep it simple
- "analog data" is an external input

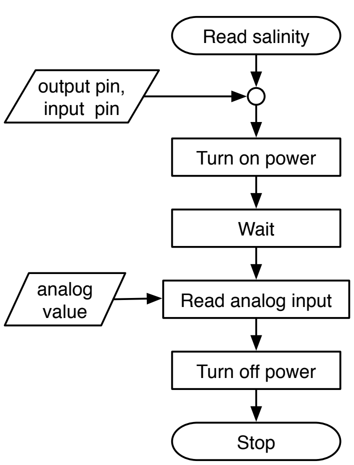



# Exercise 2

Your answer goes here.



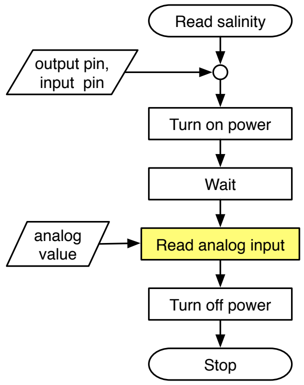
# Exercise 2



# Exercise 3

Expand the "Read analog input" step in another flow chart

- Compute the average of n readings
- "analog data" is an external input



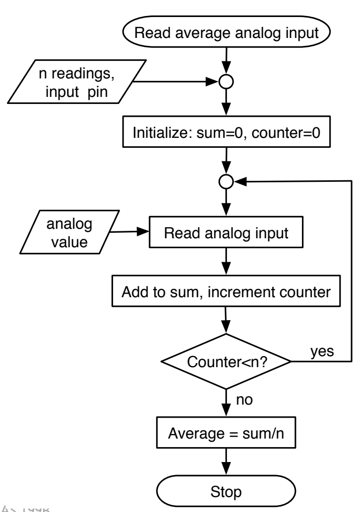



# Exercise 3

Your answer goes here.



# Exercise 3



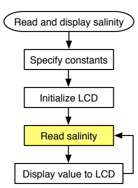
# Top-down design

1. Start with a general statement of the solution
   a. List the main steps
   b. Don't worry yet about details
2. Pick one of the steps
   a. Break this step into a manageable number of sub-steps
   b. Don't worry about too many of the details
   c. Apply step 2 to one of steps just generated



# Top-down design

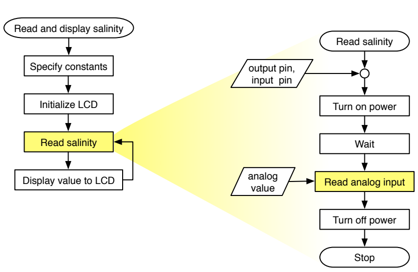
Recursive refinement: from general to specific





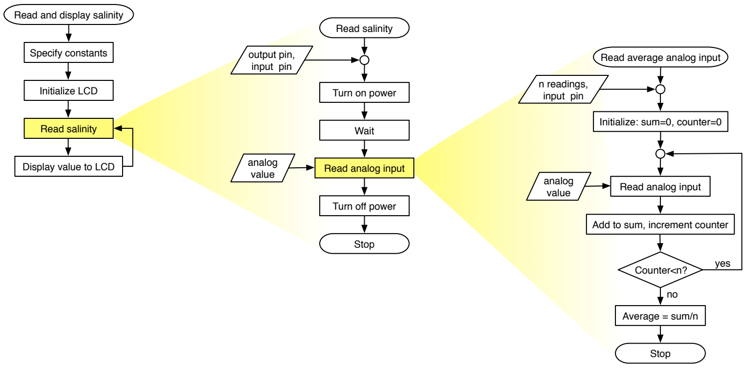

# Top-down design

Recursive refinement: from general to specific



# Top-down design

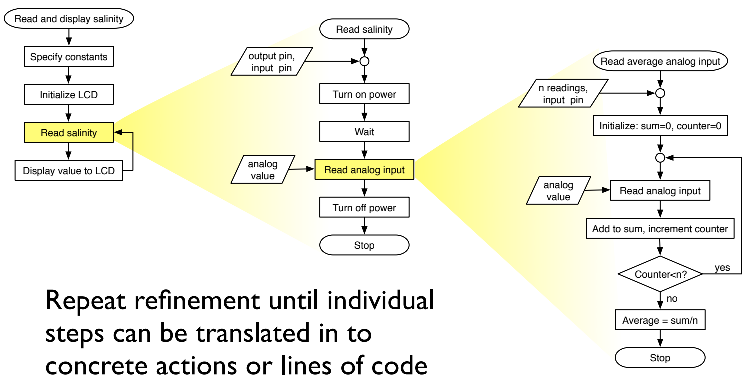
Recursive refinement: from general to specific






# Top-down design

Recursive refinement: from general to specific




Repeat refinement until individual steps can be translated in to concrete actions or lines of code



# Extending top-down design to salinity control of the fish tank

Main tasks

- Measure salinity
- Display salinity on the LCD panel
- Check: Are we in the deadtime?
  - If yes, skip to next loop iteration
  - If no, check for out of deadband condition
    - If salinity is above UCL, add fresh water
    - If salinity is below LCL, add salty water

Each of the tasks could (should!) be decomposed into smaller steps with a top-down design process

# Core control algorithm

```
//  File:  wait_for_deadtime.ino
//
//  Structure of salinity control code to implement a deadtime during which
//  no salinity correction is made.  This code is incomplete and will not compile.

unsigned long last_salinity_update;  //  Time of last correction

void setup() {
  Serial.begin(9600);
  last_salinity_update = millis();   //  Initial value; change later
}

void loop() {
  float LCL, UCL, salinity;
  int deadtime = ... ;

  salinity = salinity_reading( ... );
  update_LCD( ... );

  // -- Check for deadtime
  if ( ( millis() - last_salinity_update ) > deadtime ) {

    if ( salinity>UCL ) {
      //  add DI water:  several missing steps
      last_salinity_update = millis();
    }

    if ( salinity<LCL ) {
      //  add salty water:  several missing steps
      last_salinity_update = millis();
    }
  }
}
```



# Core control algorithm: managing deadtime

```
//  File:  wait_for_deadtime.ino
//
//  Structure of salinity control code to implement a deadtime during which
//  no salinity correction is made.  This code is incomplete and will not compile.

unsigned long last_salinity_update;

void setup() {
  Serial.begin(9600);
  last_salinity_update = millis();
}

void loop() {
  float LCL, UCL, salinity;
  int deadtime = ... ;

  salinity = salinity_reading( ... );
  update_LCD( ... );

  // -- Check for deadtime
  if ( ( millis() - last_salinity_update ) > deadtime ) {

    if ( salinity>UCL ) {
      //  add DI water:  several missing steps
      last_salinity_update = millis();
    }

    if ( salinity<LCL ) {
      //  add salty water:  several missing steps
      last_salinity_update = millis();
    }
  }
}
```

Global (and persistent) value to remember time of last change to the system

Set initial value. Update again later

Don't even consider making changes while we are in the deadtime. "In the deadtime" means "current time minus time of last system change" is greater than deadtime

Update the "time of last system change" every time the salinity is changed



# Core control algorithm: task decomposition

```
//  File:  wait_for_deadtime.ino
//
//  Structure of salinity control code to implement a deadtime during which
//  no salinity correction is made.  This code is incomplete and will not compile.

unsigned long last_salinity_update;

void setup() {
  Serial.begin(9600);
  last_salinity_update = millis();
}

void loop() {
  float LCL, UCL, salinity;
  int deadtime = ... ;

  salinity = salinity_reading( ... );
  update_LCD( ... );

  // -- Check for deadtime
  if ( ( millis() - last_salinity_update ) > deadtime ) {

    if ( salinity>UCL ) {
      //  add DI water:  several missing steps
      last_salinity_update = millis();
    }

    if ( salinity<LCL ) {
      //  add salty water:  several missing steps
      last_salinity_update = millis();
    }
  }
}
```

1. Function to read salinity sensor and convert reading to mass fraction.

2. Function to update LCD

3. Function to determine size of the correction and open the valve. One function could handle both corrections if you design it to use the right input arguments

# Recommendations

- Work in small increments
  - Identify a task, build the code to test that task independently of the entire control algorithm
- Write functions to do specific tasks
  - Read salinity sensor and convert to mass fraction
  - Update display
  - Determine size duration of valve opening, and open it
- Document your code as you write it
- Save backups of working code and testing codes
- Use Auto Format to clean up code