Modular Programming:
Where the rubber meets the Code

Gabriel Hugh Elkaim
Fall 2015
The three uses of static

1) Allocate a variable at startup and retain value between function calls = $\phi$, Permanence.

2) Scope restriction — cannot be accessed static int foo(void);

3) Pseudo-Globally — Module Level Variables
   static uint64_t memoryCounter;
   (uint64_t ReadFromMemoryCounter(void)
   return memoryCounter;
Encapsulation (1.2)

functions

int foo

char bar

FunctionA()

FunctionB()

FunctionC()

Data

functions

Implement my interface
Helper function.

Data oc specific
Encapsulation (2.2)

- interface definition
- implement the functionality

```
int foo
char bar

FunctionA()
FunctionB()
FunctionC()
```

```
RC_Init/RC_End
RC_Reset/RC_Rearm_Pins
RC_Soft_Term
RC_Cold_Term
```
What goes into a header file?

Prototypes for `PUBLIC` functions

Algorithms that are `PUBLIC`

Documentation (comment) `C++`

`PUBLIC` data structures

{ `T ypedefs`, `Structs`, `Unions`, `Enumerators`, `Bitfields` }
What shouldn’t be in a header file?

1. **Not in the .h file**
2. **No functions**
3. **No executable code**
4. **No module variables**
5. **No static functions/procedures**

- \( .h = \text{interface} \rightarrow \text{PUBLIC} \)
- \( .c = \text{implementation} \rightarrow \text{PRIVATE} \)
Where do you #include the header file?

Every time I need access:

```c
#include _motormodule_h
#define _motormodule_h

: code:

#include
```
Programming Style Issues

Layout within a module

K & R  

XUT.swift F

Module Level Comment
#includes
library /module headers
Public header for this module
Private Constant Definitions
Private Macro Definitions
Private Type Definitions
Private Variables
Private Function Prototypes
Code

One "True" Indentation Style

if (NewTime() && ((GetTime() % 1050) == 0))
    return 1;
else
    return 0;

while (1)
{
    MSES_HandleEvents();
}

void putKey(SERVICE_PARAM)
{
    putchar( GET_SHARED_BYTE() );
    putchar('.');
}

Use of White Space

Use of Comments

Naming Conventions
Module Design by Interface Specification

• View
  – The module provides Services to the rest of the code

• Design Activities
  – Specify the services
    • Describe Functionality
  – Name the Services
  – Design the implementation
Module: Communications to UI on PC

To avoid hanging up the master during the transmission or reception of messages, this module should implement buffered, interrupt driven transmit and receive. The communications routines for this module will need to be interrupt driven because the UI may send its message at any time.

Char InitializeUICommunications(void);
Do whatever hardware and software initialization necessary to prepare for communications with the UI on SC1.

Void TellUINewUserReady(void);
Should send the message to the UI that a new iButton has been inserted and read.

Unsigned char IsNameReady(void);
Should check to see if a new name is ready from the UI. Return TRUE if a new name is ready, FALSE otherwise.

Unsigned char GetNewName(unsigned char NameSpace[]);
Should copy the name gotten from the UI into the array NameSpace. The copy operation should copy no more than 16 characters, including the terminating NULL. Should return TRUE if there was a new name ready, FALSE otherwise.
Design the interfaces to modules

- Design interface for:
  - Driving the platform
  - Gathering Sensor data

- Produce
  - Public Interface specification
  - What are the details that are being hidden?
void setup() {
  Drive
  // Set up the hardware to drive my motors, read sensors, etc.
}

void loop() {
  // Serial.println("Starting Drive Setup");

  // Drive
  Drive Fill Step
  // Drive Straight (with speed)
  // Drive Turn (turn type, speed, dir)
  // - tank, pivot, arc, slow

  // Serial.println("Ending Drive Setup");
}

void Drive End() {
}
STATE EXPLOSION PROBLEM

Hierarchical State Machines

- **Harel** - 1980's

Statecharts

allows me to

3am in a
A Possible Top-Level State Diagram

- Enter
  - Finding Tape I
  - Tracking Tape
    - Found T
      - Found Tape
      - Finding Tape II
      - Found Tape
  - Stop
    - Found Tape
Work out State Diagrams to Implement Finding Tape I
Event

Top

Next level

If (end) ☹

Return to event

Next level

If (end) ☹

Return event
Let

ES Timer list timer (4, ..., 5);

LED ON

LED OFF

Turn LED off after time.
Finding Tape

- Enter
- Acquiring
- Beacon Acquired
- Driving to Tape
- Stop
- Hit Tape
Acquiring

Entry → Turning CW

Left Sensor Peaks
Set Slow Speed, Reverse

Turning CCW

Acquired

L ≈ R
Stop Turning, Post Beacon Acquired
Implementing Hierarchical State Machines

- What do you need?
State Machine Function Template

If current state is state one
  Execute During function for state one
  If an event is active
    If event is event one
      Execute action function for state one : event one
    Decide what the next state will be
  Endif
  If event is event two
    Execute action function for state one : event two
  Decide what the next state will be
  Endif
Repeat the block above as required for each of the possible events affecting this state.
If next state is different from current state
  Execute exit function for state one
  Execute entry function for new state
  Modify state variable to reflect the new state
Endif
Endif
Return from state machine function
Module

d:\me218b\Lectures\Lecture 29\SMTemplate.c

Description
This is a template file for implementing state machines.

Notes

History
When       Who       What/Why
----------  ---------  ----------
02/18/99   10:19     jec         built template from MasterMachine.c
02/14/99   10:34     jec         Began Coding

*****************************************************************************
/*----------------------------- Include Files -----------------------------*/
/* include header files for this state machine as well as any machines at the
 next lower level in the hierarchy that are sub-machines to this machine */

****************************************************************************
/ *************************************************************************/
/*----------------------------- Module Defines -----------------------------*/
// define constants for the states and event for this machine

*****************************************************************************
/ *************************************************************************/
/*----------------------------- Module Functions ----------------------------*/
/* prototypes for private functions for this machine, things like entry & exit
 functions. */

*****************************************************************************
/ *************************************************************************/
/*----------------------------- Module Variables ---------------------------*/
// everybody needs a state variable, you may need others as well
static unsigned char CurrentState;
void RunStateMachine(unsigned char CurrentEvent )
{
    unsigned char NextState = CurrentState;

    switch ( CurrentState )
    {
    case STATE_ONE :  // If current state is state one
        DuringStateOne(); //Execute During function for state one
        if ( CurrentEvent != NO_EVENT ) //If an event is active
        {
            switch ( CurrentEvent )
            {
            case EVENT_ONE : // If event is event one
                // Execute action function for state one : event one
                NextState = STATE_TWO; // Decide what the next state will be
                break;
            }
            // If next state is different from current state
            if ( NextState != CurrentState )
            {
                // Execute exit function for current state
                // Execute entry function for new state
                CurrentState = NextState; // Modify state variable
            }
        }

    return;
}
/******************************************************************************
Function
   StartStateMachine

Parameters
   None

Returns
   None

Description
   Does any required initialization for this state machine

Notes

Author
   J. Edward Carryer, 2/18/99, 10:38AM
*******************************************************************************/

void StartStateMachine ( void )
{
    CurrentState = ENTRY_STATE;
    // call the (optional) entry function for the ENTRY_STATE
    // any other initialization necessary to re-start the state machine
}
Structured Programming

- Partition the problem
- Define the interactions
- Define the interface

As a term to focus on modules.

Write pseudo-code (comments)

Code by parts - test, test, test

Integrate

Design first, code later.
Kelly Johnson — "Skunk Works"
Questions?
volatile

volatile lightweight = Read ADC()
Questions

Jobs following this class

Linked In Group "Circuit and Famous NewOrgs"

Quiz Today
By similarity

\[
\begin{bmatrix}
0.1h

c
\end{bmatrix}
\rightarrow \text{public interface}
\]

\[
\begin{bmatrix}
c
\end{bmatrix}
\rightarrow \text{private implementation}
\]

\[\text{test harness}\]
null

while True:
    cycle()
    if cycle():
        pass

    read()

    if (not cycle()):
        pass

    set()
Dead band

Locked anti-backlash drive

"Dither"