Modular Programming:
Where the rubber meets the Code

Gabriel Hugh Elkaim
Announcements

1. Uno study and remain intact.

2. Beware of shorts — deadly w5 sec.

3. Learn center —

4. Adrian Lewis — run job.
- Replaced blower on the large cutter.

- Bean Challenge - Tuesday ~ 6 pm.

3 out of 3 successful.
The three uses of static

1. Allocates a variable at startup & retains the value between function calls. = F. \text{ permanence}

2. Scope restriction - cannot be accessed from outside the module

```
static int foo (void);
```

\text{Public/Private}

3. Pseudo-Global Variable - Module level Variable

```
static uint64_t foo&_g ;
```

```
uint64_t read_foo&_g ( void ) {
    return foo&_g ;
}
```
Encapsulation (2.2)

- Interface definition
  "encoded"

- C-impls std functionality

```
RE_Init(RE_End
RE_AddPin(RE_RemovePin
RE_setPin(RE_getPin
```

Module: name.h
name.c
What goes into a header file?

```
/*
 * Header Guard
 *
 * Comment - explains module
 *
 * Prototype for public functions
 *
 * #define for public constants
 *
 * Documentation comments /* *
 *
 * Public data structures {taped, std::vector, enums, std::fields, etc.}
 */
```
What shouldn’t be in a header file?

Not going in .h file:

- No Functions
- No EXECUTABLE code
- No Module Variables
- No STATIC Function Proctotypes

.h — Interface — PUBLIC

.c — Implementation — PRIVATE
Where do you #include the header file?

Everywhere I might need access to its functionality:

```c
#include _MOTORDRIVE_H
#define _MOTORDRIVE_H

/* code */

#include "endif"
Programming Style Issues

Layout within a module

K&R

Swift: "F"

One "True" Indentation Style

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

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

Use of White Space

Use of Comments

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

TEST UNLESS

while (1)
{
    MSES_HandleEvents();
}

Naming Conventions

Gabriel Hugh Elkaim
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
A Real Example

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?
uint8_t DriveInit(void) - set up hardware/software to drive my motors, return SUCCESS or FAILURE

DriveFullStop(void) - stop all motors
DriveStraight(int8_t speed) - drive both motors at the same speed, range from -100 to 100, negative values indicate reverse.

DriveTurn(enum type, int8_t speed) - turns the robot, speed range from -100 to +100, negative indicates CCW rotation, type of turns are: tank, pivot, arc, slow
BAT_VOLTAGE 287

10:1 \div \text{ADC} \rightarrow 

\begin{align*}
\frac{287}{1023} \times 33 &= 9.258 \text{ V} \\
5 \div 8.52 &= 0.5929 \rightarrow 59\% \text{ DHT error}
\end{align*}
STATE EXPLOSION PROBLEM

Hierarchical State Machines

- Harel 1980's Statecharts

Allows me to zoom in and out based on behavior
A Possible Top-Level State Diagram

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

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

Entry ➔ Turning CW

Left Sensor Peaks
Set Slow Speed, Reverse

Turning CCW ➔ L ≈ R
Stop Turning, Post Beacon Acquired

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 (CurrentUser)
    {
    case STATE_ONE:
        // If current state is state one
        DuringStateOne(); // Execute During function for state one
        if (CurrentUser != NO_EVENT) // If an event is active
        {
            switch (CurrentUser)
            {
            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
}
Questions?