CMPE-013/L

Software Engineering

Gabriel Hugh Elkaim
Winter 2014

The three uses of static

1) Allocate & variable
- retain value
- between function calls

2) Data - Global Variable
- (Module Level Variable)
- Only available to functions within the .c file
- Static long int bar

3) Scope restriction - cannot be called outside of file
- Public/Private

Modular C Code

- good programming habits
- introduce information hiding

Encapsulation (1.2)

- int foo
- char bar

FunctionA()
FunctionB()
FunctionC()
Encapsulation (2.2)

```
.h - public interface
.c - implementation (private)
```

```c
int foo
char bar
FunctionA()
FunctionB()
FunctionC()
```

What goes into a header file?

- Function prototypes for public functions
- Definitions that are public
- Variables and comments
- How to use the module
- data types (private)

What shouldn’t be in a header file?

- Not in .h file
- No functions
- No executable code
- No typedefs
- No static variables
- No static function prototypes

WHEREANCE (PUBLIC
IMPLEMENTATION - PRIVATE

Where do you #include the header file?

```c
#include "header.h"
```
Programming Style Issues

Layout within a module
- 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

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

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

Application Programming Interfaces (API) for the ME218c Master modules

Module: Communications to UI on PC

A Real Example

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.

```c
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?
Hierarchical State Machines

- Harel Statecharts

A Possible Top-Level State Diagram

Work out State Diagrams to Implement Finding Tape I
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

**************************************************************************
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 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;
  }
}

Questions?