CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Spring 2016
Bit manipulation

- Bit masking
- Bit flags
- Bit fields
Bit manipulation

Bit packing

- Data is commonly packed into larger unsigned integers on embedded systems
- Generally a tie in to hardware or when space is critical
  - Hardware
  - Storage
  - Binary formats
Bit manipulation

Bit packing

C1CTRL1 – dsPIC33EP256MC502
Bit manipulation

Bit masks

Example

```c
// Abort the current CAN message transmission
ClCTRL1 = ClCTRL1 | 0x1000;
```
Bit manipulation

Bit masks

Example

```c
// Disable CAN message timestamping
ClCTRL1 = ClCTRL1 & 0xFFFF7;
```
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
ClCTRL1 &= ~(1 << 3);
Bit manipulation

Bit masks

- A constant that indicates which bits are relevant for a given variable
- One bits indicate significant bits
- Zero bits indicate ignore bits
Bit manipulation

Bit masks

Example

#define CxCTRL1_MASK_CANCAP (1 << 3)

// Disable CAN message timestamping
C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;
Bit manipulation

Bit masking

- Setting a bit
  - ORing with 1
    \[ \text{CCTRL1} \; |\; = \; \text{CxCTRL1\_MASK\_CANCAP}; \]

- Clearing a bit
  - ANDing with 0
    \[ \text{CCTRL1} \; &\; = \; \sim\text{CxCTRL1\_MASK\_CANCAP}; \]

- Toggling a bit
  - XORing with 1
    \[ \text{CCTRL1} \; ^\; = \; \text{CxCTRL1\_MASK\_CANCAP}; \]
Bit manipulation

Bit masking

- Setting a bit can OR multiple masks together

Example

```c
enum {
    BUTTON_EVENT_1UP = 0x01,
    BUTTON_EVENT_2UP = 0x04
};

{ 
    uint8_t event = BUTTON_EVENT_1UP | BUTTON_EVENT_2UP;
}
```
Bit manipulation

Bit masking

- Getting a bit
  - ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (C1CTRL1 & CxCTRL1_MASK_CANCAP == CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

• Getting a bit
  – ANDing with 1

Example

```c
#define CxCTRL1_MASK_CANCAP (1 << 3)

// If CAN message timestamping is enabled
if (ClCTRL1 & CxCTRL1_MASK_CANCAP) {
  ...
}
```
Bit manipulation

Bit masking

Example

// Retrieve the operating mode of the CAN hardware
int opmode = (C1CTRL1 & 0xE0) >> 5;
Bit Fields are (unsigned) int members of structures that occupy a specified number of adjacent bits from one to sizeof(int). They may be used as an ordinary int variable in arithmetic and logical operations.

- Bit Fields:
  - Are ordinary members of a structure
  - Have a specified bit width
  - Provide bit access to a variable without masking operations
Bit Fields

- Bit Fields:
  - May only be integers (short, long, __, long long)
    - No larger than the base type
  - Unsigned by default, but may be signed
  - Non-portable across architectures/compilers!
    - Just like regular structs
Bit Fields
How to Create a Bit Field

Syntax

```c
struct StructName {
    ((un)signed) int memberName_1: bitWidth;
    ...
    ((un)signed) int memberName_n: bitWidth;
}
```

Example

```c
struct ByteBits {
    unsigned int a: 1;
    long b: 1;
    short c: 2;
    unsigned d: 1;
    long long e: 3;
};
```
Bit Fields
How to Use a Bit Field

Example

typedef struct {
    unsigned int    a: 1;
    long            b: 1;
    short           c: 2;
    unsigned        d: 1;
    long long       e: 3;
} ByteBits;

ByteBits x;

bitfield struct may be declared normally or as a typedef
Bit Fields
How to Use a Bit Field

Example

```c
struct ByteBits {
    unsigned a: 1;
    unsigned b: 1;
    unsigned c: 2;
    unsigned d: 1;
    unsigned e: 3;
} x;

int main(void)
{
    x.a = 1;    // x.a may contain values from 0 to 1
    x.b = 0;    // x.b may contain values from 0 to 1
    x.c = 0b10; // x.c may contain values from 0 to 3
    x.d = 0x0;  // x.d may contain values from 0 to 1
    x.e = 7;    // x.e may contain values from 0 to 7
}
```

Byte in Data Memory (RAM):

```
    7  6  5  4  3  2  1  0
```

```
1 1 1 0 1 0 0 1
```

- e
- d
- c
- b
- a

Maxwell James Dunne – Spring 2016
Bit Fields
Microchip's SFRs

Example

```c
// SFR register declaration
extern volatile unsigned int C1CTRL1 __attribute__((__sfr__));

// SFR bitfield declaration
typedef struct {
    unsigned WIN : 1;
    unsigned : 2;
    unsigned CANCAP : 1;
    unsigned : 1;
    unsigned OPMODE : 3;
    unsigned REQOP : 3;
    unsigned CANCKS : 1;
    unsigned ABAT : 1;
    unsigned CSIDL : 1;
} C1CTRL1BITS;

extern volatile C1CTRL1BITS C1CTRL1bits __attribute__((__sfr__));
```
```c
int main(void)
{
    // Abort the current CAN message transmission
    C1CTRL1 |= 0x1000;

    // Enable CAN message timestamping
    C1CTRL1 &= 0xFFFF;

    // If CAN message timestamping is enabled
    if (C1CTRL1 & 0x0008) {
        ...
    }
}
```
item1 ⇒ data

item2 ⇒ data

Sort

\[ \text{temp} = \text{data}1 \]
\[ \text{data}1 = \text{data} + \text{a}2 \]
\[ \text{data}2 = + \text{emp} \]
\[ x = x \oplus \gamma \]
\[ y = y \oplus x \]
\[ x = x \oplus y \]
\[ x + 1 = -1 \]
Bit Fields

Signed values

Example

```c
typedef struct {
    signed int    a: 3;
    short         b: 2;
    signed short  c: 2;
    long long     d: 3;
} ByteBits;

ByteBits x;
```

X30 = 0011 0000

Bit Fields

Signed values

Example

typedef struct {
    signed int a: 3;
    short b: 2;
    signed short c: 1;
    long long d: 3;
} ByteBits;

ByteBits x;
Bit Fields
Maximum bitness

Example

typedef struct {
    signed int a: 3;
    short b: 2;
    signed short c: 1;
    long long d: 3;
} ByteBits;

ByteBits x;
Bit Fields

Maximum bitness

Example

typedef struct {
    signed short  a: 3;
    short        b: 2;
    signed short c: 1;
    short        d: 3;
} ByteBits;

ByteBits x;
Metaprogramming: The C Preprocessor

- Directives
- Constants/Macros
- Conditionals
- Debugging
Preprocessor

Preprocessor stage

C Source File

.C

C Compiler

Preprocessor

.h

C Header Files

Compiler

Assembly Source File

.s
Preprocessor

Operation of

- Preprocessor operates on all sources files before they're pass to the compiler
- Processes special **preprocessor directives** specified in the code
- Final text of the source file after all preprocessor directives are processed is then compiler
Preprocessor Directives are parts of the code that give special instructions to the compiler. They always begin with a `#` at the beginning of the line, and are used to direct the compiler with a number of specific commands.

- **Groups:**
  - `#defines`: constants, macros
  - Conditionals

- **Usage:**
  - Code organization
  - Debugging
# Preprocessor Directives

<table>
<thead>
<tr>
<th>Directive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>#define</td>
<td>Define a preprocessor macro.</td>
</tr>
<tr>
<td>#elif</td>
<td>Alternatively include some text based on the value of another expression, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#else</td>
<td>Alternatively include some text, if the previous #if, #ifdef, #ifndef, or #elif test failed.</td>
</tr>
<tr>
<td>#endif</td>
<td>Terminate conditional text.</td>
</tr>
<tr>
<td>#error</td>
<td>Produce a compile-time error with a designated message.</td>
</tr>
<tr>
<td>#if</td>
<td>Conditionally include text, based on the value of an expression.</td>
</tr>
<tr>
<td>#ifdef</td>
<td>Conditionally include text, based on whether a macro name is defined.</td>
</tr>
<tr>
<td>#ifndef</td>
<td>Conditionally include text, based on if a name is not a defined macro.</td>
</tr>
<tr>
<td>#include</td>
<td>Insert text from another source file.</td>
</tr>
<tr>
<td>#line</td>
<td>Reset the line number for compiler output.</td>
</tr>
<tr>
<td>#pragma</td>
<td>Allows for extending preprocessor directives beyond what's in the standard.</td>
</tr>
<tr>
<td>#</td>
<td>Null directive</td>
</tr>
<tr>
<td>#warning</td>
<td>Emits a warning described by the rest of the line.</td>
</tr>
</tbody>
</table>
Preprocessor Directives

Text substitution using `#define`

- Defines a text substitution label

**Syntax**

```
#define label text
```

- Each instance of `label` will be replaced with `text` by the preprocessor unless `label` is inside a string
- `text` is optional
- Uses no memory

**Example**

```
#define PI 3.14159
#define MOL 6.02E23
#define MCU "PIC32MX320F128H"
#define PI_2 2 * PI
#define _STDIO_H_
```
Preprocessor Directives

Text substitution using `#define`

- Labels must be valid identifiers

Example

```c
#define 0 1
#define _WRONG
#define __WRONG
#define RIGHT
```
Preprocessor Directives

Text substitution using `#define`

- Text goes until the end of the line
  - Unless newline is escaped with a `\`

**Example**

```
#define true false
#define true \ false
```

- Constants can be nested

**Example**

```
#define OLED_NUM_LINES (OLED_DRIVER_PIXEL_ROWS \ / ASCII_FONT_HEIGHT)
```
# Preprocessor Directives

## Predefined constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>FILE</strong></td>
<td>Full path of current file</td>
</tr>
<tr>
<td><strong>LINE</strong></td>
<td>The current line in the file</td>
</tr>
<tr>
<td><strong>DATE</strong></td>
<td>The current date as a string, like &quot;Jan 27 2014&quot;</td>
</tr>
<tr>
<td><strong>TIME</strong></td>
<td>The current time as a string, like &quot;17:20:50&quot;</td>
</tr>
<tr>
<td><strong>func</strong></td>
<td>The current function as a string, like &quot;main&quot;</td>
</tr>
<tr>
<td>__DEBUG</td>
<td>When debugging is specified in MPLAB X, not part of the standard!</td>
</tr>
</tbody>
</table>

Maxwell James Dunne – Spring 2016
Preprocessor Directives

`#undef`

**Syntax**

`#undef LABEL`

- Deletes a macro definition
- Allows you to change a macro
  - Error when macros are redefined otherwise

**Example**

```c
#define M_PI 3.14
#undef M_PI
#define M_PI 3.141592653589793238462643383279502884197
```
Preprocessor Directives

Argument Macros

• Create a function-like macro

Syntax

```c
#define LABEL(arg1, ..., argn) code
```

- The code must fit on a single line or use \ to split lines
- Text substitution used to insert arguments into code
- Each instance of LABEL() will be expanded into code
- This is not the same as a C function! No stack allocation.

Example

```c
#define MIN(x, y) ((x) < (y) ? (x) : (y))
#define SQUARE(x) ((x) * (x))
#define SWAP(x, y) { (x) ^= (y); (y) ^= (x); (x) ^= (y); }
```
Preprocessor Directives
Argument Macros – Side Effects

Example

```
#define SQUARE(x) x * x

Extreme care must be exercised when using macros. Consider the following use of the above macro:
i = 5;
a = SQUARE(i + 3);
```

\[
i + 3 \times i + 3
\]
#define SQUARE(x) ((x)*(x))

Extreme care must be exercised when using macros. Consider the following use of the above macro:

```c
i = 5;
a = SQUARE(i++);
```

\[
((i+\text{-}1) \times (i++))
\]
Macros with `#define`

Argument Macros – Side Effects

Example

```c
#define ABS(x) (((x) > 0) ? (x) : (-x))
#define NORM1(x, y) (ABS((x)) + ABS((y)))

int x = NORM1(5, 6.6);
```

```c
int x = (((5) > 0)?(5):(-5)) + (((6.6) > 0)?(6.6):(-6.6));
```
Macros with \texttt{define}

Emulating functions

- Functions provide useful features:
  - Encapsulation
  - Evaluate as an expression
  - Return values
Preprocessor Directives

Emulating functions

- For encapsulation

```
#define LABEL(arg1, ..., argn) {
    ...
    ...
}
```

- Code blocks forces all code in the macro to execute in the same context
  - Also allows for temporary variables within the macros
Preprocessor Directives

Emulating functions

Example

```c
#define INIT() TRISA = 5; LATA = 5;

if (beginStartup)
    INIT();

; ;
```
Preprocessor Directives

Emulating functions

Example

```c
#define INIT() {TRISA = 5; LATA = 5;};

if (beginStartup)
    INIT();
else
    ...
```
Preprocessor Directives
Emulating functions

• For encapsulation with expression-ness

```
#define LABEL(arg1, ..., argn) do {
    ...  
} while (0);
```

- Code blocks forces all code in the macro to execute in the same context
  - Also allows for temporary variables within the macros
- `while`-statement allows for semi-colon termination
  - Generates a single statement

Maxwell James Dunne – Spring 2016
Preprocessor Directives

Emulating functions

• To "return" values, just have the statement evaluate to a value

Example

```
define LABEL(arg₁, ..., argₙ) VALUE
```
Preprocessor Directives
Stringification of macro values

Example

```c
#define VERSION 6.3
#define TEXTIFY(x) #x

printf(""s", TEXTIFY(VERSION));
```

6.3
Preprocessor Directives

Stringification of macro values

- You need another layer of indirection

Example

```c
#define TEXTIFY(x) TEXTIFY_HELPER(x)
#define TEXTIFY_HELPER(x) #x
#define MAJOR_VER 1
#define MINOR_VER 3
#define VERSION_STRING TEXTIFY(MAJOR_VER) \ 
    "." \ 
    TEXTIFY(MINOR_VER)

printf("%s", TEXTIFY(VERSION));
```

1.3
LL sort and count

List->data = List2->data

strcmp

NULL

List->data = 0
Count

if (data = NULL)
    Swap

if (strlen(data1) > strlen(data2))
    if (strcmp(data1, data2))
\texttt{NULL, cow, cow, pig}

\begin{align*}
\text{if} & (\text{strcmp}(\text{cur} \rightarrow \text{data}, \text{cur} \rightarrow \text{prev} \rightarrow \text{data})) \\
\text{wordcount}[\text{index}] & = -\text{Count}
\end{align*}
Preprocessor Directives

Token concatenation

- To combine argument with existing token to generate identifiers

Example

```
define DEBUGIFY(x) x ## _DEBUG

printf("%s", DEBUGIFY(asdf));
```
Preprocessor Directives
Conditional compilation

• Control what code actually gets compiled
  – Already seen this with header guards

Example

```c
#define BUTTONS_H
...
#endif
```

Preprocessor Directives
Conditional compilation

• Family of if-statements
  – #if
  – #ifdef
  – #ifndef

• Ended with #endif

• #if is the general case
  – #ifdef/#ifndef only check if a macro has been defined
Preprocessor Directives

Emulating functions

Example

```c
#if INIT
\
#endif
#if 0
#endif
#if defined(_WIN32)
#endif
#if defined(__unix__) && !defined(__APPLE__)
#endif
#if __STDC_VERSION__ > 199409L
#endif
```
Preprocessor Directives
Conditional compilation

- `#ifdef text`
  - Same as `#if defined(...)`

- `#ifndef text`
  - Same as `#if !defined(...)`

- `#elif text`
  - Else-if, follows same rules as `#if`

- `#else`

- `#endif`
Preprocessor Directives

Unit testing

- Conditionally compile in test code

```c
int main(void)
{
    // Initialization code
    #if 0  #ifdef TEST_HARNESS
        // Test code
    #endif
    // Main program
}
```
Preprocessor Directives

Fatal errors

- Output location of failure and stop running

Example

```c
#define FATAL_ERROR()  
  do {  
    printf("FATAL ERROR at %s:%s():%ld\n", __FILE__, __func__, __LINE__);  
    TRISE = 0;  
    LATE = 0xFF;  
  } while (1);
```

Maxwell James Dunne – Spring 2016
Preprocessor Directives
Forcing compilation errors/warnings

- `#warning text`
  - Outputs compilation warning
- `#error text`
  - Outputs compilation error

Example

```c
#if __STDC_VERSION__ < 199901
#error "Must be compiled with C99 or greater"
#endif
```
Switch statements
switch Statement

Syntax

```
switch (expression)
{
    case const-expr₁: statements₁
    :
    case const-exprₙ: statementsₙ
    default: statementsₙ₊₁
}
```

- `expression` is evaluated and tested for a match with the `const-expr` in each `case` clause
- The `statements` in the matching `case` clause is executed
switch Statement

Flow Diagram (default)

START

Const-expr_1 = expression? YES
statement_1
NO

Const-expr_2 = expression? YES
statement_2
NO

Const-expr_n = expression? YES
statement_n
NO

statement_{n+1}

END

Notice that each statement falls through to the next

This is the default behavior of the switch statement
**switch Statement**

Flow Diagram (modified)

Adding a **break** statement to each statement block will eliminate fall through, allowing only one case clause's statement block to be executed.
switch Statement

Simple example

switch Example 1

```c
switch (channel) {
    case 2:  puts("WBBM Chicago");  break;
    case 3:  puts("DVD Player");  break;
    case 4:  puts("WTMJ Milwaukee");  break;
    case 5:  puts("WMAQ Chicago");  break;
    case 6:  puts("WITI Milwaukee");  break;
    case 7:  puts("WLS Chicago");  break;
    case 9:  puts("WGN Chicago");  break;
    case 10: puts("WMVS Milwaukee");  break;
    case 11: puts("WTTW Chicago");  break;
    case 12: puts("WISN Milwaukee");  break;
    default: puts("No Signal Available");
}
```
switch Statement

Styling

switch Example 1

```c
switch (channel) {
    case 2:
        puts("WBBM Chicago");
        break;  // 100%
    case 3:
        puts("DVD Player");
        break;
    case 4:
        puts("WTMJ Milwaukee");
        break;
    ...
}
```
switch Statement
With ASCII

switch Example 2

```c
switch (letter) {
    case 'a':
        puts("Letter 'a' found.");
        break;
    case 'b':
        puts("Letter 'b' found.");
        break;
    case 'c':
        puts("Letter 'c' found.");
        break;
    default:
        puts("Letter not in list.");
}
```
switch Statement

Fall-through

switch Example 3

```c
switch (channel) {
    case 4:
    case 5:
    case 6:
    case 7:
        puts("VHF Station");
        break;
    case 9:
    case 10:
    case 11:
    case 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```

Maxwell James Dunne – Spring 2016
switch Statement

Range syntax

switch Example 3

```c
switch (channel) {
    case 4 ... 7:
        puts("VHF Station");
        break;
    case 9 ... 12:
        puts("VHF Station");
        break;
    default:
        puts("No Signal Available");
}
```
**switch Statement**

Real-world example

```c
bool IsHex(char character)
{
    switch (character) {
    case 'a' ... 'f':
    case 'A' ... 'F':
    case '0' ... '9':
        return true;
    default:
        return false;
    }
}
```
CMPE-013/L

Introduction to “C” Programming

Maxwell James Dunne
Winter 2016
**switch Statement**

Switch versus if/else

- Subset of functionality of if/else
- Works in cases with one value to test
- Switches can be more compact
  - No need to retype variable being tested
  - Range syntax (for GCC-compatible compilers)
  - Easier to read when used properly
- Good when combined with enums
switch Statement

With enums

Example

typedef enum {
    PARAM_EVENT_NONE,
    PARAM_EVENT_REQUEST_LIST_RECEIVED,
    PARAM_EVENT_REQUEST_READ_RECEIVED,
    PARAM_EVENT_SET_RECEIVED
} ParamEvent;

Example

ParamEvent x;
switch (x) {
    case PARAM_EVENT_NONE:
        puts("PARAM_EVENT_NONE found.");
        break;
    case PARAM_EVENT_REQUEST_LIST_RECEIVED:
        puts("PARAM_EVENT_REQUEST_LIST_RECEIVED found.");
        break;
};
switch Statement
With enums

Example

ParamEvent x;
switch (x) {
case PARAM_EVENT_NONE:
    puts("PARAM_EVENT_NONE found.");
    break;

case PARAM_EVENT_REQUEST_LIST_RECEIVED:
    puts("PARAM_EVENT_REQUEST_LIST_RECEIVED found.");
    break;
}

Errors

test.c:141:1: warning: enumeration value 'PARAM_EVENT_REQUEST_READ_RECEIVED' not handled in switch
test.c:141:1: warning: enumeration value 'PARAM_EVENT_SET_RECEIVED' not handled in switch
switch Statement

With enums

Example

ParamEvent x;
switch (x) {
    case PARAM_EVENT_NONE:
        puts("PARAM_EVENT_NONE found.");
        break;
    case PARAM_EVENT_REQUEST_LIST_RECEIVED:
        puts("PARAM_EVENT_REQUEST_LIST_RECEIVED found.");
        break;
    default:
        break; /* FATAL_ERROR() */
}
switch Statement

Local variables

Example

```c
int x; int i;
switch (x) {
    case 3:
        int i;
        for (i = 0; i < 3; ++i) {
            puts("x");
        }
        break;
    ...  
}
```

Errors

**error:** a label can only be part of a statement and a declaration is not a statement
switch Statement

Local variables

Example

```c
int x;
switch (x) {
case 3:
    { 
        int i;
        for (i = 0; i < 3; ++i) {
            puts("x");
        }
    }
    break;
    ...
}
```
State machines
State machines

• Known as Finite State Machines (FSM)
• Mathematical model of computation where system has a single state
• Triggering conditions can change that state
• FSMs are defined completely by both their states and the transitions between them
State machines

State

- The system only exists in one state at a time
- State persists through time
- Certain conditions can change the state to another state
  - These are specific to the current state
State machines

Transitions

- Events trigger transitions between states
- A combination of events can be used
- Transitions are all mutually exclusive
- At any given time there must be a valid transition for a state
  - If no transition is explicitly stated, an implied loopback transition exists
State machines

Benefits

• Provides a formal way to reason about a system
  – Allows for testing before writing any code
• Can be easily visualized
• Are language independent
• States are only dependent on current state and current inputs
State machines

When to use

- Can be used whenever there are a finite set of states for the system
  - Car transmission
  - Stoplight
  - Vending machine
  - Toaster oven
  - Video games
State machines
Use in the SeaSlug

• Transmission protocol
  – Mission management
  – Parameter management

• Operating state
  – Handling errors/system faults

• Calibration
  – Rudder
  – Radio controller
State machines

Diagrams

STATE_1

condition1
action1

STATE_2

condition2
action2
typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
{
    switch (state) {
    case STATE_1:
    
    default:
    
    if (condition1) {
        Action1();
        state = STATE_2;
    }
    break;
    
    case STATE_2:
    
    if (condition2) {
        Action2();
        state = STATE_1;
    }
    
    }
State machines

Example

typedef enum { STATE_1, STATE_2 } SystemState;
static SystemState state;
int main (void) {
    // Initialize system

    // Event loop
    while (1) {
        // State machine
        switch (state) {
            ...
        }
    }
}
State machines

Bounce lab example

• Live coding example!
State machines

Bounce lab example

- State machines rely on conditions to trigger state machines
  - Fits in nicely with event-driven programming
- Use an enum datatype for your states and a switch-statement to check them
  - Provides some compile-time checks
State machines

Bounce lab example

- Live coding example!