CMPE-013/L

Introduction to “C” Programming

Bryant Wenborg Mairs
Spring 2014
Bit manipulation

Bit masking
Bit flags
Bit fields
Bit manipulation

Bit packing

C1CTRL1 – dsPIC33EP256MC502
Bit manipulation

Bit masks

Example

```c
// Abort the current CAN message transmission
C1CTRL1 = C1CTRL1 | 0x1000;
```
Bit manipulation

Bit masks

Example

```c
// Disable CAN message timestamping
C1CTRL1 = C1CTRL1 & 0xFFF7;
```

```c
061111111
```

```c
0
```
Bit manipulation

Bit masks

Example

// Disable CAN message timestamping
C1CTRL1 &= ~(1 << 3);

0th

15

0
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

• Clearing a bit
  – ANDing with 0

• Toggling a bit
  – XORing with 1

```c
C1CTRL1 |= CxCTRL1_MASK_CANCAP;
C1CTRL1 &= ~CxCTRL1_MASK_CANCAP;
C1CTRL1 ^= 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
};

if (newButtonState != oldButtonState)
{
    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 (C1CTRL1 & CxCTRL1_MASK_CANCAP) {
    ...
}
```
Bit manipulation

Bit masking

Example

```
// Retrieve the operating mode of the CAN hardware
int opmode = (C1CTRL1 & 0xE0) >> 5;
```
Bit Fields are \textbf{unsigned} \textbf{int} members of structures that occupy a specified number of adjacent bits from one to \texttt{sizeof(int)}. They may be used as an ordinary \textbf{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

```
struct StructName {
    ((un)signed) int memberName1: bitWidth;
    ...
    ((un)signed) int memberName_n: bitWidth;
}
```

Example

```
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
}
```
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__));
```
Example

```c
int main(void)
{
    // Abort the current CAN message transmission
    C1CTRL1 |= 0x1000;

    // Disable CAN message timestamping
    C1CTRL1 &= 0xFFF7;

    // If CAN message timestamping is enabled
    if (C1CTRL1 & 0x0008) {
        ...
    }
}
```
Bit Fields

Signed values

Example

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

ByteBits x;
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;
Example

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

ByteBits x;
```
Example

```c
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 Compiler

Preprocessor

Compiler

Assembly Source File

C Header Files
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

**Definition**

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

```
#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

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

- Constants can be nested

Example

```c
#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><strong>DEBUG</strong></td>
<td>When debugging is specified in MPLAB X, <strong>not part of the standard!</strong></td>
</tr>
</tbody>
</table>

*CMPE-013/L: “C” Programming*
Preprocessor Directives

#undef

Syntax

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

Argument Macros

• Create a function-like macro

Syntax

```
#define LABEL(arg1, ..., arg_n) 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

```
#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

```c
#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);
```
Preprocessor Directives
Argument Macros – Side Effects

Example

```c
#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++);
```
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);

int x = (((5) > 0)?(5):(-5)) + (((6.6) > 0)?(6.6):(-6.6));
```
Macros with `#define`

Emulating functions

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

Emulating functions

• For encapsulation

Example

```
#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

Gabriel Hugh Elkaim – Winter 2014
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

Example

```c
#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
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));
```

VERSION
Preprocessor Directives

Stringification of macro values

• You need another layer of indirection

Example

#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
Preprocessor Directives

Token concatenation

• To combine argument with existing token to generate identifiers

Example

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

Gabriel Hugh Elkaim – Winter 2014
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
#if 0
#endif
#if defined(_WIN32)
#else
#elif defined(__unix__) && !defined(__APPLE__)
#else
#endif
#endif
```

---

Gabriel Hugh Elkaim – Winter 2014
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

Example

```c
int main(void)
{

    // Initialization code
    #if 0
    // Test code
    #endif
    // Main program
}
```

- Program for debugging
- `#ifdef`
#define FATAL_ERROR()

do {
    printf("FATAL ERROR at %s:%s():%d\n", __FILE__, __func__, __LINE__);
    TRISE = 0;
    LATE = 0xFF;
} while (1);

if (!Init_CAN (ttj 250000)) {
    FATAL_ERROR();
}
Preprocessor Directives

Forcing compilation errors/warnings

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

Example

```
#if __STDC_VERSION__ < 199901
#error "Must be compiled with C99 or greater"
#endif
```

```c
for (int i = 0; i < 99; ++i) {}
```
Switch statements
**switch Statement**

**Syntax**

```c
switch (expression) {
    case const-expr\(_1\): statements\(_1\)
    : 
    case const-expr\(_n\): statements\(_n\)
    default: statements\(_{n+1}\)
}
```

- `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)

```
START

Const-expr_1 = expression?
  YES → statement_1
  NO → NO

Const-expr_2 = expression?
  YES → statement_2
  NO → NO

... (Repeat for each expression until n)

Const-expr_n = expression?
  YES → statement_n
  NO → statement_{n+1}

END
```

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) {
    return 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 (channel) {
    case 2:
        puts("WBBM Chicago");
        break;
    case 3:
        puts("DVD Player");
        break;
    case 4:
        puts("WTMJ Milwaukee");
        break;
    ...
}
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");
}
```
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

### switch Example 2

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