Metaprogramming: The C Preprocessor

- Directives
- Constants/Macros
- Conditionals
- Debugging
# 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

```
#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><code>__FILE__</code></td>
<td>Full path of current file</td>
</tr>
<tr>
<td><code>__LINE__</code></td>
<td>The current line in the file</td>
</tr>
<tr>
<td><code>__DATE__</code></td>
<td>The current date as a string, like &quot;Jan 27 2014&quot;</td>
</tr>
<tr>
<td><code>__TIME__</code></td>
<td>The current time as a string, like &quot;17:20:50&quot;</td>
</tr>
<tr>
<td><code>__func__</code></td>
<td>The current function as a string, like &quot;main&quot;</td>
</tr>
<tr>
<td><code>__DEBUG</code></td>
<td>When debugging is specified in MPLAB X, <strong>not part of the standard</strong>!</td>
</tr>
</tbody>
</table>
Preprocessor Directives

#undef

Syntax

```
#undef LABEL
```

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

Example

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

- Create a function-like macro

```
#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**

```
#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:

\[ i = 5; \]
\[ a = SQUARE(i + 3); \]
\[ a = i + 3 \times i + 3; \]
\[ a = i + 3 \times i + 3; \]
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++);
Macros with \texttt{\#define}

Argument Macros – Side Effects

\textbf{Example}

\begin{verbatim}
#define ABS(x) (((x) > 0) ? (x) : (-x))
#define NORM1(x, y) (ABS((x)) + ABS((y)))

int x = NORM1(5, 6.6);
\end{verbatim}

\begin{verbatim}
int x = (((5 > 0)?5:(-5)) + (((6.6 > 0)?6.6:(-6.6))));
\end{verbatim}
Macros with `#define`

Emulating functions

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

Emulating functions

• For encapsulation

**Example**

```c
#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

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

if (beginStartup)
    INIT();

if (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(arg_1, ..., arg_n) 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

```c
#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

```
#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
```
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
#if defined(_WIN32)
#if defined(__unix__) && !defined(__APPLE__)
#if __STDC_VERSION__ > 199409L
```

Maxwell James Dunne
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

```
int main(void)
{

    // Initialization code

#if 0

    // 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 8s:8s():8d\n", __FILE__, __func__, __LINE__);
        TRISE = 0;
        LATE = 0xFFFF;
    } while (1);
```
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
```
if define STACK_TESTING
main()
endif

CMPE-013/L

Linked Lists

Maxwell James Dunne
Linked List

Theory
typedef struct ListItem {
    struct ListItem *previousItem;
    struct ListItem *nextItem;
    char *data;
} ListItem;
Linked List

`ListItem *LinkedListNew(char *data);`

Diagram:
- `0` pointing to a box labeled `P
  N
  D
  NULL`
- `LL` pointing to `data = data + a`
Linked List

List* ListItemCreateAfter(ListItem *item, char *data);

a → p = i;
i → n = a;
Linked List

List* LinkedListCreateAfter(List* item, char* data);
Linked List

ListItem *LinkedListCreateAfter(ListItem *item, char *data);
Linked List

char *LinkedListRemove(ListItem *item);

\[ i \rightarrow N \rightarrow p = \text{NULL}; \]
Linked List

char *LinkedListRemove(ListItem *item);

```c
i
i -> p -> n = 0;
```
Linked List

char *LinkedListRemove(ListItem *item);
Linked List

char *LinkedListRemove(ListItem *item);

\[
\begin{align*}
  i \rightarrow p \rightarrow N & = i \rightarrow N; \\
  i \rightarrow N \rightarrow p & = i \rightarrow p;
\end{align*}
\]
Linked List

```c
ListItem *LinkedListGetFirst(ListItem *list);

i = i -> p;

while(i -> p != NULL)
```
Linked List

int LinkedListSize(ListItem *list); // int LinkedListPrint(ListItem *list);

i = i → N;
! = NULL
Lab 5 is due Friday (will attempt to get extra staff for section on Friday)
Lab 6 will get extended a day.
strcmp("Cot", "Cog")
malloc(sizeof(ListItem))
CMPE-013/L

Bounce (or Hardware)

Maxwell James Dunne
Bounce

- Digital I/O
- A/D
- Timers
- Debouncing
Static Events;

main()

Timer1 Int + Events - BCE();

3

3
Debouncing
Last event = Down
1) start with one button
   BTN4

2) copy & paste

3) User input