A Little Bit About the C Preprocessor

Preprocessor Directives

- All preprocessor directive begin with a # at the start of a line
- Several types:
  - macros
  - includes
  - conditional compilation

Macros

- A simple macro typically replaces first word with remainder of line
- Typical usage: to define constants
- Convention: ALL CAPS

```
#define  MAX   500
#define  TRUE  1
#define  FALSE 0;        // problem - there should be no semicolon
```

- Here, a macro is defined but not given a value:

```
#define  DEBUG
```

- A macro may have arguments
- Use parentheses; args may be complicated expressions)

```
#define  max((a), (b))     (((a) > (b)) ? (a) : (b) )
```

Include Files

- #include adds a copy of a “header file” to the program
- Use angle brackets <> for files the compiler is expected to know how to find

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

- Use angle double quotes for other files

```
#include  “hal.h”
#include  “include/info.h”
```

Conditional Compilation

- Conditional compilation - Example 1

```
#ifdef  DEBUG
    printf("The value of total is %d\n", total);
#endif
```

- Conditional compilation (recommended at top of every header file):

```
#ifndef  HAL_H
#define HAL_H
int process(int x);
#endif
```

The C Preprocessor

- Running gcc to compile a program actually invokes several programs
  - preprocesser -> compiler -> assembler -> linker
  - cpp | gcc | asm | ld
- The C preprocessor (cpp) does not understand C! It simply/blindly includes text and makes substitutions to text. IT HAS NO UNDERSTANDING OF C SYNTAX.
- You can run the preprocessor explicitly (to see what effect the preprocessor has on your code):
  - cpp prog.c
- To see the output from all the compilation steps:
  - gcc -save-temps prog.c
    // Creates intermediate files prog.i, prog.m, prog.o, and a.out
    // The first two are human-readable with cat, emacs, more, etc.