CSE 303 Concepts and Tools for Software Development

Richard C. Davis
UW CSE – 10/25/2006

Lecture 12 –
The C Preprocessor

Administravia

- HW3 is Due
- HW4 will be posted soon
- Societal Implications Discussion Monday
  - Again, must hand summary
  - Reading will be shorter

Tip of the Day

- Go to arbitrary line in Emacs
  - M-x goto-line *n*
  - Very handy when looking at compile errors

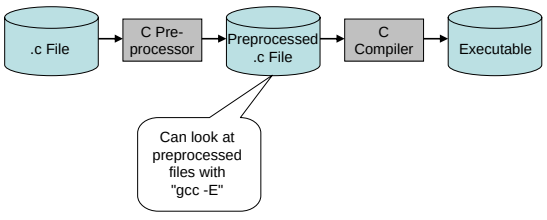
Last Time

- C Casts
- Advanced Data Structures
  - Linked Lists

Today

- The Preprocessor

Source Files are "Preprocessed"



Preprocessor Behavior

- Look for commands
  - `#include`
  - `#define`
  - `#ifdef`
- Replace strings with other strings
- Examples in *list.c, list.h, main.c*

Simple Macros

- `#define foo bar baz`
  - Replace any occurence of `foo` with "`bar baz`"
- Often used to define constants
  - `#define PI 3.14159`
  - Compare with `extern` globals

Parameterized Macros

- `#define PLUS1(x) x+1`
  - Dangerous
- `#define PLUS1b(x) ((x)+1)`
  - Better
- `#define TWICE(x) ((x)+(x))`
  - Problem?

Macros vs. Functions

- `#define TWICE(x) ((x)+(x))`
- `int twice(int x) { return x+x; }`
- A: `v=1; w=TWICE(++v);`
  - "`++`" is executed two times
- B: `v=1; w=twice(++v);`
  - "`++`" is executed once
- Rule of thumb for macro definitions:
  - Don't use an argument more than once

Macros vs. Functions (cont'd)

- Rules of thumb for macro definitions:
  - Don't use an argument more than once
  - Avoid if a function works just as well
    - Performance improvement is hard to get
- When are macros necessary?
  - "Parameterized" type
  - `#define NEW_T(t) ((t*) malloc(sizeof(t)))`

File Inclusion

- `#include <file.h>`
  - Searches in special directories
- `#include "file.h"`
  - Searches in current directory
- Runs preprocessor on file
- Replaces command with resulting text

Benefits of File Inclusion

- Provides Modularity
  - Can split functionality between files
  - Hide "private" functions, types, variables
    - Hard to do well

File Inclusion Problems

- Circular Includes
- Multiple Inclusions
  - Inefficiency
  - Redeclarations of global variables
  - Redefinitions of functions
- Solution: Conditional Compilation

Conditional Compilation

- Very Common Idiom in header files:

```
#ifndef FOO_H
#define FOO_H
…
#endif //FOO_H
```


Conditional Debugging

```
#define DEBUG

#ifdef DEBUG
…
#else
…
#endif //DEBUG
```


Odds and Ends

- Other use for conditional compilation
  - Adapting code to different architectures
  - C is a *mostly* portable language
- "Un"-defining preprocessor symbols

```
#undef SYMBOL
```


Summary

- Preprocessor
  - Header files important for modularity
  - `#ifdef` for managing multiple versions
  - Avoid macros

Reading

- Programming in C
  - Chapter 12: The Preprocessor
  - pp333-342: Working with Larger Programs

Next Time

- File I/O
- Function Pointers
- C Wrap-up