The C Preprocessor

CSE 303 Lecture #12a

Benjamin Ylvisaker
C source files are “preprocessed”

You can look at preprocessed files by running "gcc -E"
The preprocessor responds to commands

- `#include ...`
- `#define ...`
- `#ifdef ...`
- `#ifndef ...`
- `#ifndef ...`
- `...`
Simple macros

• `#define foo bar baz buz`
• Replace any occurrences of `foo` with “bar baz buz”
• Often used to define constants
  – `#define pi 3.14`
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=7; w=TWICE(++v);
• B: v=7; w=TWICE(v++);
• C: v=7; w=twice(v++);
• D: v=7; w=twice(++v);
Macros vs Functions

• #define TWICE(x) ((x)+(x))
• int twice(int x) { return x + x; }
• A: v=7;  w=TWICE(++v);
• B: v=7;  w=TWICE(v++);
• C: v=7;  w=twice(v++);
• D: v=7;  w=twice(++v);
• Good rule of thumb: in a macro definition, don’t use an argument more than once
Macros vs. Functions (cont’d)

• Good rule of thumb: don’t use a macro if a function will work just as well
  – Some people think that macros lead to higher performance. You don’t know enough yet to do that level of optimization.

• Good example of a macro that can’t be implemented as a function:
  – #define NEW_T(t,cnt)
    ((t*) malloc((cnt)*sizeof(t)))
File Inclusion

• `#include <file.h>`
  – Search in special directories
• `#include “file.h”`
  – Search in the “current” directory

• When the preprocessor sees an include command, it finds the included file, runs the preprocessor on the file and replaces the include command with the resulting text
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
...
(rest of foo.h)
...
#endif
```
Conditional Debugging

#define DEBUG

#ifdef DEBUG
    printf(...);
#else
    don’t print
#endif
Odds & Ends

• Conditional compilation also useful for manually adapting code to different architectures and operating systems
  – C is a mostly portable language

• Preprocessor symbols can also be “#undef”ined
Line Numbers

• It is possible to make multi-line macros
• How does the compiler know what line numbers to give in error messages (remember, it only sees the preprocessed file)?
• Answer: there are special "#line" directives
printf and scanf

CSE 303 Lecture #12b

Benjamin Ylvisaker
Input and Output

• `printf` is the standard C function for formatted output
  - `printf("format string", v1, v2, ...);`

• `scanf` is the standard C function for formatted input
  - `scanf("format string", v1, v2, ...);`
Format Strings

- Format strings contain special markers, beginning with %.
  - %d : int
  - %f : float, double
  - %c : char
  - %s : char *, string
  - %e : float, double in scientific form
More Formatting

- Padding (width)
  - %12d
  - %012d

- Precision
  - %12.4d
  - %12.6d

- Left/right justification
  - %12d
  - %-12d

- Similar in %f, %e, ... conversions
scanf

- `scanf` is very similar to `printf`, except people rarely get as creative with the formatting strings.
- The expressions following the formatting string must be `pointers` to the appropriate type.