Announcements

- Exam #1
  - re-grade requests due by 1:45 on Thursday
- Program #1b
  - will return hand-grading portion today
  - re-grade requests due by 1:45 one week from today
- Program #2
  - Due March 15th at 8:00 pm
- Reading
  - Chapter 14 (Today)
  - Chapter 12, 10.2 (Thursday)

Steps in compiling a program

- Converting .c to .o files
  - Pre-processor
  - Compiler (or Translator)
    - Scanner/Parser
    - Type Checker
    - Optimizer
    - Code Generator
  - Assembler
    - Converts assembly code to machine code
- Converting .o's to an executable
  - Linker
    - resolves symbols
      - function use and definitions
      - global variable declarations and use

Pre-processor

- Makes a pass over the program before the compiler
- Handles
  - #include statements
  - Symbolic constants
  - Macros
  - Pre-processor Symbols
  - and other pre-processor directives
    - thgings that start w/ #

Pre-defined Symbols

- __FILE__
  - the name of the source file being compiled
- __LINE__
  - line number of the current line in the file
- __DATE__
  - Date the file was compiled
- __TIME__
  - Time the file was compiled
- __STDC__
  - 1 if the compiler conforms to ANSI C

- Note: these start with two underscores and end with two underscores

#define

- #define name stuff
- Symbolic constants
  - Primary Use
  - #define MAX_SIZE_TABLE 1024
- Macros
  - #define name(parameter-list) stuff
  - Example:
    - #define SUM(a,b) a + b
- Note: all caps is a convention
  - not required by the preprocessor
  - makes it easier for us
- Macros are a single line
  - Can define a "continuation" by ending line with \
  - #define macro1 a very long macro \
  - that runs onto another line

Common Macros

- Square:
  - #define SQUARE(x) ((x) * (x))
  - foo = SQUARE(a + 1)
  - This is replaced to:
    - foo = ((a + 1) * (a + 1))
- Double:
  - #define DOUBLE(x) ((x) + (x))
  - foo = 10 * DOUBLE(3)
  - This is replaced to:
    - foo = 10 * ((3) + (3))
- Maximum:
  - #define MAX(a, b) ((a) > (b) ? (a) : (b))
  - printf("%d is the max", MAX(5,6));
  - This replaces to:
    - printf("%d is the max", ((5)>(6)?(5):(6)));

Macro Substitution

- Items are textually replaced so you must be careful!!
- Consider:
  - #define SQUARE(x) x * x
  - foo = SQUARE(a + 1)
  - This is replaced to:
    - foo = a + 1 * a + 1
  - #define DOUBLE(x) x + x
  - foo = 10 * DOUBLE(3)
  - This is replaced to:
    - foo = 10 * 3 + 3

#define substitution

1. Check macro invocation arguments for macros
   - If present, replace with macro values
2. Replace macros invocations by defined macro
3. Scan for any defined macros used in program
   - If found, repeat steps 1 & 2

String literals are not scanned for macro replacement

Macros vs. Functions

- They look similar, but macros
  - are textually replaced into program
    - can be faster than subroutines for short things
  - macros can not be recursive
  - do not check types of parameters
    - replaced code is type checked during compile

- Example: macro being passed a type as a param

```
#define MALLOC(n, type) \
    ((type *) malloc( (n) * sizeof( type )))
```

Other Issues with Macros

- Side Effects of Macro Parameters
  - Replace each use of parameter
  - Consider:
    - #define DOUBLE(x) ((x) + (x))
    - y = DOUBLE(++j);
    - y = ((++j) + (++j))
- Can be hard to tell macros from functions
  - Solution, use all upper case for macro names
- #undef name
  - un-define the macro named "name"
- Compiler Command Line Definition
  - gcc -DFOO=3
  - defines FOO just as though #define FOO 3 appeared in program

Conditional Compilation

- #if const-ex1
  - statements done iff “const-ex1” is true
- #elif const-ex2
  - statements done iff “const-ex1” is false and “const-ex2” is true
- #else
  - statements done iff “const-ex1” is false and “const-ex2” is false
- #endif

- defined(macro-name)
  - one example of a constant expression
  - 1 if macro-name is defined, 0 if not
- #error
  - stops the compilation at this point
- Useful if code must be different on different systems
- Tends to make code hard to read

File Inclusion

- #include "foo.h"
  - Works as we have seen earlier
- What if foo.h includes another file
  - Pre-processor can handle this
- What if "foo.h" and "bar.h" each #include stuff.h
  - This is fine, until one program includes both foo.h and bar.h
- Solution (inside stuff.h):

```
#ifndef STUFF_H
#define STUFF_H
….
#endif
```