Announcements

- Midterm #1
  - Re-grade requests due by Thursday at 1:45 PM
- Program #1B
  - Grading of Coding - returned in class
  - Re-grade requests due by next Tuesday at 1:45 PM
- Program #2
  - Due in one week
- Reading
  - Chapter 14 (Today)
  - Chapter 12, 10.2 (Tuesday)

Other Helpful Memory Routines

- char *strdup(char *string);
  - not ansi, but supported by most compilers
  - allocate enough memory to hold string (and it's null)
  - copy string into the allocated memory

- #include <alloca.h>
- void *alloca(size_t size);
  - allocate memory on the **stack**
  - memory is freed when the function returns

More Dynamic Memory Examples

```
typedef struct {
    int count;
    char *name;
    float price;
} item;
item *inventory;

inventory = (item *) calloc(sizeof(item), 20);
for (i=0; i < 20; i++) {
     inventory[i].name = strdup(newName);
}
```

Steps in compiling a program

- Converting .c to .o files
  - Pre-processor
  - Compiler
    - 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
    - puts it all into one file
  - Macros
  - Pre-processor Symbols

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

#define

- #define name stuff
- Symbolic constants
  - Primary Use
  - #define MAX_SIZE_TABLE 1024
- Macros
  - #define name(paramter-list) stuff
  - Example:
    - #define SUM(a,b) a + b
- Macros are a single line
  - Can define a "continuation" by ending line with \
  - #define macro1 a very long macro \
  - that runs onto another line

Macro Substitution

- Items are textually replaced
- 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

Solution to Macro Expressions

- Use parenthesis
  - #define SQUARE(x) ((x) * (x))
  - #define DOUBLE(x) ((x) + (x))
- A Common Macro:
  - #define MAX(a, b) ((a) > (b) ? (a) : (b))

#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 literal are not scanned for macro replacement

Macros vs. Functions

- They sort of look alike, but macros
  - are textually replaced into program
    - can be faster than subroutines for short things
  - macros may 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 a macro
- Compiler Command Line Definition
  - gcc -DFOO=3
  - defines FOO as though #define FOO 3 appeared in program

Conditional Compilation

- #if constant-expressions
  - statements
- #elif constat-expressions
  - more statements
- #else
  - yet more statements
- #endif

- defined(macro-name)
  - constant expression, 1 if macro-name is defined
- Useful if code must be different on different systems
- Tends to make code hard to read

Conditional Compilation Example

- #if defined(i386_platform)
  - something specific to i386
- #elsif defined(sparc)
  - some specific to sparc
- #else
  - #error
- #endif

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
```

Include File Search

- #include <someFile.h>
  - Search a list of places for the file (a standard set is defined)
  - On Linux includes /usr/include
  - Generally used for file that are not part of the project being compiled
- #include "anotherFile.h"
  - Search in the current directory, then the list of include directories
- Can use command line flags to change locations
  - gcc -I<some directory>
  - looks in <some directory> for header files