Pre-lab Exercises - Worksheet2: Environments

Even A. Nilsen 14.08.2016

1. Preprocessor Directives

a) The compiler will look for the header file in pre-determined directories. This method is normally used to include standard library header files.

b) The compiler will look for the header file in the same directory as the file containing the directive.

c) Assigns a name to a constant value. Everywhere the compiler encounters the word `LENGTH` it replace it with `100`.

d) Creates a macro which is a fragment of code which has been given a name.
`return CUBE(1 + 2);` would evaluate to
`return ((1 + 2) * (1 + 2) * (1 + 2));`

e) This creates a macro with multiple arguments. It also uses a previously defined macro in its code fragment.
`CALC(1,2,3)` would evaluate to:

```
((1) + ((2) * (2) * (2)) +
(((3) * (3) * (3)) * ((3) * (3) * (3)) * ((3) * (3) * (3))) )
```

I hope I got the parentheses right.

f) This is an example of conditional compiling. Meaning that the code in the `#ifdef` block would only be compiled if the directive `LENGTH` was defined.

g) This is another example of conditional compiling, only this time we are checking if the directive `THEFILE` is **NOT** defined. This is done to make sure that no function or include directive is defined twice in the same program.

2. Shell Commands

a) `ls *Ralph*`

b) `ls | grep -e "^[0-9][0-9]-...$"`

c) `ls *[^AaEeIiOoUuYy]/*^[AaEeIiOoUuYy]`

d) `alias lsc="ls -a *.c"`
The `-a` flag tells ls to include hidden files.

e) `alias gcc="gcc -Wall -ansi -pedantic"`

3. Global Variables and Static Functions

a) Global variables are bad for a number of different reasons.

- **Non-locality** - Global variables can be accessed and modified by any part of the program.
- **Implicit coupling** - The use of global variables often creates tight tight coupling between variables and functions.
- **Concurrency** - If the global variables can be accessed by multiple theads of execution, synchronization is necessary to make the system thread-safe. [Located at][1]

b) The `static` keyword is a storage class specifier, meaning that it indicates where a variable or function is stored. The reason you would never declare a static function in a header file is because it would be inaccessible from outside that file, and therefore be useless.

4. Compile Dependencies

a) `main.c` includes both `database.h` and `util.h`.

b) `main.c`, `database.c`, `database.h`, `util.c` and `interface.c` all include `util.h`.

c) `main.o`, `database.o`, `util.o` and `interface.o` would all be created during compilation.

d) `main.o` and `database.o` would need to be recompiled if `database.h` was modified.

e) `main.o`, `database.o`, `util.o` and `interface.o` would need to be recompiled if `util.h` was modified.

f) `main.o`, `database.o`, `util.o` and `interface.o` would need to be recompiled if `util.c` was modified.

[1] Rishikesh Parkhe, *c2.com*, "Global Variables Are Bad"(blog),posted July 31, 2013, accessed August 15, 2016, http://c2.com/cgi/wiki?GlobalVariablesAreBad