AMSC/CMSC 662
Computer Organization and Programming for Scientific Computing

Fall 2011

Compiling and Interpreting

Dianne P. O'Leary

Compiling vs. Interpreting

- C is a compiled language.
- MATLAB is an interpreted language.

- What does a compiler do?
- What does an interpreter do?
- What difference does it make?

Example of a compiler: gcc

gcc is a set of 4 programs:

- cpp: The C preprocessor produces a .i file (1st pass over the code).
  - Processes any "macros" you have used and brings in appropriate declarations from the libraries you specify in .h files.
  - Strips out comments.
  - Standardizes some special characters ("trigraphs").
  - Standardizes end-of-line symbols.
  - Reference: `http://gcc.gnu.org/onlinedocs/cpp`
- cc1: The compiler produces a .s file (2nd pass).
  - Creates a symbol table containing variables and external functions.
  - Parses each line, recognizing key words, operators, and symbols.
  - Produces assembly language function, complete except for addresses of externals.

- as: The assembler produces a .o file from the .s file.
  - Format: pp 10-12 of B&H Linker slides.
- ld: The linker produces an executable file with default name ”a.out”. from a set of .o files and libraries.

When you run your program, gcc is not involved.

Example of an interpreter: MATLAB

When you type `A = abs(B) + C;` at the command prompt, MATLAB

- Parses the line, recognizing key words (`abs`), operators (+, =), and symbols (`A, B, C`).
- Uses its symbol table to determine addresses of the variables, and allocates new space (on a stack) for variables when necessary.
- Evaluates the expression right to left, calling its `abs` function and saving the results, then calling its addition function, storing the result in `A`

When you type `C = myfun(A);` at the command prompt, MATLAB

- Parses the line, recognizing key words (none), operators (=), and symbols (`A, C, myfun`).
- Uses its symbol table to determine addresses of the variables, and allocates new space (on a stack) for variables when necessary.
- Looks for the necessary .m file (`myfun.m`).
- Parses your `myfun`, quitting if it finds syntax errors. If not, it proceeds through the lines of `myfun`, performs each of the operations, allocates space on a stack when necessary, saves the final results, and adjusts the stack pointer on return.

- If you type `C = myfun(A);` again, it will repeat this entire process, including the parsing.
- But if `myfun.m` calls `mysub.m` 5 times, it will parse `mysub.m` only once.

Aside: A major difference between MATLAB and C

In MATLAB when we say `[A,B] = myfun(C,d)`,

- All variables in the input argument list `(C,d)` are call-by-value, which just means that their value is not changed by `myfun.m`, since `myfun.m` gets a copy of `C, d` to work with.
- In contrast, all variables in the output argument list `[A,B]` are call-by-reference, which means that their values become whatever is returned by `myfun.m`.

In C, all arguments are call-by-value, so pointers must be used if values need to be returned.

Example: `http://www.exforsys.com/tutorials/c-language/call-by-value-and-call-by-reference.html`

Can you see why MATLAB code runs slower than C code?

Notes:

- Actually not much difference if most of the operations are LAPACK/BLAS on matrices of reasonably large size.
- It is possible to compile MATLAB code into `MEX-files` so that they run faster. See `http://www.mathworks.com/products/compiler/`
- It is also possible to call compiled C functions (and `Fortran` functions) from MATLAB. See `http://www.mathworks.com/support/tech-notes/1600/1605.html` and `http://www.mathworks.com/help/techdoc/matlab_external/f38569.html`