3342 assignment 2
1)
a) main calls func1; func1 calls func2; func2 calls func3
main -> a:main, b:main, c:main
func1-> a:main, b:func1, c:main, d:func1
func2-> a:main, b:func1, c:func2, d:func2, e:func2
func3-> a:main, b:func1, c:func2, d:func3, e:func3, f:func3

b)main calls func1; func1 calls func3
main -> a:main, b:main, c:main
func1-> a:main, b:func1, c:main, d:func1
func3-> a:main, b:func1, c:main, d:func3, e:func3, f:func3

c) main calls func2; func2 calls func3; func3 calls func1
main -> a:main, b:main, c:main
func2-> a:main, b:main, c:func2, d:func2, e:func2
func3-> a:main, b:main, c:func2, d:func3, e:func3, f:func3
func1-> a:main, b:func1, c:func1, d:func1, e:func3, f:func3

d) main calls func3; func3 calls func1
main -> a:main, b:main, c:main
func3-> a:main, b:main, c:main, d:func3, e:func3, f:func3
func1-> a:main, b:func1, c:func1, d:func1, e:func3, f:func3

e) main calls func1; func1 calls func3; func3 calls func2
main-> main -> a:main, b:main, c:main
func1-> a:main, b:func1, c:main, d:func1
func3-> a:main, b:func1, c:main, d:func3, e:func3, f:func3
func2-> a:main, b:func1, c:func2, d:func2, e:func2, f:func3

f) main calls func3; func3 calls func2; func2 calls func1
main -> a:main, b:main, c:main
func3-> a:main, b:main, c:main, d:func3, e:func3, f:func3
func2-> a:main, b:main, c:func2, d:func2, e:func2, f:func3
func1-> a:main, b:func1, c:func1, d:func2, e:func2, f:func3

7)
7.1. Max values of integers:
C: 4 byte int -> 32 bit integer -> 2^{31}-1 = 2147483647
C++: 4 byte int -> 32 bit integer -> 2^{31}-1 = 2147483647
Python: Limited only to system memory, switches to long when int becomes too large. Long has unlimited precision.
Ruby: Limited only to system memory, switches to long when int becomes too large. Long has unlimited precision.
C#: 2147483647, same reason as C/C++
Haskell: int – guaranteed to be 2^{29}-1 = 536870911. Integer -> limited only by system memory

7.2. Simplicity (Readability/Writability)
Simplest to read and write is Python and Ruby equally. They are scripting languages specifically designed to be easy to read and write, and fast to develop in.
The most difficult to develop (Aside from Haskell) is C/C++. Although C/C++ are much

faster for larger ints, they take longer to develop and are not as nice to read and write.
Haskell was difficult to write and read but only because I am not used to the functional paradigm.
C# was fairly neutral. I found it easy to read and write in because of its similarity to Java, however, it was not as easy as Ruby and Python.

7.3. Compiled or Interpreted
C/C++: Compiled (using gcc/g++)
Ruby/Python: Interpreted using each languages interpreter
C#: Compiled to byte code using mcs (Mono compiler)
Haskell: Can be compiled into machine code, or run from the interpreter

7.4. Single and multiline comments
C/C++/C#: single line: //[comments]. multiline: /* [comments] */
Python: single line: #[comment], multiline: '''[comment]''' ('Pythonic' way is to use multiple single line comments but...)
Ruby: single line: #[comments], multiline: =begin [comments] =end
Haskell: single line: --[comments], multiline: {- [comments] -}

7.5. Typing
C/C++/C#: static typed, strongly typed
Python/Ruby: Dynamic typed, duck typed (strongly typed at runtime)
Haskell: static typed, strongly typed

7.6: Scope
All static scoped

7.7: Memory
C/C++: Requires explicit memory cleanup of heap dynamic variables
C#/Python/Ruby/Haskell: Garbage collected