Assignment Part 1 – Fortran

# Program Testing Procedure

Run program, observe console output. Every 3rd line should be "Fizz", every 5th line "Buzz", every 15th line "Fizz buzz". The other lines should contain the line number.

Also make sure output formatting is correct – Fortran is finnicky with print formatting.

# Weekly Question

The most striking difference between Fortran and languages I have used before is the column-based syntax. This is due to Fortran’s origins with punch-cards; ancient technology obviously not relevant to any modern language. Most of the languages I use structure syntax with whitespace, braces, semicolons, etc. (Additionally, modern compilers’/interpreters’ greater capabilities compared to Fortran’s likely enable their handling of more flexible syntax.)  
Further exhibiting Fortran’s primitiveness are the PROGRAM, STOP, and END statements, which inform the compiler where the program starts and ends. In any modern language you would take it for granted that the compiler/interpreter could figure this information out for itself.

However, for the simple program we had to code, Fortran is not that different to something like C89, albeit more primitive. IF/ELSEIF/ELSE is the same as in modern procedural languages. There’s a DO loop, which is close enough to the modern “for” loop. The algebraic and boolean notation is largely the same as we use now (besides some differences in operator naming). The one function call to MOD() that I needed to use worked as I expected. Variables are declared with a type before use, the same as C.  
I suspect that Fortran’s difference would become even more apparent if I had to write a larger program, for example using subprograms.

# Reflection

Fortran’s violation of the labelling principle negatively affects the program’s readability, writability and reliability. Firstly, each line must obey a strict column-based format, with various columns indicating different things depending on their contents, none of which is labelled (you must know what column means what). Secondly, lines are labelled with numbers, not meaningful names (again, you must remember which number line does what). Additionally, the use of numerical line labels confuses statements such as DO: for example, it is not immediately clear what numbers mean what in the following statement: DO 10 i=1, 100  
Having to remember these arbitrary orders and labels reduces reliability; the programmer is likely to make a mistake and (knowing Fortran, silently) break the code.

Fortran violates the syntactic consistency principle, reducing readability. For example, the function call MOD(a, b) includes the parameters within the parentheses, but the WRITE statement, which looks like a function call, places the value to write outside the parentheses. Setting a variable is done with =, but equality checking is done with .EQ..

Fortran violates many other language principles, for example:

* Defence in depth: implicit variable declarations, ignored whitespace, existence of GOTO
* Information hiding: everything is global
* Security: existence of GOTO
* Structured program: existence of GOTO
* Zero-one-infinity: array dimensions restriction

But fortunately, they do not affect my program much or at all, due to the simplicity of the task. Note that while, for example, I used IF/ELSEIF/ELSE instead of GOTO, this does not mean Fortran in general adheres to the structure program principle.