Due Date: Wednesday, February 6, 2019, 11:59 PM

Assignment 1: Comparing Languages

COMP 3010 – Organization of Programming Languages
January 2019

[**Note:** *This assignment is adapted from the original version authored by Prof. Michael Scott.*]

Problem description

In this assignment, you will solve a simple problem in each of five different programming languages:

- Ada
- OCaml or Haskell
- C#, Go, Rust, or Swift
- Python or Ruby
- Prolog

Given an integer *n*, your program should output all **permutations** of the *n* numbers {1, 2, ..., *n*}. So if, for example, you specify the value 3, you should see:

```
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

You may print the members of each permutation in ascending or descending order (your choice); but please don't otherwise scramble them.

Finding permutations is a naturally recursive problem (iterative solutions are also possible). You will use **Heap's algorithm**, which is considered to be the most effective solution to this problem. The Wikipedia article linked above gives recursive and iterative versions of the algorithm.

The easiest and most elegant solutions employ a recursive set of *iterators*, which are abstractions used to drive a `for` loop. We will study iterators in Section 6.5.3; you may want to read ahead. In the terminology of that section, you'll find that Python and C# have "true" iterators. Prolog's search mechanism can be used to create the equivalent of iterators, and yields a very elegant solution. Ada and Scheme have no special iterator support; for these you'll have to work with arrays or lists (or find some other solution—e.g., *tasks* in Ada). [For what it's worth, Java and C++ have *iterator objects*, which are sort of half of what you want.]

The use of iterators is optional. **However, you must use the recursive version of the algorithm whenever possible (unless the language you choose doesn't support recursion, of course!).**

If you already knew all the languages, you'd probably find your task easiest in Prolog and hardest in Ada, with the various other languages ranging in between. (Of course, you probably don't know all the languages already, so the unfamiliar ones will be the hardest.) **A hint:** your Ubuntu VM has a folder that contains working versions of all the nontrivial examples in the required textbook. For Ada and C#, you might find it helpful to start with one of the examples: It will already import appropriate libraries and contain examples of the control constructs, I/O calls, etc.

When run, most of your programs should take *n* as a command-line argument, and then write permuations, one per line, to standard output. For Prolog, please arrange for `permutations(k, L)` to produce successive permutations (values for `L`) in response to a semicolon prompt.

Your programs should all run in time proportional to the length of the output. Not all "obvious" programs will do so. In the **Examples** content area on the Blackboard page for this course, you'll find a Python program for the related problem of **combinations** of *n* elements taken *k* at a time. This program produces the correct output, but takes time exponential in *n*, regardless of *k*. In particular, it takes exponential time when *k* = *n*, even though the output is only one line long (try it for, say, *k* = *n* = 20). Take a look at this program for combinations, and try to determine why it takes exponential time!

Division of labor and submission procedure

You may work alone on this project, or in teams of two. If you split up the languages, whoever takes Ada should probably do two; the other person should do three. In any case, ***each team member must write his or her own README file (no sharing of text on this allowed), and turn in the project separately*** (with all five programs, which will be the same as the partner's code). This means, of course, that you'll need to really understand your partner's code.

Be sure your write-up (README file) describes any features of your code that the TA might not immediately notice. In addition, for this assignment, your README file must compare and contrast the programming experience in the different languages you used (all five of them):

- What was easy?
- What was hard?
- Are there noticeable differences in speed?
- What do you like/dislike?
- Did you find iterators to be helpful?

To turn in your code, use the following procedure, which will be the same for all assignments this semester:

1. Your code for all five programs should be in a directory called "`<YourName>_OPL_A1`" (for example, "`TomWilkes_OPL_A1`"). Put your write-up in a `README.txt` or `README.pdf` file in the same directory as your code.
2. In the parent directory of your A1 directory, create a `.tar.gz` file that contains your A1 directory (e.g., "`TomWilkes_OPL_A1.tar.gz`"). If you don't know how to create a `.tar.gz` file, read the `man` pages for the `tar` and `gzip` commands (or ask a friend!).
3. In the page for Assignment 1 on Blackboard, use the Submit button to upload your `.tar.gz` file. When you submit, give the name of your partner (if you had one) in the submission comments field.

Resources

We will be using the following language implementations. All of these are available pre-installed in your Ubuntu VM.

Ada

- Compile your code with `gnatmake` (a wrapper for the GNU Ada translator). It produces native executables.

C#

- Compile your code with `mcs` (the Mono project C# compiler) and run with the `mono` JIT/run-time system.

Go

- Run your code from the command line with `go`.

Haskell

- Run your code under the `ghci` interpreter, or compile with `ghc` (the Glasgow Haskell Compiler) to produce native binaries.

Prolog

- Run your code under the `swipl` interpreter.

Python

- Run your code under the `python3` interpreter.

OCaml

- Run your code under the `ocaml` interpreter, or compile with `ocamlc` to produce native binaries.

Ruby

- Run your code under the `ruby` interpreter.

Rust

- Compile your code with `rustc`.

Swift

- Run your code under the `swift` interpreter, or compile with `swiftc`.

We won't be devoting lecture time to how to use these languages. You'll need to find on-line tutorials or other resources and teach yourself. Here are some decent starting points:

Ada

- www.adacore.com
  Principal web repository for everything Ada, including documentation, compilers, and tools.
- www.adahome.com
  An older repository, now going somewhat stale, but still with some excellent content.
- www.adaic.org/ada-resources/standards/ada12/
  The Ada 2012 Reference Manual. (Note: this is a tough slog. You'll want to find a gentler tutorial—do a Google search.)

C#

- www.hitmill.com/programming/dotNET/csharp.html
  There isn't a canonical single site for C#, but this one is pretty good.
- www.ecma-international.org/publications/standards/Ecma-334.htm
  The international language standard. Covers C# 2.0.
- www.mono-project.com/
  The C# implementation we are using. Open source; supported largely by Xamarin, with a particular emphasis on high quality compilation of C# for the x86.
- msdn.microsoft.com/en-us/vstudio/hh341490.aspx
  Microsoft's main page for C# resources.

Go

- go-lang.org/
  Principal web site for Go. Includes a nifty in-the-web-page interpreter with which to experiment.

Haskell

- haskell.org/
  Principal web site for Haskell, with documentation, tutorials, implementations, tools, etc.

- Hoogle <https://www.haskell.org/hoogle/>
  Type or name based function search through all the library documentation.
- Learn You a Haskell for Great Good! <http://learnyouahaskell.com/chapters/>
  An easy-to-follow online book.

Prolog

- www.swi-prolog.org/
  The Prolog implementation we're using. Includes documentation and downloads (if you want to install it on your personal machine).

Python

- www.python.org/
  Principal web site for Python, with documentation, tutorials, implementations, tools, and news from the user community.

OCaml

- ocaml.org/
  Principal web site for OCaml, with documentation, tutorials, implementations, tools, etc.
- dev.realworldocaml.org/
  A free online version of a textbook on OCaml.

Ruby

- ruby-lang.org/
  Principal web site for Ruby, with documentation, tutorials, implementations, tools, etc.

Rust

- rust-lang.org/
  Principal web site for Rust, with documentation, tutorials, implementations, tools, etc.

Swift

- developer.apple.com/swift/
  Apple's official page for Swift.