SWEN90010: HIGH INTEGRITY SYSTEMS ENGINEERING
DEPARTMENT OF COMPUTING AND INFORMATION SYSTEMS
THE UNIVERSITY OF MELBOURNE

Workshop 5

Introduction

The aim of this workshop is to start getting you familiar with:

- Ada syntax;
- problem solving in Ada and methods for expressing certain computations; and
- compiling and executing Ada programs.

This first Ada workshop aims to give you a look at strings, character handling, specifications, package bodies, and generic packages. The program that you will be working with is simple; the aim is to learn Ada and not to test your programming ability (although this will come later).

It is not necessarily intended that you complete workshops in the allocated time slot. The workshops in the subject are set to a length that you may often have to complete the final tasks outside of contact hours.

Tools

All tools we use in this subject are available on lab machines. The tools we will be using in this workshop are the GNAT Ada compiler and GNAT Programming Studio (GPS).

If you want to install the tools on your local machine, they can be downloaded from `http://libre.adacore.com/tools/spark-gpl-edition/`. Later in the subject, we will be using the SPARK toolkit in combination with these, so you should download and install the following two packages:

- GNAT Ada GPL 2016
- SPARK GPL 2016

If you are running these tools on the lab machines you might need to run them under Cygwin. Open the Cygwin64 Terminal application from the Start Menu. Then run, from the command prompt:

```
$ cd /cygdrive/c/Gnat/bin/
$ ./gps
```

Specification

Overview The *WordCount* program accepts a string from the standard input and delivers a count of words in the input string.

The following is a list of requirements:

1. The program should accept a string characters on the standard input and count the number of words in the input.

2. The input is terminated by a '#' character.
3. Words are separated by white spaces. To simplify the task, assume that white spaces are the space character (i.e. ignore tabs and new line characters).
4. For the purposes of this workshop, a character that is not a whitespace character.
5. The output is a single integer displaying the word count.

Input and output

In the subject notes, we have already encountered the `Put` and `Put_Line` procedures for printing string to standard output (defined in the `Ada.Text_IO` and `Ada.Integer_Text_IO` packages).

To receive a *single character* from standard input, use:

- `Get(Ch : out Character)`: Gets the next character from standard input.

Subprograms and Packages

Recall that Ada does not require the main procedure to be called `main`, but each compilation unit must be stored in a file of the same name. If you are going to call your procedure for this workshop `WordCount` then you must store your procedure in the file named `wordcount.adb` or `WordCount.adb`. Recall that the `.adb` extension is for implementations and the `.ads` extension is for specifications.

Some supporting files

There are two supporting files available from the subject repository:

1. `ProtectedStack.ads` — containing the *specification* of a protected generic stack abstract data type.
2. `ProtectedStack.adb` — containing the *implementation* of the protected generic stack.
3. `WordCount.adb` — a compilation unit that gets your started on the word counting program.

Tasks

1. Your first task is to download the supporting source code for the workshop, available from the LMS.
2. Your next task is to test that you can run the Ada compiler. You can do this using `gnatmake` from the terminal or using GNAT Professional Studio (GPS).

   Run `gnatmake` command on the protected stack package using `gnatmake protectedstack` to see if the compiler executes successfully. `gnatmake` can be found at `C:\GNAT\2014\bin\gnatmake`

   To run on GPS, you must set up a project, and import the files. First, open GPS. This can be found under the main Windows menu. If you cannot find it there (sometimes this just happens!), the executable file itself is at `C:\GNAT\2014\bin\gnat-gps`.

   Instructions for how to create, build, and run the `WordCount` skeleton program can be found in the Appendix.

3. Implement the word counting program specified in the *Specification* section, and run a few tests to convince yourself that the program works on most inputs.

   Tip: it is of course a good idea to implement your program incrementally. Start by writing and running a small program that extends the program given.

   Note the following in the skeleton provided. `ProtectedStack` is importaed by adding the following:

   (a) At the top of word count program, including the following:

   ```
   with Ada.Strings.Unbounded;
   ```

   This provides us with a data type representing a string of an unbound size. Recall that strings are just character arrays, so have a bounded length. The `Ada.Strings.Unbounded` package allows us to simulate variable length strings in Ada. There are some tips on how to use this package below.

   (b) In the declarations of the program:

   ```
   package ASU renames Ada.Strings.Unbounded;
   use ASU;
   package StringStack is new ProtectedStack(100, ASU.Unbounded_String);
   ```

   The final line introduces some new syntax and a new concept that we have not seen before: *generics*. The protected stack module uses a generic type, which means that the type can be instantiated at compile time, rather than design time.

   In the above, we create a new package called `StringStack`, which is the same as `ProtectedStack`, except it accepts strings; specifically, unbounded strings.

   To declare a stack, use:

   ```
   St :  StringStack.Stack;
   ```

4. Every time a word is recognised, *push* the recognised word on the protected stack.

5. Write a loop to *pop* every word from the protected stack and print it using `Put`. The result should be the words from the input text printed in reverse order.

Tips

Strings, and unbounded strings, have some procedures and functions that are relevant to this workshop:

- To append a character, `Ch`, to a string, `Str`, use: `Str := Str & Ch;`
- To check if a character, `Ch`, is a tab, use `Ch = Ada.Characters.Latin_1.HT`. Remember to import the package!
- To convert from a string to an unbound string and vice-versa, use `ASU.To_Unbounded_String(Str : in String)` and `ASU.To_String(UBStr :  in Unbound_String)` (where `ASU` is the renamed package above).

A Creating, building, and running a project in GNAT Professional Studio

Here are the steps to follow to set up and build a project in GPS:

1. When you first start it up, select the option to create a new project. Select the path to be the path that points to the workshop-1 source code.
2. To identify a "main" file for the project, select *Project* → *Edit Project Properties* → *Main files* → *Add*, and select `wordcount.adb`. Click *Ok*.

3. To build the project, select *Build* → *Project*→ *Build all*, which will compile the `wordcount.adb` file, and the related files (protected stack).

4. To run the program, select *Build* → *Run*→ *wordcount*, and just click *Execute*. For the template word count program, it accepts a single character and prints "Hello world!:", followed by the character.