FSA21 - SRTS - SW Exercise

Raquel Guitiérrez, Jordi Valls, Francesc Rocher

June 4, 2021

1 Description

The objective is to implement a program for sorting a list of integers in ascending order in a reliable way by employing either of the SW redundancy principle of *N-Version Programming* with 3 diverse versions and majority voting.

2 Environment Setup

This application has been developed, tested and compiled in Ubuntu-20.04 and GNAT Ada compiler version 9. For testing, the project uses GNATtest, which is contained in the `asis-programs` package and must be installed separately. The whole environment can be installed with:

```
$ apt install gnat gnat-gps asis-programs
```

The package `gnat-gps` is highly recommended for developers.

2.1 Installation

Simply use the 'gprbuild' command to compile the program. No installation instructions required. Run it from the `bin` directory:

```
$ cd <project_rot>
$ gprbuild
$ ./bin/n_sort_voting
```

2.2 Unit Test Environment

By using `gnattest` and `aunit` we adhere to the directory structures used by these tools. To simplify unit test development we prefer to keep the same sub-directory structure of the `src` directory. Thus, the generation of the test harness and unit test stubs is as:

```
$ cd <project_rot>
$ gnattest -P n_sort_voting.gpr
```

Necessary setting are specified in the `Gnattest` package of the `n_sort_voting.gpr` project file.

3 Assumptions

Because the requirements don't impose any constraint regarding efficiency or performance in the sorting algorithms to be implemented, some participants deliberately decided to choose *experimental* algorithms. Of course, these algorithms are never used in real environments, but are interesting pieces of code on how *not* to implement a sorting algorithm.

Also, the requirements are not clear on how to proceed with the error injection of duplicates for lists of length $L = 10$. On one hand, all the memory must be statically allocated, but on the other hand the injection of duplicates must increase the length of the list by one element, so both conditions cannot be fulfilled for such lists.

Thus, some constraint that we impose are:

- You cannot use input integers $\{x_1, x_2, ..\}$ where the distance between maximum and minimum is such that $max(\{x_i\}) - min(\{x_i\}) > StackSize$, where $StackSize$ is the maximum stack size that your system allows per running program. Please find your system limitations with the command `ulimit`. There might be some small discrepancies in this limitation because not only lists are using stack memory, but also other parameters.

- The voter does not takes into account whether the lists are sorted or not, it just compares them. Thus, in case you enter a list of two integers and injects the wrong order error in two of them, the erroneous one will win (this error injected in a list of two elements simply swaps them).

- Similarly, if you inject the omission error in three lists of length two, the erroneous one will win.

- The duplication error can be injected only in lists with length $L \leq 9$. This error injected in a list of length $L = 10$ has no effect because the lists have statically allocated memory and duplication means increasing the length of the list by one element.

4 Final Considerations

Finally, we tried to do at our best with the correctness of the implementation. For this, we provide a formal specification in form of a *contract* in Ada 2012, in the file `Implementation_Interface.ads`. Also, we considered that at least some parts of the implementation required unit testing.

For example, we provide both a formal contract and unit testing for the error injection. If in principle this may seem too much, we must admit that we found a bug in the formal specification thanks to the unit testing, and vice-versa.