Spark Ada and GPS for CSC313/CSM13

Cheat Sheet (updated 03/11/2017)

Compiled by: Olga Petrovska (843356@swan.ac.uk)

Note: most code is taken from Anton Setzer's Clock example

Some Simple Data Types[1]

Integer	-2,147,483,648 to 2,147,483,647
Float	Real numbers with an approximate range of 8.43E-37 to 3.37E+38 and an accuracy of about 6 decimal digits
Boolean	An enumeration type with the values (False, True)
Character	A character in the ANSI (8-bit) character set.

Logical operators:

conjunction	`and`
inclusive disjunction	`or`
inclusive disjunction	`xor`
implication	`(if _ then _ )`

Note: there is no `end if` in case of implication. Not to be confused with the if statement. Implication is used in verification conditions.

Relational operators:

equals	=
doesn't equal	/=
less than	<
less or equal than	<=
greater than	>
greater or equal than	>=

Note: make sure you use >= correctly as => is used to declare dependencies.

Arithmetic operators:

addition	`+`
subtraction	`-`
multiplication	`*`
division	`/`
modulus	`mod`
reminder	`rem`

Function syntax:

```
function FunctionName (InputParameter : TypeOfInputParameter) return
                                                    TypeOfValueReturned is
OptionalVariableInstantiatedWithinTheFunction : TypeOfTheOptionalVariable;
begin
...(body of the function)...
end FunctionName;
```

Procedure syntax:

```
function ProcedureName (InputParameter : in TypeOfInput;
                                        OutputParameter : out TypeOfOutput) is
OptionalVariableInstantiatedWithinTheFunction : TypeOfTheOptionalVariable;
begin
...(body of the procedure)...
end FunctionName;
```

Assigning a value:

```
X := 22;
Y := X + 1;
```

If-statement:

```
if ….
then …. ;
else …. ;
end if;
```

Loop syntax:

```
loop
pragma Loop_Invariant((BooleanCondition1 and BooleanCondition2) or
                                                    BooleanCondition3);
...(body of the loop)…
exit when BooleanCondition4;
end loop;
```

Adding a new type (in .ads):

```
type Status_Cooling_System_Type is (Activated, Not_Activated);
type Hour12 is new Integer range 0..11;
type Ampm is (Am,Pm);
```

Adding a new record type (in .ads):

```
type Hours_Ampm is
     record
        Hours    : Hour12;
        Mode     : Ampm;
     end record;
```

Converting a value of one type (e.g., T : Hour) into another type (e.g. Integer):

```
Integer(T)
```

or in case `T` is an input parameter (of an arbitrary type) of a function that returns an `Hour` and you need to convert it into an `Integer`:

```
Integer(function(T))
```

Function with two input parameters:

if the type of input parameters is the same:

```
function To12 (X,Y : Integer) return Integer is …
```

if the types differ:

```
function To12 (X: Integer, Y: Hour) return Integer is …
```

Procedure with two input and two output parameters:

```
procedure To12Proc (X,Y: in Integer; U,Z: out Integer) is …
```

Procedure with input / output parameters:

If you have a procedure which updates the value of your input parameter, you can use the following syntax, i.e. `X` is both an input and an output parameter:

```
procedure Proc (X: in out Integer) is …
```

Adding dependencies (in .ads file):

```
procedure To12Proc (T : in Hour; U : out Hour12) with
Depends => (U => T);
```

For multiple dependencies, the syntax is as follows:

```
procedure Proc (A, B, C : in Integer; D, E : out Integer) with
Depends => (D => (A, B),
            E => (A, C));
```

Adding post conditions (in .ads file):

```
function To12 (T : Hour) return Hour12 with
Post => (T = Hour(To12'Result) or T = Hour(To12'Result + 12));
```

* Functions return values. Thus, `To12'Result` is the value that the function `To12` returns. This post condition says that after the function has been called the input parameter `T` is either equal to the value that `To12` returns or that value + 12. Refer to the function definition in the Clock example.

```
procedure To12Proc(T : in Hour; U : out Hour12) with
Depends => (U => T),
Post => (T = Hour(U) or T = Hour(U) + 12);
```

* Procedures do not return values, they assign values to variable(s), for example to `U` in this case. The above post condition makes sure that the value of input parameter `T` matches the value of the output parameter after the procedure has been executed.

Adding a Loop_Invariant:

Loop Invariant is checking whether conditions defined in it are met when you enter the loop. After the last iteration the conditions should be False. You can use the usual logical operators and, or, xor and add as many conditions as you want.

```
pragma Loop_Invariant((BooleanCondition1 and BooleanCondition2) or
                                                    BooleanCondition3);
```

Loop invariant does not need to be the last statement in a loop.

GPS Commands

This information is taken from the Basic Usage of SPARK Ada page[2]:

For **checking the syntax** (only Ada not SPARK Ada)

Build -> Check Syntax

For **compiling all ada files** in a directory use

Build -> Project -> Build All

This will generate files which can then be linked to other projects. It will not create executable files. The main purpose for use this command in the labs is to check that the files are correct ada files.

For **compiling an executable file** (e.g. with input/output) click on the executable .adb file, make sure that it is shown and the active tab, and then use

Build -> Project -> Build < current file >

You need to have an .adb file open which has no package and only one procedure with the same name as the file and with no arguments.

For **running an executable file** use

Run -> Custom

Then choose command `xterm -hold -e ./filename`, where filename.adb is the file you were running the build current file command on. In most cases filename is main. Please cut and paste the command in as it is (note blanks in between the components, no other command in front, the part ./). SPARK Ada remembers your command, so you only need possibly to change filename when running it again.

To **run data flow and examination flow analysis**:

SPARK 2014 -> Examine All

To **run all 3 levels of SPARK Ada including verification conditions**:

SPARK 2014 -> Prove all

In the pop-up dialog, before clicking *Execute*, select "Progressively Split" (3rd option) in the drop-down menu on the right.

FAQs

What is the difference between a function and a procedure?

Function returns a value, *e.g. it returns a value that is equal to either T or T-12:*

```
function To12 (T : Integer) return Integer is
begin
   if T >= 12 then return T - 12;
   else return T;
   end if;
end To12;
```

Procedure does not return anything. It assigns a value to a variable or variables,
e.g. a value that is equal to either T or T-12 is assigned to the variable U.

```
procedure To12Proc (T : in Integer; U : out Integer) is
begin
   if T >= 12 then U := T - 12;
   else U := T;
   end if;
end To12Proc;
```

What are .ads and .adb files?

.adb and .ads files always come in pairs. In plain words, in an .ads file you specify what your functions, procedures and types are and in an .adb file you define how your functions and procedures work.

What does main.adb do and what should I know when updating main.adb?

In your main.adb you specify how the user interface works.
When updating main.adb, make sure you update the package that it works with, add correct type declarations and call functions and procedures that you have declared in your program.
When working with types other than Integers, make sure you convert types where needed.

Note: `AS_Get` is used to assign a value to a variable, e.g. `T`, when users input a number. This value is saved as an `Integer`. If your function/procedure needs this value as an input but the type of the input that is expected is different from `Integer`, you need to convert it accordingly. For example, if you function expects a number `T` of the type `Hour` (i.e. it is declared as `function To12 (T : Hour) return Hour12;`), then you call it in the following way: `To12(Hour(T))`.
`AS_Put_Line` is used to print values in your program dialog. It expects an `Integer` as its input. If the value that your function returns is not an Integer, then you need to convert it: `AS_Put_Line(Integer(To12(Hour(T))));`

Why do my files not appear in the GPS menu?

All file names should use lower case letters and the path to your project should not contain any spaces (folder names should not contain any spaces).

How do I run my program?

In order to run your program, you need to have main.adb and main.ads files added to your project. You can copy main.ads file from an example project, it does not need to be modified.

```
pragma SPARK_Mode (On);

procedure Main;
```

Main.adb should be updated to work with your program. You also need to add IO library created by Anton to your folder. You can download the library using this short URL: http://bit.do/asLibraryIO (full url in references[3])

Before you run your program, make sure you've compiled all files. Click *Build Project* > *Run* and type `xterm -hold -e ./main` (please note the blanks after xterm, -hold and -e), then click *Execute*. This should open a new dialog with your program.

Why do I get the 'No such file or directory' message when I execute my program?

Make sure you have opened main.adb file and compiled it separately (*Build -> Project -> Build <current file>*) before you run the program.

References:

1. http://www.modula2.org/sb/env/index117.htm
2. Basic Usage of SPARK Ada
http://www.cs.swan.ac.uk/~csetzer/lectures/critsys/current/SPARK_Ada/basicUsageOfSparkAda.html
3. IO Library
http://www.cs.swan.ac.uk/~csetzer/lectures/critsys/current/SPARK_Ada/examplesFromLecture/src/examplesAdaSparkAdaCriticalSystems/asLibraryIO.zip