Programming Languages
CSCI-GA.2110.001 Fall 2016

Ada Programming Assignment
Due Friday, October 7

You must work on this assignment by yourself. If you need help, you may ask me or Deepti.

Your assignment is to write a concurrent Adaptive Quadrature program in Ada. As you may know, Adaptive Quadrature is a method for approximating the area under a curve (i.e. the definite integral of the corresponding function). You do not need to be familiar with Adaptive Quadrature or integration in order to complete this assignment. Here are the steps that you should follow.

- **Step 0. The Algorithm**

  Read the adaptive quadrature code written in Python using Simpson's method found on the Wikipedia page here. Essentially, you will be translating this Python code into Ada and adding concurrency. Note that because you cannot pass a procedure as a parameter (such as the parameter `f` in the Python code) in Ada, you will supply a function `F` as the generic parameter in the instantiation of a generic package.

- **Step 1. Implementing a generic Adaptive Quadrature package specification**

  Write the package specification for a generic package, `AdaptiveQuad`, for performing adaptive quadrature, parameterized by the function `F` whose definite integral is being approximated. The package should export a function `AQuad(A,B,Eps)` that approximates the definite integral of `F` in the interval from `A` to `B`, using `Eps` as the epsilon value. `A`, `B`, and `Eps` are all of type `Float`, and `F` should take a `Float` as a parameter and return a `Float` as a result.

  The package specification for `AdaptiveQuad` should be of the form:

  ```
  generic
     -- function F declared here as the generic parameter.
  package AdaptiveQuad is
     -- function AQuad declared here.
  end AdaptiveQuad;
  ```

  This code should be placed in a file named `AdaptiveQuad.ads` .

- **Step 2. Implementing the `AdaptiveQuad` package body**

  To implement the body of the `AdaptiveQuad` package, you can define procedures in the package that correspond to the procedures found in the Python code on the above Wikipedia page. The `AQuad` function, above, corresponds to the `adaptive_simpsons_rule` function found in the Python code (remember that `F` is not a parameter to the function, instead the package is instantiated with `F`).

  Notice that in the Python function `recursive_asr`, there are two recursive calls. In your corresponding Ada code, those two recursive calls must execute concurrently. Thus, you must use Ada task(s) within your corresponding recursive procedure (Hint: there should be no entry calls in this package, since no communication between tasks is required).

  Thus, your package body might look something like:

```
package body AdaptiveQuad is

   function SimpsonsRule(...) ...

   function RecAQuad(...) ...

   function AQuad(...) ...

end AdaptiveQuad;
```

This code should be placed in a file named `AdaptiveQuad.adb` .

- **Step 3. Implementing the main procedure**

  In a separate file, `AQMain.adb`, you should define the procedure `AQMain`, which will be called when the program starts. Within `AQMain`, you should define the following:

  – A `Float` constant, `Epsilon`, defined to be 0.000001 .

  – A procedure `MyF` that takes a `Float` x and returns the value of $sin(x^2)$.

  – A new package resulting from instantiating your generic package `AdaptiveQuad` with the parameter `MyF`.

  – A task `ReadPairs` for reading in the `A` and `B` values representing the lower and upper bounds of the region of the curve (see the `a` and `b` values in the Python code). Your program should perform adaptive quadrature on five different intervals, so within a loop that iterates 5 times, the following should occur:

    ∗ Read in an `A` value and a `B` value (using the `Get` procedure)

    ∗ Provide the `A` and `B` values to the `ComputeArea` task (below), so it can perform adapture quadrature.

    It is important that `ReadPairs` can proceed to read the next pair of `A` and `B` values concurrently with the `ComputeArea` task performing the adaptive quadrature. That is, `ReadPairs` should not wait for the adapative quadrature to be completed before reading the next `A` and `B` values.

  – A task `ComputeArea` for performing adaptive quadrature. In a loop, it should peform the following actions:

    ∗ wait for the `A` and `B` values from `ReadPairs`.

    ∗ call the `AQuad` procedure (above), passing `A`, `B`, and `Epsilon`.

    ∗ once `AQuad` returns, provide `A`, `B`, and `AQuad`'s result to the `PrintResult` task (below).

    As mentioned above, it is important that `ReadPairs` not have to wait while `AQuad` is being called. Additionally, `ComputeArea` should not have to wait for `PrintResult` to finish printing the result before getting the next `A` and `B` values from `ReadPairs`.

  – A task `PrintResult` for printing the results of adapative quadrature. In a loop, it should perform the following actions:

    ∗ wait for the `A`, `B`, and adaptive quadrature result values from `ComputeArea`.

    ∗ print a message such as:

    ```
    The area under sin(x^2) for x = ... to ... is  ...
    ```

    where the "..." are replaced by the values of `A`, `B`, and the result.

    Remember that printing should happen concurrently with the other tasks running (see above).

- **Step 4. Submit the files**

  To submit your program, upload your three files to the course web site.

Helpful Hints

1. Be sure to review the sample Ada programs provided in class, especially the program illustrating the use of a generic package.

2. To compile your program, in a shell type

   ```
   gnatmake AQMain
   ```

   To run your program, type

   ```
   ./AQMain
   ```

   and then enter two numbers at a time, representing `A` and `B`.

   However, rather than typing five pairs of numbers every time you run your program, you can put 10 numbers (on one line) in a file, e.g. input.txt, and run the program by typing

   ```
   ./AQMain < input.txt
   ```

   I have provided such an input.txt for you to use.

3. In order to print `Float`s using the Put function, put these two lines near the top of your `AQMain.adb` file:

   ```
   with Ada.Float_Text_IO;
   use Ada.Float_Text_IO;
   ```

4. In order to use the Ada's `Sin` function, you'll need to use the `Generic_Elementary_Functions` generic package. To do so, put this line near the top of your `AQMain.adb` file:

   ```
   with Ada.Numerics.Generic_Elementary_Functions;
   ```

   and put these two lines within your `AQMain` procedure:

   ```
   package FloatFunctions is new Ada.Numerics.Generic_Elementary_Functions(Float);
   use FloatFunctions;
   ```