Jan 22nd, 2016

Assignment 1:3

This week, we are practicing writing proper definitions while keeping our target audience in mind. Being able to write proper definitions is an important part of the technical writing process. When we communicate among our peers, it can be very easy to use jargon or terminology that is of common knowledge to that peer group, while sounding foreign to another peer group. This assignment will allow us to practice and refine our efforts in writing proper definitions for a given audience. The definitions must introduce a new term to a group of non-technical readers.

Here I have provided examples of a parenthetical, sentence, and expanded definitions of the term ‘recursion’, which is a common term used in the field of Computer Science. The intended reading audience is students entering their first year of study in the subject of Computer Science. Students starting their first course in Computer Science are learning how to write their first programs. The term ‘recursion’ will be new to these students.

**Parenthetical Definition of ‘Recursion’**

We will solve this problem using a program that relies on recursion (functions that, when used, call back on themselves, running in an apparent loop).

**Sentence Definition of ‘Recursion’**

Recursion is a type of function behaviour that relies on a specific set of rules which is used repeatedly on an object or variable. This will reduce the object to its simplest form, which is often referred to as the base case.

**Expanded Definition of ‘Recursion’**

*What is recursion?*

The concept of recursion has been around for many years, dating back as early as the late 19th century1. Mathematicians would often use the concept of recursion to solve complex mathematical operations. As an introduction to programming concepts, recursion is much simpler. Recursion occurs when we use functions that, when executed, run another function, which is itself. These types of functions are called recursive functions, and when they are used, it results in recursion.

*How does it work?*

Recursion works on data of arbitrary size. The recursion should function in the same manner whether the data size is small or large. This type of data is usually represented in the form of a list. When we work with large amounts of data, the problems we are presented with often become large and complex. Recursion allows us to find a solution to the problem efficiently, avoiding the need to write complicated programs to solve a complicated set of data. For example, if we have a large list of numbers, and we would like to know how many of those numbers are divisible by 3. We do not know how large this list is, so there is no way to look at each object individually in the list and write a program to look at that object. Instead, we can write a recursive function that will look at the list as a whole, and go through each object in the list until it reaches the end. The steps for this problem are listed below:

* Take the list as a whole, and look at the first item in the list
* Is the first number of the list divisible by 3?
  + If yes, do something (that we specified), then call the function itself using the rest of the list as the item that the function is using
  + If no, do something (that we specified), then call the function itself using the rest of the list as the item that the function is using
* When the function gets to the last number, the call to itself will receive an empty list as the item it is using (since there are no numbers left in the list). The function calls end and a result is produced.

The large problem is reduced down into many small problems as the function continues to call on itself2. Once the problem is in its simplest form (base case), which for lists of data would be an empty list, then a separate execution runs, and the program is finished.

*Let’s see it in action!*

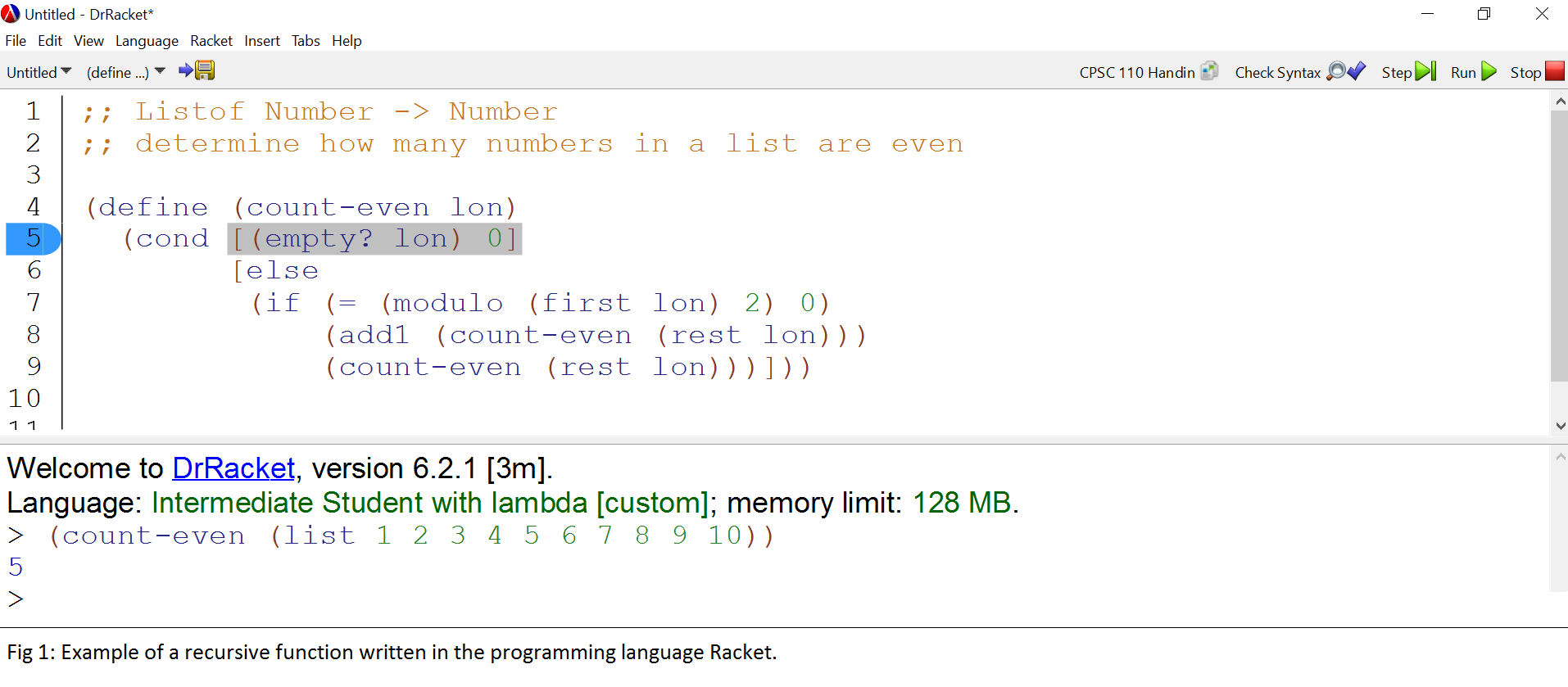


Figure 1 represents an example of recursion. As you can see, the function acts on the first part of the list, does something, then sends the rest of the list back to the function itself (this is the recursion). The ‘something’ in this case is that every time the function sees a number that is even (when divided by two, the remainder is zero).

*When is recursion required?*

Generally, recursion is required when operating on data of arbitrary size3 (in the previous example, a list of numbers that could be of any size). That is why the concept of recursion is so useful – it can operate on data that is small or large. Recursion will always work through a list of data, reducing the list until there is nothing left (or the data is in its simplest form), and a result will be produced. However, caution must be taken when designing a program that uses recursion. If there is no separate path for the function to take when it reaches the simplest form of the input, it will continue to run forever in what is called an infinite loop, or it will result in an error when there is no data left. The recursion is still happening, but there is no way for the program to determine when it has done through all the data.

Works Cited:

1. Daylight, Edgar. “The Advent of Recursion in Programming, 1950’s – 1960’s” Institute of Logic, Language and Computation, University of Amsterdam, The Netherlands. April 2010. 23 Jan 2016.
2. Kiczales, Gregor. *Systematic Program Design.* University of British Columbia. Sept 2015. Web. 23 Jan 2016.
3. Felleisen, Findler, Flatt and Krishnamurthi. “How to Design Programs”. Cambridge, Massachusetts and London, England: The MIT Press. Web. 23 Jan 2016.