CnC-Scala Partition String example

Please refer to the Find-Primes example before starting with this example.

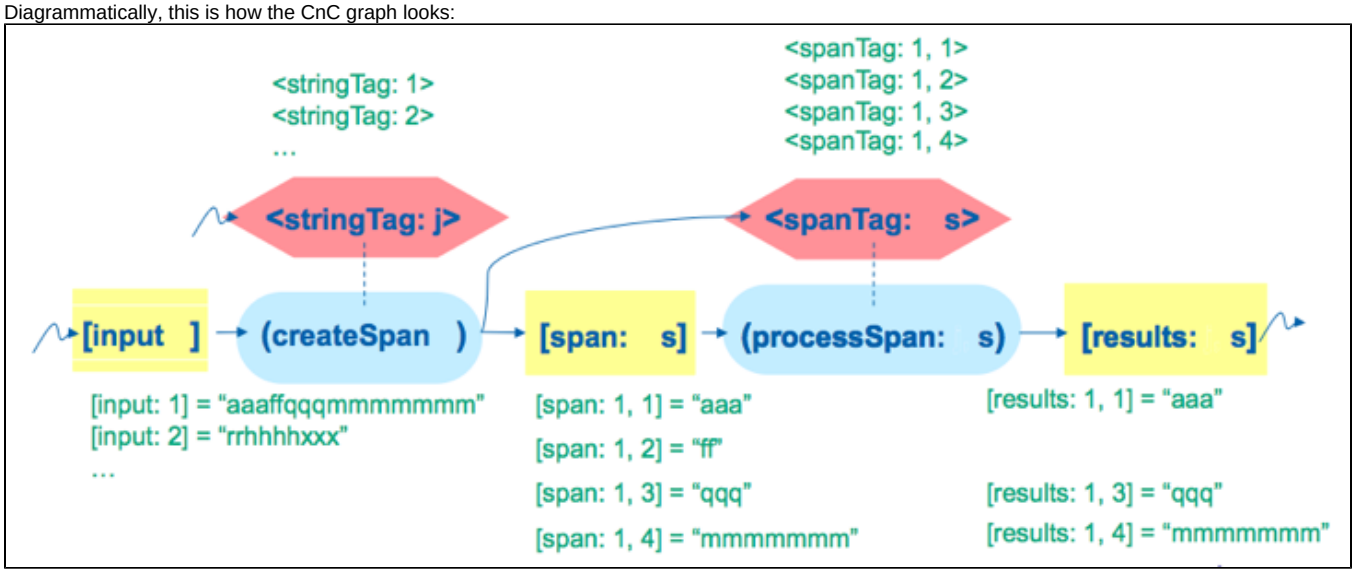
First, define the CnC graph for the partition string example:

PartitionStringCncDefinition.scala

```
object PartitionStringCncDefinition {
  def main(args: Array[String]): Unit = {
    ApplicationName("PartitionString")
    TargetPackage("partitionstring")
    TargetDirectory("partitionstring")

    Comment("Item Collections")
    ItemCollection[String, String]("input")
    ItemCollection[String, String]("span")
    ItemCollection[String, String]("results")

    Comment("Tag Collections")
    TagCollection[String]("stringTag")
    TagCollection[String]("spanTags")

    Comment("Step Prescriptions")
    Presription("singletonTag", List("createSpan"))
    Presription("spanTags", List("processSpan"))

    Comment("Step Dependences")
    StepDependence("createSpan", List[String]("input"), List("spanTags", "span"))
    StepDependence("processSpan", List[String]("span"), List("results"))

    Comment("Environment Collections")
    EnvironmentDependence(
      List("input", "stringTag"),
      List("results")
    )
  }
}
```

This graph is a little more interesting than the Find-Primes example since it has two Steps: `createSpan` and `processSpan`. `createSpan` produces Items for the `span` Item Collection. This in turn triggers the prescription of the `processSpan` step which writes results to the `results` Item Collection. The environment reads off these values from the `results` Item Collection.

Diagrammatically, this is how the CnC graph looks:



(Image from Kathleen Knobe's CnC Tutorial)

Next, run the CnC-Scala translator:

```
cnc_scala_translate PartitionStringCncDefinition
```

Next, we need to provide the implementations for the Steps using the generated templates (as explained in the Find-Primes example).

Note for experienced CnC users: the implementation of the `compute()` method reads the input dependences on values from the Item Collections without explicitly stating its dependences in a `ready()` or `awaitsList()` method as required in some of the other CnC implementations.

UserPartitionStringSteps.scala

```
package partitionstring

import edu.rice.cnc.api._
import java.lang.String
import util.continuations.cps

class UserCreateSpanStep extends CreateSpanStep {
  def compute(
              tag: String,
              inInput: InputCollection[String, String],
              outSpanTags: TagCollection[String],
              outSpan: OutputCollection[String, String]
              ): Unit@cps[Any] = {

    // Get input string
    val in: String = inInput.get(tag)

    if (in.length() != 0) {
      var ch: Char = in.charAt(0)
      var len: Int = 0
      var i: Int = 0
      var j: Int = 0
      while (i < in.length()) {
        if (in.charAt(i) == ch) {
          i += 1
          len += 1
        } else {
          val jStr = String.valueOf(j)
          outSpan.put(jStr, in.substring(j, j + len))
          outSpanTags.put(jStr)
          ch = in.charAt(i)
          len = 0
          j = i
        }
      }
      val jStr: String = String.valueOf(j)
      outSpan.put(jStr, in.substring(j, j + len))
      outSpanTags.put(jStr)
    }
  }
}

class UserProcessSpanStep extends ProcessSpanStep {
  def compute(
              tag: String,
              inSpan: InputCollection[String, String],
              outResults: OutputCollection[String, String]
              ): Unit@cps[Any] = {

    val toProcess = inSpan.get(tag)

    if ((toProcess.length() % 2) != 0) {
      outResults.put(tag, toProcess)
    }
  }
}
```

Running this program with an input of `aaaaaabbbbbccddddddsssrrrrrw` should produce the following output:

```
Contents of Collection'results'
  19=sss
  22=rrrrr
  27=w
  6=bbbbb
HabaneroRuntime:
  num workers=4
  executor service=jsr166y.ForkJoinPool@647109c4[Terminating, parallelism = 4, size = 4, active = 4, running =
4, steals = 6, tasks = 0, submissions = 0]
  async instances=9
  activity instances=9
```