Bref3 format specification

General information and overview:

1. Bref3 (pronounced “bee-ref three”) stands for “binary reference version 3”. Bref3 format is a binary format for storing phased, non-missing genotypes for a list of samples.

2. This document provides pseudocode for reading for reading Bref3 format. The pseudocode defines the structure of a Bref3 file.

3. Integer values are read using the readByte() , readUnsignedShort(), and readInt() methods described in the documentation for the Java DataInput interface in the java.io package.
    a. The readByte() method reads a signed one-byte integer in the range: [-128, 127].
    b. The readUnsignedByte() method reads an unsigned one-byte integer in the range: [0, 255].
    c. The readUnsignedShor()) method reads an unsigned two-byte integer in the range: [0, 65535].
    d. The readInt() method stores a signed four-byte integer in the range: $[-2^{31}, 2^{31} - 1]$.

4. String values are read stored in the Modified UTF-8 format, and read using the readUTF() method described in the documentation for the Java DataInput interface in the java.io package.

5. The Bref3 format stores the genotype data in data blocks. Each data block contains the marker and genotype information for a set of consecutive markers. Each marker is either “allele-coded” or “sequence-coded”.
    a. If a marker is allele-coded, the indices of haplotypes carrying non-major alleles are stored. This is an efficient storage format for markers whose non-major alleles have low frequency
    b. For markers that are sequence-coded, the list of distinct allele sequences present in the sequence-coded markers in the data block is stored, and the index of the distinct allele sequence carried by each haplotype is stored.
    c. The number of distinct allele sequences for the sequence-coded markers in a data block must be < 65535.

6. In the following pseudocode:
    - **in** implements java.io.DataInput and reads from a bref3 file.
    - **nHaps** denotes the number of haplotypes.
    - **nRecs** denotes the number of records in a data block. Each marker corresponds to one VCF record.
    - **nSeqs** denotes the number of distinct allele sequences present in the sequence-coded records in a data block.
    - **nAlleles** denotes the number of alleles (including the REF allele) for a marker.

7. The following code for the readRecords() method returns the list of records in a bref3 file.

Pseudocode for reading Bref3 format:

```
in = <java.io.DataInput reading from a bref3 file>
snvPerms = <list of lexicographically-sorted permutations of [“A”,“C”,“G”,“T”]>

def readRecords(in):
  // Read “magic number” and confirm the file format and version
  if in.readInt() != 2055763188:
    exit                                // file is not a bref3 file
  program = readString(in)              // program used to create bref3 file
  samples = readStringArray(in)         // sample IDs
  nHaps = 2*samples.length              // number of haplotypes

  recList = []                          // list of VCF records read
  nRecs = in.readInt()                  // number of records in next data block
  while (nRecs != 0):
    readDataBlock(in, samples, recList, nRecs)
    nRecs = in.readInt()


  return recList



def readString(in):
  return in.readUTF()



def readStringArray(in):
  length = in.readInt()
  array = []
  for j in range(0, length):
    array.add(readString(in))
  return array



def readByteLengthStringArray(in):
  length = in.readUnsignedByte()
  array = []
  for j in range(0, length):
    array.add(readString(in))
  return array
```

```
def readDataBlock(in, samples, recList, nRecs):
  chrom = readString(in)                  // CHROM for all records in data block
  nSeqs = in.readUnsignedShort()          // number of distinct allele sequences
                                             in sequence-coded records
  hap2Seq = []
  for j in range(0, 2*samples.length):
    hap2Seq.add(in.readUnsignedShort())   // index of sequence carried by each
                                             haplotype at sequence-coded records
  for j in range(0, nRecs):
    rec = readRecord(in, chrom, samples, nSeqs, hap2Seq)
    recList.add(rec)



def readRecord(in, chrom, samples, nSeqs, hap2Seq):
// returns marker, list of samples, allele carried by each haplotype

  marker = readMarker(in, chrom)
  coding = in.readByte()
  if coding == 0:
    return readSeqCodedRecord(in, samples, marker, nSeqs, hap2Seq)
  else if coding == 1:
    return readAlleleCodedRecord(in, samples, marker)



def readMarker(in, chrom):
  marker.chrom = chrom
  marker.pos = in.readInt()                   // POS field
  marker.id = readByteLengthStringArray(in)  // ID field

  alleleCode = in.readByte()                  // encodes SNV alleles != -1
  if alleleCode == -1:
    marker.alleles = readStringArray(in)     // number of alleles (REF + ALT)
    marker.end = in.readInt()
  else:
    marker.nAlleles = 1 + (alCode & 0b11)    // number of alleles (REF + ALT)
    permIndex = (alleleCode >> 2)
    marker.alleles = snvPerms[permIndex][0:nAlleles]   // REF is first allele
    marker.end = -1

  return marker
```

```
def readSeqCodedRecord(in, samples, marker, nSeqs, hap2Seq):
  seq2Allele = []
  for j in range(0, nSeqs):
    seq2Allele.add(in.readUnsignedByte())
  hap2Allele = []
  for j in range(0, hap2Seq.length):
    hap2Allele.add(seq2Allele[hap2Seq[j]])
  record.marker = marker
  record.samples = samples
  record.hapToAllele = hap2Al
  return record



def readAlleleCodedRecord(in, samples, marker):
  nHaps = 2*samples.length
  nAlleles = marker.alleles.length
  int[][] hapIndices
  majorAllele = -1
  for j in range(0, nAlleles):
    hapIndices.add(readIntArray(in))
    if hapIndices[j]==null:
      majorAllele = j;

  hap2Allele = []
  for j in range(0, nHaps):
    hap2Allele.add(major)
  for j in range(0, hapIndices.length):
    if hapIndices[j] != null:
      for hap in hapIndices[j]:
        hap2Allele[hap] = j

  record.marker = marker
  record.samples = samples
  record.hapToAllele = hap2Al
  return record


def readIntArray(in):
  length = in.readInt()
  if length == -1:
    return null
  else:
    array = []
    for j in range(0, length):
      array.add(in.readInt())
    return array
```