Lab Exercises 17/05/2021: BitTorrent

Esercizio 1

BitTorrent is a peer-to-peer protocol that allows the distribution and sharing of files on the Internet, which uses small metainfo files with the *.torrent* extension to describe shared resources and their division into blocks. The encoding used for metainfo files is called *Bencode*.

In Bencode encoding an element can be of type string (sequence of bytes), signed integer, list or dictionary.

The string is coded as:

```
<string length in ASCII base ten>: <string data>
```

e.g.:

```
4:spam                          "spam"
10:mytest.txt                   "mytest.txt"
0:                              ""
```

Note that there is no initial or final delimiter. Text strings use UTF-8 encoding, while binary data should not be encoded in any way, but simply inserted after the colon.

Integers are coded as:

```
i<integer encoded in ASCII base 10>e
```

e.g.:

```
i3e                             3
i-146e                          -146
i0e                             0
```

Note that numbers must be stored with 64-bits in order to handle files larger than 4 GB.

The list is coded as:

```
l(<bencode element>)e
```

With the parentheses above (round brackets) we indicate 0 or more occurrences of the content (they are not bytes in the file, i.e. the parenthesis are not present in the file), e.g.:

```
li1e4:ciaoi-2ee                 [ 1, "ciao", -2 ]
l3:byeli1ei2eee                 [ "bye", [ 1 2 ] ]
le                              [ ]
```

Note that the first character is a lowercase "L" letter. There may be an arbitrary number of other elements in the list.

The dictionary is coded as:

```
d(<bencode string><bencode element>)e
```

For example:

```
d3:cow3:moo4:spam4:eggse        {
                                    "cow"  => "moo"
                                    "spam" => "eggs"
                                }
d4:spaml1:a1:bee                {
                                    "spam" => [
                                        "a"
                                        "b"
                                    ]
                                }
de                              {
                                }
```

There can be an arbitrary number of pairs in the dictionary (key, value). Dictionary entries will always be sorted by the key (which is always a string).

All data in a metainfo file uses Bencode encoding. A metainfo file consists of **a single dictionary**, which must contain, in addition to many other optional fields, the **info** field, a dictionary that contains the **pieces** field. This is a string obtained by concatenating the 20 bytes of the hash value obtained with the SHA1 algorithm of all the pieces in which the file is fragmented. That is, if we had a file consisting of two pieces and the SHA1 hash of the first was `ABCDEFGHIJKLMNOPQRST` and the SHA1 hash of the second was `12345678901234567890`, in the torrent we would find:

```
d…4:infod…6:pieces40:ABCDEFGHIJKLMNOPQRST12345678901234567890…e…e
```

The length of the pieces string will therefore always be a multiple of 20.

Your task is to create a program that has the following syntax:

```
torrent_dump <file .torrent>
```

The program accepts a .torrent file as input and sends an ASCII textual representation on standard output in the following way:

1. Integers are displayed in decimal;

2. The strings are displayed as `<double quote><characters><double quote>` where all bytes with a value less than 32 or greater than 126 must be displayed with the character "." (the point), while the others are displayed as they are.

3. Lists are displayed as:

```
[
    1
    "ciao"
    -2
]
```

   That is, the square brackets indicate the opening and closing of the list, then the elements must be inserted in successive lines and indented with a tab character more than the brackets.

4. The dictionaries are displayed as:

```
{
    "airplane" => "Boeing"
    "list" => [
        4
        -13
    ]
    "first" => 1
}
```

   That is, the braces indicate the opening and closing of the dictionary, then the pairs of strings and elements must be inserted in successive lines and indented with a tab character more than the brackets. Each couple is represented by:

```
<string><space><equal><greater><space><element>
```

The only peculiarity is the output of the *pieces* field, which, despite being a sequence of bytes, must be displayed differently from the others. The bytes of the pieces string must be shown in hexadecimal divided into groups of 20 bytes (therefore 40 hexadecimal characters) and divided into lines. For example:

```
{
    announce => "udp://tracker.coppersurfer.tk:6969/announce"
    announce-list => [
        [
            "udp://tracker.coppersurfer.tk:6969/announce"
        ]
        [
            "udp://tracker.opentrackr.org:1337/announce"
        ]
        [
            "udp://tracker.leechers-paradise.org:6969/announce"
        ]
        [
```

```
            "http://tracker.aletorrenty.pl:2710/announce"
        ]
    ]
    comment => "Torrent downloaded from torrent cache at http://torrage.info"
    created by => "uTorrent/3.4.7"
    creation date => 1469197489
    encoding => "UTF-8"
    info => {
        length => 1038206544
        name => "Kick Ass 2010 ITALIAN DVDRip H264 AC3 5.1 Srt Ita By Jack90.mkv"
        piece length => 1048576
        pieces =>
            5d02bd860b76b28787479a169cc4eaa283256a7c
            ab82c144d608fea24f383ecc3d04fd4b615a5920
            c29936f791045911ac649b83674cc3d08733a90d
            4f3a9e1a73f1830af5a849fb1acba9838409a44b
            ...
            59a45b84bfa2fc3080e416c190195cf4babf72ee
    }
}
```

For solving the exercise, it is recommended to proceed in steps first of all by reading the entire recursive structure (using recursion is not mandatory, but it simplifies the work a lot. Feel free to shoot yourself in a foot and don't use it). Then produce the output of the single standard elements, then add the output of the pieces field and finally worry about the tabulations. Solving this exercise should take less than four hours, without any previous code available, i.e. starting from scratch.

If the input file does not exist or any error occurs during parsing, the program must terminate with errore code `-1`.