Programming Exam 16/07/2021

Exercise 1

The Snappy library created by Google is a compression and decompression library that uses an algorithm similar to LZSS, which does not aim for the maximum compression ratio but favors speed. The format of the compressed output stream produced by Snappy is byte aligned and it is specified in the following.

The first element encountered in the stream is the *Preamble*. It contains the original size of the stream before being compressed and it is represented using little-endian *varints*. A *varint* consists of a series of bytes in which the 7 least significant bits are part of the data and the most significant bit, if set to 1, is used to indicate whether there are additional bytes to read. For example a length of 64 will be represented using a single byte with the value 0x40, while a length of 2097150 (0x1FFFFE in hexadecimal) will be represented using 3 bytes: 0xFE 0xFF 0x7F:

```
0x1FFFFE -> 1.1111.11|11.1111.1|111.1110 ->
            c.cccc.cc|bb.bbbb.b|aaa.aaaa

         -> 1111.1110 1111.1111 0111.1111 -> 0xFE 0xFF 0x7F
            1aaa.aaaa 1bbb.bbbb 0ccc.cccc
```

After the preamble there is the compressed stream. There are two types of elements in a Snappy stream, *Literals* and *Copies*, and there are no restrictions on the order in which they can appear, other than the fact that a stream must start with a Literal. Each element begins with a Byte *Tag* whose two **least significant** bits (bits in positions 1 and 0) indicate its type:

Value	Type
00	Literal
01	Copy with 1-byte offset
10	Copy with 2-byte offset
11	Copy with 4-byte offset

The interpretation of the 6 **most significant** bits (bits in positions 7,6,5,4,3 and 2) of the tag changes depending on the type of element.

Literals

Literals contain uncompressed data stored directly in the stream. The way in which the `length` of a literal is stored changes depending on the length of the literal itself:

- For literals up to and including 60 bytes long, the value of `length - 1` (length minus 1) is stored in the 6 most significant bits of the byte tag.
- For longer literals the value of `length - 1` (length minus 1) is stored in the bytes following the byte tag in little-endian. The 6 most significant bytes of the byte tag indicate how many bytes

must be read:

Value	Number of bytes
60	1 byte
61	2 bytes
62	3 bytes
63	4 bytes

The literal data follows directly after the length bytes.

Copies

The copies are references to the already decompressed stream, they contain an *offset*, which indicates how many bytes to go back from the current position, and a *length*, which indicates how many bytes must be copied starting from that position. As in the LZ77 and LZSS algorithms, it is possible to have lengths larger than the offset, for example the "xababab" sequence could be encoded as:

```
<literal: "xab"> <copy: offset = 2 length = 4>
```

You can have different types of copy elements depending on the distance of the offset and the length to be copied:

- **Copy with 1-byte offset**: these elements can encode lengths between 4 and 11 (4 and 11 included) and offsets between 0 and 2047 (0 and 2047 included).
  The value of `length - 4` (length minus 4) takes up 3 bits and it is stored in bits 4, 3 and 2 of the byte tag.
  The `offset` value takes up 11 bits of which the 3 most significant are stored in bits 7, 6 and 5 of the tag byte (the 3 most significant bits) and the 8 least significant are stored in the byte following the tag.
  For example a copy with offset 2 and length 4 would be stored as:

```
tag: 01
length: 000 (3 bits, minus 4)
offset: 00000000010 (11 bits)
+-----------+-----------+
| 0000.0001 | 0000.0010 |
+-----------+-----------+
```

- **Copy with 2-byte offset**: these elements can encode lengths between 1 and 64 inclusive and offsets between 0 and 65535 inclusive. The value of `length - 1` (length minus one) occupies the 6 most significant bits of the byte tag and the offset is stored as a 16-bit little-endian integer in the two bytes following the byte tag.
- **Copies with 4-byte offset**: they work like copies with 2-byte offset with the only difference that the offset is stored as a 32-bit little-endian integer in the 4 bytes following the tag.

In the stream there is no method to signal the end of the compressed data, simply the end of the compressed data marks the end of the stream.

Your task

Write a command line program with the following syntax:

```
snappy_decomp <input file> <output file>
```

The program must open a file compressed with the Snappy algorithm, decompress its contents and save them to the decompressed file.