LZ4 explained

The compressed stream is composed of sequences.
Each sequence starts with a **token**.
The token is a one byte value, separated into two 4-bits fields (which therefore range from 0 to 15).
The first field is to indicate the *length of literals*. If it is 0, then there is no literal. If it is 15, then we need to add some more bytes to indicate the full length. Each additionnal byte then represent a value of 0 to 255, which is added to the previous value to produce a total length. When the byte value is 255, another byte is output.
There can be any number of bytes following the token. There is no "size limit". As a sidenote, here is the reason why a not-compressible input stream can be expanded by up to 0.4%.

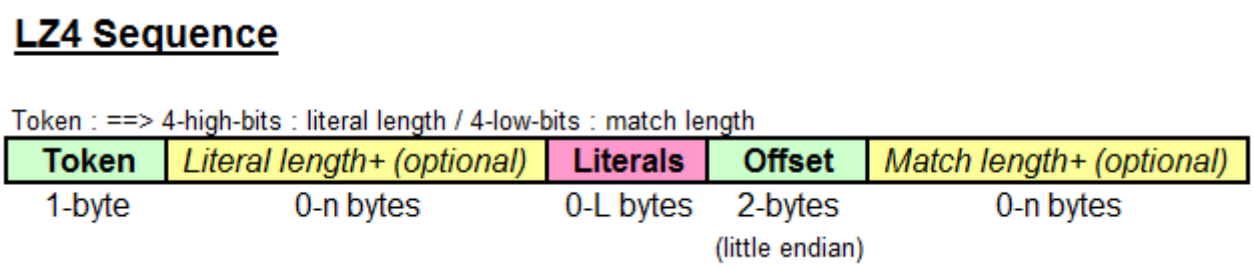
Following the token and optional literal length bytes, are the literals themselves.
They are exactly as numerous as previously decoded in length of literals. It's possible that there are zero literal.

Following the literals is the offset. This is a 2 bytes value, between 0 and 65535. It represents the position of the match to be copied from. Note that 0 is an invalid value, never used. 1 means "current position - 1 byte". 65536 cannot be coded, so the maximum offset value is really 65535. The value is stored using "little endian", which means the lower byte is the first one in the stream.

Then we need to extract the matchlength. For this, we use the second token field, a 4-bits value, from 0 to 15. There is an offset to apply, which is the minimum length of a match, called minmatch. This minimum is 4. As a consequence, a 0 value means 4 bytes, and a value of 15 means 19+ bytes.
Similar to literal length, on reaching the highest possible value (15), we output additional bytes, one at a time, with values ranging from 0 to 255. They are added to total to provide the final matchlength. A 255 value means there is another byte to read and add. There is no limit to the number of optional bytes that can be output this way (This points towards a maximum achievable compression ratio of ~250).

With the offset and the matchlength, the decoder can now proceed to copy the repetitive data from the already decoded buffer. On decoding the matchlength, we reach the end of the sequence, and start another one.

Graphically, the sequence looks like this :



Note that the last sequence stops right after literals field.

There are specific parsing rules to respect in order to remain compatible with assumptions made by the decoder :
1) The last 5 bytes are always literals
2) The last match cannot start within the last 12 bytes
Consequently, a file with less then 13 bytes can only be represented as literals
These rules are in place to ensure that the decoder will never read beyond the input buffer, nor write beyond the output buffer.