CHEDS String Formatting:

This type of formatting is used in CHEDS-file-types which are used for loading in saved games via the CHEDS engine.

The string will be iterated through from first to last index. The protocol changes states and saves data as the string index progresses. This type of structured data extraction is my favourite way to tackle this procedure and opens up possibilities for flexibilities in encoding and decoding.
Think of it as reading a book, you get introduced to the entire story. Within main chapters you get context to understand the next part.

The “Introduction” in this case is the...

Header:

Ongoing projects will always change. Therefore, attributes may get added or removed to/from classes and structures. Therefore the header must not be of fixed byte size. We can identify the correct encoding format by including a version in our header string, which should – of course – be the first element (In our case the third, because there is the very first identifier “CHEDS”, which is here for good measure, and the second determines the version byte length). This allows for backwards compatibility.

The way that individual header bytes are handled is determined by the compiled program itself.
Of course a version may require a different handler than another version. However the universal header format will always remain as follows:

BYTE#	0	1	2	3	4	5	6..(6+VLEN)	...
DATA	‘CHEDS’					VLEN	VERSION	ATTRIBUTES

- ‘CHEDS’
  - File identifier, this is nothing but raw ASCII-characters
- VLEN
  - Version length tag, this is the int-8 value of amount of bytes the version tag will take
- VERSION
  - Version tag, in ASCII used for compatibility
- ATTRIBUTES
  - All attributes, compressed from a mess of bits into bytes

Notice that we require a version byte length identifier. It is bad practice to assume that a version has e.g. only 4 bytes. What would happen if we hypothetically reached the version after ’99.9’? How would we identify revisions? In general, never assume anything that’s ambiguous, neither in social media nor in file headers.
The second thing to notice is that our attributes will not have an “ending character” (e.g. simply ‘\0’) as this could cause issues for certain or rare cases, which just won’t do. Instead of using an attribute length byte, the version itself can identify the length of attribute bytes.

After the attributes comes the...

Body:

The body is a stream of bytes that determines the positions of pieces, which colour is selected, and which pieces are selected.

Chess has two coordinates that span from 0 to 7, which is perfect for our applications:
As the two coordinates only use 6 bits, we can use the first 2 bits to pipe additional commands or attributes into context. As such, this sheet will refer to the first 2 bits as “context” and to the last 6 as “data”.

BIT#	0	1	2	3	4	5	6	7
WHITE	0	1	FILE			RANK		
BLACK	1	0	FILE			RANK		
P. SEL.	1	1	PIECE ID					
EXIT	0	0	X	X	X	X	X	X

- WHITE
  - Place a white piece at FILE, RANK
- BLACK
  - Place a black piece at FILE, RANK
- P. SEL.
  - Select a type of piece such as a pawn. This value can be reused until P.SEL. is triggered again. Very useful for many pawns.
- EXIT
  - The exit byte which will most likely just be ‘\1’, or at least bigger than ‘\0’. Closes and validates the file handler and returns with success (1).