Algorithm

1. Start
2. Import necessary libraries: Import the yaml, random, and struct libraries.
3. Define functions:
    1. pack_list(values, endianness): Packs a list of values into binary data based on their types and endianness.
    2. handle_meta(meta): Handles metadata such as endianness and file extension.
    3. random_based_on_size(size, endianness): Generates random binary data based on the specified size and endianness.
    4. random_based_on_type(size, type_field, endianness, encoding=None): Generates random binary data based on the specified type, size, endianness, and encoding.
    5. handle_seq(seq, endian, parent): Handles sequence items by generating binary data based on the provided sequence attributes and parent types.
    6. handle_type(types, endianness, user_defined_type): Handles user-defined types by recursively generating binary data based on the defined sequences.

4. Read YAML data: Read the YAML data from the provided file path.Process data and generate output
    1. Iterate over a range of iterations (e.g., 10) to generate multiple output files.
    2. Extract metadata (endianness and file extension) from the YAML data.
    3. Handle the sequence defined in the YAML data, generating binary data based on the specified attributes and types.
    4. Write the generated binary data to output files with sequential names.
5. Stop

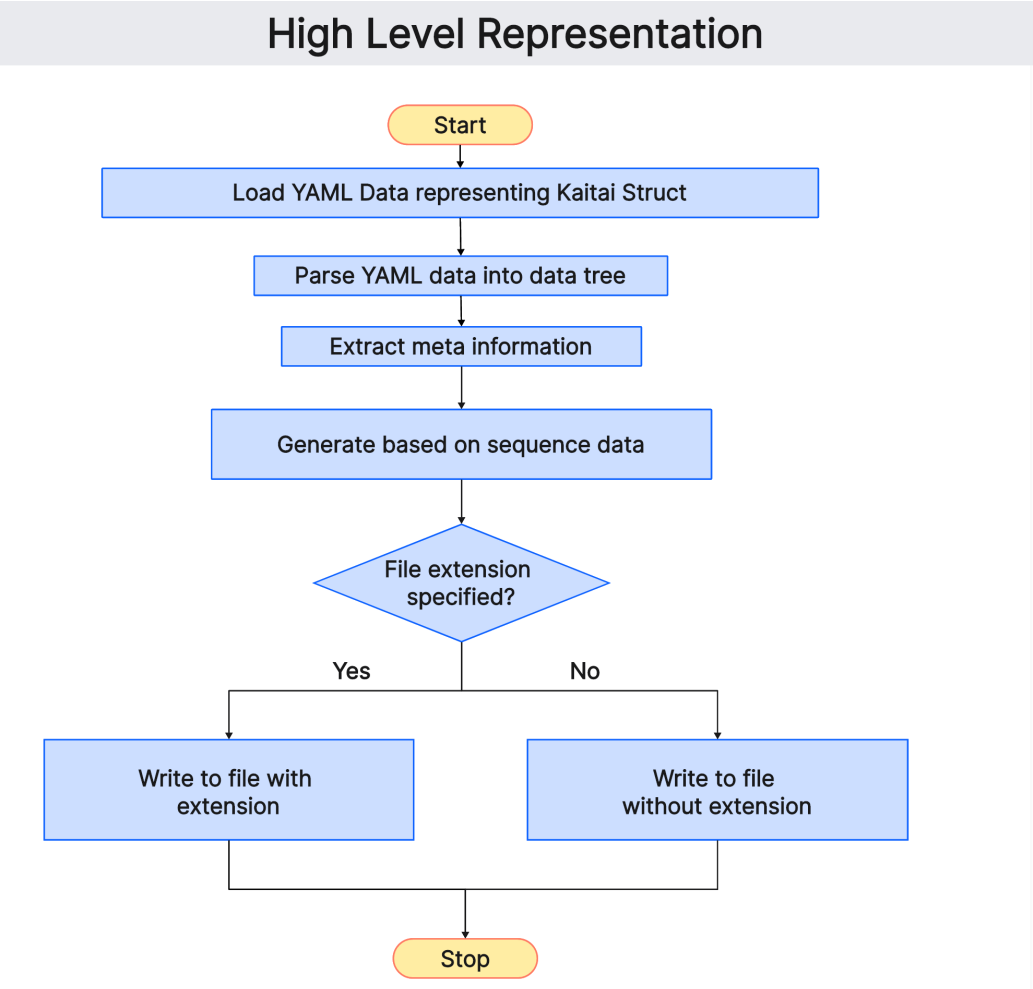
Pseudocode

1. Read metadata from the YAML data:
   a. Initialize endianness and file_extension variables.
   b. Iterate through each key-value pair in the metadata.
   c. If the key is 'file-extension', assign its value to the file_extension variable.
   d. If the key is 'endian', assign its value to the endianness variable.
   e. Return the endianness and file_extension.

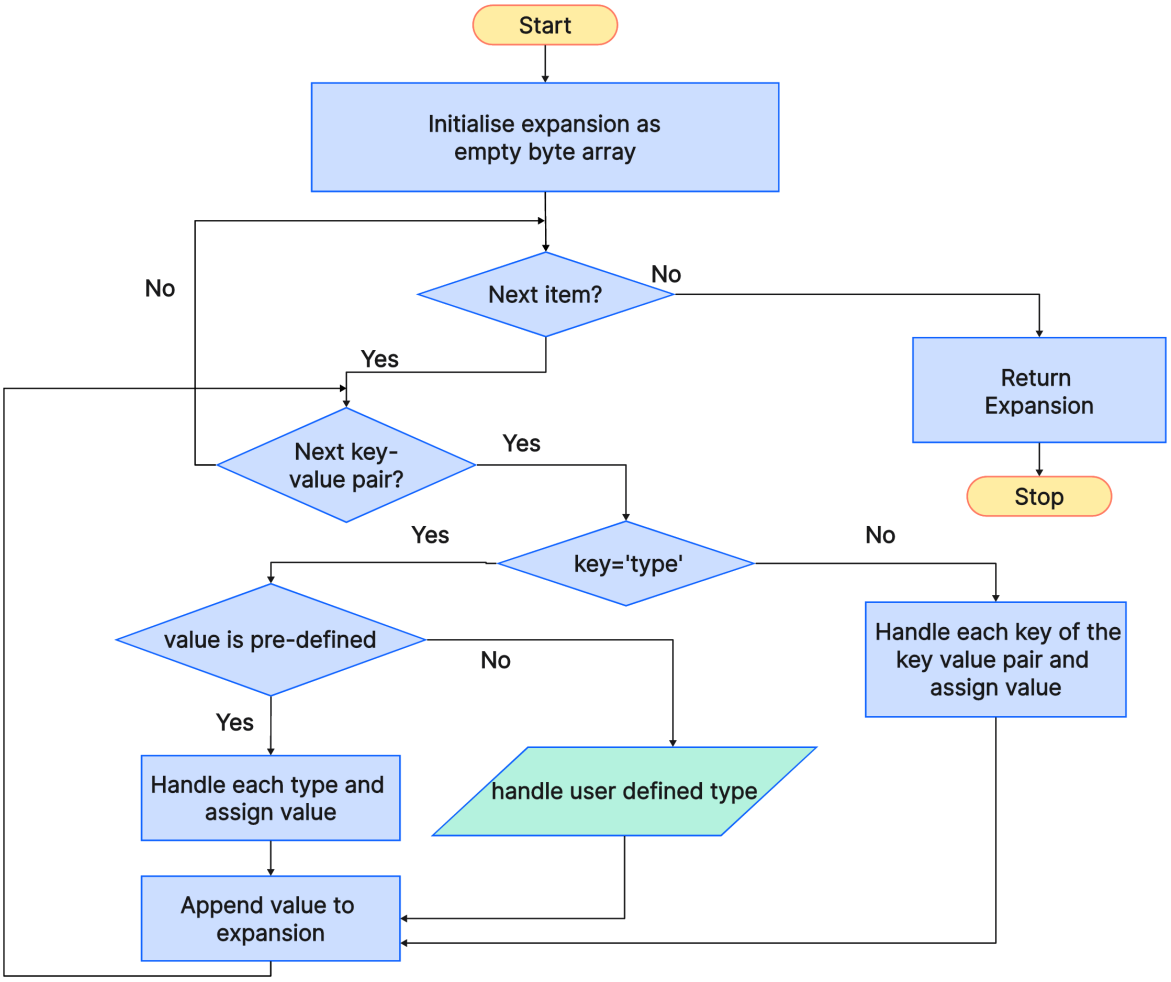
2. Handle sequence data:
   a. Initialize expansion variable to an empty byte string.
   b. Iterate through each item in the sequence.
   c. Initialize type, size, encoding, endianness, and valid_endian variables.
   d. For each item in the sequence:
      i. If the key is 'contents', call the pack_list function to pack the values into binary data.
      ii. If the key is 'size', assign its value to the size variable.
      iii. If the key is 'type', assign its value to the type variable.
      iv. If the key is 'encoding', assign its value to the encoding variable.

v. If the key is 'valid', convert its value to bytes and append it to the expansion.
e. If the type is None:
i. Generate random binary data based on the specified size and endianness using the random_based_on_size function.
ii. Append the generated binary data to the expansion.
f. If the type is one of the predefined types (e.g., 'u2', 'u4', 's2', 'str'), generate random binary data based on the type, size, and endianness using the random_based_on_type function.
g. If the type is a user-defined type, recursively call the handle_type function to handle the user-defined type.
h. Return the expansion.

3. Handle user-defined types:
a. Initialize expansion variable to an empty byte string.
b. Iterate through each key-value pair in the types dictionary.
c. If the key matches the user_defined_type, recursively call the handle_seq function to handle the sequence defined for the user-defined type.
d. Return the expansion.



Handling Sequence Data



Handling User-defined Types