MEMO

Author: Xiaocheng Zhang | Tian Xia

API:

- We implemented serialization into several files:
    - one file contains a class used to **serialize** is called `serializer.h`
    - one file contains a class used to **deserialize** is called `deserializer.h`
    - one file contains a class used to save the data that has been serialized is called `keeper.h`
    - one main file used to generate several example that used to test is called `main.cpp`.

Methods we have in serializer.h:

- methods in `class Serializer`:
    1. `virtual char *serialize(String *s)`: Used to serialize a string type into char* that used to send.
    2. `virtual char *serialize(int i)`: Used to serialize a int type into char* that used to send.
    3. `virtual char *serialize(float f)`: Used to serialize a float type into char* that used to send.
    4. `virtual char *serialize(double f)`: Used to serialize a double type into char* that used to send.
- methods in `class AdvanceSerializer` (**No implementation because of time.**):
    1. `virtual char *serialize(StringArray *sa)`: Used to serialize a StringArray type into char*.
    2. `virtual char *serialize(DoubleArray *da)`: Used to serialize a DoubleArray type into char*.
    3. `virtual char *serialize(Message *msg)`: Used to serialize a Message type into char*.

4. `virtual char *serialize(Ack *ack)` : Used to serialize a Ack type into char*.
5. `virtual char *serialize(Status *st)` : Used to serialize a Status type into char*.
6. `virtual char *serialize(Register *r)` : Used to serialize a Register type into char*.
7. `virtual char *serialize(Directory *d)` : Used to serialize a Directory type into char*.
   All Methods above are returning a char that used to send*

Methods we have in deserializer.h:

- method `basic_deserialize` : Used to deserialize five basic type of data (failed to deserialize double since it failed in serializing). Return a `void*` type used to save all kinds of data.
- methods in `class Deserializer` :
  1. `virtual void deserialize(String *sa)` : Used to deserialize the data read by method `read_data` , `read_data` will read data as String for four basic types and StringArray for all class types.
  2. `virtual void deserialize(StringArray *sa)` : Used to deserialize all class types (Did not implementation)
  3. `virtual void *read_data(char *filename)` : Used to read the serialized data which saved in the file filename. It keeps reading until reach the end of file. It will translate all data which signed as basic types (start with a serial number 0000 to 0999) into single String type, and all data which signed as class types (start with a serial number 1000 to 9999) into StringArray (This part did not implement).

Methods we have in keeper.h:

- methods in `class Keeper` (abstract class):
  1. `virtual void write_file(char *filename)` : Used to write all serialized data which saved as a StringArray in this class.
  2. `virtual void append(char *s)` : Used to append a serialized data to `serialized_buffer_list` . char *s is the data returned by method `serialize` in `class Serializer` .

3. `virtual void append(String *s)`: Same with (2) but accept a complete String type.

- In order to call those methods, user needs to create a new class called `BasicKeeper` or classes made by user self.

Usage:

- Our methods are all defined in classes which are Function Objects. This is our design choice that is easy for user to add more classes which they want.
- Ways we serialize data is:
    - In `types.h`, we defined many "types" that used to check data types.
    - We saved data as a format "data-type data-value\n" as basic types.
    - We want to save class types data as a format:
      "data-type { data-type first-field | data-type second-field | ... }"
      where there is an array, we want to save it as a format:
      "data-type [
      data-type first-element\n
      data-type second-element\n
      ...
      ]"