CIS*2750
Assignment 2, Module 1

Updates to GEDCOMparser.h

Replace `enum eCode` with:

```
typedef enum eCode {OK, INV_FILE, INV_GEDCOM, INV_HEADER, INV_RECORD, OTHER_ERROR, WRITE_ERROR} ErrorCode;
```

New functions

1. writeGEDCOM

```
/** Function for writing a GEDCOMobject into a file in GEDCOM format.
*@pre GEDCOMobject object exists, is not null, and is valid
*@post GEDCOMobject has not been modified in any way, and a file representing the GEDCOMobject contents in GEDCOM format has been created
*@return the error code indicating success or the error encountered when writing the GEDCOM
*@param obj – a pointer to a GEDCOMobject struct
**/

GEDCOMerror writeGEDCOM(char* fileName, const GEDCOMobject* obj);
```

This function takes a GEDCOM object and saves it to a file in GEDCOM format. This might seem daunting, but it will be somewhat similar to `printGEDCOM` (traversing the various lists and references), so you can borrow a lot of ideas (and code) from it.

You need to make sure that the text output is in correct GEDCOM format. You can select any line ending you want - just be consistent and use the same line ending throughout the entire file.

Testing this function will be simple. You can read in a GEDCOM, parse, and save. The output file should be identical to the input file. Alternatively, you can read a GEDCOM as `object1`, save it, then read in the file you have just saved as `object2`. The two objects must be identical.

To make this work, we will only deal with a subset of full GEDCOM functionality. Basically, if we don't have a struct for it, we don't need to know how to read/write it. As a result, we will make the a few assumptions.

XREFs

You will need to turn pointers into XREFs. How you choose to name your XREFs is up to you. Do **not** try to save the XREFs from a file you have parsed and rely on them. These are discarded as soon as `createGEDCOM()` returns. In Assignment 3, you will be modifying GEDCOM objects and creating them from scratch, so you will need to create new XREFs.

Events

Events will be saved as part of the Individual or Family record to which they belong. You can assume that the `otherFields` list of an Event struct is empty, and you do not need to save anything from it. Make sure that the levels in front of the event tags are correct.

Individuals

All entries in the `otherFields` list are simple level 1 fields, i.e. RESN, SEX, RFN, AFN, or RIN - not components of sub-structures. The only exception is that `otherFields` may contain components of the name structure, specifically GIVN and SURN. If these are present, you will need to save them correctly as level 2 tags in the output.

For example, imagine you have an Individual struct with the following data:
- givenName = "Anne", surname = "Hathaway"
- otherFields list has two Field structs:
  - tag = "GIVN", value = "Anne"
  - tag = "SURN", value = "Hathaway"

You would save this portion of the Individual record as

```
1 NAME Anne /Hathaway/
2 GIVN Anne
2 SURN Hathaway
```

NOTE: `otherFields` **must not** contain XREFs, or any other junk left over from parsing the original file. This means you may need to clean up your parser if you have misused `otherFields` in Assignment 1.

Individuals may have Events in the `events` list, which will need to be saved to the file in the appropriate format - i.e. according to the GEDCOM specification.

The entries from the `families` list will need to be converted into correct FAMC and FAMS tags. For example, the `families` list for Anne Hathaway may contain two entries
1. Pointer to a family where Anne is the spouse
2. Pointer to a family where Anne is the child

These would be saved to the file as something like:
```
1 FAMS @F1234@
1 FAMC @F2345@
```

Family

You can assume that the `otherFields` list of a Family struct is empty, and you do not need to save anything from it. The pointers from the two spouse fields and the children list will need to be converted into appropriate new XREFs.

Families may have Events in the `events` list, which will need to be saved to the file in the appropriate format - i.e. according to the GEDCOM specification.

Submitter

You can assume that the `otherFields` list of a Submitter struct is empty, and you do not need to save anything from it.

Header

You can assume that the `otherFields` list of a Header struct is empty, and you do not need to save anything from it. While we do not have a field for the FORM tag, its value will always be LINEAGE-LINKED.

Return value: writeGEDCOM returns WRITE_ERROR if writing the file fails for any reason (NULL arguments, unable to open the file for writing, etc.).

2. validateGEDCOM

```
/** Function for validating an existing GEDCOM object
 *@pre GEDCOM object exists and is not null
 *@post GEDCOM object has not been modified in any way
 *@return the error code indicating success or the error encountered when
validating the GEDCOM
 *@param obj – a pointer to a GEDCOMobject struct
 **/

ErrorCode validateGEDCOM(const GEDCOMobject* obj);
```

This function is used to validate a GEDCOMobject before saving it. It returns an error code indicating the problem (if any) with the GEDCOM.

The validation is straightforward. Return an error that corresponds to the component in which the error occurs:

- INV_GEDCOM if any of the required struct pointers - i.e. `header` and `submitter` in GEDCOOMobject - are NULL
- INV_HEADER if one of the fields in Header is not initialized (empty otherFields list is OK)
- INV_RECORD if
  - Submitter name is empty
  - One of the pointers that you need to dereference when saving a GEDCOM object to file is NULL (e.g. a NULL pointer to a Family in the families list)
  - A value of some field is too long, and would cause the gedcom_line to exceed 255 characters. To keep things simple, we will set the max length of a value to 200 characters, unless the GEDCOM description has a lower limit for that value.

3. Getting started

Implement writGEDCOM() in stages. Start by writing a function to save a minimal GEDCOM (header, submitter, and nothing else). Once it compiles and runs - correctly! - add more functionality, once step at a time. Make sure you test after adding each increment. Suggested order:

- Add functionality for saving Individuals (don't worry about events yet)
- Then add functionality for saving Families (again, don't worry about events yet), and create correct XREF references.
- Finally, add the functionality for saving Events.