CIS*2750
Assignment 2, Module 2

New functions

1. *getDescendantListN*

```
/** Function to return a list of up to N generations of descendants of an individual in a
GEDCOM
 *@pre GEDCOM object exists, is not null, and is valid
 *@post GEDCOM object has not been modified in any way, and a list of descendants has been
created
 *@return a list of descendants.  The list may be empty.  All list members must be of type
List.
 *@param familyRecord - a pointer to a GEDCOMobject struct
 *@param person - the Individual record whose descendants we want
 *@param maxGen - maximum number of generations to examine (must be >= 1)
 **/

List getDescendantListN(  const GEDCOMobject* familyRecord,
                                const Individual* person,
                                unsigned int maxGen);
```

This function will be somewhat similar to `getDescendants()` in that it will return a list of descendants. However, the returned list will be much more structured, and we will use this structure later. Note that an individual might have more than one FAMS tag (e.g. individual remarried).
Details:

- The entries of the descendant list will be "generation lists".
- Each generation list will contain a list of Individual records - a single generation of an individual's descendants.
- The descendant list must be sorted by generation. So if an individual has children and grandchildren, the first generation list will contain the list of children, and the second generation list - the list of grandchildren
- Individuals in every generation list must be be sorted by their last name (`surname` field):
  - If a generation contains multiple individuals with the same last name, sort these individuals by first name.
  - If a generation contains multiple individuals with the same last name and first name, they can appear in any order.
  - If the `surname` field of an `Individual` struct is an empty string, place this individual at the end of the generation list.
- The list will accept an argument `maxGen`, which will limit the depth of the search
  - If `maxGen = 0`, return all descendants
  - If `maxGen > 0`, return all descendants up to generation maxGen. So if `maxGen = 1`, return a descendant list containing only one generation list - the individual's children. If `maxGen = 2` - the descendant list will contain two lists (children and grandchildren), etc.
  - If `maxGen` is exceeds available data, return all descendants. For example, if `maxGed = 7` and the record for the individual only had children of that individual, the descendant list will contain only one generation list - the individual's children.
- Return an empty list of no descendants found or if the `person` argument is NULL.

Individuals in each generation must be unique. Make sure you check for duplicates when inserting individuals into a generation list. You can use the address of the Individual struct to determine uniqueness. So if someone is a parent in two families, the generation list for the children will contain unique records - the same child cannot appear twice. This situation is unlikely to occur, but make sure you account for it.

Make sure that the generation lists contain **copies** of the Individual records, rather than the references to actual records.

See examples on page 3 for some sample pseudocode output.

2. *getAncestorListN*

```
/** Function to return a list of up to N generations of ancestors of an individual in a
GEDCOM
 *@pre GEDCOM object exists, is not null, and is valid
 *@post GEDCOM object has not been modified in any way, and a list of ancestors has been
created
 *@return a list of ancestors.  The list may be empty.
 *@param familyRecord - a pointer to a GEDCOMobject struct
 *@param person - the Individual record whose descendants we want
 *@param maxGen - maximum number of generations to examine (must be >= 1)
 **/
List getAncestorListN(    const GEDCOMobject* familyRecord,
                          const Individual* person,
                          int maxGen);
```

This function is complementary to `getDescendantsListN`. It will return an ancestor list with similar organization. An individual's ancestors are defined to be all spouses in the families pointed to by the individual's FAMC tags. Note that an individual might have more than one FAMC tag (e.g. parents got remarried).
Details:

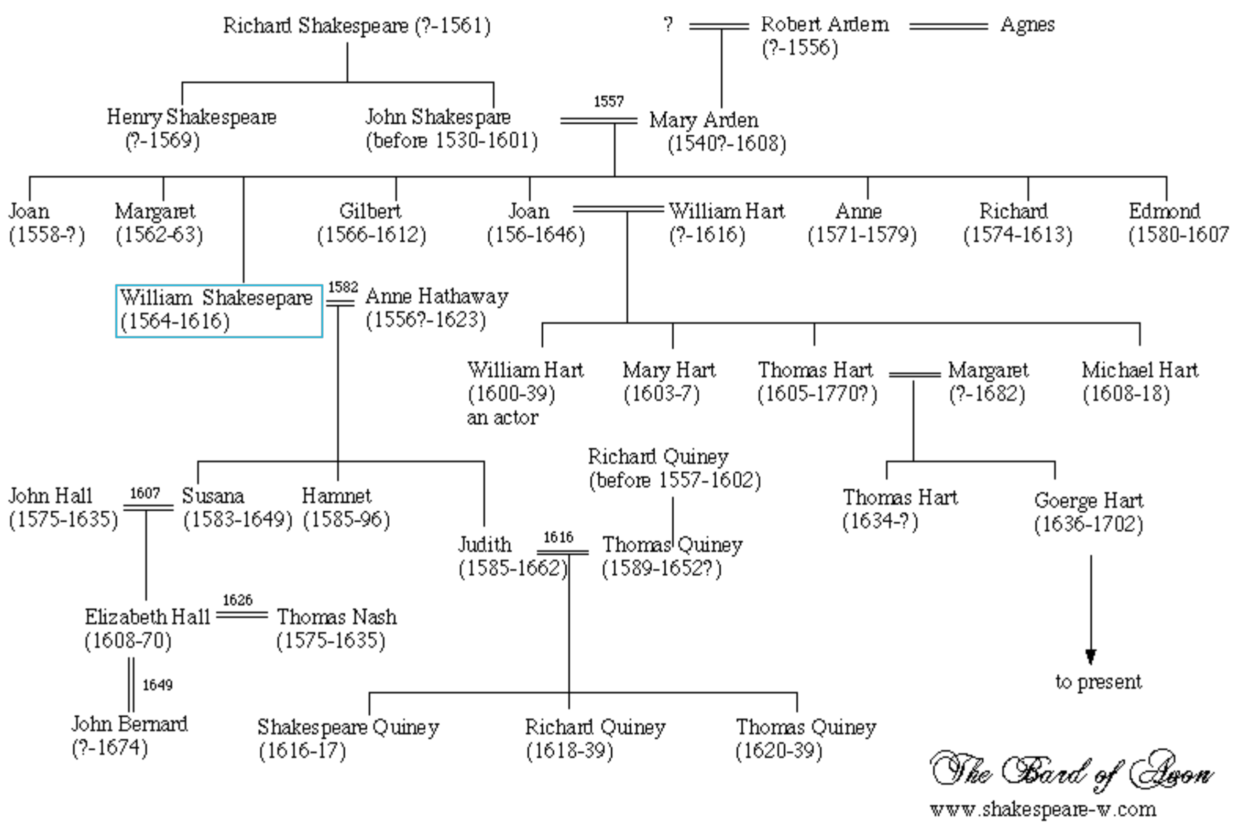
- The entries of the ancestor list will be "generation lists".
- Each generation list will contain a list of Individual records - a single generation of an individual's ancestors.
- The ancestor list must be sorted by generation, in reverse chronological order. So if an individual has parents and grandparents, the first generation list will contain the list of parents, and the second generation list - the list of grandparents.
- Individuals in every generation list must be be sorted by their last name (`surname` field):
  - If a generation contains multiple individuals with the same last name, sort these individuals by first name.
  - If a generation contains multiple individuals with the same last name and first name, they can appear in any order.
  - If the `surname` field of an `Individual` struct is an empty string, place this individual at the end of the generation list.
- The list will accept an argument `maxGen`, which will limit the depth of the search
  - If `maxGen = 0`, return all ancestors
  - If `maxGen > 0`, return all ancestors up to generation maxGen. So if `maxGen = 1`, return a ancestor list containing only one generation list - the individual's parents. If `maxGen = 2` - the ancestor list will contain two lists (parents and grandparents), etc.
  - If `maxGen` is exceeds available data, return all ancestors. For example, if `maxGed = 7` and the record for the individual only had parents of that individual, the ancestor list will contain only one generation list - the individual's parents.
- Return an empty list of no ancestors found or if the `person` argument is NULL.

Individuals in each generation must be unique. Make sure you check for duplicates when inserting individuals into a generation list.

For example, if someone is linked to two families as a child - e.g. mother remarried - the generation list for the parents will contain the father, the mother, and the mother's new spouse (e.g. stepfather). The mother must not appear twice.

Make sure that the generation lists contain **copies** of the Individual records, rather than the references to actual records.

See examples on page 3 for some sample pseudo-code output.



Given the genealogy record above, here are some sample results. The output lists are in pseudocode.

If we look at descendants of William Shakespeare:

- `getDescendantsListN` with `maxGen=1` would return a descendants list with one generation list:

```
{  { Hamnet Shakespeare, Judith Shakespeare, Susana Shakespeare}  }
```

- `getDescendantsListN` with `maxGen=2` would return a descendants list with two generation lists:

```
{ { Hamnet Shakespeare, Judith Shakespeare, Susana Shakespeare}
  { Elizabeth Hall, Richard Quiney, Shakespeare Quiney, Thomas Quiney} }
```

If we look at ancestors of Mary Hart

- `getAncestorsListN` with `maxGen=1` would return an ancestors list with one generation list:

```
{  { Joan Hart, William Hart }  }
```

- `getAncestorsListN` with `maxGen=3` would return an ancestors list with three generation lists. The second generation list only contains Joan Hart's parents, since William Heart's parents are not in the record. If they were, they would be in the second generation list. Also, we assume that Mary Arden's FAMC tags point to two families: one where Robert Arden is the husband and the wife is unknown, one where Robert Arden is the husband and Agnes (without the last name) is the wife.

```
{  { Joan Hart, William Hart }
   { Mary Arden, John Shakespeare}
   { Robert Arden, Richard Shakespeare, Agnes }
}
```