Multimedia Management System

In this application you code classes to load from files information about books, movies, and TV shows, create objects with the loaded information and manage those objects using STL containers and algorithms. If something goes wrong during the execution, you will report it to clients using exceptions.

Put all the global variables, global functions/operator overloads, and types inside the `seneca` namespace and include the necessary guards in each header file.

`settings` Module

The `settings` will contain functionality regarding configuration of the application. Design and code a structure named `Settings`; in the header, *declare* a global variable of this type named `g_settings` and define it in the implementation file.

For simplicity reasons, this type will contain only public data-members and no member-functions.

Public Members

- `m_maxSummaryWidth` an integer in 2 bytes that will store the maximum width of text then printing the summary of a media item. By default, the width is 80.
- `m_tableView` as a Boolean attribute; when `true`, print to screen the information about the media items formatted as a table. By default, this attribute is `false`.

`mediaItem` module (supplied)

This module contains information about a generic multimedia item (a book, movie, or TV show).

Do not modify this module! Study the code supplied and make sure you understand it.

book Module

Design and code a class named `Book` derived from `MediaItem` that can store the following information (for each attribute, chose any type that you think is appropriate--you must be able to justify the decisions you have made):

- `m_author`: the author of the book
- **title** (inherited)
- `m_country`: the country of publication
- **the year of publication** (inherited)
- `m_price`: the price of the book
- **the summary** (inherited): a short description of the book

Private Members

- add any constructors that are necessary for your design

This class will not offer any public constructors.

Public Members

- `void display(std::ostream& out) const override`: override this function to print the information about a single book. Use the following implementation:

```
void display(std::ostream& out) const
{
  if (g_settings.m_tableView)
  {
    out << "B | ";
    out << std::left << std::setfill('.');
    out << std::setw(50) << this->getTitle() << " | ";
    out << std::right << std::setfill(' ');
    out << std::setw(2) << this->m_country << " | ";
    out << std::setw(4) << this->getYear() << " | ";
    out << std::left;
    if (g_settings.m_maxSummaryWidth > -1)
    {
      if (static_cast<short>(this->getSummary().size()) <=
g_settings.m_maxSummaryWidth)
        out << this->getSummary();
      else
        out << this->getSummary().substr(0, g_settings.m_maxSummaryWidth - 3) <<
"...";
    }
    else
      out << this->getSummary();
    out << std::endl;
  }
  else
  {
    size_t pos = 0;
    out << this->getTitle() << " [" << this->getYear() << "] [";
    out << m_author << "] [" << m_country << "] [" << m_price << "]\n";
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << "" << '\n';
    while (pos < this->getSummary().size())
    {
      out << "    " << this->getSummary().substr(pos, g_settings.m_maxSummaryWidth)
<< '\n';
      pos += g_settings.m_maxSummaryWidth;
    }
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << ""
      << std::setfill(' ') << '\n';
  }
}
```

- `Book* createItem(const std::string& strBook)`: a **class function** that receives as parameter the representation of the book as a string and builds a dynamically allocated object of type `Book` using the information from the string and returns it to the client. The parameter contains a single line of text extracted from the file `books.csv`. The format of the line is as following:

```
AUTHOR,TITLE,COUNTRY,PRICE,YEAR,SUMMARY
```

This function should remove all spaces from the **beginning and end** of any token in the string.

If the parameter is an empty string or a string starting with `#`, raise an exception with the message `Not a valid book.`. (lines in the file starting with `#` are considered comments and should be ignored).

If all the data is correctly loaded, create a dynamic object of type `Book` using your private constructors and return its address to the client.

When implementing this function, consider the following functions:

  - std::string::substr()
  - std::string::find()
  - std::string::erase()
  - std::stoi()
  - std::stod()

Add any other private member that is required by your design!

`movie` Module

Design and code a class named `Movie` that stores the following information for a single movie (for each attribute, chose any type that you think is appropriate--you must be able to justify the decisions you have made):

- **title** (inherited)
- **the year of release** (inherited)
- **the summary** (inherited)

Private Members

- add any constructors that are necessary for your design

This class will not offer any public constructors.

Public Members

- `void display(std::ostream& out) const override`: override this function to print the information about a single book. Use the following implementation:

```
void display(std::ostream& out) const
{
  if (g_settings.m_tableView)
  {
    out << "M | ";
    out << std::left << std::setfill('.');
    out << std::setw(50) << this->getTitle() << " | ";
    out << std::right << std::setfill(' ');
    out << std::setw(9) << this->getYear() << " | ";
    out << std::left;
    if (g_settings.m_maxSummaryWidth > -1)
    {
      if (static_cast<short>(this->getSummary().size()) <=
g_settings.m_maxSummaryWidth)
        out << this->getSummary();
      else
        out << this->getSummary().substr(0, g_settings.m_maxSummaryWidth - 3) <<
"...";
    }
    else
      out << this->getSummary();
    out << std::endl;
  }
  else
  {
    size_t pos = 0;
    out << this->getTitle() << " [" << this->getYear() << "]\n";
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << "" << '\n';
    while (pos < this->getSummary().size())
    {
      out << "    " << this->getSummary().substr(pos, g_settings.m_maxSummaryWidth)
<< '\n';
      pos += g_settings.m_maxSummaryWidth;
    }
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << ""
      << std::setfill(' ') << '\n';
  }
}
```

- `Movie* createItem(const std::string& strMovie)`: a **class function** that receives as parameter the representation of the movie as a string and builds a dynamically

allocated object of type `Movie` using the information from the string and returns it to the client. The parameter contains a single line of text extracted from the file `movies.csv`. The format of the line is as following:

```
TITLE,YEAR,SUMMARY
```

This function should remove all spaces from the **beginning and end** of any token in the string.

If the parameter is an empty string or a string starting with `#`, raise an exception with the message `Not a valid movie.` (lines in the file starting with `#` are considered comments and should be ignored).

If all the data is correctly loaded, create a dynamic object of type `Movie` using your private constructors and return its address to the client.

When implementing this function, consider the following functions:

- std::string::substr()
- std::string::find()
- std::string::erase()
- std::stoi()
- std::stod()

Add any other private member that is required by your design!

`tvShow` Module

Design and code a class named `TvShow` that stores the following information for a TV show (for each attribute, chose any type that you think is appropriate--you must be able to justify the decisions you have made):

- `m_id`: the identifier of this show
- **title** (inherited)
- **the year of release** (inherited)
- **the summary** (inherited)
- `m_episodes`: a list of episodes

To represent an episode, use the following structure:

```
struct TvEpisode
{
  const TvShow* m_show{};
  unsigned short m_numberOverall{};
```

```
  unsigned short m_season{};
  unsigned short m_numberInSeason{};
  std::string m_airDate{};
  unsigned int m_length{};
  std::string m_title{};
  std::string m_summary{};
};
```

Private Members

- add any constructors that are necessary for your design

This class will not offer any public constructors.

Public Members

- `void display(std::ostream& out) const override`: override this function to print the information about a single book. Use the following implementation:

```
void TvShow::display(std::ostream& out) const
{
  if (g_settings.m_tableView)
  {
    out << "S | ";
    out << std::left << std::setfill('.');
    out << std::setw(50) << this->getTitle() << " | ";
    out << std::right << std::setfill(' ');
    out << std::setw(2) << this->m_episodes.size() << " | ";
    out << std::setw(4) << this->getYear() << " | ";
    out << std::left;
    if (g_settings.m_maxSummaryWidth > -1)
    {
      if (static_cast<short>(this->getSummary().size()) <=
g_settings.m_maxSummaryWidth)
        out << this->getSummary();
      else
        out << this->getSummary().substr(0, g_settings.m_maxSummaryWidth - 3) <<
"...";
    }
    else
      out << this->getSummary();
    out << std::endl;
  }
  else
  {
    size_t pos = 0;
    out << this->getTitle() << " [" << this->getYear() << "]\n";
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << "" << '\n';
    while (pos < this->getSummary().size())
```

```
    {
      out << "    " << this->getSummary().substr(pos, g_settings.m_maxSummaryWidth)
<< '\n';
      pos += g_settings.m_maxSummaryWidth;
    }
    for (auto& item : m_episodes)
    {
      out << std::setfill('0') << std::right;
      out << "    " << 'S' << std::setw(2) << item.m_season
        << 'E' << std::setw(2) << item.m_numberInSeason << ' ';
      if (item.m_title != "")
        out << item.m_title << '\n';
      else
        out << "Episode " << item.m_numberOverall << '\n';

      pos = 0;
      while (pos < item.m_summary.size())
      {
        out << "            " << item.m_summary.substr(pos,
g_settings.m_maxSummaryWidth - 8) << '\n';
        pos += g_settings.m_maxSummaryWidth - 8;
      }
    }
    out << std::setw(this->getTitle().size() + 7) << std::setfill('-') << ""
      << std::setfill(' ') << '\n';
  }
}
```

- `TvShow* createItem(const std::string& strShow)`: a **class function** that receives as parameter the representation of the TV Show as a string and builds a dynamically allocated object of type `TvShow` using the information from the string and returns it to the client. The parameter contains a single line of text extracted from the file `tvShows.csv`. The format of the line is as following:

```
ID,TITLE,YEAR,SUMMARY
```

This function should remove all spaces from the **beginning and end** of any token in the string.

If the parameter is an empty string or a string starting with `#`, raise an exception with the message `Not a valid show.` (lines in the file starting with `#` are considered comments and should be ignored).

If all the data is correctly loaded, create a dynamic object of type `TvShow` using your private constructors and return its address to the client.

When implementing this function, consider the following functions:

  - std::string::substr()
  - std::string::find()
  - std::string::erase()
  - std::stoi()
  - std::stod()

- `template<typename Collection_t> void addEpisode(Collection_t& col, const std::string& strEpisode)`: a **class function** that function builds an episode with the information from the string, searches in the collection for a TV show with the specified id, and adds it to the list of episodes of the found show. The string parameter contains a single line of text extracted from the file `episodes.csv`. The format of the line is as following:

```
ID,EPISODE_NUMBER,SEASON_NUMBER,EPISODE_IN_SEASON,AIR_DATE,LENGTH,TITLE,SUMMARY
```

This function should remove all spaces from the **beginning and end** of any token in the string.

If the parameter is an empty string or a string starting with `#`, raise an exception with the message `Not a valid episode.` (lines in the file starting with `#` are considered comments and should be ignored).

If an episode is missing the season info, it's considered to be in season 1.

Assumptions about the template parameter `Collection_t`:

  - has a member function called `size()` that returns the number of items in the collection
  - overloads `operator[]` that receives an unsigned integer as parameter and returns an object of type `MediaInfo*`

- `double getEpisodeAverageLength() const`: get the average length in seconds of an episode.

Important

This is a TASK that you must accomplish using STL Algorithms and no manual loops. The STL algorithms used for this task should not be used for another task. Your lambda expressions should not capture anything from the context by reference.

- `std::list<std::string> getLongEpisodes() const`: create a list with episode names that are at least 1 hour long.

Important

This is a TASK that you must accomplish using STL Algorithms and no manual loops. The STL algorithms used for this task should not be used for another task. Your lambda expressions should not capture anything from the context by reference.

Add any other private member that is required by your design!

`spellChecker` Module (functor)

Add a `SpellChecker` class to your project. This class holds two parallel arrays of strings, both of size 6 (statically allocated):

- `m_badWords`: an array with 6 misspelled words
- `m_goodWords`: an array with the correct spelling of those 6 words
- any other member required by your design to accomplish the goals described below.

Public Members

- `SpellChecker(const char* filename)`: receives the address of a C-style null-terminated string that holds the name of the file that contains the misspelled words. If the file exists, this constructor loads its contents. If the file is missing, this constructor throws an exception of type `const char*`, with the message `Bad file name!`. Each line in the file has the format `BAD_WORD GOOD_WORD`; the two fields can be separated by any number of spaces.
- `void operator()(std::string& text)`: this operator searches `text` and replaces any misspelled word with the correct version. It should also count how many times **each** misspelled word has been replaced.

When implementing this operator, consider the following functions:

  - std::string::find()
  - std::string::replace()

- `void showStatistics(std::ostream& out) const`: inserts into the parameter how many times each misspelled word has been replaced by the correct word using the current instance. The format of the output is:

```
BAD_WORD: CNT replacements<endl>
```

where `BAD_WORD` is to be printed on a field of size 15, aligned to the right.

You will have to design a method to remember how many times each bad word has been replaced.

`collection` Module

Add a `collection` module to your project. The purpose of this module is to manage a collection of media items. This class will take ownership of **ALL** `MediaItem` objects provided by the client and becomes responsible to manage their life. It is assumed, that all pointers received from the client store the address of **dynamically allocated** objects. When a new item is added to the collection, this class informs the client using a *callback function*.

This class provides two overloads of the subscripting operator (`operator[]`) to access a stored item.

Private Data

- the name of the collection;
- an STL container to store the media items
- a pointer to a function that returns `void` and receives two parameters of type `const Collection&` and `const MediaItem&` in that order.

This is the **observer** function (it *observes* an event): when an item has been added to the collection, the class `Collection` will call this function informing the client about the addition.

Public Members

- `Collection(const std::string& name)`: sets the name of the collection to the string referred to by the parameter and sets all other attributes to their default value
- this class doesn't support any copy/move operations; delete all of them.
- a destructor
- `const std::string& name() const`: a query that returns the name of the collection.
- `size_t size() const`: a query that returns the number of items in the collection.
- `void setObserver(void (*observer)(const Collection&, const MediaItem&))`: stores the address of the callback function (`observer`) into an attribute. This parameter is

a pointer to a function that returns `void` and accepts two parameters: a collection and an item that has just been added to the collection. This function is called when an item is added to the collection.

- `Collection& operator+=(MediaItem* item)`: adds the `item` to the collection, only if the collection doesn't contain an item with the same title. If `item` is already in the collection, this function deletes the parameter. If the item is added to the collection and an observer has been registered, this operator calls the observer function passing the current object (`*this`) and the new item as arguments.
- `MediaItem* operator[](size_t idx) const`: returns the item at index `idx`.
    - if the index is out of range, this operator throws an exception of type `std::out_of_range` with the message `Bad index [IDX]. Collection has [SIZE] items.`. Use operator `+` to concatenate strings.

When implementing this operator, consider the following:

    - std::to_string()
    - std::out_of_range
- `MediaItem* operator[](const std::string& title) const`: returns the address of the item with the title `title`. If no such item exists, this function returns `nullptr`.

Important
This is a `TASK` that you must accomplish using STL Algorithms and no manual loops. The STL algorithms used for this task should not be used for another task. Your lambda expressions should not capture anything from the context by reference.

- `void removeQuotes()`: for all the items stored in this collection, remove the quotation marks from the beginning/end of the title and summary.

Important
This is a `TASK` that you must accomplish using STL Algorithms and no manual loops. The STL algorithms used for this task should not be used for another task. Your lambda expressions should not capture anything from the context by reference.

- `void sort(const std::string& field)`: sort in ascending order the collection of items based on the field specified as parameter.

Important
This is a `TASK` that you must accomplish using STL Algorithms and no manual loops. The STL algorithms used for this task should not be used for another task. Your lambda expressions should not capture anything from the context by reference.

The results of this procedure will be different between various compilers. Check the provided sample output for your compiler.

FREE Helpers

- overload the insertion operator to insert the content of a `Collection` object into an **ostream** object. Iterate over all elements in the collection and insert each one into the `ostream` object (do not add newlines).

Important

The class `Collection` should have no knowledge of the custom types you have defined: `Book`, `Movie`, `TvShow`, `SpellChecker`, and `Settings`.

Sample Output

When the program is started with the command (the files are provided):

```
ws
```

the output should look like the one from the `sample_output.txt` file.