**Workshop 5: Functions and Error Handling**

In this workshop, you code a function object and a lambda expression, and handle exceptions.

You are to create a class template that manages a family of collection of objects of type T. This template's client can register a callback function (an observer) that will be called every time a new item is added successfully to a collection.

Your design works with collections of books and collections of movies, both loaded from files. The file information contains mistakes:

* You are to create a lambda expression that fixes the price information about a book
* You are to create a function object (functor) that fixes some spelling mistakes in the descriptions and titles for books or movies.

In case of exceptional circumstances, your design generates exceptions and subsequently handles them

* the functor loads the misspelled words from a file, but if that file is missing, generates an exception.
* iterating over the collection using indices, your design generates an exception if an index is invalid.

**Learning Outcomes**

Upon successful completion of this workshop, you will have demonstrated the abilities to:

* design and code a templated class that allocates and deallocates dynamic memory
* design and code a function object
* design and code a lambda expression
* code a member function that receives the address of a callback function to execute
* throw exceptions of different types
* distinguish exception types from one another

**Submission Policy**

The workshop is divided into two coding parts and one non-coding part:

* *Part 1*: worth 0% of the workshop's total mark, is optional and designed to assist you in completing the second part.
* *Part 2*: worth 100% of the workshop's total mark, is due on **Sunday at 23:59:59** of the week of your scheduled lab. Submissions of *Part 2* that do not contain the *reflection* are not considered valid submissions and are ignored.
* *reflection*: non-coding part, to be submitted together with *Part 2*. The reflection does not have marks associated to it, but can incur a **penalty of max 40% of the whole workshop's mark** if your professor deems it insufficient (you make your marks from the code, but you can lose some on the reflection).

The workshop should contain ***only work done by you this term*** or provided by your professor. Work done in another term (by you or somebody else), or work done by somebody else and not **clearly identified/cited** is considered plagiarism, in violation of the Academic Integrity Policy.

Every file that you submit must contain (as a comment) at the top **your name**, **your Seneca email**, **Seneca Student ID** and the **date** when you completed the work.

* If the file contains only your work, or work provided to you by your professor, add the following message as a comment at the top of the file:

I have done all the coding by myself and only copied the code that my professor provided to complete my workshops and assignments.

* If the file contains work that is not yours (you found it online or somebody provided it to you), **write exactly which parts of the assignment are given to you as help, who gave it to you, or which source you received it from.** By doing this you will only lose the mark for the parts you got help for, and the person helping you will be clear of any wrong doing.

**Compiling and Testing Your Program**

All your code should be compiled using this command on matrix:

/usr/local/gcc/10.2.0/bin/g++ -Wall -std=c++17 -g -o ws file1.cpp file2.cpp ...

* -Wall: compiler will report all warnings
* -std=c++17: the code will be compiled using the C++17 standard
* -g: the executable file will contain debugging symbols, allowing *valgrind* to create better reports
* -o ws: the compiled application will be named ws

After compiling and testing your code, run your program as following to check for possible memory leaks (assuming your executable name is ws):

valgrind ws

To check the output, use a program that can compare text files. Search online for such a program for your platform, or use *diff* available on matrix.

**Part 1 (0%)**

The first part of this workshop consists of two modules:

* w5 (partially supplied)
* Book

Enclose all your source code within the sdds namespace and include the necessary guards in each header file.

**Book Module**

This module holds information about a single book.

Design and code a class named Book 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):

* **author**
* **title**
* **the country of publication**
* **the year of publication**
* **the price of the book**
* **the description**: a short summary of the book

***Public Members***

* a default constructor
* const std::string& title() const: a query that returns the title of the book
* const std::string& country() const: a query that returns the publication country
* const size\_t& year() const: a query that returns the publication year
* double& price(): a function that returns the price **by reference**, allowing the client code to update the price
* Book(const std::string& strBook): a constructor that receives a reference to an unmodifiable string that contains information about the book; this constructor extracts the information about the book from the string by parsing it and stores the tokens in the object's attributes. The string always has the following format:
* AUTHOR,TITLE,COUNTRY,PRICE,YEAR,DESCRIPTION

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

When implementing the constructor, consider the following functions:

* + [std::string::substr()](https://en.cppreference.com/w/cpp/string/basic_string/substr)
  + [std::string::find()](https://en.cppreference.com/w/cpp/string/basic_string/find)
  + [std::string::erase()](https://en.cppreference.com/w/cpp/string/basic_string/erase)
  + [std::stoi()](https://en.cppreference.com/w/cpp/string/basic_string/stol)
  + [std::stod()](https://en.cppreference.com/w/cpp/string/basic_string/stof)

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

***Friend Helpers***

* overload the insertion operator to insert the content of a book object into an **ostream** object, in the following format:
* AUTHOR | TITLE | COUNTRY | YEAR | PRICE | DESCRIPTION
  + the **author** printed on a field of size 20;
  + the **title** printed on a field of size 22;
  + the **country** printed on a field of size 5;
  + the **year** printed on a field of size 4;
  + the **price** printed on a field of size 6, with 2 decimal digits;
  + see alignment in the sample output.

**w5 Module (partially supplied)**

This module has some missing statements. The missing parts are marked with TODO, which describes the code that you should add and where to add it. **Do not modify the existing code, only add the code that is missing!**

**Sample Output**

When the program is started with the command (the file book.txt is provided):

ws book.txt

the output should look like that is the sample\_output.txt file.

**Test Your Code**

To test the execution of your program, use the same data as shown in the output example above.

Upload your source code to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/10.2.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 345\_w5\_p1

and follow the instructions.

***This part represents a milestone in completing the workshop and is not marked!***

**Part 2 (100%)**

The second part of this workshop upgrades your solution to incorporate three additional modules:

* Movie
* SpellChecker
* Collection

**SpellChecker Module (functor)**

Add a SpellChecker module to your project. This module 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()](https://en.cppreference.com/w/cpp/string/basic_string/find)
  + [std::string::replace()](https://en.cppreference.com/w/cpp/string/basic_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.**

**Book Module**

Add a public templated function to your Book class:

* void fixSpelling(T& spellChecker): this function calls the overloaded operator() on the instance spellChecker, passing to it the book description.

***ASSUMPTION***: In this design, type T must have an overload of the operator() that accepts a string as a parameter.

**Since this is a templated function, it must be implemented in the header file!** The class itself is not templated; only the function is templated.

**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**
* **the year of release**
* **the description**

***Public Members***

* a default constructor
* const std::string& title() const: a query that returns the title of the movie
* Movie(const std::string& strMovie): receives the movie through a reference to a string. This constructor extracts the information about the movie from the string and stores the tokens in the attributes. The received string always has the following format:
* TITLE,YEAR,DESCRIPTION

This constructor removes all spaces from the **beginning and end** of any token in the string.

When implementing this constructor, consider the following functions:

* + [std::string::substr()](https://en.cppreference.com/w/cpp/string/basic_string/substr)
  + [std::string::find()](https://en.cppreference.com/w/cpp/string/basic_string/find)
  + [std::string::erase()](https://en.cppreference.com/w/cpp/string/basic_string/erase)
  + [std::stoi()](https://en.cppreference.com/w/cpp/string/basic_string/stol)
* void fixSpelling(T& spellChecker): a templated function. This function calls the overloaded operator() on instance spellChecker, passing to it the movie title and description.

***ASSUMPTION***: In this design, type T must have an overload of the operator() that accepts a string as a parameter.

**Since this is a templated function, it must be implemented in the header!** The class is not a templated class.

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

***Friend Helpers***

* overload the insertion operator to insert the content of a movie object into an **ostream** object, in the following format:
* TITLE | YEAR | DESCRIPTION
  + the **title** printed on a field of size 40;
  + the **year** printed on a field of size 4;

**Collection Module**

Add a Collection module to your project. The purpose of this module is to manage a collection items of template type T. Since this is templated class, it doesn't need a .cpp file.

The Collection class manages a **dynamically allocated** array of objects of type T, resizing it when a new item is added. 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;
* a dynamically allocated array of items T
* the size of the array
* a pointer to a function that returns void and receives two parameters of type const Collection<T>& and const T& in that order.

This is the **observer** function (it *observes* an event): when an item has been added to the collection, the class Collection<T> 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 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<T>&, const T&)): 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<T>& operator+=(const T& item): adds a copy of 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 does nothing. If the item is not yet in the collection, this function:
  + resizes the array of items to accommodate the new item
  + if an observer has been registered, this operator calls the observer function passing the current object (\*this) and the new item as arguments.
  + **ASSUMPTION**: type T has a member function called title() that returns the title of the item (std::string).
* T& 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()](https://en.cppreference.com/w/cpp/string/basic_string/to_string)
  + [std::out\_of\_range](https://en.cppreference.com/w/cpp/error/out_of_range)
* T\* operator[](const std::string& title) const: returns the address of the item with the title title (type T has a member function called title() that returns the title of the item). If no such item exists, this function returns nullptr.

***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 any of the custom types you have defined (Book, Movie, SpellChecker).**

**Sample Output**

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

w5.exe books.txt movies.txt missing\_file.txt words.txt

the output should look like the one from the sample\_output.txt file.

**Reflection**

Study your final solution, reread the related parts of the course notes, and make sure that you have understood the concepts covered by this workshop. **This should take no less than 30 minutes of your time and the result is suggested to be at least 150 words in length.**

Create a **text** file named reflect.txt that contains your detailed description of the topics that you have learned in completing this particular workshop and mention any issues that caused you difficulty and how you solved them. Include in your explanation—**but do not limit it to**—the following points:

* the difference between the implementations/utilizations of a functor and a lambda expression. When is appropriate to use each one?
* the constructor for SpellChecker generates an exception in certain conditions. How would you change your code to achieve the same result, but without exceptions (both on the constructor and on the client side)? Compare the two solutions.
* the classes Movie and Book contain almost identical logic in loading data from the file. How would you redesign the classes in order **not** to duplicate that logic?
* the classes Movie and Book are instantiated from the main() function using a custom constructor, but they also contain a default constructor. Is the default constructor necessary? Could you remove it? Justify your answer.

To avoid deductions, refer to code in your solution as examples to support your explanations.

**Submission**

To test and demonstrate execution of your program use the same data as shown in the output example above.

Upload the source code and the reflection file to your matrix account. Compile and run your code using the latest version of the g++ compiler (available at /usr/local/gcc/10.2.0/bin/g++) and make sure that everything works properly.

Then, run the following command from your account (replace profname.proflastname with your professor’s Seneca userid):

~profname.proflastname/submit 345\_w5\_p2

and follow the instructions.