Lab 5 – Exception Handling and Templates

Maggie Lyon, Nyla Spencer, Ember Roberts

In this lab the concepts explored include exception handling and templates. Exception handling allows the handing of errors and exceptions in our code. In task one, we created a base class. In task two, the user interactive prompts were created in main. Then, in task three, exceptions were handled by throwing and catching errors. Finally, in task four, a template is created from the EntertainmentCollection class, and VideoGame class with attribute Genre. The handling of errors built into a program encourages the constant testing of code, since it provides an error message instead of crashing the program. This gives the code more reliability and increases maintainability. Templets allow for the creation of functions and classes that can operate on different data types without having to rewrite code. Templates greatly increase the reusability of code. Templates also aid in efficiency and the flexibility of said code. Exception handling and templates are important features of cpp. Understanding them is essential for comprehending the language and writing high-quality code, which is necessary for anyone pursuing CS/engineering.

In task 1, we had to create both an add function, to add movies to the shelf, and a remove function, to remove movies from the shelf. This needed to be done in a last in first out manner, meaning the last object placed on the sheld must also be the first object to be removed from the shelf. For both we needed to keep track of the number of items already in the shelf, and to do so we maintained the index of the next place to put a movie. In the add function, the movie was passed as an argument to the function, and added to the array at this index. Then, the index was incremented to point to the next open place on the shelf. The remove function had the opposite functionality – first, the index was decremented to point to the last object added to the shelf, and then the data stored at that location was returned by the function.

Show below, Screenshot 1 demonstrated movies being added sequentially to the shelf, as well as the functionality to see how many movies are on the shelf. The second screenshot demonstrated the ‘remove movie’ functionality of the program.

```
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
1
Enter the name of the movie you would like to add:
cars
Enter the description of the movie you would like to add:
pixar
Enter the opening credits of the movie you would like to add: lightning mcqueen
Added cars to the shelf.
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
1
Enter the name of the movie you would like to add:
barbie
Enter the description of the movie you would like to add:
the movie
Enter the opening credits of the movie you would like to add: greta gerwig
Added barbie to the shelf.
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.

3
There are 2 movies on the shelf.
```

```
There are 2 movies on the shelf.
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
2
Movie being removed: barbie
Movie Description: the movie
Movie Opening Credits: greta gerwig
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
3
There are 1 movies on the shelf.
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
```

In task 3, we wrote code to throw and catch errors. These errors included trying to add a movie to a full shelf, or trying to remove a movie from an empty shelf. This was done so by creating two exception classes, fullShelf and emptyShelf. Exception handling is important because it allows us to create custom error messages so that we can catch specific errors as they arise, instead of ending up with an ambiguous error message or code that does not do what we intend it to do. Capturing these errors within their own exception classes has the added benefit of modularity and reuse, so custom exceptions can be reused whenever teh programmer decides to check for them. As seen in the screenshots below, when the user tries to add a movie to a full shelf, the exception fullShelf is thrown, and when they try to remove a movie from an empty shelf, the exception emptyShelf is thrown.

```
There are 5 movies on the shelf.
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
1
Enter the name of the movie you would like to add:
cars
Enter the description of the movie you would like to add:
racing
Enter the opening credits of the movie you would like to add: pixar
terminate called after throwing an instance of 'fullShelf'
  what():  std::exception
```

```
Choose an option to proceed:
1. Add a movie to the shelf.
2. Remove a movie from the shelf.
3. See how many movies are currently on the shelf.
4. Quit.
2
terminate called after throwing an instance of 'emptyShelf'
  what():  std::exception
```

In task 4, we created a template class to handle a similar problem. The following screenshots show the output of this task. Using a template has the advantage of the code being reusable for multiple data types passed to it, rather than just one.

```
Choose an option:
1. Add a video game to the collection
2. Remove a video game from the collection
3. Show number of video games in the collection
4. Exit
1
Enter the title of the video game:
mario
Enter the genre of the video game:
racing
terminate called after throwing an instance of 'fullCollection
'
  what():  std::exception
```

```
Choose an option:
1. Add a video game to the collection
2. Remove a video game from the collection
3. Show number of video games in the collection
4. Exit
3
Number of video games in the collection: 1
Choose an option:
1. Add a video game to the collection
2. Remove a video game from the collection
3. Show number of video games in the collection
4. Exit
2
The video game being removed is minecraft
Choose an option:
1. Add a video game to the collection
2. Remove a video game from the collection
3. Show number of video games in the collection
4. Exit
2
terminate called after throwing an instance of 'std::bad_alloc'
  what():  std::bad_alloc
```

https://replit.com/join/nmdfkayquo-nylarspencer

https://replit.com/@emberxroberts/Template-Task-4

Contributions:

Maggie – Task 2 and 3

Nyla – Task 1

Ember – Task 4