C.W:  
Task 1:

**Type-Safe Heterogeneous Container with Exception**

Create a class template TypeSafeContainer that can store any type but only allows retrieving an object of the correct type.

* If a user requests the wrong type, throw a BadTypeException.
* Internally, you must use std::any, std::variant, or custom type erasure (depending on difficulty you want).

**Hints**:

* Use templates for store<T> and get<T>.
* Ensure safe retrieval, with exception on mismatch.

Task 2:

Write a generic Matrix<T> template class supporting basic operations like addition and multiplication.

* If the dimensions do not match, throw a DimensionMismatchException.
* Also implement safe element access with bounds checking.

**Hints**:

* Template operator overloading (operator+, operator\*).
* Proper exception handling for invalid operations.

Task 3:  
Design a class template SafeArray<T> that dynamically allocates an array of type T.

* Overload the operator[] to allow element access.
* If an invalid index is accessed (negative or >= size), throw a custom exception ArrayIndexOutOfBounds.

**Task**:  
Implement SafeArray<T>, the exception class, and a small demo program that tests the exception.

H.W:

Task 4:

Write a C++ Program of templated class derived from another templated class. Consider two classes one base class with two variables and another derived class with two variables multiply the values in the classes and print their outputs respectively. (The template classes must inherit from each other)

Task 5:

Create a **stack class template** Stack<T> that supports push, pop, and top operations.  
If pop() or top() is called on an empty stack, throw an exception StackUnderflowException.

**Task**:

* Implement generic stack behavior.
* Write a main function to push/pop elements and handle exceptions gracefully.