Restrictions:

For **all** problems mentioned in this assignment, you may **only** use
`#include<iostream>`
and no other external header files. No further clarifications should be asked.

Problem 1: Simple unit test (10)

Writing tests is one of the most important tasks of a Software Developer. In this assignment, you learn how to write unit tests.

- Create directory called simple_unit_test
- Create a header file called *operations.h*
- Create a file called *main.cpp* with the following content.

```
#include "operations.h"

int main(){
    Operations operations;
    std::cout << operations.add(5, 3) << std::endl;
    std::cout << operations.multiply(5, 3) << std::endl;
    return 0;
}
```

- Implement *operations.h* such that *main.cpp* compiles and executes successfully and write 8 15 to the console.
- Use the following command `g++ -o main main.cpp -std=c++17`
- Now, create a file called *test.cpp*. On the top of the file write the following. The zip file should contain catch.hpp file.

```
#define CATCH_CONFIG_MAIN
#include "catch.hpp"
#include "operations.h"
```

Here, we are using a popular C++ unit testing framework called catch2 which can be used by simply including the header file. #define CATCH_CONFIG_MAIN instructs catch to create a main function required for running the tests. We also include "operations.h" because we will write unit tests for our code implemented in "operations.h".

- Here is a sample Test case that you should add to *test.cpp*

```
TEST_CASE("Checking Operations", "[operations]")
{
    Operations operations;
    SECTION("Addition")
    {
        REQUIRE(operations.add(5, 3) == 8);
        REQUIRE(operations.add(8, 3) == 11);
        REQUIRE(operations.add(5, 3) != -8);
    }
```

```
}
```

- Now compile and create executable for *test.cpp* using the command:
  `g++ -o test test.cpp -std=c++17`
- Running ./test should produce the following output:

```
===============================================================
All tests passed (3 assertions in 1 test case)
```

- **Problem** : **[10]**
  - Write **five** more assertions to check if operation.add has been implemented correctly.
  - Add a section for testing operation.multiply and write **ten** assertions.

Problem 2: Create List of Integers with basic operations (90)

Create directory called lists_in_cpp and implement the following functionality. You may assume that the list will contain only integers. You need the concepts of dynamic memory management, operator overloading etc. for this task.

- Create a file called *list.cpp* containing the following:

```
#include "list.h"

int main(){
    List list1, list2, list3;
    list1.append(1);
    list1.append(2);
    list2.append(30);
    list2.append(40);
    list3 = list1;
    list3.append(3);
    list1.append(4);
    List list4 = list3 + list2;
    std::cout << list1 << std::endl;
    std::cout << list3[0] << std::endl;
    std::cout << list4 << std::endl;
    std::cout << "Length of list -  " << list4.getSize() << std::endl;
    return 0;
}
```

- Executing list.cpp should produce the following:

```
[1, 2, 4]
3
[1, 2, 3, 30, 40]
Length of list -  5
```

- A sample list.h file is provided as part of the assignment. You may use it and implement the required functionalities. The following are some guidelines for the problem. **[70]**
    - You should not change list.cpp or ask for any other information from the user. In case you fail to implement some features, you may comment them out in list.cpp but should not delete them.
    - You should assume that the size of the list is not fixed and the list should get resized dynamically. You should follow proper memory management i.e., delete any unused data.
    - Here is a grade distribution:
        - Correctly implement the append method **[15]**
        - Correctly override = operator **[20]**
        - Correctly override [] operator **[10]**
        - Correctly override + operator **[10]**
        - Complete the code for overriding << **[10]**
        - Correctly implement getSize() **[5]**
- Writing tests: **[20]**
    - Create a file called test_list.cpp and write three test cases to check your implementation. A skeleton of the three required test cases is provided as part of the zip.
    - For the test case: "List Basic Operations" add any required code and for: **[12]**
        - "Append Elements" – Add five new assertions
        - "Access Elements with []" – Add five new assertions
        - "Out of Range Access Throws Exception" – Add five new assertions
    - For test case: "List Concatenation with + Operator" **[6]**
        - Implement the required code and add five assertions to check "Concatenate Two Lists"
    - For test case: "List Stream Output with <<" finish the rest of the code and add one more assertion. **[2]**
    - **Reference:**
        - REQUIRE_THROWS_AS expects a specific type of error.
    - Running your test_list.cpp should report all passed tests.