Data Structures Lab

FALL - 2024

LAB 01

FAST National University of

Computer and Emerging Sciences

Learning Outcomes

In this lab you are expected to learn the following:

- Templates
- Templates with unit testing

Unit Testing:

Google test is a framework for writing C++ unit tests.

- When using googletest, you start by writing assertions, which are statements that check whether a condition is true.
- **Tests** use assertions to verify the tested code's behavior. If a test crashes or has a failed assertion, then it fails; otherwise it succeeds.
- A **test suite** contains one or many tests. You should group your tests into test suites that reflect the structure of the tested code.
- A **test program** can contain multiple test suites.

This is how a test suite looks like,

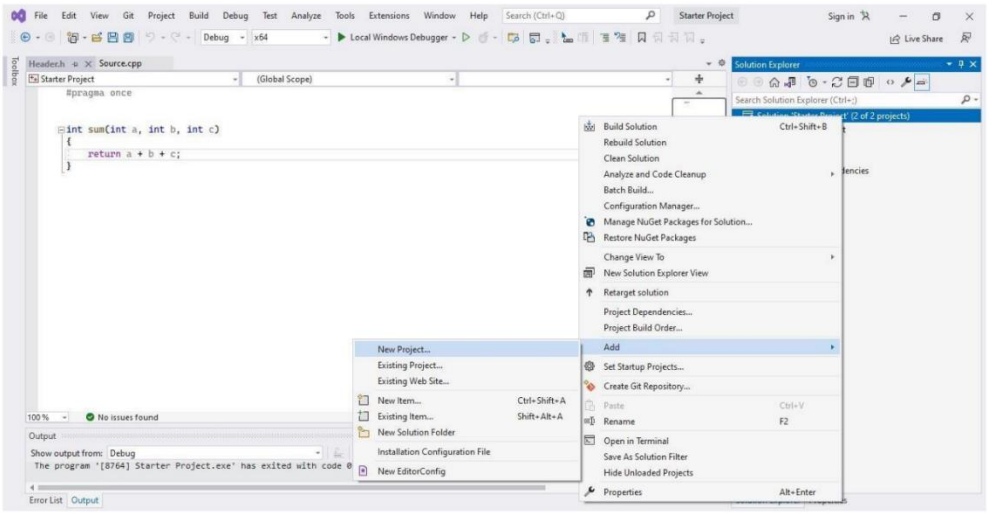
```
TEST_F(TestFixtureName, TestName) {
  ... test body ...
}
```

Example

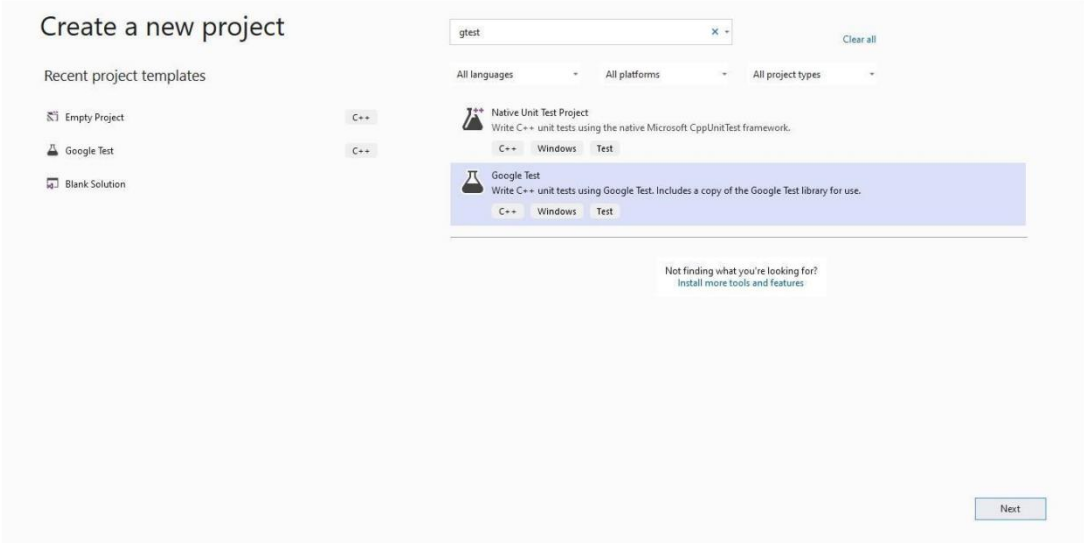
```
// Tests factorial of positive numbers.
TEST(FactorialTest, HandlesPositiveInput) {
  EXPECT_EQ(Factorial(1), 1);
  EXPECT_EQ(Factorial(2), 2);
  EXPECT_EQ(Factorial(3), 6);
  EXPECT_EQ(Factorial(8), 40320);
}
```

In Visual Studio 2017 and later, Google Test is integrated into the Visual Studio IDE as a default component of the Desktop Development with C++ workload.

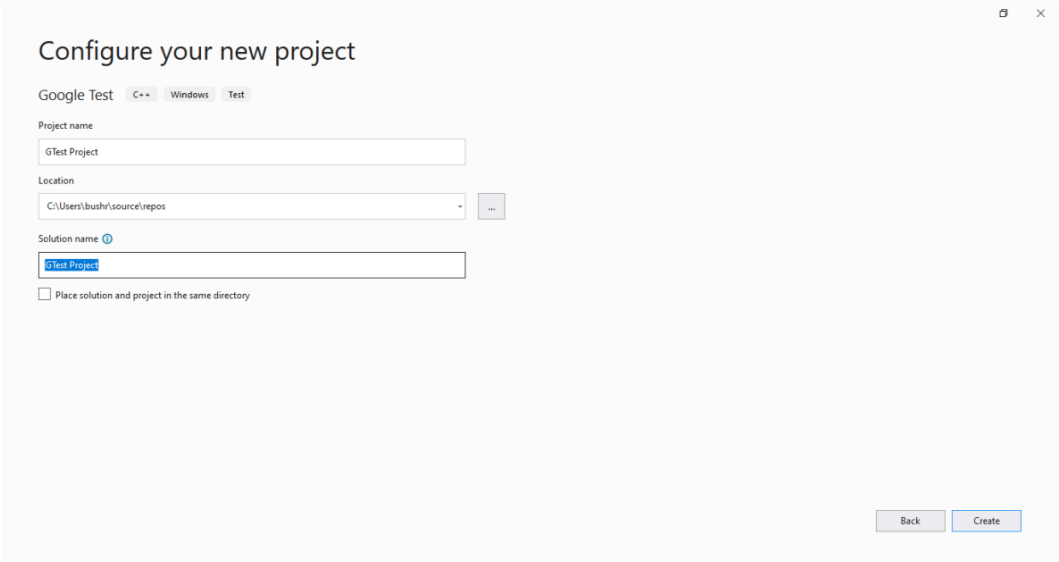
1. In Solution Explorer, right-click on the solution node and choose Add > New Project. Add a Google Test project in Visual Studio. Click on File and create a new project



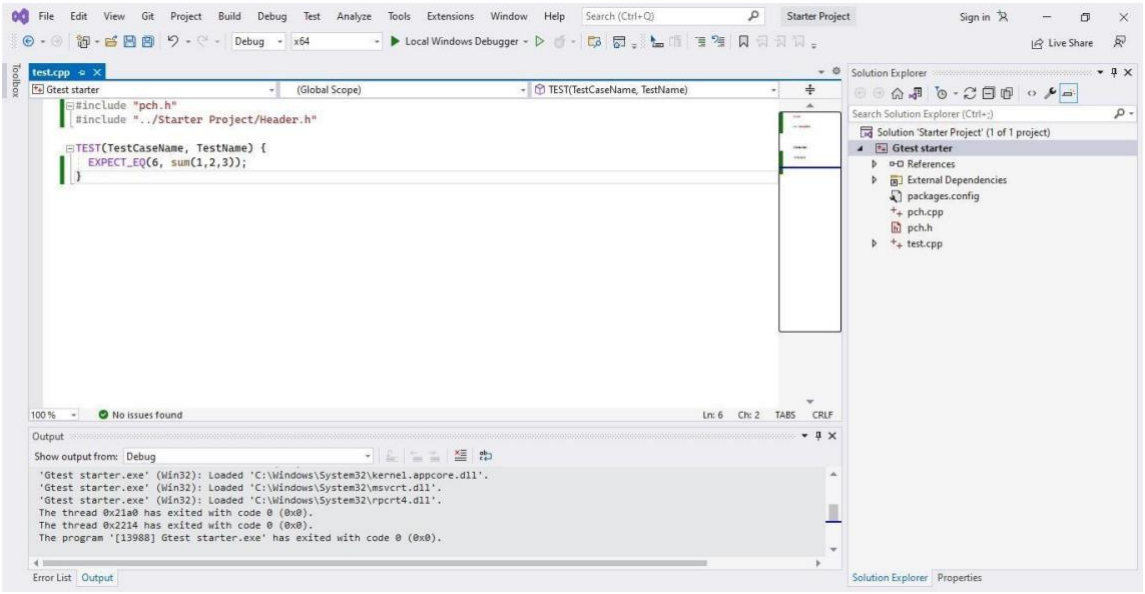
2. Search for google test and click next



3. Configure your project and click create



4. You are now ready to write and run Google Tests. Include the header file you want to test and write the test case



5. Compile and run the code using the Local Windows Debugger option from the quick access toolbar followed by the green play button

Templates

Templates are a powerful feature in C++ that allow you to write generic code capable of working with different data types. They enable you to define functions and classes without specifying the exact data type they will operate on. Templates facilitate code reusability and flexibility, as they eliminate the need to duplicate code for different data types.

Advantages of templates

1. **Code Reusability:** Templates enable you to write a single piece of code that works with various data types, reducing code duplication.

2. **Flexibility:** You can use templates to create versatile functions and classes that adapt t0 different data types as needed.

3. **Performance**: Templates allow the compiler to generate specialized code for each data type, resulting in efficient code execution.

Types of Templates:

There are two main types of templates in C++:

1. **Function Templates:** These are used to define generic functions that can operate on different data types.

   Syntax of function templates

```
template <typename T>
T Add(T a, T b) {
        return a + b;
}
```

   Example 1: Function Template for Addition

```
#include <iostream>

template <typename T>
T Add(T a, T b) {
        return a + b;
}

int main() {
    int sum_int = Add(5, 7);
    double sum_double = Add(3.14, 1.618);

    std::cout << "Sum (int): " << sum_int << std::endl;
    std::cout << "Sum (double): " << sum_double << std::endl;

    return 0;
}
```

2. **Class Templates:** These are used to define generic classes that can store and manipulate different data types.

Syntax of function templates

```
template <typename T>
class MyContainer {
     private:
     T elements[100];
     int size; public:
    //Constructor, methods, etc.
};
```

Example 2: Class Template for Container

```
#include <iostream>
template <typename T, int Size>
class MyContainer {
     private:
     T elements[Size];
     int count;

public:
    MyContainer() : count(0) {}

    void AddElement(T element) {
     if (count < Size) {
          elements[count] = element;
          count++;
        }
    }

    void PrintElements() {
     for (int i = 0; i < count; i++) {
               std::cout << elements[i] << " ";
        }
        std::cout << std::endl;
    }
};
```

```cpp
int main() {
    MyContainer<int, 5> intContainer;
    intContainer.AddElement(42);
    intContainer.AddElement(73);
    std::cout << "Elements (int): ";
    intContainer.PrintElements();

    MyContainer<std::string, 3> stringContainer;
    stringContainer.AddElement("Hello");
    stringContainer.AddElement("World");
    std::cout << "Elements (string): ";
    stringContainer.PrintElements();

    return 0;
}
```

Lab Tasks

Submission Instructions:

1. Create a separate .h and .cpp file for each class.
2. Now create a new folder with name *ROLLNO_SEC_LAB01* **e.g. i23XXXX_A_LAB01**
3. Move all of your files to this newly created directory and compress it into a **.zip file**.
4. Now you have to submit this zipped file on Google Classroom.
5. If you don't follow the above-mentioned submission instruction, you will be marked **zero.**
6. Plagiarism in the Lab Task will result in **zero** marks in the whole category.

Q 1. Let A and B be two sets: (Data of sets can be integer, double or string)

A = {10.43,4.3,5.61,6.90,11.57,12.11,3.8,2.4,9.5}

B = {11.01,12.34,16.5,3.8,8.1,2.4,9.11,12.11,6.75,10.43,20.2,2.1,4.3}

Write the following template functions:

a. **Union**

   Find the union of A and B. The union of A and B is the set that contains those elements that are either in A or in B, or in both

   Prototype: T* unionSets(T *set1, int size1, T *set2, int size2)

b. **Intersection**

   Find the intersection of A and B. The intersection of A and B is the set that contains those elements that are in both A and B.

   Prototype: T* intersectionSets(T *set1, int size1, T *set2, int size2)

c. **Disjoint**

   Return true if A and B are disjoint sets (sets having no common elements).

   Prototype: bool areDisjoint(T *set1, int size1, T *set2, int size2)

d. **Find Element**

   Return True if the Element exists in the set

   Prototype: bool elementExists(T *set, int size, T element)

Q2. You are provided with the dataset of university students. This datasets contains names, ages, course major and pending fee. Your task is to write a function that removes duplicates of any type

of dataset provided. Write a template function **RemoveDups()** that removes duplicate vales form the provided dataset.

Example :
Age_data = {21, 32, 22 , 33, 22}
Name data = {sara, mary, larry, sara}

The function should return {21,32,22,33} and {sara, mary,larry} when provided with the respective dataset.

Prototype: T* RemoveDups(T *, int)

Q3. Define a template class **Calculator** to perform the following operations. The calculator class consist of two member variables num1 and num2 that can be of any type.

1. Addition

   Prototype: T add()

2. Multiplication

   Prototype: T multiply()

3. Subtraction

   Prototype: T subtract()

4. Division

   Prototype: T divide()

5. X Power Y

   Prototype: T power()

6. Square

   Prototype: T square()

Q4 Define a template class **Container** for storing different type of values e.g. int, double, float etc. This container would keep same type of values/elements at a time. The **Container** class should consist of the following member variables:

- **T * values;** A pointer to set of values it contains. This member points to the dynamically allocated array (which you will be allocating in constructor).
- **capacity;** An integer that shows total capacity of the container
- **counter;** An integer counter variable which increments upon every insertion operation and decrements upon removal of any element; representing total number of elements currently present in the container.

The container should consist of following functions:

- **constructor** with capacity as parameter
- **isFull** which will return true if counter reaches capacity otherwise false
- **insert** which will receive a value of some type (int/double/float) and insert into the container if it's not already full

  Prototype: void insert(T)
- **search** which will return true if the container contains value passed as parameter

  Prototype: bool search(T)
- **remove** which will remove a value, if exists, from the container and shifts all subsequent values one index back.

  Prototype: void remove(T)
- **print** which will print all values contained in the container