Programming Practicum Report: Meeting #7

R. Ethan Halim

October 21st, 2024

1 Addition inside of a Function

The entire source file is hosted on a GitHub repository **here**.

1.1 Explanation

$$\text{add}(a, b) : (\mathbb{R}, \mathbb{R}) \to \mathbb{R} = a + b$$

This is the only function that the problem necessitates. It implements addition as a function, whereby the two operands of the arithmetic operation are represented as the arguments of a function, accepting real values through the `double` data type. Similarly, it returns a real number as the result of the opaque operation. The code below is equivalent to the mathematical expression above.

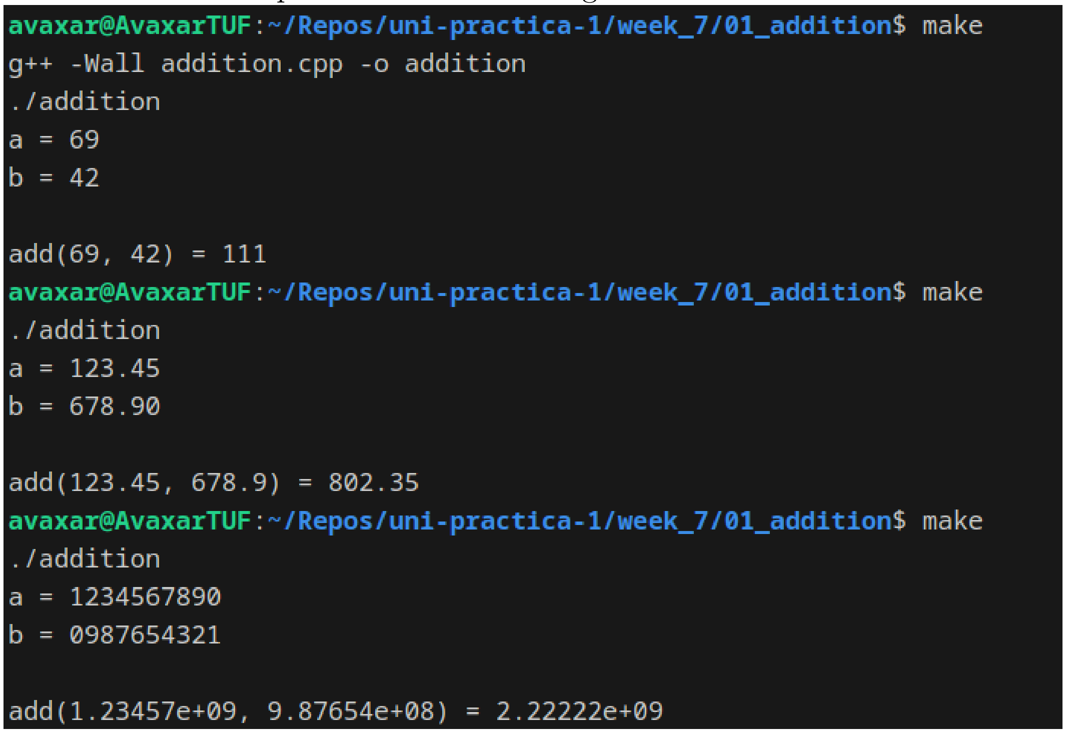
```
// Requirement: an addition function
double add(double a, double b) {
    return a + b;
}
```

The main function only provides the command-line interface to interact with the function.

```
int program(std::istream& cin, std::ostream& cout) {
    double a;
    cout << "a = ";
    cin >> a;

    double b;
    cout << "b = ";
    cin >> b;

    cout << "\nadd(" << a << ", " << b << ") = " << add(a, b)
    ↪  << '\n';
    return 0;
}
```

1.2 Manual Testing

Below is the compilation and the testing of the source code.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/01_addition$ make
g++ -Wall addition.cpp -o addition
./addition
a = 69
b = 42

add(69, 42) = 111
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/01_addition$ make
./addition
a = 123.45
b = 678.90

add(123.45, 678.9) = 802.35
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/01_addition$ make
./addition
a = 1234567890
b = 0987654321

add(1.23457e+09, 9.87654e+08) = 2.22222e+09
```

1.3 Test Cases

1.3.1 Tests

Below is copied directly from the `tests.txt` file.

```
%INPUT
90
94
%OUTPUT
a = b =
add(90, 94) = 184
%END

%INPUT
-12
-17
%OUTPUT
a = b =
add(-12, -17) = -29
%END

%INPUT
25
-15
%OUTPUT
a = b =
add(25, -15) = 10
%END

%INPUT
87
-30
%OUTPUT
a = b =
add(87, -30) = 57
%END

%INPUT
88
-34
%OUTPUT
a = b =
add(88, -34) = 54
%END
```

```
%INPUT
-52
-88
%OUTPUT
a = b =
add(-52, -88) = -140
%END

%INPUT
-23
-47
%OUTPUT
a = b =
add(-23, -47) = -70
%END

%INPUT
-76
-37
%OUTPUT
a = b =
add(-76, -37) = -113
%END

%INPUT
-9
-83
%OUTPUT
a = b =
add(-9, -83) = -92
%END

%INPUT
24
-12
%OUTPUT
a = b =
add(24, -12) = 12
%END

%INPUT
-47
-25
```

```
%OUTPUT
a = b =
add(-47, -25) = -72
%END

%INPUT
-80
-47
%OUTPUT
a = b =
add(-80, -47) = -127
%END

%INPUT
64
33
%OUTPUT
a = b =
add(64, 33) = 97
%END

%INPUT
46
-61
%OUTPUT
a = b =
add(46, -61) = -15
%END

%INPUT
98
24
%OUTPUT
a = b =
add(98, 24) = 122
%END

%INPUT
-70
54
%OUTPUT
a = b =
add(-70, 54) = -16
%END
```

```
%INPUT
96
-87
%OUTPUT
a = b =
add(96, -87) = 9
%END

%INPUT
18
34
%OUTPUT
a = b =
add(18, 34) = 52
%END

%INPUT
2
-48
%OUTPUT
a = b =
add(2, -48) = -46
%END

%INPUT
1
-24
%OUTPUT
a = b =
add(1, -24) = -23
%END

%INPUT
51
78
%OUTPUT
a = b =
add(51, 78) = 129
%END

%INPUT
8
9
```

```
%OUTPUT
a = b =
add(8, 9) = 17
%END

%INPUT
40
64
%OUTPUT
a = b =
add(40, 64) = 104
%END

%INPUT
90
-27
%OUTPUT
a = b =
add(90, -27) = 63
%END

%INPUT
2
35
%OUTPUT
a = b =
add(2, 35) = 37
%END

%INPUT
100
-15
%OUTPUT
a = b =
add(100, -15) = 85
%END

%INPUT
24
-33
%OUTPUT
a = b =
add(24, -33) = -9
%END
```

```
%INPUT
-85
4
%OUTPUT
a = b =
add(-85, 4) = -81
%END

%INPUT
-87
60
%OUTPUT
a = b =
add(-87, 60) = -27
%END

%INPUT
-50
-15
%OUTPUT
a = b =
add(-50, -15) = -65
%END

%INPUT
59
71
%OUTPUT
a = b =
add(59, 71) = 130
%END

%INPUT
-80
89
%OUTPUT
a = b =
add(-80, 89) = 9
%END

%INPUT
-3
35
```

```
%OUTPUT
a = b =
add(-3, 35) = 32
%END

%INPUT
-36
13
%OUTPUT
a = b =
add(-36, 13) = -23
%END

%INPUT
-24
60
%OUTPUT
a = b =
add(-24, 60) = 36
%END

%INPUT
32
7
%OUTPUT
a = b =
add(32, 7) = 39
%END

%INPUT
3
100
%OUTPUT
a = b =
add(3, 100) = 103
%END

%INPUT
63
87
%OUTPUT
a = b =
add(63, 87) = 150
%END
```

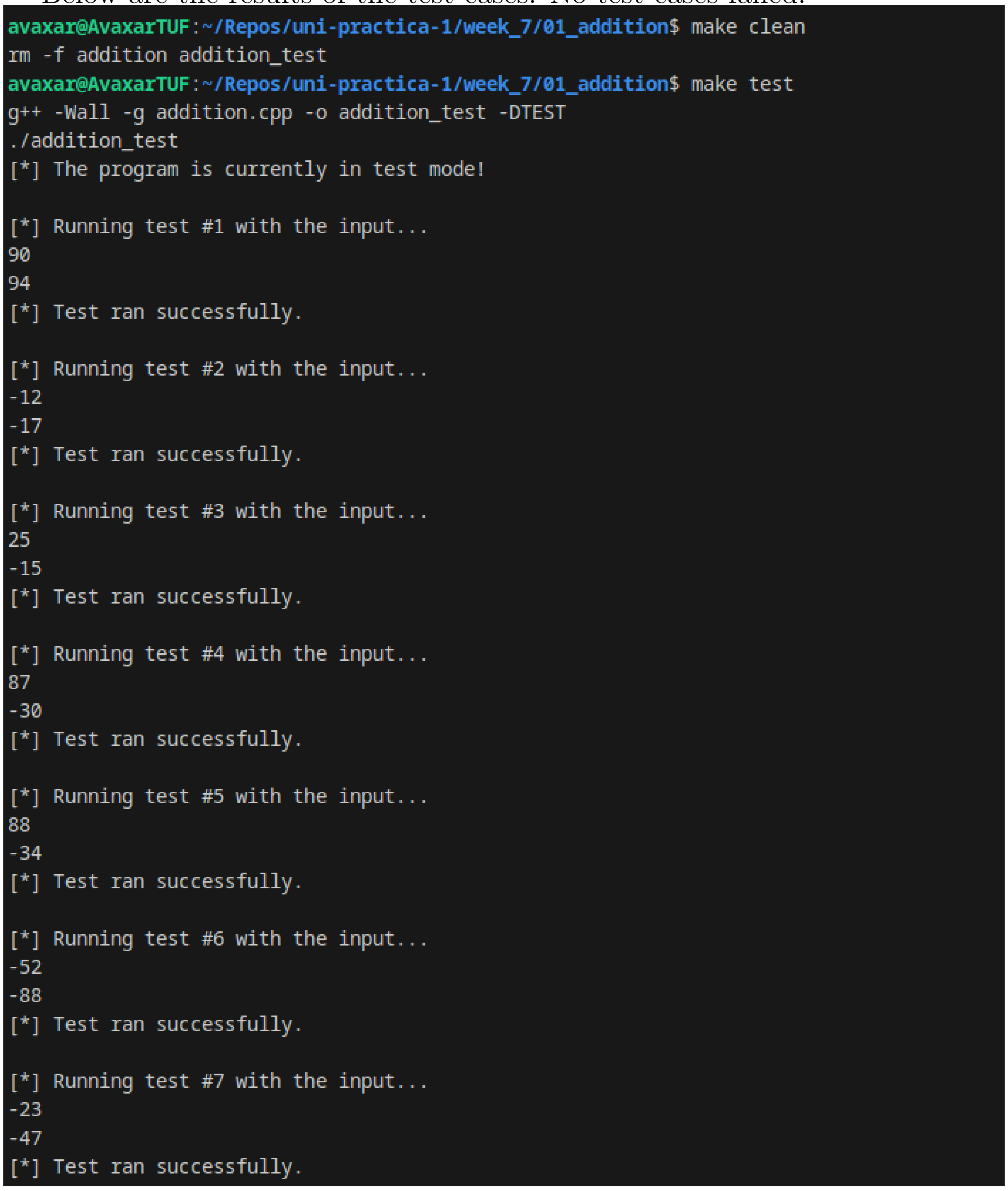
```
%INPUT
-46
0
%OUTPUT
a = b =
add(-46, 0) = -46
%END

%INPUT
-79
53
%OUTPUT
a = b =
add(-79, 53) = -26
%END

%INPUT
87
63
%OUTPUT
a = b =
add(87, 63) = 150
%END

%INPUT
52
31
%OUTPUT
a = b =
add(52, 31) = 83
%END

%INPUT
-63
-42
%OUTPUT
a = b =
add(-63, -42) = -105
%END

%INPUT
-95
-94
```

```
%OUTPUT
a = b =
add(-95, -94) = -189
%END

%INPUT
53
26
%OUTPUT
a = b =
add(53, 26) = 79
%END

%INPUT
-96
-83
%OUTPUT
a = b =
add(-96, -83) = -179
%END

%INPUT
63
91
%OUTPUT
a = b =
add(63, 91) = 154
%END

%INPUT
13
-39
%OUTPUT
a = b =
add(13, -39) = -26
%END

%INPUT
52
-23
%OUTPUT
a = b =
add(52, -23) = 29
%END
```

```
%INPUT
43
98
%OUTPUT
a = b =
add(43, 98) = 141
%END
```

1.3.2 Execution

Below are the results of the test cases. No test cases failed.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/01_addition$ make clean
rm -f addition addition_test
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/01_addition$ make test
g++ -Wall -g addition.cpp -o addition_test -DTEST
./addition_test
[*] The program is currently in test mode!

[*] Running test #1 with the input...
90
94
[*] Test ran successfully.

[*] Running test #2 with the input...
-12
-17
[*] Test ran successfully.

[*] Running test #3 with the input...
25
-15
[*] Test ran successfully.

[*] Running test #4 with the input...
87
-30
[*] Test ran successfully.

[*] Running test #5 with the input...
88
-34
[*] Test ran successfully.

[*] Running test #6 with the input...
-52
-88
[*] Test ran successfully.

[*] Running test #7 with the input...
-23
-47
[*] Test ran successfully.
```

2 Recursive Function to Compute Factorials

The entire source file is hosted on a GitHub repository **here**.

2.1 Explanation

Mathematically, the factorial of a given natural number (an integer starting from zero) is defined as below using recursion. The end of the recursion is defined as when it hits the base case, which—in this case—is when 0! occurs.

$$\text{factorial}(n) : \mathbb{N} \to \mathbb{N} = n! = \begin{cases} 1 & \text{if } n = 0 \\ n \cdot (n-1)! & \text{if } n \neq 0 \end{cases}$$

Programmatically, it is implemented as below. In C/C++, natural numbers can be exclusively represented using unsigned integers, so the program uses the highest storing unsigned integer type—`uint64_t`. The scenario of the base case 0! is checked in the initial if-statement, and an early return will be invoked following such scenario. The general case, $n \cdot (n-1)!$, is placed following the if-statement as a return call utilizing recursion by calling its own function.

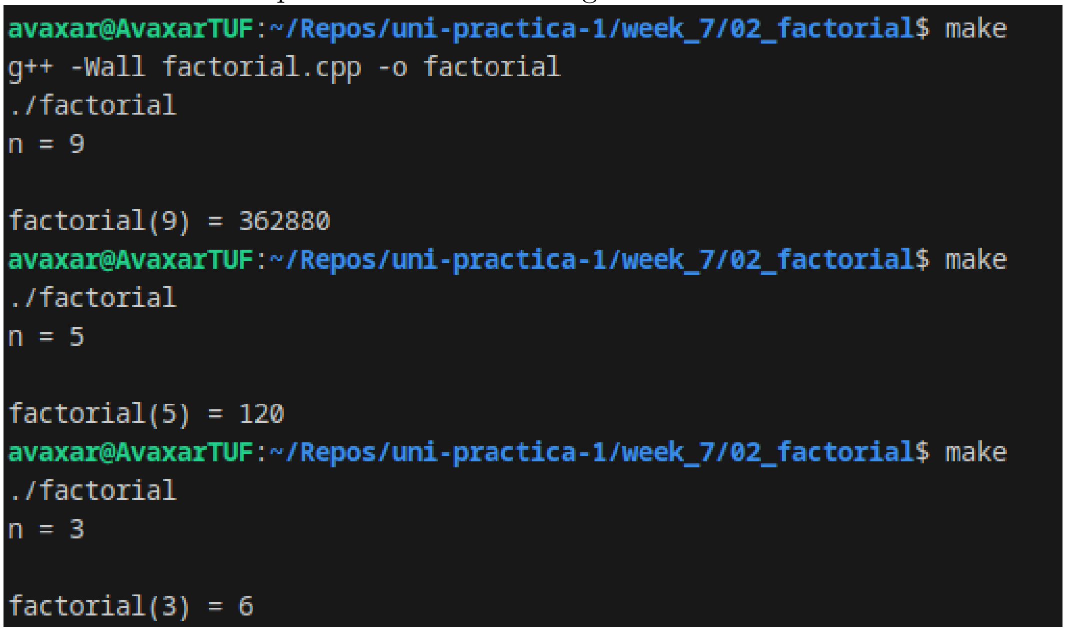
```cpp
uint64_t factorial(uint64_t n) {
    // Base case: 0! = 1
    if (n == 0) {
        return 1;
    }

    return n * factorial(n - 1);
}
```

The main function only provides the command-line interface to interact with the function.

```cpp
int program(std::istream& cin, std::ostream& cout) {
    uint64_t n;
    cout << "n = ";
    cin >> n;

    cout << "\nfactorial(" << n << ") = " << factorial(n) <<
     ↪  '\n';
    return 0;
}
```

2.2 Manual Testing

Below is the compilation and the testing of the source code.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/02_factorial$ make
g++ -Wall factorial.cpp -o factorial
./factorial
n = 9

factorial(9) = 362880
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/02_factorial$ make
./factorial
n = 5

factorial(5) = 120
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/02_factorial$ make
./factorial
n = 3

factorial(3) = 6
```

2.3 Test Cases

2.3.1 Tests

Below is copied directly from the `tests.txt` file.

```
%INPUT
-5
%OUTPUT
n =
factorial(-5) = 1
%END

%INPUT
-4
%OUTPUT
n =
factorial(-4) = 1
%END

%INPUT
-3
%OUTPUT
n =
factorial(-3) = 1
```

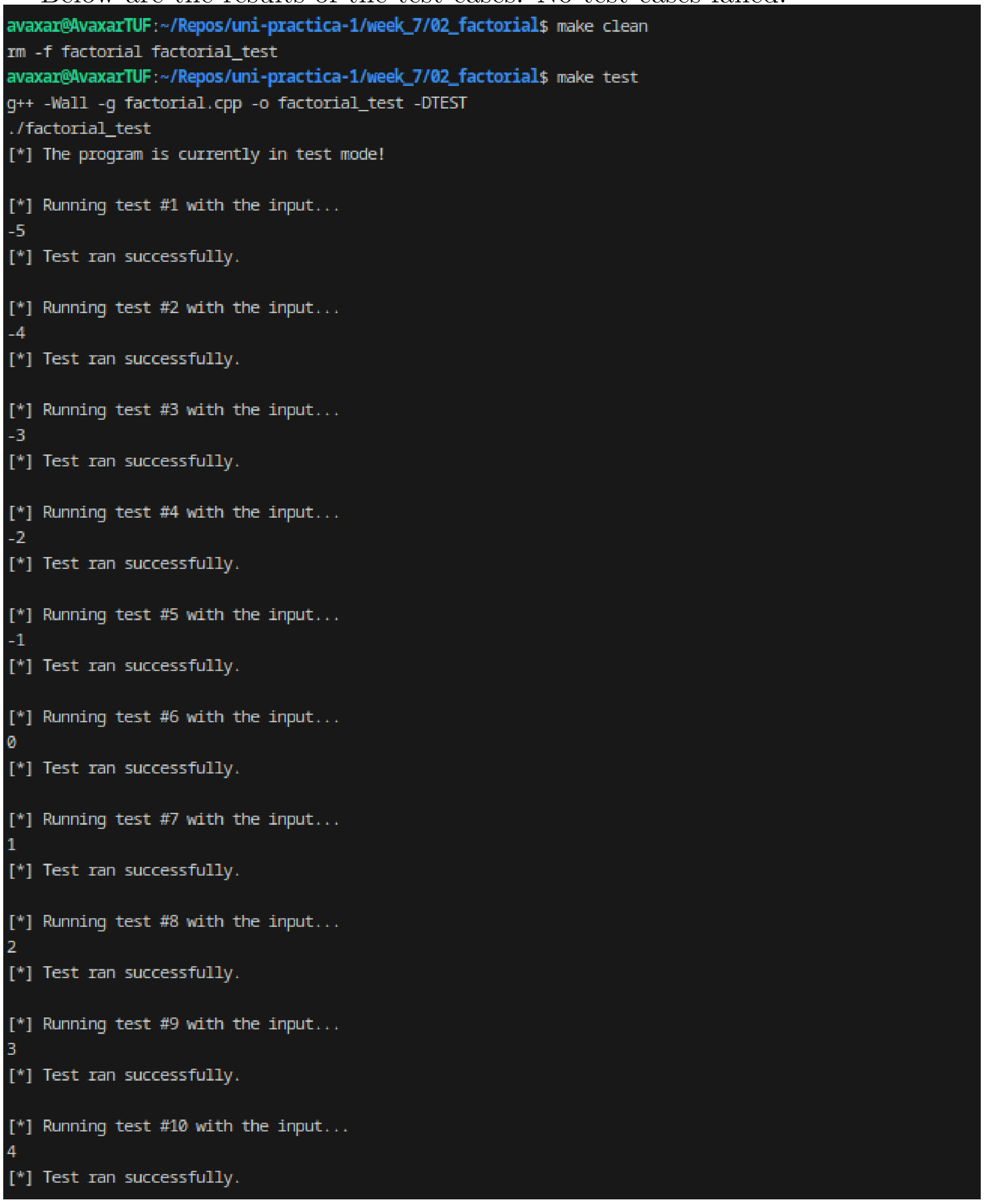
```
%END

%INPUT
-2
%OUTPUT
n =
factorial(-2) = 1
%END

%INPUT
-1
%OUTPUT
n =
factorial(-1) = 1
%END

%INPUT
0
%OUTPUT
n =
factorial(0) = 1
%END

%INPUT
1
%OUTPUT
n =
factorial(1) = 1
%END

%INPUT
2
%OUTPUT
n =
factorial(2) = 2
%END

%INPUT
3
%OUTPUT
n =
factorial(3) = 6
%END
```

```
%INPUT
4
%OUTPUT
n =
factorial(4) = 24
%END

%INPUT
5
%OUTPUT
n =
factorial(5) = 120
%END

%INPUT
6
%OUTPUT
n =
factorial(6) = 720
%END

%INPUT
7
%OUTPUT
n =
factorial(7) = 5040
%END

%INPUT
8
%OUTPUT
n =
factorial(8) = 40320
%END

%INPUT
9
%OUTPUT
n =
factorial(9) = 362880
%END

%INPUT
10
```

```
%OUTPUT
n =
factorial(10) = 3628800
%END

%INPUT
11
%OUTPUT
n =
factorial(11) = 39916800
%END

%INPUT
12
%OUTPUT
n =
factorial(12) = 479001600
%END

%INPUT
13
%OUTPUT
n =
factorial(13) = 6227020800
%END

%INPUT
14
%OUTPUT
n =
factorial(14) = 87178291200
%END
```

2.3.2 Execution

Below are the results of the test cases. No test cases failed.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/02_factorial$ make clean
rm -f factorial factorial_test
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/02_factorial$ make test
g++ -Wall -g factorial.cpp -o factorial_test -DTEST
./factorial_test
[*] The program is currently in test mode!

[*] Running test #1 with the input...
-5
[*] Test ran successfully.

[*] Running test #2 with the input...
-4
[*] Test ran successfully.

[*] Running test #3 with the input...
-3
[*] Test ran successfully.

[*] Running test #4 with the input...
-2
[*] Test ran successfully.

[*] Running test #5 with the input...
-1
[*] Test ran successfully.

[*] Running test #6 with the input...
0
[*] Test ran successfully.

[*] Running test #7 with the input...
1
[*] Test ran successfully.

[*] Running test #8 with the input...
2
[*] Test ran successfully.

[*] Running test #9 with the input...
3
[*] Test ran successfully.

[*] Running test #10 with the input...
4
[*] Test ran successfully.
```

3 Recursive Function to Compute GCDs

The entire source file is hosted on a GitHub repository **here**.

3.1 Explanation

Mathematically, the greatest common divisor of two given natural numbers (integers starting from zero) can be computed using the Euclidean algorithm (alternatively called "Euclid's algorithm"), which is defined below. The algorithm utilizes recursion, wherein the final base case is when $b = 0$.

$$\gcd(a, b) : (\mathbb{N}, \mathbb{N}) \to \mathbb{N} = \begin{cases} a & \text{if } b = 0 \\ \gcd(b, a \bmod b) & \text{if } b \neq 0 \end{cases}$$

Programmatically, it is implemented as below. In C/C++, natural numbers can be exclusively represented using unsigned integers, so the program uses the highest storing unsigned integer type—`uint64_t`. Similar to the previous problem, the base case is checked for by an if-statement, which would perform an early return, and the general case follows it.

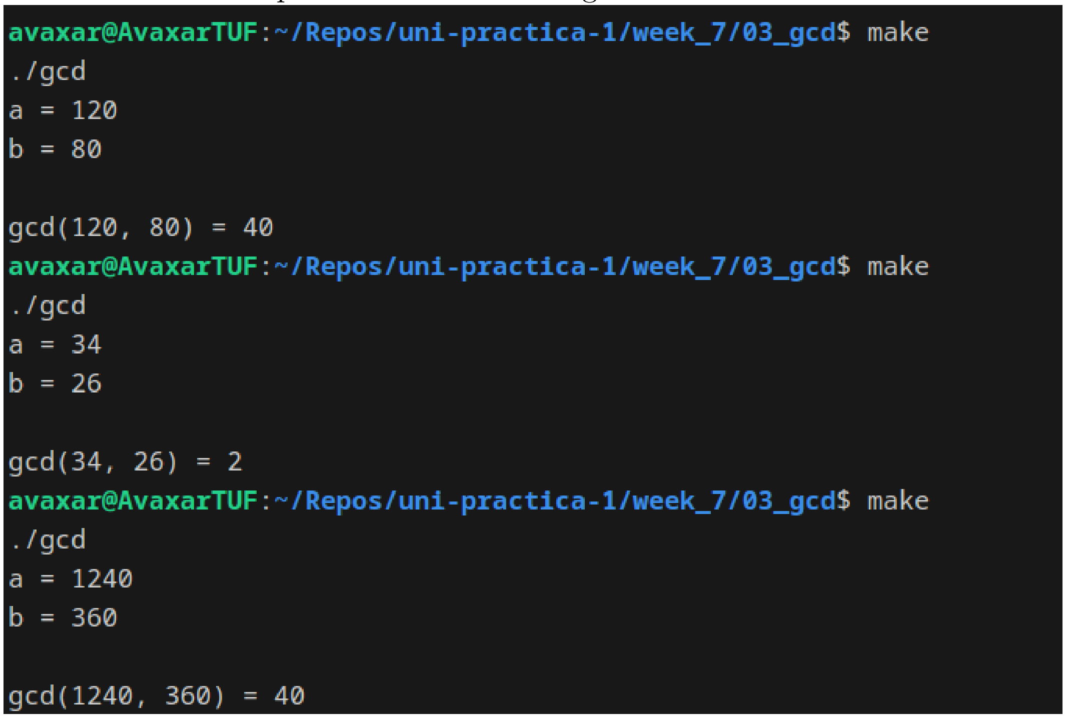
```cpp
// Implements the Euclidean algorithm
uint64_t gcd(uint64_t a, uint64_t b) {
    // Base case: gcd(a, 0) = a
    if (b == 0) {
        return a;
    }

    return gcd(b, a % b);
}
```

The main function only provides the command-line interface to interact with the function.

```cpp
int program(std::istream& cin, std::ostream& cout) {
    uint64_t a;
    cout << "a = ";
    cin >> a;

    uint64_t b;
    cout << "b = ";
    cin >> b;

    cout << "\ngcd(" << a << ", " << b << ") = " << gcd(a, b)
    ↪  << '\n';
    return 0;
}
```

3.2 Manual Testing

Below is the compilation and the testing of the source code.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/03_gcd$ make
./gcd
a = 120
b = 80

gcd(120, 80) = 40
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/03_gcd$ make
./gcd
a = 34
b = 26

gcd(34, 26) = 2
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/03_gcd$ make
./gcd
a = 1240
b = 360

gcd(1240, 360) = 40
```

3.3 Test Cases

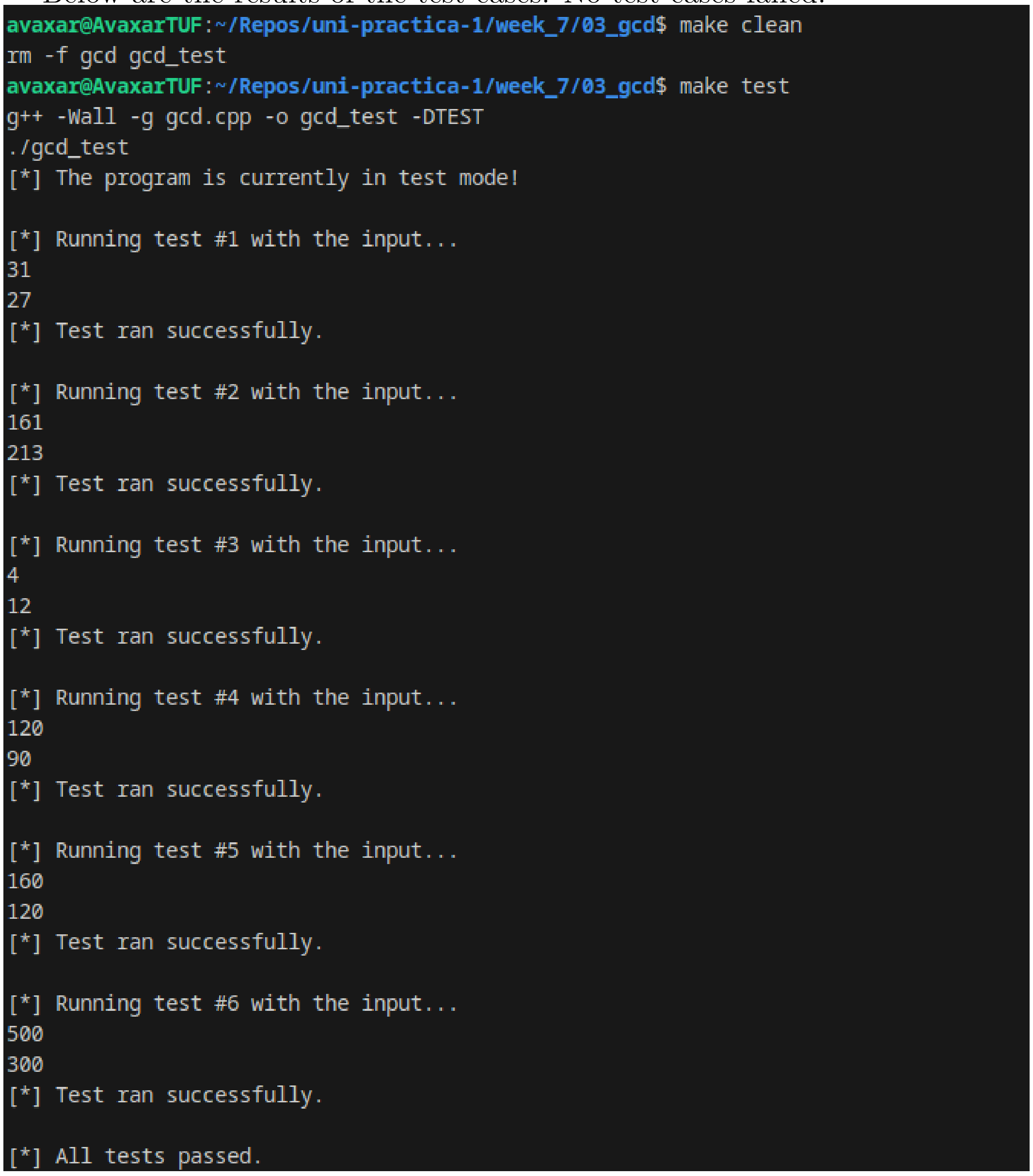
3.3.1 Tests

Below is copied directly from the `tests.txt` file.

```
%INPUT
31
27
%OUTPUT
a = b =
gcd(31, 27) = 1
%END

%INPUT
161
213
%OUTPUT
a = b =
gcd(161, 213) = 1
%END

%INPUT
4
12
%OUTPUT
a = b =
gcd(4, 12) = 4
%END


%INPUT
120
90
%OUTPUT
a = b =
gcd(120, 90) = 30
%END

%INPUT
160
120
%OUTPUT
a = b =
gcd(160, 120) = 40
%END

%INPUT
500
300
%OUTPUT
a = b =
gcd(500, 300) = 100
%END
```

3.3.2 Execution

Below are the results of the test cases. No test cases failed.



```
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/03_gcd$ make clean
rm -f gcd gcd_test
avaxar@AvaxarTUF:~/Repos/uni-practica-1/week_7/03_gcd$ make test
g++ -Wall -g gcd.cpp -o gcd_test -DTEST
./gcd_test
[*] The program is currently in test mode!

[*] Running test #1 with the input...
31
27
[*] Test ran successfully.

[*] Running test #2 with the input...
161
213
[*] Test ran successfully.

[*] Running test #3 with the input...
4
12
[*] Test ran successfully.

[*] Running test #4 with the input...
120
90
[*] Test ran successfully.

[*] Running test #5 with the input...
160
120
[*] Test ran successfully.

[*] Running test #6 with the input...
500
300
[*] Test ran successfully.

[*] All tests passed.
```