

Exercise 3

String Manipulations

Overview

- This exercise is to be conducted by adopting a **Pair Programming**.
[What is pair programming?](https://youtu.be/oBraLLybGDA) (<https://youtu.be/oBraLLybGDA>)
- You and your partner will be coding collaboratively using VS Code

Plagiarism Warning

You may discuss with others and refer to any resources. However, any kind of plagiarism will lead to your submission being dismissed. No appeal will be entertained at all.

Late Submission and Penalties

- The submission must be done via eLearning. Other than that (such as telegram, email, google drive, etc.), it will not be entertained at all.
- Programs that CANNOT COMPILE will get a 50% penalty.
- Programs that are submitted late will get a 10% penalty for every 10-minutes late.

Problem

In this exercise, you will be writing a C++ program that defines a custom string class. You will write the program into separate files.

You are provided with a starter code containing a class named `CustomString` consisting of an attribute of type `string`, named `data`, several methods, and operators. The class declaration has already been given fully in the specification file, `custom_string.hpp`. Your job in this exercise, is to complete the class implementation.

The main file, `main.cpp` has also been given fully. The purpose of the main file is to perform automatic testing on the tasks you have accomplished. When you run the program with the original codebase, you should notice that all tests FAIL (as shown in Figure 1). Your job is to turn the tests to "PASS" (like in Figure 2) by accomplishing the tasks in this exercise.

```
-----
Testing Task 1: Mutator methods
FAIL    1(a). Testing pushFront(). It should add the string in front.
FAIL    1(b). Testing pushBack(). It should add string at the back
FAIL    1(c)-i. Testing pop(). It should return the extracted string.
FAIL    1(c)-ii. Testing pop(). The extracted string should be erased from the attribute accordingly.
FAIL    1(d)-i. Testing popFront(). It should return the extracted string.
FAIL    1(d)-ii. Testing popFront(). The extracted string should be erased from the attribute accordingly.
FAIL    1(e)-i. Testing popBack(). It should return the extracted string.
FAIL    1(e)-ii. Testing popBack(). The extracted string should be erased from the attribute accordingly.
-----

Testing Task 2: Overloaded operators
FAIL    2(a). Testing operator!. It should return a CustomString with reversed data.
FAIL    2(b). Testing operator*. It should return a CustomString with repeated string.
-----

Testing Task 3: Conversion methods
FAIL    3(a). Testing toDouble(). It should return the corresponding value of type double
FAIL    3(b). Testing toUpper(). It should return a CustomString with upper case string

Testing Summary:
Number of Test: 12
Number of Pass: 0
Number of Fail: 12
Pass Rate      : 0%
```

Figure 1: Run with the original starter code

```

-----
Testing Task 1: Mutator methods
PASS    1(a). Testing pushFront(). It should add the string in front.
PASS    1(b). Testing pushBack(). It should add string at the back
PASS    1(c)-i. Testing pop(). It should return the extracted string.
PASS    1(c)-ii. Testing pop(). The extracted string should be erased from the attribute accordingly.
PASS    1(d)-i. Testing popFront(). It should return the extracted string.
PASS    1(d)-ii. Testing popFront(). The extracted string should be erased from the attribute accordingly.
PASS    1(e)-i. Testing popBack(). It should return the extracted string.
PASS    1(e)-ii. Testing popBack(). The extracted string should be erased from the attribute accordingly.
-----

Testing Task 2: Overloaded operators
PASS    2(a). Testing operator!. It should return a CustomString with reversed data.
PASS    2(b). Testing operator*. It should return a CustomString with repeated string.
-----

Testing Task 3: Conversion methods
PASS    3(a). Testing toDouble(). It should return the corresponding value of type double
PASS    3(b). Testing toUpper(). It should return a CustomString with upper case string

Testing Summary:
Number of Test: 12
Number of Pass: 12
Number of Fail: 0
Pass Rate      : 100%

```

Figure 2: Expected Final Result

Modify only the class implementation file, **custom_string.cpp** to do the following tasks.

1. Complete the implementations of the mutator methods:

a. **void pushFront(const string &s);**

This method merges the string **s** to the object's attribute **data** in front. For example, if we have an object `cs: CustomString cs("World");` then this operation: `cs.pushFront("Hello ");` will modify the **data** of **cs** to "Hello World".

b. **void pushBack(const string &s);**

This method merges the string **s** to the object's attribute **data** at the end. For example, if we have an object `cs: CustomString cs("C++ ");` then this operation: `cs.pushBack("Programming");` will modify the **data** of **cs** to "C++ Programming".

c. `string pop(int index, int count);`

This method extracts and returns a substring from the object's attribute **data** starting from the **index** and taking as many as **count** characters. The extracted substring will also be erased from the object's data. For example, if we have an object `str`:

```
CustomString str("ABCDEF");
```

```
then this operation: string result = str.pop(2,3);
```

will give the `result = "CDE"` and leaving the `str`'s data to `"ABF"`.

d. `string popFront(int count);`

This method extracts and returns a substring from the object's attribute **data** starting from the beginning and taking as many as **count** characters. The extracted substring will also be erased from the object's data. For example, if we have an object `str`:

```
CustomString str("ABCDEF");
```

```
then this operation: string result = str.popFront(3);
```

will give the `result = "ABC"` and leaving the `str`'s data to `"DEF"`.

You should implement this method by making use of the method `pop()` from 1(c).

e. `string popBack(int count);`

This method extracts and returns a substring from the object's attribute **data** at the end by taking as many as **count** characters. The extracted substring will also be erased from the object's data. For example, if we have an object `str`:

```
CustomString str("ABCDEF");
```

```
then this operation: string result = str.popBack(2);
```

will give the `result = "EF"` and leaving the `str`'s data to `"ABCD"`.

You should also implement this method by making use of the method `pop()` from 1(c).

2. Complete the implementations of the overloaded operators:

a. CustomString operator!() const;

This operator returns a new CustomString object with a reversed data. This method does not change at all the original data of the current object. For example, if we have an object cs: CustomString cs("abcdef"); then this operation: CustomString result = !cs; will give the result's data = "fedcba" and the cs's data remains as "abcdef".

b. CustomString operator*(int count) const;

This operator returns a new CustomString object with repeated string from the data of the current object. This method does not change at all the original data of the current object. For example, if we have an object cs: CustomString cs("XYZ "); then this operation: CustomString result = cs * 4; will give the result's data = "XYZ XYZ XYZ XYZ " and the cs's data remains as "XYZ ".

3. Complete the implementations of the conversion methods:

a. double toDouble() const;

This method returns the corresponding value the string in double type. For example, if we have an object str: CustomString str("99.50"); then this operation: double number = str.toDouble(); will give the number = 99.50.

b. CustomString toUpper() const;

This method returns a new CustomString object with capitalized string from the data of the current object. This method does not change at all the original data of the current object. For example, if we have an object str: CustomString str("One Plus Two"); then this operation: CustomString result = str.toUpper(); will give the result's data = "ONE PLUS TWO" and the str's data remains as "One Plus Two".

Assessment

This exercise carries **3%** weightage for the final grade of this course. The breakdown weightage is as follows (out of 100 points):

Criteria	Points
The code	
1. Task 1 – Implementations of the mutator methods:	
a. pushFront()	10
b. pushBack()	10
c. pop()	10
d. popFront()	10
e. popBack()	10
2. Task 2 – Implementations of the overloaded operators:	
a. operator !	15
b. operator *	15
3. Task 3 – Implementations of the conversion methods:	
a. toDouble()	10
b. toUpper()	10

Submission

- Deadline: **Saturday, 10 Dec 2022, 11:59 PM**
- Only one member from each pair needs to do the submission.
- Submission must be done on eLearning. Any other means such as email, telegram, google drive will not be accepted at all.
- Submit only the implementation file for the class, i.e., **custom_string.cpp**.