# String class

Implement a class for representation a character string in C++.

You should not use the STL library containers and algorithms.

Extend python interpreter functionality with implemented class (you can use Boost, SWIG, SIP, Pybind, CXX or whatever you want).

The basic class must follow next requirements:

Class MyString // Basic functionality

{

Constructor() // default constructor

Constructor(MyString) //copy constructor

Constructor(“char string”) // char array constructor

Constructor({‘s’,’t’,’r’,’i’,’n’,‘g’})// initializer list constructor

Constructor(std::string) // std::string constructor

Constructor(“char string”, count) // init class with count characters of “char string”,Constructor(“hello”, 4) -> “hell”

Constructor( count, char) // init class with count of characters

Constructor(4, ‘a’) -> “aaa”

Destructor()

Operator +(Mystring) // concatenate with Mystring

Operator +(“char string”) // concatenate with char array

Operator +(std::string) // concatenate with std::string

Operator +=(“char string”) // assignment concatenate with char array

Operator +=(std::string) // assignment concatenate with std::string

Operator=(“char string”) // char string assignment

Operator=(std::string) // std::string assignment

Operator=(‘char’) // char assignment

Opearator[](int) // index operator, Mystring(“012345”)[5] ->‘5’

Operator>(), Operator<(), Operator>=(), Operator<=(), \

Operator!=(), Operator==() // lexicographically comparing

c\_str() // return a pointer to null-terminated character array

data() // return a pointer to array data that not required to be null-terminated

length() // same as size

size() // return the number of char elements in string

empty() // true if string is empty

capacity() // return the current amount of allocated memory for array

shrink\_to\_fit() // reduce the capacity to size

clear() // remove all char element in string

Operator<<(std::basic\_ostream), Operator>>(std::basic\_istream) // like operator<<,>>(std::basic\_string)

**insert(index, count, char) // insert count of char in index position**

s = “aaaaa”

s.insert(0,1,’!’) -> “!aaaaa”

s.insert(3,2,’@’) -> “!aa@@aaa”

**insert(index, “string”) // insert null-terminated char string at index position**

s = “aaaaa”

s.insert(1,”@@@@@”) -> “a@@@@@aaaa”

**insert(index, “string”, count) // insert count of null-terminated char string at index position**

s = “aaaaa”

s.insert(1,”@@@@@”, 2) -> “a@@aaaa”

**insert(index, std::string) // insert std::string at index position**

s = “aaaaa”

string = “@@@@@”

s.insert(1, std::string(“@@@@@”)) -> “a@@@@@aaaa”

**insert(index, std::string, count) // insert count of std::string at index position**

s = “aaaaa”

s.insert(1, std::string(“@@@@@”)), 2) -> “a@@aaaa”

**erase(index, count) // erase count of char at index position**

s = “aa@@@@@aaa”

s.erase(2, 3) -> “a@@aaaa”

**append(count, char) // append count of char**

str.clear() -> “”

str.append(3,’!’) -> “!!!”

str.append(3,’@’) -> “!!!@@@”

**append(“string”) // append null-terminated char string**

str.clear() -> “”

str.append(“hello ”) -> “hello ”

str.append(“world”) -> “hello world”

**append(“string”, index, count) // append a count of null-terminated char string by index position**

str.clear() -> “”

str.append(“hello world”,0, 6) -> “hello ”

str.append(“hello world”,6, 5) -> “hello world”

**append(std::string) // append std:: string**

str.clear() -> “”

str.append(std::string(“hello ”)) -> “hello ”

str.append(std::string(“world”)) -> “hello world”

**append(std::string, index, count) // append a count of std:: string by index position**

str.clear() -> “”

str.append(std::string(“hello world”),0, 6) -> “hello ”

str.append(std::string(“hello world”),6, 5) -> “hello world”

**replace(index, count, “string”) // replace a count of char at index by “string”**

s = “hello amazing world”

s.replace(6,7,”wonderful”) -> “hello wonderful world”

**replace(index, count, std::string) // replace a count of char at index by std::string**

s = “hello amazing world”

s.replace(6,7,std::string(”wonderful”)) -> “hello wonderful world”

**substr(index) //return a substring starts with index position**

s = “hello amazing world”

s.substr(6) -> “amazing world”

**substr(index,count) // return a count of substring’s char starts with index position**

s = “hello amazing world”

s.substr(6, 7) -> “amazing”

**find(“string”) // if founded return the index of substring**

**find(“string”, index) // same as find(“string”) but search starts from index position**

**find(std::string) // if founded return the index of substring**

**find(std::string, index) // same as find(“string”) but search starts from index position**

}

# Additional tasks

# Unit test

Implement unit tests for you class <https://docs.microsoft.com/en-us/visualstudio/test/unit-test-your-code?view=vs-2019>

Each implemented method should be covered by 2-3 test cases.

# Tier1 String class

Class MyString // Tier 1 – implement addition functionality for Basic class

{

Constructor(&&) //move constructor

Constructor(0x1234568) // conversion constructor for integer types -> “0x12345678”

Constructor(0.05) // conversion constructor for float types -> “0.05”

Operator=&& // move assignment

Operator<<(std::basic\_ofstream), Operator>>(std::basic\_ifstream) //file IO read and write

find() // Implement Aho-Corasick Algoritm

**Support basic std::exceptions**

at(index) // return the index element of string if exist, otherwise throw an

exception

**substr(), replace(), erase(),Insert() should throw an out\_of\_range exception**

to\_int() // perform conversion from string to integer types: char, short,

int an other.

to\_float() // perform conversion from string to float types: float, double

and other

**Implement your own exception type for invalid conversion**

}

# Tier2 String class

Class MyString // Tier 2 – implement additional functionality for Tier 1 class

{

**//Implement the Iterator pattern**

iterator begin()

iterator end()

const\_iterator cbegin()

const\_iterator cend()

reverse\_iterator rbegin()

reverse\_iterator rend()

const\_reverse\_iterator rcbegin()

const\_reverse\_iterator rcend()

**Support range-based loop for(auto &el: container)**

All methods that takes index argument must have an iterator version

**insert(index, “string”) -> insert(iterator, “string”)**

**append(std::string, index, count) -> append(std::string, iterator, count)**

… and so on

}

# Python Wrapping

Extend python interpreter functionality with implemented class by 2 or more ways