# 1 Part: Dynamic array of strings 40%

## 1.1 Description

In this part of the assignment, your task is to implement several functions for manipulating dynamic arrays of strings. Intuitively, the array represents a text, and each of the elements of the array represents a word (an arbitrary string without whitespaces).

You have already seen the `string` class used in the lecture. We will study it in more detail later, but you know enough to use it here. As it handles the memory allocation for the individual strings, you will not have to worry about it. Copy constructor, `operator=`, as well as `operator==` are implemented for strings, and you can use them.

Each function that you will have to implement, has a parameter `text` for the array of strings and a parameter `length` for the number of strings stored in the array, passed either by value or by reference depending on the specific function. Some functions have a parameter for the capacity (number of elements) allocated for the array. Some of the functions will be allocating memory for the array. The code using the functions will be responsible for deallocating the returned array's memory.

**A detailed description of the functions is given in the header file `Text.h`.** Below, we illustrate the desired behaviour of some functions on a few examples.

## 1.2 Example Usage

**Please refer to the file `Text.h` for the functions' declaration and description.**

- `resize`

  Parameters

`text:`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | *bb* | *ccc* | | |

`length:` 6 `capacity:` 8

Result of `resize(text, length, capacity, 10)`

`text:`

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | *bb* | *ccc* | | | | |

`length:` 6 `capacity:` 10

**Note:** Calling `resize` with new capacity value that is smaller or equal to the current value of `capacity` leaves all of `text,length` and `capacity` unchanged.

- `readText`

  Example input:

```
a bb,     ccc a bb
    ccc?    EOT
```

  Example result of `readText(cin,length,capacity)`

  `text:`

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| *a* | *bb,* | *ccc* | *a* | *bb* | *ccc?* |  |  |

  `length:` 6  `capacity:` 8

- `getWords`

  Parameters:

  `text:`

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | *bb* | *ccc* | 11 | 22 |

  `length:` 8  `capacity:` 8

  Result of `getWords(text,length,number_unique,capacity_unique)`

  `text:`

|   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | 11 | 22 |  |  |  |

  `number_unique:` 5  `capacity_unique:` 8

- `appendText`

Parameters:

`text:`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | *bb* | *ccc* | | |

`length:` 6 `capacity:` 8

`other:`

| | | | |
|---|---|---|---|
| 11 | 22 | 33 | |

`length_other:` 3

Example result of `appendText(text,length,capacity,other,length_other)`

`text:`

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | *bb* | *ccc* | 11 | 22 | 33 |

`length:` 9 `capacity:` 9

**Note:** If the sum of the current value of `length` and `length_other` exceeds the current value of `capacity`, then `text` is resized to `length+length_other`. Otherwise, `length` is updated to the new length of `text`, but `capacity` remains the same. In either case, the array `other` and its length and capacity are not modified.

- `deleteWordAll`

Parameters:

`text:`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *a* | 11 | *ccc* | *a* | 11 | *ccc* | | |

`length:` 6

Example result of `deleteWordAll(text,length,"11");`

`text:`

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| *a* | *ccc* | *a* | *ccc* | | | | |

`length:` 4

**Note:** The capacity of the array `text` remains unchanged, while the length might decrease, and remaining elements are shifted to the positions of the deleted elements.

- `replacePhraseFirst`

Parameters:

`text:`

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | 11 | *ccc* | *a* | 11 | |

`length:` 8 `capacity:` 9

`phrase:`

| | |
|---|---|
| *a* | 11 |

`new_phrase:`

| |
|---|
| *bb* |

`phrase_length:` 2

`new_phrase_length:` 1

Example result of

```
replacePhraseFirst(text,length,capacity,phrase,phrase_length,
                                              new_phrase, new_phrase_length)
```

`text:`

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *bb* | *ccc* | *a* | 11 | | |

`length:` 7 `capacity:` 9

- `replacePhraseFirst` (another example)

Parameters:

`text:`

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | *a* | 11 | *ccc* | *a* | 11 | |

`length:` 8 `capacity:` 9

`phrase:`

| | |
|---|---|
| *a* | 11 |

`new_phrase:`

| | | | |
|---|---|---|---|
| 11 | 22 | 33 | 44 |

`phrase_length:` 2 `new_phrase_length:` 4

Example result of

```
replacePhraseFirst(text,length,capacity,phrase,phrase_length,
                                    new_phrase, new_phrase_length)
```

`text:`

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| *a* | *bb* | *ccc* | 11 | 22 | 33 | 44 | *ccc* | *a* | 11 |

`length:` 10 `capacity:` 10

**Note:** When the length of the text resulting from the replacement exceeds the capacity of `text`, the capacity of `text` needs to be increased enough to accommodate the new text. Otherwise, the capacity of `text` remains the same. The length of the text might increase, decrease or remain the same, depending on the phrases.