The assessed exercise has 2 assessed tasks: Task A and Task B

Task A

Before completing the task please reads the "hints to complete the task you find below".

Pointer arithmetic is commonly used in C++ and is particularly efficient with arrays.

Create a function in C++ called *substitute* that takes as parameters two chars *char1* and *char2*, and a char* string *s*.

```
void substitute(char * s, char c1, char c2)
{

 // to be completed

}
```

The function *substitute* should *replace* in the string *s* **each** occurrence of *c1* **that is followed by the special symbol *'!'*** with *c2*. Implement the function *substitute* **using pointer arithmetic**.

For example, if

c1 = 'a', c2 = 'x'

s = 'la!belabella!bel',

then (after the function *substitute* has been called) the string *s* should be *'lx!belabellx!bel'* (i.e., replace each 'a' present in the string *s* with an 'x' only when the symbol 'a' is followed in the string *s* by the special symbol '!').

Hints to complete the task:

To complete the task you need to implement a Console Application for C++. Watch the recorded materials in "Introduction to C++" starting from the "How to start with C++" which shows also the initial steps on how to create a Console App for C++ in Visual Studio.

To implement the method you will need to use pointer arithmetic. Watch the video in "Introduction to C++" which discusses pointers (there are also examples on how to use pointer arithmetic). We are also going to discuss about pointers in the webinars, so please consult also the slides and the recorded webinar when completing the task.

Task B

C++ is particularly efficient when implementing algorithms that manipulate strings and arrays.

An important function studied in biology is DNA recombination. DNA can be represented (in an abstract way) as *strings* over the 4 letters A,C,G, and T.

Create a function in C++ called *recombination* that takes as parameters three strings *s1 and s2 and s3* and **display <u>all</u> the strings that can obtained as recombination of s1 and s2 using the string s3**.

Assume the set of letters A = {A,T,C,G} and s1, s2 and s3 strings obtained using the symbols in A.

If s1 and s2 **can be split** using s3 in the following way :

s1 = s1_ls3s1_r and s2 = s2_ls3s2_r (1)

(this means that s1 is the concatenation of s1_l, s3, and s1_r; while s2 is the concatenation of s2_l, s3 and s2_r)

then the strings s1' and s2' are obtained as **recombination of s1 and s2 using the string s3** in the following way :

s1' = s1_ls3s2_r and s2' = s2_ls3s1_r

(this means that s1' is the concatenation of s1_l, s3, and s2_r; while s2' is the concatenation of s2_l, s3, and s1_r)

The string s3 may occur in multiple occurrences within the strings s1 and s2. Therefore to display **<u>all</u>** the strings that can be obtained as **recombination of s1 and s2 using the string s3** one needs to consider **all** possible ways in which s1 and s2 can be split using s3 following (1).

Hints to complete the task:

The program should manipulate strings. It is not a requirement but you are allowed to use the class "string" of C++; or you could choose to use strings as arrays of chars. Watch the recorded materials in "introduction to C++, in particular the videos on strings. We are going to discuss more about C++ in the webinars so please consult also the slides and the recorded webinars when completing the task. There are two examples below.

EXAMPLE1

s1 = 'AGCGADA',

s2 = 'ATTGCG'

s3 = 'GC'

In this example there is **one way** to split s1 and s2 using s3 (because s1 and s2 contain only a single occurrence of s3 as substring). To easily visualize the split, the occurrence of the string s3 used for the split is underlined.

s1 can be split using s3 as

s1 = A<u>GC</u>GADA

this means s1_l = A, s1_r = GADA
(s1 is the concatenation of s1_l, s3, and s1_r)

s2 can be split using s3 as

s2: ATT<u>GC</u>G

this means s2_l =ATT, s2_r = G
(s2 is the concatenation of s2_l, s3 and s2_r)

Then, as described above

s1' = s1_ls3s2_r
(s1' is the concatenation of s1_l, s3 and s2_r)

and

s2' = s2_ls3s1_r
(s2' is the concatenation of s2_l, s3 and s1_r)

This means that the strings s1' and s2' that can be obtained after recombination are :

s1' = AGCG
s2' = ATTGCGADA

So the function `recombination` with input:

s1 = 'AGCGADA',

s2 = 'ATTGCG'

s3 = 'GC'

should display these strings (the order is irrelevant):

'AGCG'

'ATTGCGADA'

EXAMPLE 2

s1 = 'AGCGAGCA',

s2 = 'TAGCTTGCGAT'

s3 = 'GC'

In this case there **are 4 different possibilities** to split s1 and s2 using s3 (as in the example above to better visualize the split, the chosen occurrence of the string s3 is highlighted). For each of this possibility we obtain two strings s1' and s2'.

First possibility

s1 = AGCGAGCA
s2 = TAGCTTGCGAT

This means that
s1_l = A, s1_r = GAGCA, s2_l = TA, s2_r = TTGCGAT

Then that strings that can be obtained after recombination are

s1'= AGCTTGCGAT
and
s2' = TAGCGAGCA

Second Possibility

s1 = AGCGAGCA
s2 = TAGCTTGCGAT

This means that
s1_l = A, s1_r=GAGCA, s2_l = TAGCTT, s2_r = GAT

Then the strings that can be obtained after recombination
are:

s1' = AGCGAT
and
s2' = TAGCTTGCGAGCA

Third possibility

s1 = AGCGAGCA
s2 = TAGCTTGCGAT

This means that
s1_l = AGCGA, s1_r = A, s2_l= TA, s2_r = TTGCGAT

The strings that can be obtained after recombination are then

s1'= AGCGAGCTTGCGAT
and
s2'= TAGCA

Fourth Possibility

s1= AGCGAGCA
s2 = TAGCTTGCGAT

This means that
s1_l = AGCGA, s1_r = A, s2_l = TAGCTT, s2_r =GAT

The strings that can be obtained after recombination are then

s1' = AGCGAGCGAT
and
s2' = TAGCTTGCA

So the function `recombination` with input:

s1 = 'AGCGAGCA',

s2 = 'TAGCTTGCGAT'

s3 = 'GC'

should display these strings (the order is irrelevant):

'AGCTTGCGAT'

'TAGCGAGCA '

'AGCGAT '

'TAGCTTGCGAGCA'

'AGCGAGCTTGCGAT'

'TAGCA'

'AGCGAGCGAT'

'TAGCTTGCA'