Hyperloop II

Submission deadline:	**2018-11-25 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**5.5000**
Max. assessment:	**5.0000** (Without bonus points)
Submissions:	8 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)
Advices:	2 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to implement a function (not a whole program, just a function) that plans hyperloop tracks. The function will be given the lengths of the available track segments and the length of the desired track. Based on these parameters, the function computes whether the track can be constructed, or not. This problem is an extension of the simple variant. Up to three segments are combined in this version.

The hyperloop track is composed of individual segments of certain lengths. We assume there are three manufacturers, each of them produces segments of one fixed length. Each pair of adjacent segments must be connected by exactly one bulkhead; the whole track must start and end with a bulkhead. All bulkheads have the same length. The segments and bulkheads must be used in their original length, i.e., they cannot be cut or extended. The function is given the lengths of the segments, bulkheads, and the length of the track. Based on the parameters, the function decides whether there is a valid combination of segments and bulkheads that gives the exact length of the desired track. If the track can be constructed, the function counts how many different variants are available.

Valid combinations of bulkheads and segments are:

```
bulkhead
bulkhead segment bulkhead
bulkhead segment bulkhead segment bulkhead
bulkhead segment bulkhead segment bulkhead segment bulkhead
bulkhead segment bulkhead segment bulkhead segment bulkhead segment bulkhead
...
```

The required interface is:

```
unsigned long long int  hyperloop                          ( unsigned long long int len,
                                                             unsigned int s1,
                                                             unsigned int s2,
                                                             unsigned int s3,
                                                             unsigned int bulkhead,
                                                             unsigned int * c1,
                                                             unsigned int * c2,
                                                             unsigned int * c3 );
```

`len`
: an input parameter - the length of the track being constructed,

`s1, s2, s3`
: input parameters - the lengths of the segments produced by the first/second/third manufacturer. The parameter may be zero, meaning the segments from the respective producer(s) are unavailable and cannot be used (i.e., only the segments from the other manufacturers),

`bulkhead`
: an input parameter - the length of the bulkhead. The parameter may be zero, meaning the bulkheads do not add to the overall length of the track,

`c1, c2, c3`
: output parameters - the required number of segments from the first/second/third manufacturer. If the function cannot find the solution, it shall not modify this parameter,

return value

is either zero or the number of different solutions found. If the function does not succeed (i.e., no valid combination of segments and bulkheads results in the desired track length), the return value must be 0. Otherwise, if some solutions do exist, the function returns the number of different solutions that lead to the desired track length. The output parameters `c1`, `c2`, and `c3` must be set to some valid solution.

Two solutions are considered identical if the required number of segments is the same for each segment length. Conversely, two solutions are considered different if they differ in the number of segments in at least one length. In other words:

- the order of segments in the track is irrelevant. For example:

```
len = 27
s1 = 3
s2 = 9
s3 = 1
bulkhead = 5
```

gives the following 5 combinations:

```
[bulkhead] [segment 3] [bulkhead] [segment 3] [bulkhead] [segment 1] [bulkhead]
[bulkhead] [segment 3] [bulkhead] [segment 1] [bulkhead] [segment 3] [bulkhead]
[bulkhead] [segment 1] [bulkhead] [segment 3] [bulkhead] [segment 3] [bulkhead]

[bulkhead] [segment 3] [bulkhead] [segment 9] [bulkhead]
[bulkhead] [segment 9] [bulkhead] [segment 3] [bulkhead]
```

Of the above combinations, the first three are equivalent (three segments of length 3, zero segments of length 9, and one segment of length 1) and the last two combinations are equivalent (one segment of length 3, one segment of length 9, and zero segments of length 1):

```
2 * s1 + 0 * s2 + 1 * s3 + 4 * bulkhead = 27
1 * s1 + 1 * s2 + 0 * s3 + 3 * bulkhead = 27

hyperloop ( 27, 3, 9, 1, 5, &c1, &c2, &c3 ) == 2
```

- if two (or all three) manufacturers produce segments of the same length, then the segments are indistinguishable. For example:

```
len = 40
s1 = 10
s2 = 10
s3 = 20
bulkhead = 0
```

There is a total of 9 possible combinations:

```
4 * s1 + 0 * s2 + 0 * s3 + 5 * bulkhead = 40
3 * s1 + 1 * s2 + 0 * s3 + 5 * bulkhead = 40
2 * s1 + 2 * s2 + 0 * s3 + 5 * bulkhead = 40
1 * s1 + 3 * s2 + 0 * s3 + 5 * bulkhead = 40
0 * s1 + 4 * s2 + 0 * s3 + 5 * bulkhead = 40

2 * s+ + 0 * s2 + 1 * s3 + 4 * bulkhead = 40
1 * s+ + 1 * s2 + 1 * s3 + 4 * bulkhead = 40
0 * s+ + 2 * s2 + 1 * s3 + 4 * bulkhead = 40

0 * s+ + 0 * s2 + 2 * s3 + 3 * bulkhead = 40
```

Of them, only three are considered different. The first solution requires four segments of length 10 and zero segments of length 20. The second solution requires two segments of length 10 and one segment of length 20. Finally, the third solution requires zero segments of length 10 and two segments of length 20:

```
hyperloop ( 40, 10, 10, 20, 0, &c1, &c2, &c3 ) == 3
```

Submit a source file with the implementation of `hyperloop`. Further, the source file must include your auxiliary functions which are called from the required function. The `hyperloop` function will be called from the testing environment, thus, it is important to adhere to the required interface. Use the attached sample code as a basis for your development, complete the required function and add your required auxiliary functions. There is an example `main` with some test in the attached code. These values will be used in the basic test. Please note the header files as well as `main` is nested in a conditional compile block (`#ifdef/#endif`). Please keep these conditional compile block in place. They are present to simplify the development. When compiling on your computer, the headers and `main` will be present as usual. On the other hand, the header and `main` will "disappear" when compiled by Progtest. Thus, your testing `main` will not interfere with the testing environment's `main`.

Your function will be executed in a limited environment. There are limits on both time and memory. The exact limits are shown in the test log of the reference. A reasonable implementation of the naive algorithm shall pass both limits without any problems. There is a bonus test, the bonus test requires an efficient algorithm to pass the time limit.

Advice:

- Download the attached sample code and use it as a base for your development.
- The `main` function in your program may be modified (e.g. new tests may be included). The conditional compile block must remain, however.
- There is `assert` macro used in the example `main` function. If the value passed to `assert` is nonzero (true), the macro does nothing. On the other hand, if the parameter is zero, the macro stops the execution and reports line, where the test did not match (and shall be fixed). Thus, the program ends silently when your implementation passes the tests correctly.
- The total track length and the resulting number of different solutions may exceed the range of the `int` data type. Therefore, the interface uses `unsigned long long int` data type instead. Functions `printf` and `scanf` use `%llu` conversion to display/read values of this data type.

- Do not try to generate all possible permutations and subsequently de-duplicate them. Generate only the unique solutions. Arrays are not required to solve the problem. Indeed, the array would add unnecessary complexity (it would have to be dynamically allocated) and the de-duplication would be too time expensive.
- The `unsigned long long int` data type is an extension of the C90/C++03 standard; the data type is included in the newer standards. If the compiler uses the old standard, it displays warnings. Progtest compiler is configured to suppress this warning, the compilation includes `-Wno-long-long` flag.

Sample data: **Download**

☐ **Reference**

8	**2018-11-18 11:58:25**	**Download**
Submission status:	Evaluated	
Evaluation:	5.5000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.027 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.011 s (limit: 2.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.628 s (limit: 1.989 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Bonus test (speed)': Abnormal program termination (Time limit exceeded)
    - Cumulative test time exceeded, killed after:: 2.003 s (limit: 2.000 s)
    - Bonus test - failed, evaluation: No bonus awarded
  - Overall ratio: 100.00 % (= 1.00 * 1.00 * 1.00)
- Advices used: 2
- Penalty due to advices: None (2 <= 2 limit)
- Total percent: 100.00 %
- Early submission bonus: 0.50
- Total points: 1.00 * (5.00 + 0.50) = 5.50

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**2**	--	--	--
Lines of code:	**281**	**140.50 ± 134.50**	**275**	`hyperloop`
Cyclomatic complexity:	**110**	**55.00 ± 53.00**	**108**	`hyperloop`

7	**2018-11-18 11:40:27**	**Download**
Submission status:	Evaluated	
Evaluation:	5.3900	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.030 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 25.00 %
    - Total run time: 0.245 s (limit: 2.000 s)
    - Optional test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 98.00 %, required: 25.00 %
    - Total run time: 0.246 s (limit: 1.755 s)
    - Optional test success, evaluation: 98.00 %
    - Failed (invalid output) **[Unlock advice (65 B)]**
    - ☐ Failed (invalid output)