Employees

Submission deadline:	**2018-12-30 23:59:59**
Late submission with malus:	**2019-01-06 23:59:59** (Late submission malus: 100.0000 %)
Evaluation:	**3.6000**
Max. assessment:	**3.0000** (Without bonus points)
Submissions:	11 / 20 Free retries + 10 Penalized retries (-10 % penalty each retry)
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)

The task is to implement a set of functions to handle linked lists. The linked list represents the list of employees in a company. Each employee is represented by name and a link to his/her substitute. Given such a list, the functions are used to add a new employee, copy the list, and dispose the list.

`TEMPLOYEE`
: is a data structure defined in the testing environment. Your implementation must use the structure, however, it must not modify it in any way. The structure represents one employee. The employees are organized in a single linked list. The structure has the following fields:

- `m_Next` - a link to the next employee in the list. The last employee in the list must set `m_Next` to the `NULL` value.
- `m_Bak` - is a link to the substitute employee. The link is either `NULL` (meaning there is no substitute for this particular employee), or it references some element in the linked list. In particular, the link may even refer to itself, meaning the employee is indeed its own substitute.
- `m_Name` - an ASCIIZ (zero terminated) employee name.

`newEmployee ( name, next )`
: the function creates a new element in the linked list and places the new element at the first position of the list. The parameters are `name` - name of the new employee and `next` - a link to the first element of an existing employee list. Return value is the first element of the newly updated linked list. The function is responsible for the allocation of the employee structure, moreover it must initialize the fields. The newly listed employee does not have any substitute defined, i.e. `m_Bak` must be set to `NULL`.

`freeList ( list )`
: the function frees all resources allocated by the given list. The parameter is a link to the first element of a list previously created using `newEmployee` calls.

`cloneList ( list )`
: the function creates an independent copy of the given list. The newly created list must preserve the employees, the order of employees, their names, and their substitutes. Caution: the newly created list must be independent, thus the links to the substitutes must be updated to refer to the corresponding elements in the newly created list. Return value is the link to the first element in the newly created list.

Submit a source file with the implementation of the above functions. Further, the source file must include your auxiliary functions which are called from the required functions. Your functions 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 functions and add your required auxiliary functions. There is an example `main` with some tests in the attached code. These test cases 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 copy the lists.

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. a new test may be included). The conditional compile block must remain, however.
- There is macro `assert` 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 the line, where the test did not match (and shall be fixed). Thus, the program ends silently when your implementation passes all the tests correctly.
- There are not any known limitations on th eemployee names. In particular, employee names may be unique, but they do not have to be unique.

Sample data: **Download**

Reference

11	**2018-12-30 18:50:17**	**Download**
Submission status:	Evaluated	
Evaluation:	3.6000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.000 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.019 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Random test + memory usage test': success
    - result: 100.00 %, required: 50.00 %
    - Total run time: 0.026 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Bonus - speed': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.570 s (limit: 1.000 s)
    - Bonus test - success, evaluation: 120.00 %
  - Overall ratio: 120.00 % (= 1.00 * 1.00 * 1.00 * 1.00 * 1.20)
- Total percent: 120.00 %
- Total points: 1.20 * 3.00 = 3.60

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**5**	--	--	--
Lines of code:	**179**	**35.80 ± 33.91**	**102**	`main`
Cyclomatic complexity:	**65**	**13.00 ± 17.08**	**47**	`main`

10	**2018-12-30 18:17:17**	**Download**
Submission status:	Evaluated	
Evaluation:	0.0000	

- **Evaluator: computer**
  - Program compiled
  - Test 'Basic test with sample input data': success
    - result: 100.00 %, required: 100.00 %
    - Total run time: 0.000 s (limit: 1.000 s)
    - Mandatory test success, evaluation: 100.00 %
  - Test 'Borderline test': Abnormal program termination (Segmentation fault/Bus error/Memory limit exceeded/Stack limit exceeded)
    - Total run time: 0.006 s (limit: 1.000 s)
    - Mandatory test failed, evaluation: 0.00 %
  - Overall ratio: 0.00 % (= 1.00 * 0.00)
- Total percent: 0.00 %
- Total points: 0.00 * 3.00 = 0.00

SW metrics:

	Total	Average	Maximum	Function name
Functions:	**9**	--	--	--
Lines of code:	**222**	**24.67 ± 27.67**	**102**	`main`
Cyclomatic complexity:	**71**	**7.89 ± 13.86**	**47**	`main`

9	**2018-12-30 18:16:23**	**Download**