Marshal commands

Submission deadline:	**2011-12-18 23:59:59**	632810.953 sec
Evaluation:	**0.0000**	
Max. assessment:	**3.0000** (Without bonus points)	
Submissions:	0 / 10 Free retries + 20 Penalized retries (-2 % penalty each retry)	
Advices:	0 / 2 Advices for free + 2 Advices with a penalty (-10 % penalty each advice)	

Your task is to develop a a set of functions to manipulate single linked lists. Our linked lists will represent soldiers, and the functions may be considered marshal drill commands.

Our program will represent soldiers in the form of a structure -- `TSOLDIER`. A platoon is represented by a list of soldiers. The implementation makes use of single linked lists, where each soldier references his next colleague. The last soldier in the platoon has the link set to `NULL`. Therefore, to manipulate the entire platoon, it is enough to have the reference to the first soldier.

The following commands shall be implemented:

`TSOLDIER * mergePlatoons ( TSOLDIER *a, TSOLDIER *b );`
: The function merges two platoons. The source lists (platoons) will vanish, the result is one bigger platoon, returned by the function. The merging, of course, has strict guidelines. The first soldier in source platoon `a` will become the first in the merged platoon. The next will be soldier #1 from `b`, soldier #2 from `a`, soldier #2 from `b`, ...

`void splitPlatoon (TSOLDIER *src, TSOLDIER ** a, TSOLDIER ** b )`
: The function can be used to split a platoon `src` into two smaller platoons. The first half of the list will become platoon `a`, the second half of the list will become platoon `b` (`a` and `b` are out parameters). Both `a` and `b` will be equally sized. If the number of soldiers in the source platoon was odd, the last soldier in the source will be discarded (i.e. killed, armies permit some collateral damages).

`void destroyPlatoon (TSOLDIER * x );`
: The function will free memory allocated to represent the soldiers in the platoon. The testing environment will call the function for each platoon created if it is no longer needed.

structure `TSOLDIER`
: The structure describes a soldier. There are three fields:

- `m_Next` is a link to the next soldier in the platoon, NULL for the last soldier in the list,
- `m_PersonalID` is a personal numbers,
- `m_SecretRecord` this is a secret record describing the soldier. Your function (your rank) is not allowed to read/write the data. The data must be preserved unmodified.

The input lists (the platoons) will be prepared by the testing environment. The structures will be allocated in the heap, using `malloc` function. If your function needs to free the memory, (i.e. in the `destroyPlatoon ()` function), you shall use `free`. Do NOT use C++ operator delete. The functions that manipulate the platoons shall not create new soldier records. The task is to re-organize the lists such that the resulting lists do match the description. Your implementation MUST modify only the links to the next soldiers, it MUST NOT copy the contents of the structures (i.e. do not copy m_PersonalID and m_SecretRecord among existing structures). Copying of structures is discouraged (it is much slower than just link manipulation), moreover, the testing environment detects the copied structures and evaluates them as failures.

Submit a source file containing the implementation of the required functions. The source file must contain the functions themselves and all your supplementary functions needed (called from) the functions. On the other hand, the source file shall not contain `#include` preprocessor directives and `main` function (if the `#include` definitions and `main` function are inside a conditional compile block, they may stay in the submitted file). Use the code below as a basis for your development. If the preprocessor definitions and conditional compilation remain unmodified, the file may be submitted to the Progtest.

```
#ifndef __PROGTEST__
#include <stdio.h>
#include <stdlib.h>
```

```
typedef struct TSoldier
 {
   struct TSoldier   * m_Next;
   int                 m_PersonalID;
   char                m_SecretRecord[64];
 } TSOLDIER;

#endif /* __PROGTEST__ */

TSOLDIER         * mergePlatoons                           ( TSOLDIER        * p1,
                                                             TSOLDIER        * p2 )
 {
 }

void               splitPlatoon                            ( TSOLDIER        * src,
                                                             TSOLDIER       ** p1,
                                                             TSOLDIER       ** p2 )
 {
 }

void               destroyPlatoon                          ( TSOLDIER        * src )
 {
 }

#ifndef __PROGTEST__
int main ( int argc, char * argv [] )
 {
   /* tests */
 }
#endif /* __PROGTEST__ */
```

The functions will be tested in a limited environment. The problem does not have any high memory/time requirements. The difficulty is in the pointers and memory allocation. Be careful when handling the ends of the lists -- NULL must be set in the `m_Next` to indicate the end of a platoon. Next, an empty platoon is a valid platoon description which may be used in place of input parameters. The functions must handle this "empty" platoon properly and must not crash. Merging of an empty platoon does not add anything, splitting an empty platoon results in two empty platoons, and destroying of an empty platoon again does not do anything.

Sample program use (comments describe input linked lists, numbers represent soldier personal numbers):

```
TSOLDIER  * a, * b, * c;

/* list a: 0 -> 1 -> 2 -> 3 -> 4 */
/* list b: 10 -> 11 -> 12 -> 13 -> 14 */
c = mergePlatoons ( a, b );
/* list c: 0 -> 10 -> 1 -> 11 -> 2 -> 12 -> 3 -> 13 -> 4 -> 14 */
splitPlatoon ( c, &a, &b );
/* list a: 0 -> 10 -> 1 -> 11 -> 2 */
/* list b: 12 -> 3 -> 13 -> 4 -> 14 */
destroyPlatoon ( a );
destroyPlatoon ( b );

/* list a: 0 -> 1 -> 2 */
/* list b: 10 -> 11 -> 12 -> 13 -> 14 */
c = mergePlatoons ( a, b );
/* list c: 0 -> 10 -> 1 -> 11 -> 2 -> 12 -> 13 -> 14 */
splitPlatoon ( c, &a, &b );
/* list a: 0 -> 10 -> 1 -> 11 */
/* list b: 2 -> 12 -> 13 -> 14 */
destroyPlatoon ( a );
destroyPlatoon ( b );
```

```
/* list a: 0 -> 1 -> 2 */
/* list b: 10 -> 11 -> 12 -> 13 */
c = mergePlatoons ( a, b );
/* list c: 0 -> 10 -> 1 -> 11 -> 2 -> 12 -> 13 */
splitPlatoon ( c, &a, &b );
/* list a: 0 -> 10 -> 1 */
/* list b: 11 -> 2 -> 12 */
destroyPlatoon ( a );
destroyPlatoon ( b );
```

Submit: [redacted] Submit

☐ **Reference**