X-mas tree decoration

Submission deadline:	**2011-12-18 23:59:59**	1057741.324 sec
Evaluation:	**0.0000**	
Max. assessment:	**5.0000** (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 set of functions to help with X-mas tree decoration.

Surprisingly, the decoration of a X-mas tree may be a quite difficult task. It involves experiments with the decorations to achieve an aesthetic output. Even if it sounds strange, there may be safety issues related to the decoration. For instance, flammable decorations may be located too close to an open flame, involving a risk of fire. To eliminate this jeopardy, your function will be used to automatically check for the fire hazard.

For the sake of simplicity, we will model X-mas tree as a ternary tree in our program. That is, each node has at most three sons (at most three branches branched from the trunk). Decorations may be placed to the nodes. At most one decoration may be in a node, the possible choices are a candle or a sparkler.

The node is represented by `TNODE` structure, see below. The entire tree is determined by a pointer to the root node. The following interface will be used to access the tree:

```
int setDecoration ( TNODE ** root, char * path, int decor );
```

The function creates or modifies the tree and the decorations. The parameters are:

- `root` (inout), a pointer to the root of the tree modified,
- ASCIIZ string determining a node in the tree (see below) and
- the type of the decoration to be placed into the target node.

The function starts traversing the tree from the root node. As the function traverses the tree, it has to choose the branches. The branch number is provided by the path string. Based on digits '0' to '2', the function chooses the correct branch. The path string is read from left to the right, one digit in each tree node. The function sets the decoration in the target node. If the target node was already decorated, the old decoration will be replaced with the new one. If a node is not present in the path, the function silently creates the required nodes and decorates the newly inserted intermediate nodes with `DECORATION_NONE`. Caution: only create the required nodes, do not create any extra node. The function shall return 1 on success and 0 on failure. A failure means that the path contained characters different from digits '0', '1', and '2'. If the input path is invalid, the function shall not modify the tree in any way and shall return immediately.

```
int cutBranch ( TNODE ** root, char * path );
```

The function will cut off a node of a tree. This may remove an entire subtree, including all connected branches and decorations. The function shall free the memory that was allocated to represent the subtree being removed. Moreover, the link to the removed subtree shall be replaced by NULL in the parent. The parameters are:

- `root` (inout), a pointer to the tree to modify and
- ASCIIZ string determining the node to be cut off (the node itself and all nodes below will be removed).

The path has the same meaning as in the function above. If the function succeeds, return value will be 1. Otherwise (i.e. the path is invalid or the target node does not exist at all), the function will not modify the tree and returns 0 immediately. Please note that an empty string passed to the function destroys the entire tree.

```
int easyToCatchFire ( TNODE * root );
```

The function will test whether the tree is a fire hazard. If there is a candle and a sparkler close to each other, the tree is considered a fire hazard and the function shall return 1. Otherwise, the function shall return 0. A candle and a sparkler are too close to each other if they are either in a direct parent/child relation, or if they are immediate brothers.

```
void destroyTree ( TNODE * root );
```

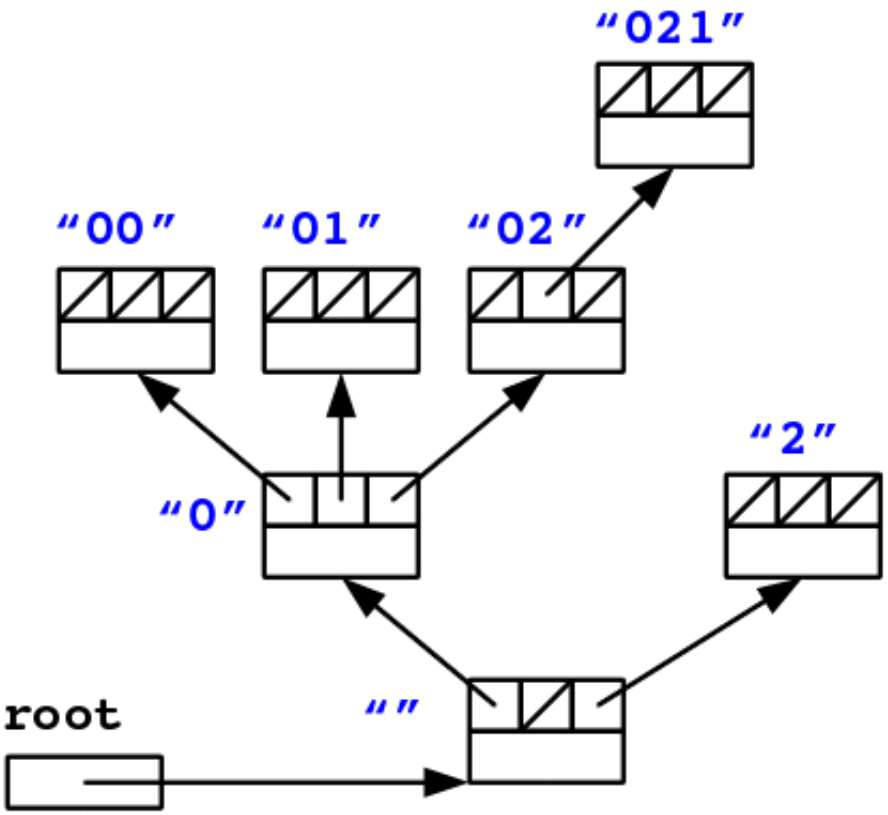
The function shall free all memory allocated to represent the tree, The testing environment will call the function at the end, when the tree is about to be destroyed.

`TNODE`

the structure describes a node in the tree. The fields are:

- `m_Decoration` - the type of decoration in this node (DECORATION_xxx constants),
- `m_Parent` - a link to the parental node, or NULL for the root node,
- `m_Branches` - links for the child nodes, NULL if the node does not have any child in the particular direction.

The paths are depicted in the following figure:



Submit a source file containing the implementation of the requierd functions and all your supplementary functions needed (called from) the required 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.

The problem does not have any high memory/time requirements. The difficulty is in the pointers and memory allocation. Some functions may require recursion. The testing environment will test the memory structure your functions create. If the structure is not exactly the same as in the reference solution (i.e. some nodes missing/extra nodes), your implementation will be considered invalid.

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

#define MAX_BRANCHES             3
#define DECORATION_NONE          0
#define DECORATION_CANDLE        1
#define DECORATION_SPARKLER      2

typedef struct TNode
 {
   struct TNode   * m_Parent;
   struct TNode   * m_Branches[MAX_BRANCHES];
   int              m_Decoration;
 } TNODE;
#endif /* __PROGTEST__ */
```

```
/* your auxiliary functions */


int  setDecoration   ( TNODE ** root, char * path, int decor )
 {

 }
int  cutBranch       ( TNODE ** root, char * path )
 {

 }
int  easyToCatchFire ( TNODE * root )
 {

 }
void destroyTree     ( TNODE * root )
 {

 }

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

Sample usage:

```
TNODE * n;
int     x;

n = NULL;
x = setDecoration ( &n, (char*) "000", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "001", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "002", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "1", DECORATION_CANDLE ); /* x = 1 */
x = setDecoration ( &n, (char*) "01", DECORATION_NONE );  /* x = 1 */
x = setDecoration ( &n, (char*) "", DECORATION_CANDLE ); /* x = 1 */
x = easyToCatchFire ( n ); /* x = 0 */
destroyTree ( n );

n = NULL;
x = setDecoration ( &n, (char*) "000", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "002", DECORATION_CANDLE ); /* x = 1 */
x = setDecoration ( &n, (char*) "2", DECORATION_CANDLE ); /* x = 1 */
x = easyToCatchFire ( n ); /* x = 0 */
destroyTree ( n );

n = NULL;
x = setDecoration ( &n, (char*) "0001", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "000", DECORATION_CANDLE ); /* x = 1 */
x = easyToCatchFire ( n ); /* x = 1 */
destroyTree ( n );

n = NULL;
x = setDecoration ( &n, (char*) "012001", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "012002", DECORATION_CANDLE ); /* x = 1 */
x = easyToCatchFire ( n ); /* x = 1 */
x = cutBranch ( &n, (char*) "0120" ); /* x = 1 */
```

```
x = easyToCatchFire ( n ); /* x = 0 */
destroyTree ( n );

n = NULL;
x = setDecoration ( &n, (char*) "0123", DECORATION_SPARKLER );  /* x = 0 */
x = cutBranch ( &n, (char*) "012" ); /* x = 0 */
x = easyToCatchFire ( n ); /* x = 0 */
destroyTree ( n );

n = NULL;
x = setDecoration ( &n, (char*) "012", DECORATION_SPARKLER );  /* x = 1 */
x = setDecoration ( &n, (char*) "011", DECORATION_CANDLE ); /* x = 1 */
x = easyToCatchFire ( n ); /* x = 1 */
x = cutBranch ( &n, (char*) "" ); /* x = 1, n = NULL */
x = easyToCatchFire ( n ); /* x = 0 */
x = cutBranch ( &n, (char*) "" ); /* x = 0 */
destroyTree ( n );
```

Submit: [redacted] **Submit**

Reference