Christmas tree

CPSPC 2007 Praha, Day 1

Submit:	`xmas.c` / `xmas.cpp` / `xmas.pas`	Time limit:	seconds
Input:	*library*	Memory limit:	64 MB
Output:	*library*	Points:	100

You have a nice, new Christmas tree (well, it's not Christmas time now, but the tree was in a discount) and a box full of decorations of four kinds: Bells, Drops, Spheres and Angels.

The tree consists of points where a decoration can be put. Some of the places are connected by a branch. The whole tree is one piece (all points are interconnected by branches) and there are no cycles.

You made a bet with your friend. You said he will not be able to stop you from decorating the whole tree legally. (Don't dare say you did not know there is a law on decorating trees!) The tree is decorated legally if there is no pair of directly connected points decorated with the same decoration.

You and your friend will keep in changing turns (you start), decorating a previously undecorated point at a time. Neither of you can decorate a point with the same decoration already used for one of its neighbors (it would break the law).

If you manage to reach the state when all the points of the tree are decorated, you win. Otherwise, your friend wins.

Game library:

This task has no input and output file. You should call a special library, which will tell you the structure of the tree and emulate your opponent.

The library is called `tree_lib` and you should link it by specifying either `#include "tree_lib.h"` or `use tree_lib;` (depending on your language of choice). It provides the following functions:

- `int init( void );`
  `function init: longint;`
  You have to call this function exactly once and prior to calling anything else from the library. It will return N ($1 \leq N \leq 1\,000\,000$)—the number of decorable points. From this time on, every point is in range $1, \ldots, N$.
- `int neighbor_count( int point );`
  `function neighbor_count( point: longint ): longint;`
  This function will tell you the number of points directly connected to the given point. If the *point* parameter is out of the $1, \ldots, N$ interval, 0 is returned.
- `int neighbor( int point, int index );`
  `function neighbor( point, index: longint ): longint;`
  It will tell you the *index*-th neighbor of *point*. The *index* starts at 1. If any of the parameters is out of bounds, 0 is returned. (The first neighbor has index 1.)
- `char decoration( int point );`
  `function decoration( point: longint ): char;`
  This function returns the decoration of *point*, which means one of '`A`', '`B`', '`D`' or '`S`' for an angel, bell, drop or sphere, or '␣' for a yet undecorated point and also for a non-existent point.
- `int enemy( void );`
  `function enemy: longint;`
  The function can be called only when it is your opponent's move. It will return which point he decorated. It is up to you to request the decoration of the point.
- `void decorate( int point, char decoration );`
  `procedure decorate( point: longint; decoration: char );`
  Use this to decorate a *point* by *decoration* (one of '`A`', '`B`', '`D`' or '`S`'). You can use this only when it is your turn.

If you decorate a non-existent point, try to overwrite any decoration, place any decoration against the law, pass the tree to your opponent in a state where there are some non-decorated points, but none of them can be decorated legally, or do not obey the turns, your program will be terminated and you lose. When the whole tree is decorated, the program is terminated as well, but you win (by the function that decorated the last point).