



**Red-Black Trees**

**Due: Wednesday, April 29 at midnight in WebCT**

## Problem Specification

An excellent description of red-black trees is given in Wikipedia at:  
[http://en.wikipedia.org/wiki/Red-black\\_tree](http://en.wikipedia.org/wiki/Red-black_tree)

The Wikipedia includes C source code for various cases of insertion into a red-black tree.

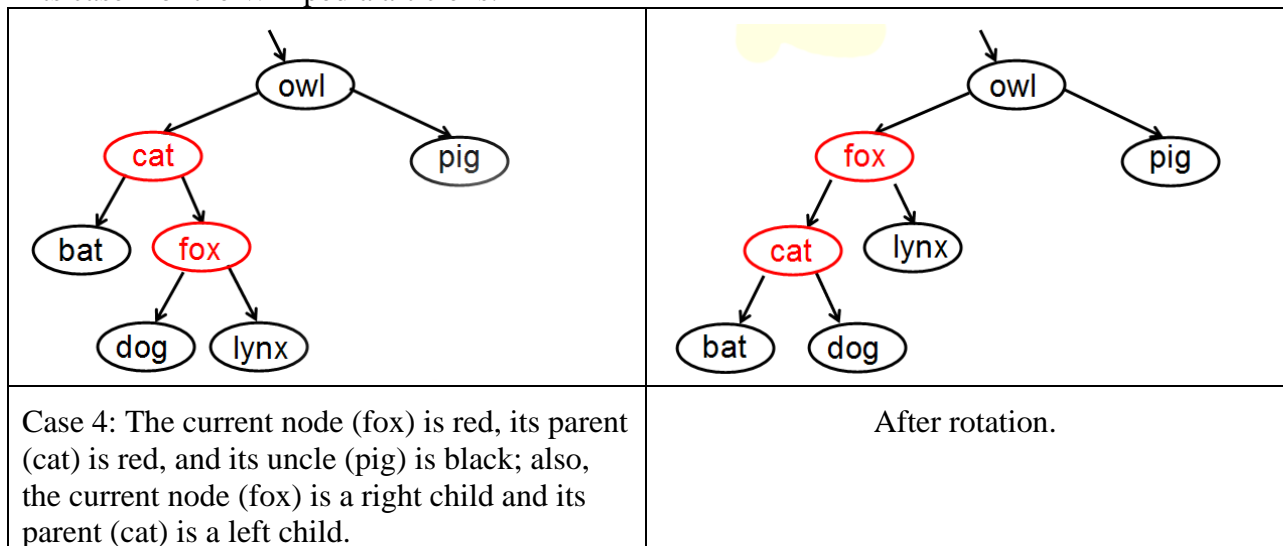
Using the source code provided in Wikipedia, write a C program `YourFirstName_YourLastName_redBlackTree.c` which reads a given file and builds a red-black tree of the words (using lexicographical order) in the file. Each time a node is added to the tree, print the node's text and its parent. For example: if the string "lynx" is read and placed as a leaf in the right side of fox, then print:

**lynx: CHILD OF fox**

If the tree had been empty when fox was read, then print:

**fox: ROOT**

After a node is added to the tree, the tree may need to be adjusted by the red-black algorithm. Sometimes this adjustment requires changing one or more of a node's children (cases 4 and 5 in the Wikipedia article). For each such change, your task is to display the each node that has a changed child, its original child, and its new child. Furthermore, display all such changes for a single case on the same line and in breath-first order of the pre-transformed tree: changed node nearest the root to left-most changed node farthest from the root. For example, part of a tree that fits case 4 of the Wikipedia article is:



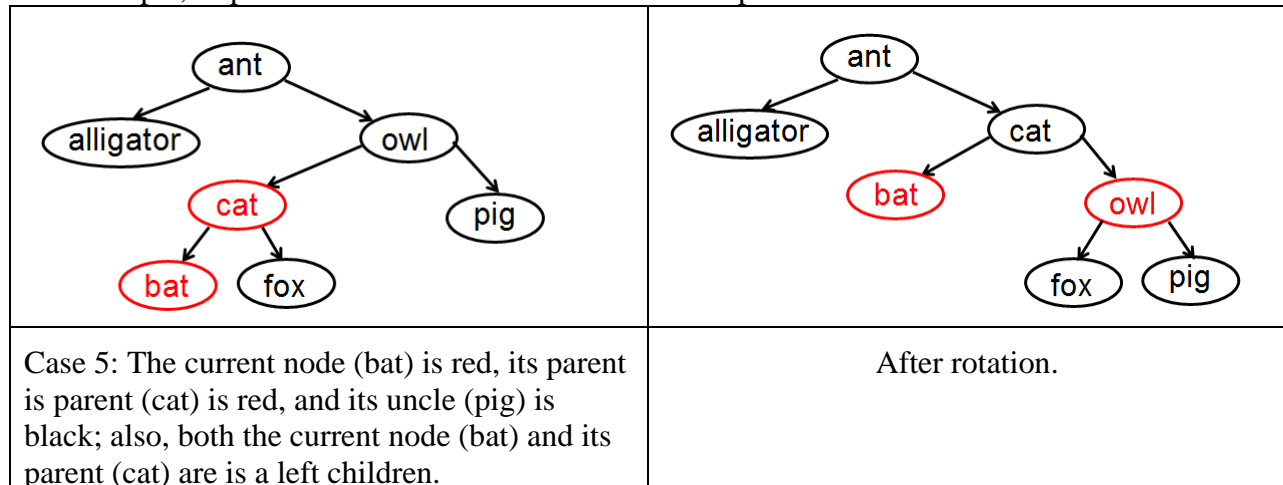
The output for this rotation is:

**(owl: cat => fox), (cat: fox => dog), (fox: dog => cat)**

Because:

1. All nodes are printed that have at least one of their children change. In the above case that is “owl”, “cat” and “fox”.
2. All changes caused by a single rotation are printed on one line.
3. The first node printed is “owl” because it is closer to the root than either of the other two nodes. The next node printed is “cat” because it is closer to the root than “fox”.
4. Before the rotation, owl’s left side child was “cat”, after the rotation it became “fox”. Thus, print: `(owl: cat => fox)`
5. Before the rotation, cat’s right side child was “fox”, after the rotation it became “dog”. Thus, print: `(cat: fox => dog)`
6. Before the rotation fox’s left side child was dog, after the rotation in becomes “cat”. Thus, print: `(fox: dog => cat)`

An example, of part of a tree that fits case 5 of the Wikipedia article is:



The output for this rotation is:

`(ant: owl => cat), (owl: cat => fox), (cat: fox => owl)`

John Franco of the University of California has created a nice Java applet that animates each step of insertion into a red-black tree. The applet is on the web at:

<http://www.ece.uc.edu/~franco/C321/html/RedBlack/redblack.html>

The full output of you program must:

- 1) Remove all trailing whitespace from the input line, then echo the input line within square brackets []. Then print a blank line.
- 2) Print each node as it is first inserted into the tree followed by a line for each rotation that occurs.
- 3) Skip a line. Then print a depth first traversal of the tree including each node's depth, color, children and parent.

An example showing the full output:

```
[any body can do exceptional programming]
```

```
any: ROOT
```

```
body: CHILD OF any
```

```
can: CHILD OF body
```

```
(any: body ==> NULL), (body: NULL ==> any)
```

```
do: CHILD OF can
```

```
exceptional: CHILD OF do
```

```
(body: can ==> do), (can: do ==> NULL), (do: NULL ==> can)
```

```
programming: CHILD OF exceptional
```

Depth First Traversal:

```
1: any(BLACK) c=[NULL, NULL], p=[body]
```

```
2: can(BLACK) c=[NULL, NULL], p=[do]
```

```
3: programming(RED) c=[NULL, NULL], p=[exceptional]
```

```
2: exceptional(BLACK) c=[NULL, programming], p=[do]
```

```
1: do(RED) c=[can, exceptional], p=[body]
```

```
0: body(BLACK) c=[any, do], p=[NULL]
```

You may assume that:

1. All input will consist of single line of space delimited words.
2. Each word will consist solely of lowercase letters.
3. No input file will contain repeated words.
4. Any characters after the last lowercase letter may be considered whitespace (i.e. extra spaces, newlines, carriage returns, tabs, etc.)