Problem II. In C, C++ or Python: Implement the recursive-descend parser for the language #1's = #0's. The grammar for this language is provided in figure 17 of weeks 1-2 file. Prepare a framework of tests demonstrating that your code works properly. The code you submit should include your tests.

Hint: Adapt the recursive-descend parser of FCS section 11.6 constructing parse trees of the grammar of balanced parentheses (FCS figure 11.25; students are allowed to use directly the code of FCS section 11.6). Introduce additional global binary variable indicating whether 0's or 1's play the role of '(' in the parser for the grammar of balanced parentheses

All programming targets were achieved to the 3rd degree. Below is proof of program correctness.

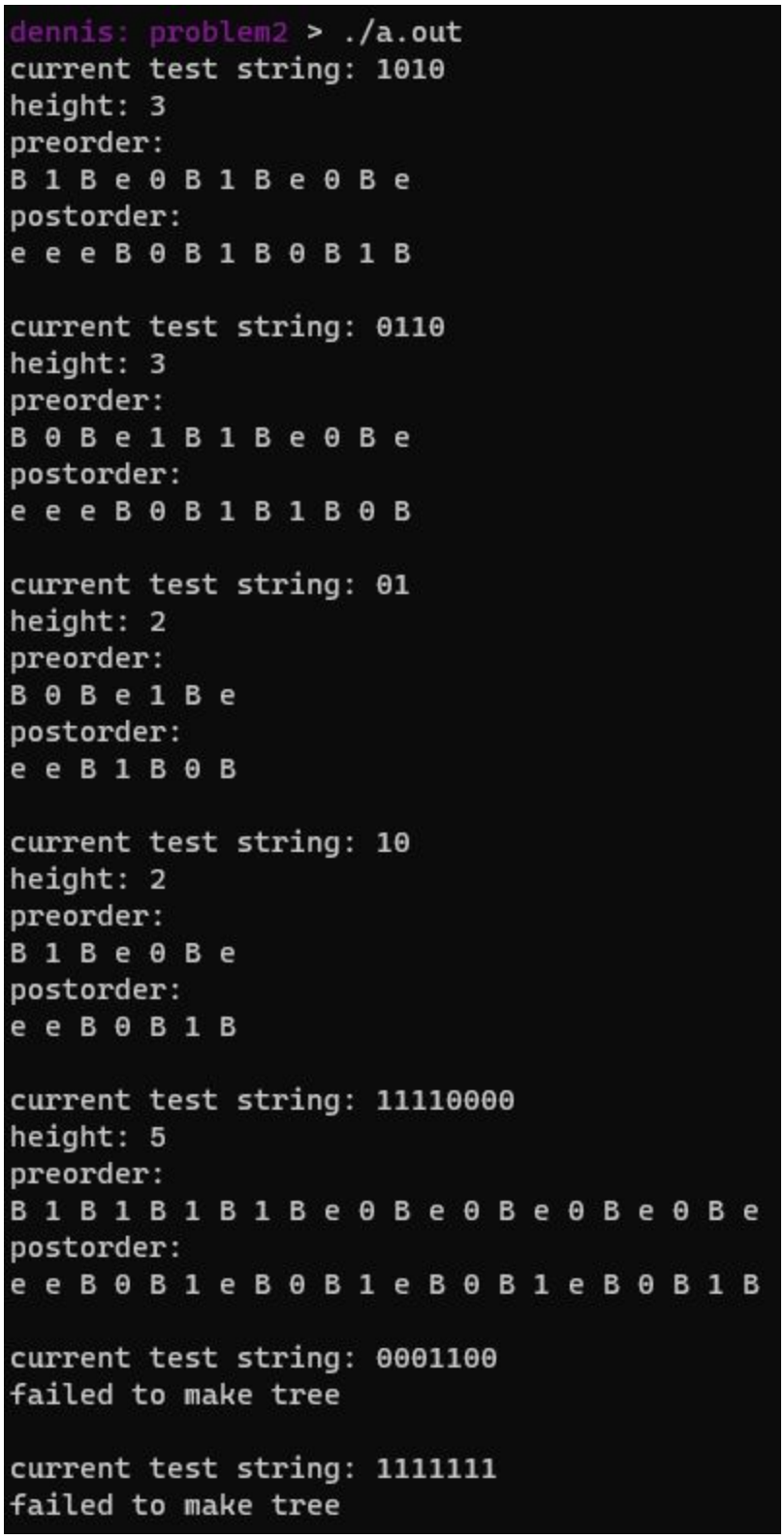
This screenshot shows the tests that were used. The output is pasted below.

```
void main(int argc, char *argv[])
{
    char *tests[] = {"1010", "0110", "01", "10", "11110000", "0001100", "1111111"};
    for(int i = 0; i < 7; i++){
        char* cur = tests[i];
        printf("current test string: %s\n", cur);
        nextTerminal = cur;
        parseTree = B('\0');

        if(parseTree!=NULL){
            int h = computeHeight(parseTree);
            printf("height: %d\n", h);
            printf("preorder:\n");
            printPreOrder(parseTree);
            printf("\n");
            printf("postorder:\n");
            printPostOrder(parseTree);
            printf("\n");
        } else
            printf("failed to make tree\n");
        printf("\n");
    }

}
```

We attempted to parse the following binary strings: "1010", "0110", "01", "10", "11110000", "0001100", and "11111111". We expect that a tree can be made for all of the strings except the last two.



As shown in this screenshot, an input string that has an unbalanced amount of 1's and 0's fails to make a parse tree. If the parse is successful, the tree that is returned is appropriate. The height is calculated and the tree is printed in both pre and post order.