#4 LOGO 解答

代码

```
#include <iostream>
#include <string>
using namespace std;

string skim_spaces_before(const string &str)
{
    for (auto i = str.begin(); i != str.end(); i++) {
        if (*i != ' ')  return string(i, str.end());
    }

    return string();
}

int find_match_bracket(const string &str) // str: a string begins
with [
{
    int cnt = 0;
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == '[') cnt++;
        if (str[i] == ']' && --cnt == 0) return i;
    }
    return -1;
}

int eval(const string &s)
{
    string str = skim_spaces_before(s);
    if (str.empty()) {
        return 0;
    }

    if (str[0] == 'F' || str[0] == 'B') {
        int coe = str[0] == 'F' ? 1 : -1;
        int cur = 3;
        char ch = str[cur];
        int d = 0;
        while (ch <= '9' && ch >= '0') {
            d = d * 10 + ch - '0';
            ch = str[++cur];
        }
```

```
        return coe * d + eval(string(str.begin() + cur, str.end()));
    }
    else { // REPEAT
        int cur = 7;
        char ch = str[cur];
        int d = 0;
        while (ch <= '9' && ch >= '0') {
            d = d * 10 + ch - '0';
            ch = str[++cur];
        }
        int sub_begin = cur + 1;
        int sub_end = find_match_bracket(string(str.begin()+cur,
str.end())) + cur;
        int dist = eval(string(str.begin() + sub_begin, str.begin() +
sub_end));
        return d * dist + eval(string(str.begin() + sub_end + 1,
str.end()));
    }
}

int abs(int x)
{
    return x > 0 ? x : -x;
}

int main()
{
    string str;
    getline(cin, str);
    cout << abs(eval(str)) << endl;

    return 0;
}
```

解析

首先要强调的是，这段代码可能不是最优的写法，效率不够高，内存占用也不够精简。它看起来是很C++的写法——使用了C++专属的类，但又不那么C++——代码中并没有面向对象的体现。

但也许这段代码的可读性是较之其他写法较好的。使用了 `string` 并且毫不吝啬时间空间资源的拷贝让代码免于传递类似于起始位置与长度之类的参数。采用了递归让语句的读取免于要命的循环。（当然了，递归还是要考虑递归深度的问题，但题目中说字符串长度不超过254，所以理应问题不大。）

所以解释这段代码也很简单，只要解释明白 `eval` 函数即可。

一句话说明 `eval` 做的事情：接受一段代码（字符串形式），返回海龟在这段代码控制下移动的距离。

更确切些：

1. 去除代码段之前的空格，得到处理后的字符串，下面的处理都是针对处理后的字符串
2. 若为空字符串，返回0
3. 若为FD或BK命令，读取后面的数字，返回移动的步数+eval(剩下的代码段)
4. 若为REPEAT命令，读取后面的数字作为循环次数，返回循环次数*eval(中括号之间的代码段)+eval(剩下的代码段)