Suppose that the program to the right is compiled into the executable `a.out`. In questions 1 through 3, please tell me the output of that command when typed into the shell.

Question 1: `echo B | ./a.out`

Question 2: `echo AA | ./a.out`

Question 3: `echo BAD | ./a.out`

Question 4: If the string entered on standard input has *n* characters, what is the big-O running time of the program?

Question 5: If the string entered on standard input has *n* characters, what is the big-O memory usage of "the stack"?

Bonus Question: Suppose the parameter **s** were not a reference parameter. Then what is the big-O memory usage of "the stack"?

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

void a(const string &s, int index)
{
  if (index == s.size()) return;

  cout << s[index];

  if (s[index] == 'A') cout << "1";

  a(s, index+1);

  cout << s[index];
}

int main()
{
  string s;

  cin >> s;
  a(s, 0);
  cout << endl;
  return 0;
}
```

Answers to today's clicker questions:

Question 1: This makes one recursive call, which returns instantly, so it simply prints 'B' twice. The answer is "BB".

Question 2: Let's go through what happens here:

- **a("AA",0)** first prints 'A'.
- **a("AA",0)** then prints '1', because character 0 is 'A'.
- **a("AA",0)** calls **a("AA",1)**.
- **a("AA",1)** prints 'A'
- **a("AA",1)** then prints '1', because character 1 is 'A'.
- **a("AA",1)** calls **a("AA",2)**.
- **a("AA",2)** returns instantly.
- **a("AA",1)** prints 'A' again, and then returns.
- **a("AA",0)** prints 'A' again, and then returns.

So the answer is "A1A1AA".

Question 3: Let's go through what happens:

- **a("BAD",0)** first prints 'B'.
- **a("BAD",0)** calls **a("BAD",1)**.
- **a("BAD",1)** prints 'A' and then '1'.
- **a("BAD",1)** calls **a("BAD",2)**.
- **a("BAD",2)** prints 'D'.
- **a("BAD",2)** calls **a("BAD",3)**.
- **a("BAD",3)** returns instantly.
- **a("BAD",2)** prints 'D' again and then returns.
- **a("BAD",1)** prints 'A' again and then returns.
- **a("BAD",0)** prints 'B' again and then returns.

The answer is "BA1DDAB"

Question 4: There are *n* recursive calls, and each call to **a()** does *O(1)* work. The answer is *O(n)*.

Question 5: The stack contains local variables and procedure parameters for each recursive call. There are no local variables, and two procedure parameters, each of which is *O(1)* (the reference parameters is a pointer, which is most commonly 8 bytes, and the integer is 4 bytes). So each context on the stack is *O(1)* and there are *n* of these. The answer is therefore *O(n)*.

Bonus Question: If *s* is not a reference parameter, then a copy of the string will be stored on each stack context. That means each recursive call consumes *O(n)* on the stack. The answer is therefore $O(n^2)$.