TA2 CS354 Aaron Woods

3.2

Pascal:

```
program Fail;
begin
     staticInt: integer = 10;
     setInt(staticInt, 15);
     writeln(staticInt);
end.
```

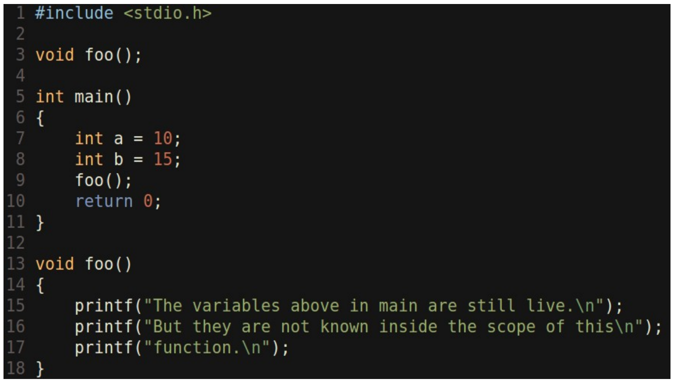
When setInt returns, the value of staticInt will still be 10 because the function only received a copy of the variable, instead of being able to manipulate the variable itself. The reason for this is because the variable was allocated statically.

Scheme:

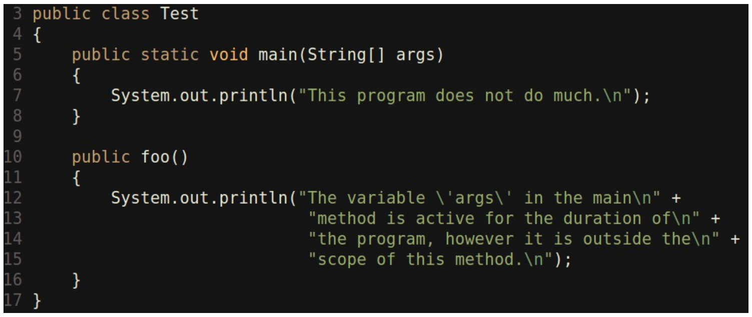
```
(define useless1
     (let (a (cons 1 2)))
     (useless2 a)
)

(define useless2 somethingUseless
     (add (car somethingUseless (cdr somethingUseless)))
)
```

Because 'a' is defined on the stack, the operations performed in useless2 will not affect 'a'.

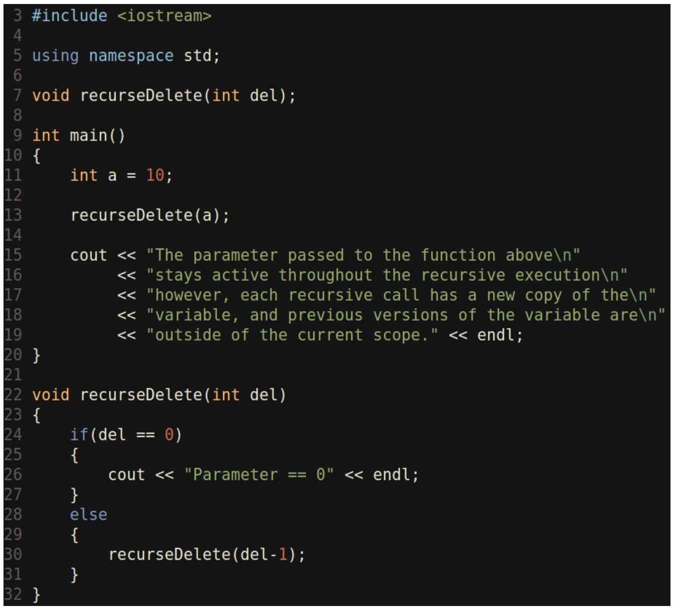
3.4



```
#include <stdio.h>

void foo();

int main()
{
    int a = 10;
    int b = 15;
    foo();
    return 0;
}

void foo()
{
    printf("The variables above in main are still live.\n");
    printf("But they are not known inside the scope of this\n");
    printf("function.\n");
}
```



```
public class Test
{
    public static void main(String[] args)
    {
        System.out.println("This program does not do much.\n");
    }

    public foo()
    {
        System.out.println("The variable \'args\' in the main\n" +
                           "method is active for the duration of\n" +
                           "the program, however it is outside the\n" +
                           "scope of this method.\n");
    }
}
```



```
#include <iostream>

using namespace std;

void recurseDelete(int del);

int main()
{
    int a = 10;

    recurseDelete(a);

    cout << "The parameter passed to the function above\n"
         << "stays active throughout the recursive execution\n"
         << "however, each recursive call has a new copy of the\n"
         << "variable, and previous versions of the variable are\n"
         << "outside of the current scope." << endl;
}

void recurseDelete(int del)
{
    if(del == 0)
    {
        cout << "Parameter == 0" << endl;
    }
    else
    {
        recurseDelete(del-1);
    }
}
```

3.5

C-rules

Line 7: “1 1”
Line 11: “1 1”
Line 14: “1 2”

C#-rules

Line 7: “3 1”
Line 11: “1 1”
Line 14: “1 2”

3.7

She will tell him that L is never reset to its beginning index in reverse. Because of this, when delete_list() is called on L, the while loop is never executed. As a result, there are now two copies of the list left in memory. To fix the problem, he needs to return L to the initial list item before returning the reversed list. He might even consider deleting L at the end of the reverse() function.

3.14

The program prints “1 1 2 2” for static scoping and “1 1 2 1” for dynamic scoping. The reason for this is because in the dynamic scope, the function second() is only modifying its local variable 'x'. It sets it to 2, and then prints it. Once the function exits however, that local 'x' is gone, and the subsequent print_x is now referring to the global variable.

3.18

The program prints “1 0 2 0 3 0 4 0” for shallow binding and “1 0 5 2 0 0 4 4” for deep binding. The reason for this is, in the shallow binding, both set_x() and S() are affecting the global variable. In the deep binding implementation however, set_x() only affects the local variable, and S() only affects the global variable.