Utsav Roy
TextBook Aassignment 2
Shane Panter

Excercise 3.4(Page 167)

a. C example, static variable in a function is live when the execution is not in the function, but it is not in the scope. when for loop is executed in the main block. the value of 'x' is available for the whole function but the does not initiate again and again everytime the for loop is called.

```
void add(void);
    int main(){
    printf(enter any number);
        for(x = 0; x<<b; x++){
            add();
        }
    }
    void add(void){
        static int x;
    int i;
    printf(enter a number);
    scanf("%d",&i);
    x = x+i;
    printf("the total%d", x);
```

b. Java example, global variable x is live upon declaration, however is redeclared within the method double and is not in the scope of the method.

```
    int x = 5;
    public int double(int value){
        int x = 5;
        int x = t *value;
        return x;
    }
```

c. python example, this is similar to java the variable x is live but not in the local scope of add() function.

```
    def algebra(){
        x = 5;
        add();
    }
    def add(){
    x = 6;
    print x + 1;;
    add();
    }
```

Excercise 3.5(Page 167)

declaration order rules of C

- at line 7, it prints **1 1.** a is declared at line 2 and b is at line 5.
- at line 11, it prints **3 1.** a is declared again at 8 and b is at line 5.
- at line 14, it prints **1 2.** a is declared in line 2 and b is at line 3.

Thus the final answer will be **113112.**

declaration order rules of C#

- at line 7, the program will throw error because name are declared before the use. no assignment to a or b is within scope of inner, so the program will brake while compiling. Since line 7 of procedure inner() is asking to print a and b before declaring them.

declaration order rules of Modula-3

- at line 7, it prints **3 3.** a is declared at line 8 since middle is called before to run inner(), b is at line 5. since order of declaration does not matter.
- at line 11, it prints **3 3.** similarly to line 7.
- at line 14, it prints **1 2.** a is declared at line 2 and b is at line 3.

Thus the final answer will be **333312.**

Excercise 3.7(Page 169)

a. Brad made the new list, where rtn is pointing to the beginning of the new list and as he L passes while reversing the list. with the return statement, “return rtn”. He is making the copy of the L and does not have any access to the memory containing the old list.. Thus after multiple iteration of the main() function, the memory is filled and the program crashes.
b. When Brad calls function delete_list(), it frees all the pointers to the data within it, thus the new list will not have any concrete reference. In, the heap might not be corrupted but the output would be because of the reference getting corrupted.

Excercise 3.14(Page 171)

Static Scoping:

first() = 1
print_x = 1 (since x is global variable and is set to 1 on calling first())
second() = 2 (x is defined in procedure second, but while calling setx(2), value 2 is assigned to the global variable x)
print_x = 2 (no on calling second() the new value of global variable x is set to 2)
so, the final output for the static scoping is 1122.

Dynamic Scoping:

first() = 1
print_x = 1(since x is global variable and is set to 1 on calling first())

second() = 2 (x is defined in procedure second, but while calling setx(2), value 2 is assigned to the global variable x)
print_x = 1 (prints the global variable of x found within the scope of print call, which is 1 after first() was called)
so, the final output for the Dynamic scoping is 1121.

Excercise 3.18(Page 169)

Shallow binding:
set_x(0);foo(set_x, print_x,1);print_x

when set_x(0) is called it initializes global x to 0, on calling set_x the value of x is set to 5. if 1 is passed in as the argument which is present in {1,3} , the first part of the conditional statement. after the if statement due to the sallow dynamic binding the value of x is set to 1. Now the second conditional statement is executed and since n is 1 and the condition is {1,2}. The block is executed and local x which is 1 is printed. After this foo is executed global x is set to 0.

Similarly, the final output with shallow binding is **1 0 2 0 3 0 4 0.**

Deep binding:
set_x(0);foo(set_x,print_x,1);print_x

when set_x(0) is called it initializes global x to 0, on calling set_x the value of x is set to 5. if 1 is passed in as the argument which is present in {1,3} , the first part of the conditional statement. after the if statement due to the sallow dynamic binding the value of x is set to 1. Now the second conditional statement is executed and since n is 1 and the condition is {1,2}. The block is executed and local x which is 1 is printed. After this foo is executed global x is set to 0.

Thus the result is **1 0**.

set_x(0);foo(set_x,print_x,2);print_x

when set_x(0) is called it initializes global x is 0. then the function foo(set_x,print_x,2). This function initializes global x as 2 and the execution of else part, since 2 is there in {1,2}. the if part of the second condition statement is executed that is 5.

Thus the result is **5 2**

set_x(0);foo(set_x,print_x;3);print_x

when set_x(0) is called it initializes global x is 0, then the function foo(set_x,print_x,3) is called. This function initiates the local x to 3, after the execution of if part in the condition statement, since 3 is not there in {1,2} the else part of second part is executed and print_x is called. it prints global x, which is 3. After this, the foo() loop returns the global x as 3.

Thus the result is **3 3**

set_x(0);foo(set_x,print_x,4);print_x

when set(0) is called is initializes global x as 0, then the function foo(set_x,print_x;3) is called. This function initializes the global x to 4 after the execution of else part, since 4 is not there int {1,2}, the else part of the second condition statement is executed and print_x is called. it prints global x which is 4. after this, the foo() loop returns the global x as 4.

Thus the result is **4 4.**

The final output with deep binding is **1 0 5 2 3 3 4 4.**