HW3

3.7, 3.14 and 3.39

3.7

As part of the development team at MumbleTech.com, Janet has written a list manipulation library for C that contains, among other things,
the code in Figure 3.16.

```
typedef struct list_node {
    void* data;
    struct list_node* next;
} list_node;

list_node* insert(void* d, list_node* L) {
    list_node* t = (list_node*) malloc(sizeof(list_node));
    t->data = d;
    t->next = L;
    return t;
}

list_node* reverse(list_node* L) {
    list_node* rtn = 0;
    while (L) {
        rtn = insert(L->data, rtn);
        L = L->next;
    }
    return rtn;
}

void delete_list(list_node* L) {
    while (L) {
        list_node* t = L;
        L = L->next;
        free(t->data);
        free(t);
    }
}
```

Figure 3.16 List management routines for Exercise 3.7.

(a) Accustomed to Java, new team member Brad includes the following code in the main loop of his program:

```
list_node* L = 0;
while (more_widgets()) {
    L = insert(next_widget(), L);
}
L = reverse(L);
```

Sadly, after running for a while, Brad's program always runs out of memory and crashes.
Explain what's going wrong.

When he assigns reverse(L) to L, He still has data of L, because he didn't free the data of L

(b) AfterJanet patiently explains the problem to him, Brad gives it another try:

```
list_node* L = 0;
while (more_widgets()) {
    L = insert(next_widget(), L);
}
list_node* T = reverse(L);
delete_list(L);
```

L = T;

This seems to solve the insufficient memory problem, but where the program used to produce correct results (before running out of mem- ory),
now its output is strangely corrupted, and Brad goes back to Janet for advice.
What will she tell him this time?

“You deleted L before assignin T, assign T to L and then delete L. To free the memory.”.

3.14
Consider the following pseudo code:

```
x : integer                                  ––global

procedure set_x(n : integer)
    x := n

procedure print_x()
    write_integer(x)

procedure first()
```

```
    set_x(1)                                S.2. x = 1
                        D.2. x = 1
    print_x()
S.Output1: 1                    D.Output1: 1

procedure second()
    x : integer
    set_x(2)                                S.3. x =
2                       D.3. x = 2
    print_x()
S.Output3: 2                D.Output3: 2

set_x(0)                                    S.1. x =
0                       D.1. x = 0
first()
print_x()
S.Output2: 0                D.Output2: 1
second()
print_x()
S.Output4: 0                D.Output4: 2
```

What does this program print if the language uses static scoping? What does it print with dynamic scoping? Why?

Static scoping outputs: 1020
Because static scoping distinguishes global and local variables, it doesn't keep the track of local data.

Dynamic scoping outputs: 1122
Because dynamic scoping does not distinguish global and local variables. it keeps the record of local data.

3.39
Do you think coercion is a good idea? Why or why not?

It has its strength and weaknesses.

strength would be, as programmers we don't need to worry about what the type of variables we will need or get from the user or API.

the weakness would be that it's not clear, when we're reading through the code. Variable x can be string at one point, but it'll be a integer down the lines of code.