Concepts of Programming Languages, CSCI 305, Fall 2020
Homework #10, complete by class time Dec. 3

Exercises 3.7 and 3.13 from the text (pages 167-171), attribute grammar problem, and Exercise 4.13 (pages 212-213).

1. Exercise 3.7 As part of the development team at MubleTech.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 member and crashes, Explain what's going wrong.

b. After Janet 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 memory), now its output is strangely corrupted, and Brad goes back to Janet for advice. What will she tell him this time?

2. Exercise 3.13 Consider this problem in Scheme:

```
(define A
  (lambda()
    (let* ((x 2)
           (C  (lambda (P)
                 (let  ((x  4))
                       (P))))
           (D  (lambda ()
                 x ))
           (B  (lambda ()
                 (let  ((x  3))
                       (C  D)))))
     (B))))
```

What does this program print?

What would it print if Scheme used dynamic scoping and shallow binding?

Dynamic scoping and deep binding?

Explain your answers.

5. The following grammar parses binary numbers.

binNum → bits | + bits | - bits
bits → bits bit | bit
bit → 0 | 1

a.) Add attribute grammar rules to the grammar to accumulate the decimal value of the binary number into a 'value' attribute of binNum.

binNum → bits

binNum → + bits

binNum → - bits

$bits_1$ → $bits_2$ bit

bits → bit

bit → 0

bit → 1

b.) Show a parse tree for the string -1101, decorate it and show the flow of the values.

4.13 (page 212) Consider the following context-free grammar:

decl → ID decl_tail
decl_tail → , decl
decl_tail → : ID ;

with the attribute grammar

decl → ID decl_tail	
decl.t := decl_tail.t	1.
decl_tail.in_tab := insert (decl.in_tab, ID.n, decl_tail.t)	2.
decl.out_tab := decl_tail.out_tab	3.
decl_tail → , decl	
decl_tail.t := decl.t	4.
decl.in_tab := decl_tail.in_tab	5.
decl_tail.out_tab := decl.out_tab	6.
decl_tail → : ID ;	
decl_tail.t := ID.n	7.
decl_tail.out_tab := decl_tail.in_tab	8.

Show a parse tree for the string A, B : C;. Then, using arrows and textual description specify the attribute flow required to fully decorate the tree.