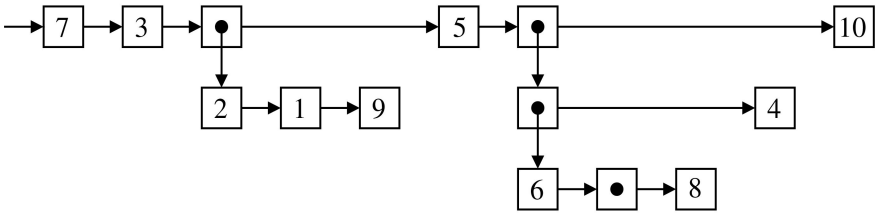
CS19001/CS19002 PROGRAMMING AND DATA STRUCTURES LABORATORY
Assignment No: 9
Last Date of Submission: 13–April–2015

In this assignment, you make a linked-list implementation of a *recursive list*. A recursive list consists of an ordered list of items. Each item in the list can be an element (like an integer) or a recursive list. A recursive list is allowed to be empty. Here is an example of a recursive list:

(7, 3, (2, 1, 9), 5, ((6, (), 8), 4), 10)

The list contains six items. The first, second, fourth, and sixth items are elements (7, 3, 5, and 10). The third and fifth items are recursive lists. The first of these two sublists is a list of three elements (2, 1, and 9), whereas the second consists of two items: a recursive list (6, (), 8) and an element 4. The subsublist (6, (), 8) consists of two elements 6, and 8, and a subsubsublist () which is empty. The outermost list (7, 3, … , 10) is at *level* zero, the two sublists (2, 1, 9) and ((6, (), 8), 4) are at level one, the subsublist (6, (), 8) is at level two, and the subsubsublist () is at level three.

A linked-list implementation of recursive lists is suggested by the following figure:



Each node is the linked list should be capable of storing either an element or a pointer to a sublist. In order to identify what item it is (element or sublist), a type field should also be stored in a node. Finally, we should have a next pointer to move down the list. Use a structure as suggested below:

```
#define ELEMENT 0
#define SUBLIST 1

typedef struct _node {
   int type;
   int element;
   struct _node *sublist;
   struct _node *next;
} node;

typedef node *list;
```

If the type of a node is ELEMENT, then its element field is considered (the sublist pointer may be set to NULL), whereas for a node of type SUBLIST, the element field is ignored, and the sublist pointer stores the sublist. In the figure above, the vertical pointers are the sublist pointers. The node between the elements 6 and 8 has its sublist pointer NULL, since it points to an empty sublist. The horizontal pointers are the next pointers used for browsing down a list.

Part 1

The user enters (or you #define) a size *n* of a recursive list. Write a function to create a random recursive list containing exactly *n* elements (integers). Make random decisions whether the next element to be added is an element or a sublist. Moreover, sublists at levels > 0 should also be allowed to terminate before all the *n* elements are added. Here follows an outline of your list-creation function.

Set L to an empty list.
Repeat so long as the requisite number of elements are not added:
    If level > 0, take a random decision. If the decision is to terminate, break the loop.
    Randomly take a decision whether the next element is an element or a sublist.
    If an element is to be added, append a randomly generated integer to *L* in a node of type ELEMENT.
    If a sublist is to be added, append a node of type SUBLIST to *L*, and its sublist pointer is set recursively.
Return L.

You may take small probabilities (like 0.25) for termination at level > 0 and for sublist creation at each level.

Part 2

Write a function to print a recursive list (with all parentheses). Invoke this function to print the recursive list created in Part 1. The print function should again be recursive.

Part 3

Write a function to flatten a recursive list. The flattened list corresponding to the example on the last page is the list

(7, 3, 2, 1, 9, 5, 6, 8, 4, 10)

consisting only of elements in the same order as they appear in the original list. Your function should not change the original list. It should make fresh memory allocations for creating the flattened list. Use the same list structure for representing this flattened list. Now, each node in the flattened list will be of type ELEMENT. Print the flattened list by invoking the function of Part 2.

Submit a single C source file solving all the parts.

Sample outputs

```
n = 25
Original list:  (2,4,9,8,6,1,1,9,(),8,5,(2,2,7,8),8,8,(4,(),(9),(),(6,(5,3,6,7,7,2))))
Flattened list: (2,4,9,8,6,1,1,9,8,5,2,2,7,8,8,8,4,9,6,5,3,6,7,7,2)

n = 25
Original list:  ((8),(9),(2,4,1),1,7,(4),2,(),1,8,6,5,(2,()),6,8,((((),3,2,(),5),9),5,3,3,(4,(6)))
Flattened list: (8,9,2,4,1,1,7,4,2,1,8,6,5,2,6,8,3,2,5,9,5,3,3,4,6)
```