CS117 (Fall 2015) Programming Assignment 5 30 pts. Due: Oct. 23 (F), before lab starts

Practice for dynamic memory allocation/deallocation (garbage collection)

In the memory allocation part, linked-list version of the insertion sort algorithm is used. For the memory deallocation, we use the lazy garbage collection mechanism that has **mark** and **sweep** phases.

Assume that the size of the simulated dynamic memory is 10 cells and each cell contains three fields, i.e. key, next, and mark_bit (initialized with 0).

For this practice, we use two linked lists whose head pointers are named H1 and H2.
Initially, the free-list (head pointer name: Free) contains all the cells and H1 = -1, H2 = -1, and Free = 1.
The initial memory configuration is:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
key										
next	2	3	4	5	6	7	8	9	10	-1
mark	0	0	0	0	0	0	0	0	0	0

H1 = -1, H2 = -1, Free = 1 //indices of head nodes of list1, list2, and free-list

Part1: memory allocation

After processing the following insertion operations consecutively, using the insertion-sort algorithm,

insert (List1, 3); //insert a node with key value 3 into List1, using insertion sort algorithm
insert (List1, 1);
insert (List2, 4);
insert (List1, 5);
insert (List2, 2);
insert (List2, 9);
insert (List2, 8);
insert (List1, 4);

the resulting memory configuration and head indices are:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
key	3	1	4	5	2	9	8	4		
next	8	1	7	-1	3	-1	6	4	10	-1
mark	0	0	0	0	0	0	0	0	0	0

H1 = 2, H2 = 5, Free = 9
List1 and List2 are sorted (by insertion sort).

- Write a menu-driven program for the following menu options:
  Print_memory //displays memory contents and values of indices (H1, H2, Free)
  Insert (Head_index, key) //gets a new node from the free-list, and inserts it into the list using // insertion sort algorithm

Test your program using the above 8 insertion operations (keep the order). Show the memory and index values after each operation, by invoking the print-memory option.

Part2: memory deallocation

Include the following two additional options into the program developed in part1:

Delete (Head_index, key) //deletes the node with key from the sorted list pointed to by Head_index
Garbage_Collect (Head_index1, Head_index2, Free) //mark-and-sweep garbage collection

For the garbage-collection, use the ***mark-and-sweep*** mechanism, i.e.,

Mark phase: Trace all reachable nodes starting from all head pointers (H1, H2, Free), and mark them.
Sweep phase: Starting from the lowest memory address (memory[1]), collect all unmarked nodes and return them to the free-list (to the head of the free-list one at a time).

Operations on the free-list are LIFO (last in first out, like stack), i.e., garbage collector collects a garbage node and returns it to the head of the free-list (push like). When a memory allocation is requested, the 1st free node is assigned (*pop like*).

Test your program (part2) using the following sequence of operations:

delete (List1, 4);
print_memory;
delete (List2, 8);
print_memory;
delete (List1, 1);
print_memory;
delete (List2, 4);
print_memory;
delete (List1, 5);
print_memory;
garbage-collect (List1, List2, Free)
print_memory

After performing the above five delete operations (right before the garbage collection), the memory configuration is:

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
key	3	1	4	5	6	9	8	4		
next	-1	1	6	-1	2	-1	6	4	10	-1
mark	0	0	0	0	0	0	0	0	0	0

H1 = 1, H2 = 5, Free = 9 //indices of head nodes of list1, list2, and free-list

Memory configuration after the garbage collection is not shown in this assignment sheet. Please make it by your self.

- **Write a menu-driven program (make one program which includes both part1 and part2);**
- **Include good documentation and submit hardcopies of the source code and output by due date.**