SE 2S03 — Assignment 3

Ned Nedialkov

2 November 2018

Due date: 14 November, in class.

Consider the linked lists in Figure 1. Each node of the horizontal linked list has a pointer to a singly-linked (SL) "bottom" list, or NULL, e.g. L4 is empty. Call a structure of this type HB list, from horizontal-bottom list.

Each node contains an integer key. In the horizontal and bottom lists, the keys are sorted in non-decreasing order. The goal is to take a pointer to the beginning of an HB list and return a

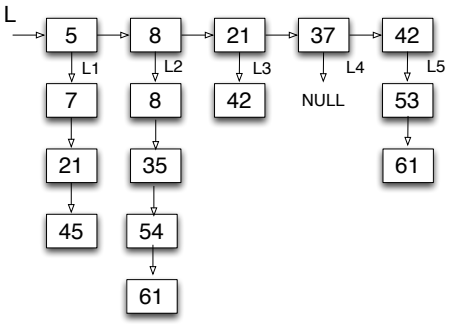

Figure 1: Example of HB list

pointer to a SL list such that the keys are in non-decreasing order as in shown in Figure 2.

Figure 2: The flattened HB list as a SL list

Implement the functions in

```
/* File assignment3.h */
#ifndef ASSIGNMENT3_H
#define ASSIGNMENT3_H

/* Node in a singly linked list */
struct SLnode {
  int            key;
  struct SLnode *next; /* pointer to the next item in a list */
```

```
};

typedef struct SLnode *SLnodePtr;

/* Node in the "horizontal" list */
struct HBnode {
  int            key;
  struct HBnode *next; /* pointer to the next item in the
                          horizontal list */
  SLnodePtr bottom;    /* pointer to the bottom list */
};

typedef struct HBnode *HBnodePtr;

HBnodePtr createHBlist(int n, int m);
SLnodePtr flattenList(const HBnodePtr L);
void freeSLlist(SLnodePtr L);
void freeHBlist(HBnodePtr L);
void printHBlist(const HBnodePtr L);
void printSLlist(const SLnodePtr L);

#endif /* ASSIGNMENT3_H */
```

Descriptions of these functions follow.

```
HBnodePtr createHBlist(int n, int m);
```

Returns a pointer to an HB list with $n \geq 0$ horizontal nodes, and each of the bottom lists has a number of nodes that is a random number in $[0, m]$, $m \geq 0$. The keys should be randomly generated and each of the lists should be sorted as in Figure 1. The goal of this function is to generate valid data so you can work with the remaining functions.

```
SLnodePtr flattenList(const HBnodePtr L);
```

Converts an HB list into a SL list and returns a pointer to the latter. For examples, when called with L from Figure 1 it should return a pointer to the list in Figure 2.

- The input HB list must not be modified.
- You are not allowed to copy the keys from the HB list into an array, sort the array, and then create the SL list.
- You are not allowed to use arrays.
- This function must not use any sorting, as the lists are already sorted.

```
void freeSLlist(SLnodePtr L);
```

Frees all the memory allocated for the list at L.

```
void freeHBlist(HBnodePtr L);
```

Frees all the memory allocated for the list at L.

```
void printHBlist(const HBnodePtr L);
```

Outputs the keys in an HB list at L. The format of the output is up to you.

```
void printSLlist(const SLnodePtr L);
```

Outputs the keys in a SL list at L. The format of the output is up to you.

For example, my main program and implementation

```
#include <stdio.h>
#include "assignment3.h"

int main()
{
  int n = 5, m = 5;
  HBnodePtr L = createHBlist(n,m);
  printf("HB␣list\n");
  printHBlist(L);
  printf("\n");

  printf("SL␣list\n");
  SLnodePtr P = flattenList(L);
  printSLlist(P);
  printf("\n");

  freeSLlist(P);
  freeHBlist(L);
  return 0;
}
```

produce

```
HB list
383-> 555 809 923 1119
777-> 836 1540
793-> 820 1155 1214
886->
915-> 1301

SL list
383 555 777 793 809 820 836 886 915 923 1119 1155 1214 1301 1540
```

You can create your own functions to help implement the above. Make such functions **static**. Store all your code, except the main program, in a file with name `list.c`. It must start with **#include** `"assignment3.h"`.

20 points. When evaluating this assignment, we will test the functions

```
HBnodePtr createHBlist(int n, int m);
SLnodePtr flattenList(const HBnodePtr L);
void freeSLlist(SLnodePtr L);
void freeHBlist(HBnodePtr L);
```

and will not test the print functions.

3 points. After you get your program working, run it through `valgrind`, e.g. `valgrind ./main`. To obtain full marks, `valgrind` must not report any memory problems, e.g. the output should look like

```
==22333==
==22333== HEAP SUMMARY:
==22333==     in use at exit: 0 bytes in 0 blocks
==22333==   total heap usage: 17 allocs, 17 frees, 1,276 bytes allocated
==22333==
==22333== All heap blocks were freed -- no leaks are possible
==22333==
==22333== For counts of detected and suppressed errors, rerun with: -v
==22333== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

1 point Format `list.c` using `clang-format`.

`clang-format list.c`

will output on the screen formatted output. You can save it into a temporary file and then move it to `list.c` by

`clang-format list.c > tmp; mv tmp list.c`

Bonus 3 points To qualify, valgrind must show no errors and no memory leaks, and all of our tests must pass.

Then the formatted `list.c` with the smallest number of lines of C code receives 3 marks, the second smallest 2 marks, the third smallest 1 mark.

Count the number of C lines with the `cloc` program; see http://cloc.sourceforge.net.

You should see output like

```
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
C                                1             20             21            111
-------------------------------------------------------------------------------
```

The number of code lines is 111.

Hints

a. Creating lists. You may create an array with integers and then sort it using the `qsort` function. See `man qsort`. Then you can insert the entries into a linked list(s).

b. Flattening. Consider writing a merge function that merges two sorted lists.

c. Code size. This assignment is short, but requires a lot of thinking and understanding. For example, my `list.c` contains 111 lines of C code.

d. Recursion is very useful for this assignment.

e. Test your code extensively. It should work with boundary cases, e.g. when one or both of the parameters in `createHBlist` are 0.

f. It is not necessary that all the keys are distinct.

Submit

- the formatted `list.c` to SVN in a subdirectory with name `A3`. This directory must contain only `list.c`. (The main program must be in a different file; do not submit it.)
- hard copy of `list.c`
- hard copy of your `valgrind` output

Remarks

- You must not share code.
- I will be following strictly the course policy (from the course outline) about late submissions.
  - To avoid late penalty, both the hard copy and the SVN submission must be before 12:40 on the 14th.
  - Any submission (part or complete) between 12:41 on the 14th and 12:40 on the 15th leads to 10% penalty for this assignment.

    Between 12:41 on the 15th and 12:40 on the 16th the penalty is 20%, and so on. That is, 10% per day.
  - The above excludes special accommodations by SAS.