Jatan Bhatt
400086432
2S03

Assignment 3

list.c

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

static int randIntMin(int min) {
  return min +
         (rand() %
          1024); // carefully chosen modulus to ensure uniformity (power of 2)
}

void freeSLlist(SLnodePtr L) {
  SLnodePtr curr = L;
  while (curr != NULL) {
    SLnodePtr temp = curr;
    curr = curr->next;
    free(temp);
  }
}

void freeHBlist(HBnodePtr L) {
  HBnodePtr curr = L;
  while (curr != NULL) {
    freeSLlist(curr->bottom);
    HBnodePtr temp = curr;
    curr = curr->next;
    free(temp);
  }
}

void printSLlist(const SLnodePtr L) {
  SLnodePtr curr = L;
  while (curr != NULL) {
    printf("%d->", curr->key);
    curr = curr->next;
  }
}

void printHBlist(const HBnodePtr L) {
  HBnodePtr currH = L;
  while (currH != NULL) {
    printf("%d->", currH->key);
    printSLlist(currH->bottom);
    printf("\n");
    currH = currH->next;
  }
```

```
}

/*
   Creates a new SLnode with the given key
   Mot using minKey here as the function is used when flattening
*/
static SLnodePtr createSLnode(int key) {
  SLnodePtr node = malloc(sizeof(struct SLnode));
  node->key = key;
  node->next = NULL;
  return node;
}

/*
   Creates a random list of SLnodes, between [0,m]
   Generates a key with a minimum value to gaurantee ascending order
*/
static SLnodePtr createSLlist(int m, int minKey) {
  int numNodes = rand() % (m + 1); // randomly generate a number between [0,m]
  if (numNodes == 0)
    return NULL; // numNodes = 0

  SLnodePtr head = createSLnode(randIntMin(minKey)); // numNodes = 1

  SLnodePtr curr = head;
  for (int i = 0; i < (numNodes - 1); i++) { // numNodes = 2...numNodes
    curr->next = createSLnode(randIntMin(curr->key));
    curr = curr->next;
  }

  return head;
}

/*
   Creates a new HBnode along with its bottom SL list, given an m
   Generates a key with a minimum value to gaurantee ascending order
*/
static HBnodePtr createHBnode(int m, int minKey) {
  HBnodePtr node = malloc(sizeof(struct HBnode));
  node->key = randIntMin(minKey);
  node->bottom = createSLlist(m, node->key);
  node->next = NULL;
  return node;
}

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

```
HBnodePtr createHBlist(int n, int m) {
  if (n == 0)
    return NULL; // n = 0

  HBnodePtr head =
      createHBnode(m, 0); // n = 1 (min key value is 0 for first node)

  HBnodePtr curr = head;
  for (int i = 0; i < (n - 1); i++) { // n = 2...n
    curr->next = createHBnode(
        m, curr->key); // min key value for next node is current key
    curr = curr->next;
  }

  return head;
}

static SLnodePtr addSorted(SLnodePtr list, SLnodePtr toInsert) {
  // printf("Sorting: %d\n", toInsert->key);

  if (list == NULL)
    list = toInsert;
  else {
    SLnodePtr curr = list;
    while (curr->next != NULL) {
      if (curr->next->key > toInsert->key)
        break;
      curr = curr->next;
    }
    toInsert->next = curr->next;
    curr->next = toInsert;
  }
  return list;
}

SLnodePtr flattenList(const HBnodePtr L) {
  SLnodePtr head = NULL;
  HBnodePtr currHB = L;
  while (currHB) {
    head = addSorted(head, createSLnode(currHB->key));
    SLnodePtr currSL = currHB->bottom;
    while (currSL) {
      head = addSorted(head, createSLnode(currSL->key));
      currSL = currSL->next;
    }
    currHB = currHB->next;
  }
  return head;
}
```

(main function commented out)

```
/*
int main(void) {
  srand(time(NULL));
  HBnodePtr head = createHBlist(5, 10);
  printHBlist(head);
  printf("Done HB\n\n");

  SLnodePtr flat = flattenList(head);
  printSLlist(flat);
  printf("\nDone Flat\n\n");

  freeHBlist(head);
  freeSLlist(flat);
  printf("Done Cleanup\n");
  return 0;
}
*/
```

Valgrind output

```
==8240== HEAP SUMMARY:
==8240==     in use at exit: 0 bytes in 0 blocks
==8240==   total heap usage: 33 allocs, 33 frees, 1,576 bytes allocated
==8240==
==8240== All heap blocks were freed -- no leaks are possible
==8240==
==8240== For counts of detected and suppressed errors, rerun with: -v
==8240== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```