Lab # 6

Collaboration is encouraged. You may discuss the problems with other students, but you must write up your own solutions, including all your C programs, by yourself. If you submit identical or nearly identical solutions to someone else, this will be considered a violation of the code on academic honesty.

In this lab, we’ll study a very useful data structure: a linked list. A linked list consists of a sequence of data records. In each record, there is a pointer (i.e., a “link”) to the next record. Linked lists are often used to implement other data structures such as stacks, queues and, hash tables. The principal benefit of a linked list over an array is that items can be easily added to and removed from the list. The disadvantage is that one cannot jump to a given point in the list; one has to follow all the links to get there.

We’ll use a simple structure for each record: some data (an int) and a pointer to the next record. (Instead of an int, imagine that we had a big chunk of data.)

```
struct record {
    int data; /* data */
    struct record *next; /* pointer to next record */
};
```

Note that, in C, a pointer can have a special value called NULL which indicates it doesn’t point to anything. Consider the following three functions. These add a record to the list, remove a record from the list, and search the list for a specific data value. Note that pointers and pointers to pointers are used.

```
struct record *list_add(struct record *p, int i)
{
    struct record *n = (struct record *) malloc(sizeof(struct record));
    if (n == NULL) return NULL;

    n->next = p; /* the previous record (*p) now becomes the next record */
    p = n;      /* add new empty record to the head of the list */
}
n->data = i;

return p;
}

void list_remove(struct record *p) /* remove head */
{
    if (p != NULL) {
        struct record *n = p;
        p = p->next;
        free(n);
    }
}

struct record *list_search(struct record *n, int i)
{
    while (n != NULL) {
        if (n->data == i) return n;
        n = n->next;
    }
    return NULL;
}

We can print the list, starting at the record pointed by n, with the following function (note that %p prints out a pointer value):

void list_print(struct record *n)
{
    if (n == NULL) {
        printf("list is empty\n");
    }
    while (n != NULL) {
        printf("address %p, next address, %p, data %d\n", n, n->next, n->data);
        n = n->next;
    }
}

In order to create a list populated with data, we’ll use the rand() function in C. This function returns a sequence of pseudo-random numbers (i.e., random looking numbers). It returns a different sequence for every seed value. You can set the seed value as follows:

srand(atoi(argv[1]));
Then you can call the `rand()` function as follows to create records with pseudo-random numbers between 0 and 99:

```c
for (i = 0; i < atoi (argv[2]); i++) {
    list_add(&l, rand() % 100);
}
```

Here is a `main()` function that uses the first argument as the seed; the second argument as the list length; and the third argument as an integer to search for. If the list contains a record with the third argument, the program prints “found”.

```c
#include "stdafx.h"
#include <stdio.h> /* for printf */
#include <stdlib.h> /* for malloc */

int main(int argc, char **argv) {
    int i;
    struct record *l = NULL;
    struct record *s;

    srand(atoi(argv[1]));
    for (i = 0; i < atoi(argv[2]); i++) {
        l = list_add(l, rand() % 100);
    }
    list_print(l);
    s = list_search(l, atoi(argv[3]));
    if (s != NULL) {
        printf("found\n");
    }

    return 0;
}
```

**Problem**

- Write a program that accepts two arguments: the seed value and the list length. It prints the list; then it deletes all records with value 13; then it prints out the list again.

- Write a program that accepts two arguments: the seed value and the list length. It
prints the list; then it deletes each record before a record with value 13; then it prints out the list again.

(Isn’t it idiotic that some buildings don’t have a floor 13? The floors are numbered 1 - 12, and then 14 and above.)