Recitation 10 Solutions

Practice!

(Credit for this section goes to CMU alumna Caroline Buckey; it has been updated since by Alex Cappiello
and Rob Simmons.)

Suppose you have the implementation using linked lists shown in lecture. Specifically, you have the
following structs:

```c
struct list_node {
    int data;
    struct list_node* next;
};
typedef struct list_node list;

struct linkedlist_header {
    list* start;
    list* end;
};
typedef struct linkedlist_header linkedlist;
```

In lecture, we talked about the is_segment(start, end) function that tells us we can start at start,
follow next pointers, and get to end without ever encountering a NULL. (We won’t worry about the
problems with getting is_segment to terminate in this recitation.) A linkedlist is a non-NULL pointer
that captures a reference to both the start and end of a linked list.

```c
bool is_linkedlist(linkedlist* L) {
    if (L == NULL) return false;
    return is_segment(L->start, L->end);
}
```

Recall from lecture that we always have one “dummy” node at the end of our linked list segments. Its
fields are uninitialized; it simply ensures that we never need to worry about start or end being null.

Creating a new linked list

Here’s the code that creates a new linked list with one non-dummy node. Suppose linkedlist_new(12)
is called. For each of lines 4-9 (inclusive) draw a diagram that shows the state of the linked list after
that line executes. Use X for struct fields that we haven’t initialized yet.

```c
linkedlist* linkedlist_new(int data) {
    //@ensures is_linkedlist(result);
    {
        list* p = alloc(struct list_node);
        p->data = data;
        p->next = alloc(struct list_node);
        linkedlist* L = alloc(struct linkedlist_header);
        L->start = p;
        L->end = p->next;
        return L;
    }
```
4.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
\end{tikzpicture}
\end{figure}

5.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \node[draw, rectangle] (12) at (1.5,-1) {12};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
    \draw[->] (12) -- (next);
\end{tikzpicture}
\end{figure}

6.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \node[draw, rectangle] (12) at (1.5,-1) {12};
    \node[draw, rectangle] (data1) at (2,0) {data};
    \node[draw, rectangle] (next1) at (2,-1) {next};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
    \draw[->] (12) -- (next);
    \draw[->] (12) -- (data1);
    \draw[->] (data1) -- (next1);
\end{tikzpicture}
\end{figure}

7.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \node[draw, rectangle] (12) at (1.5,-1) {12};
    \node[draw, rectangle] (data1) at (2,0) {data};
    \node[draw, rectangle] (next1) at (2,-1) {next};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
    \draw[->] (12) -- (next);
    \draw[->] (12) -- (data1);
    \draw[->] (data1) -- (next1);
    \node[draw, rectangle] (start) at (3,0) {start};
    \node[draw, rectangle] (end) at (3,-1) {end};
    \draw[->] (start) -- (p);
    \draw[->] (next1) -- (end);
\end{tikzpicture}
\end{figure}

8.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \node[draw, rectangle] (12) at (1.5,-1) {12};
    \node[draw, rectangle] (data1) at (2,0) {data};
    \node[draw, rectangle] (next1) at (2,-1) {next};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
    \draw[->] (12) -- (next);
    \draw[->] (12) -- (data1);
    \draw[->] (data1) -- (next1);
    \node[draw, rectangle] (start) at (3,0) {start};
    \node[draw, rectangle] (end) at (3,-1) {end};
    \draw[->] (start) -- (p);
    \draw[->] (next1) -- (end);
\end{tikzpicture}
\end{figure}

9.

\textit{Solution:}

\begin{figure}[h]
\centering
\begin{tikzpicture}
    \node[draw, rectangle] (p) at (0,0) {p};
    \node[draw, rectangle] (l) at (0,-1) {L};
    \node[draw, rectangle] (data) at (1,0) {data};
    \node[draw, rectangle] (next) at (1,-1) {next};
    \node[draw, rectangle] (12) at (1.5,-1) {12};
    \node[draw, rectangle] (data1) at (2,0) {data};
    \node[draw, rectangle] (next1) at (2,-1) {next};
    \draw[->] (p) -- (data);
    \draw[->] (l) -- (next);
    \draw[->] (data) -- (next);
    \draw[->] (12) -- (next);
    \draw[->] (12) -- (data1);
    \draw[->] (data1) -- (next1);
    \node[draw, rectangle] (start) at (3,0) {start};
    \node[draw, rectangle] (end) at (3,-1) {end};
    \draw[->] (start) -- (p);
    \draw[->] (next1) -- (end);
\end{tikzpicture}
\end{figure}
Adding to the end of a linked list

We can add to either the start or the end of a linked list. The following code adds a new list node to the end.

```c
void add_end(linkedlist ∗ L, int x)
{//@requires is_linkedlist(L);
 //@ensures is_linkedlist(L);
 { list ∗ p = alloc(struct list_node);
   L−>end−>data = x;
   L−>end−>next = p;
   L−>end = p;
 }
```

Suppose `add_end(L, 3)` is called on a linked list `L` that contains before the call, from start to end, the sequence `(1, 2)`. Draw the state of the linked list after each of lines 5 - 8 (inclusive). Include the list struct separately before it has been added to the linked list.

5.  

![Diagram](https://via.placeholder.com/150)

**Solution:**

6.  

![Diagram](https://via.placeholder.com/150)

**Solution:**

7.  

![Diagram](https://via.placeholder.com/150)

**Solution:**

8.  

![Diagram](https://via.placeholder.com/150)

**Solution:**

3
Adding to the start of a linked list

With the previous example in mind, can you think about what code would be necessary if we instead wanted to add a new list node to the start of a linked list?

```c
void add_start(linkedlist* L, int x)
{
    list* p = alloc(struct list_nodes);
    p->data = x;
    p->next = L->start;
    L->start = p;
}
```

Removing the first item from a linked list

This is the code that removes the first element from a linked list. If it were not for the second precondition, we might remove the dummy node! This would almost certainly cause the postcondition to fail.

```c
int remove(linkedlist* L)
{
    int x = L->start->data;
    L->start = L->start->next;
    return x;
}
```

Suppose `remove(L)` is called on a linked list `L` that contains before the call, from start to end, the sequence `(4, 5, 6)`. Draw the state of the linked list after lines 6 and 7 execute. Include an indication of what data the variable `x` holds.

6. 

![Diagram of linked list before removal]  

**Solution:**

7. 

![Diagram of linked list after removal]  

**Solution:**