

2 Linus Torvald's Elegant Linked-List Removal¹

Consider first the following typical C definition for a singly-linked list with a typical definition of list removal:

```
struct Node {
    struct Node *next;
};

struct List {
    struct Node *head;
};

void remove(struct List *l, struct Node *toremove)
{
    // assume "toremove" is not null and is part of "l"
    struct Node *curr = l->head;
    struct Node *prev = 0;
    while (curr != toremove) {
        prev = curr;
        curr = curr->next;
    }
    if (prev) {
        prev->next = curr->next;
    } else {
        l->head = curr->next;
    }
}
```

Note the case distinction at the end of `remove`: if the element to be removed happens to be the first in the list, then we update `l->head`, otherwise `prev->next` is updated.

Linus Torvalds² prefers another way of defining the removal:

```
void removeBetter(struct List *l, struct Node *toremove)
{
    struct Node **p = &l->head;
    while (*p != toremove)
        p = &(*p)->next;
    *p = toremove->next;
}
```

This is more elegant, because there are no special cases to consider: at the end, we always update `*p`. This works because `p` points to either `l->head` (if the first element should be

¹We warmly thank Ștefan Ciobăcă from Alexandru Ioan Cuza University for contributing the idea for this challenge.

²Discussed during a Ted Talk: https://www.ted.com/talks/linus_torvalds_the_mind_behind_linux, at timestamp 14:30.

removed), or to the previous element in the list (when an element other than the first should be removed).

Tasks

- (a) Define a data structure for linked lists in your verification tool of choice.
- (b) Implement the `removeBetter` subprogram.
Note: if the tool you use does not support indirect references, you may simulate an indirect reference by passing the heap as an explicit argument to the subprogram.
- (c) Prove that `removeBetter` finishes and does not have memory errors when called with a pointer to a linked list and `toremove` is part of the list.
- (d) Prove that `removeBetter` is functionally correct: it changes the list such that `toremove` is actually removed.
- (e) Can you also make the verification conditions of `removeBetter` not use special cases?

Extra Task

- (f) Using the same trick of using indirect references to avoid special cases, create and prove an `insertBefore` subprogram that inserts a new element in the list just before another.