MTH 4300
Spring 2025
E. Fink

Homework 7

---

1. First, read Lecture 19a, about `vector`s. Then, utilize what you've learned in solving this problem:

   Write code which allows the user to enter in a sequence of names (you may assume no spaces in each name), until they enter the word `STOP`. Your program should print out the number of *different* names in the list (not including the `STOP`). For example, if the user entered in `Alice Bob Alice Carol Bob Alice STOP` the program should output `3`, for there are three different names, `Alice`, `Bob` and `Carol`.

2. I have the following code:

```
int main() {
   vector <string> guest_list = {"Alice", "Bob", "ADD3", "Carol", "David", "ADD3", "Evan"}; // CONTENTS MAY
                                                                                            // CHANGE
   vector <string> vips = {"Jane", "Kevin", "Larry", "Marcus", "Nancy"}; // CONTENTS MAY CHANGE
}
```

   Write code that goes through `guest_list`, and for every appearance of the string `ADD3`, replaces it with the first 3 entries from `vips` that have not yet been added – and if there are fewer than 3 entries from `vips` left, only add the remaining people (possibly none, in which case `"ADD3"` should simply be erased from the list).

   For example, with the given lists, the final contents of `guest_list` should be

   `Alice Bob Jane Kevin Larry Carol David Marcus Nancy Evan`

   Your code should still work even if `guest_list` and `vips` were each replaced with different non-empty lists of strings.

   (**Warning:** do NOT use iterators for `guess_list`, as they will break, since `guest_list` gets modified as you loop through it. Instead, use an index variable.)

3. Consider the following code:

```
struct Node {
    string data;
    Node *next;
    Node(string s, Node* n = nullptr): data{s}, next{n} {}
};

int main(){
    Node *the_list;
    Node e("Elephant");
    Node c("Camel");
    Node g("Giraffe");
    Node o("Okapi", &c);
    the_list = &g;
    g.next = &o;
}
```

   a. What would print out if the following code were added to `main()`?

```
    Node *current = the_list;
    while(current != nullptr) {
        cout << current->data << endl;
        current = current->next;
    }
```

   b. What would happen if the following code were added to `main()` instead? Explain.

```
    Node *current = the_list;
    while(current != nullptr) {
        cout << current->data << endl;
        current = current->next->next;
    }
```

4. a. Write a function `string get(Node *head, int n)` (where the struct `Node` is as in the previous problem). The first parameter to this function should be thought of as a pointer to the first element of a linked list. This function should retrieve the value of the `n`th element of the given linked list (where `n` should NOT be 0-based: so `get(head, 1)` should return the data of the first `Node` in the linked list. If `n` is out-of-bounds, your function should return an empty string.

   b. If you answered part a *with* recursion, provide a non-recursive implementation. If you answered part a *without* recursion, provide a recursive implementation.

5. Write a function `void remove_Rs(Node* &)` (where the struct `Node` is as in the previous problems). The parameter to this function should be thought of as a pointer to the first element of a linked list. The function should delete each element of the list which starts with the letter `R`.

   So for example, if we had the following code:

```
int main(){
    Node n5("Rich");
    Node n4("Joe", &n5);
    Node n3("Rob", &n4);
    Node n2("Ryan", &n3);
    Node n1("Okapi", &n2);
    Node *head = &n1;
    remove_Rs(head);
}
```

   then the list pointed to by `head` would contain `Okapi Joe` in it at the end.

   I suggest a recursive solution. Hint: consider three cases – when the `head` equals `nullptr`, when the `head` is the address of a Node whose data starts with `R`, and when the `head` is the address of a Node whose data does not start with `R`. Also, be sure to have the parameter be passed by REFERENCE, since this function needs to modify the list.