

Section Handout

Problem 1: Removing Duplicates

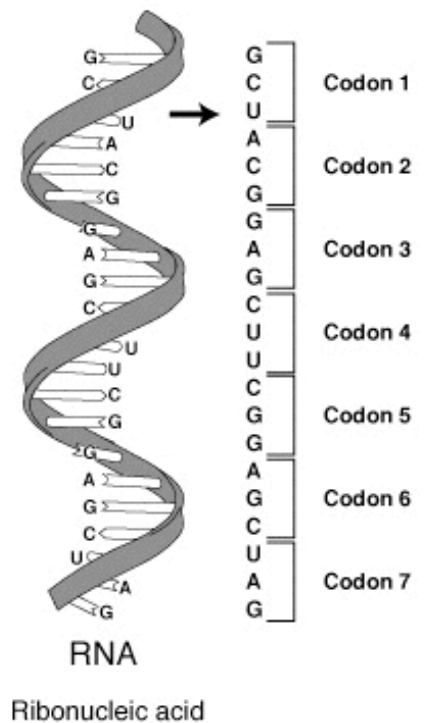
Write a function `RemoveDuplicates` that given a linked list will remove and free the second of all neighboring duplicates found in the list. If the incoming list is (5 5 22 37 89 89 15 15 22) the function will destructively modify the list to contain (5 22 37 89 15 22). Don't worry about duplicate sequences longer than 2 or duplicates that aren't right next to each other in the list.

```
struct node {
    int value;
    node *next;
};

void RemoveDuplicates(node *list);
```

Problem 2: Ribonucleic Acid and Codons

Ribonucleic acid—more commonly referred to as RNA—consists of a series of molecular subunits chained together to form a polymer. Each of these subunits is constrained to be one of four **nucleotides**: adenine (**A**), guanine (**G**), uracil (**U**), or cytosine (**C**). Genetic information is expressed via tri-nucleotide units (such as **UAC** or **GGU**) called **codons**, so that an RNA strand of 36 nucleotides, for example, is really a sequence of 12 codons. By crawling over a strand of our own RNA, one codon at a time, we can get information about how our body is instructed to chain together amino acids to build proteins.



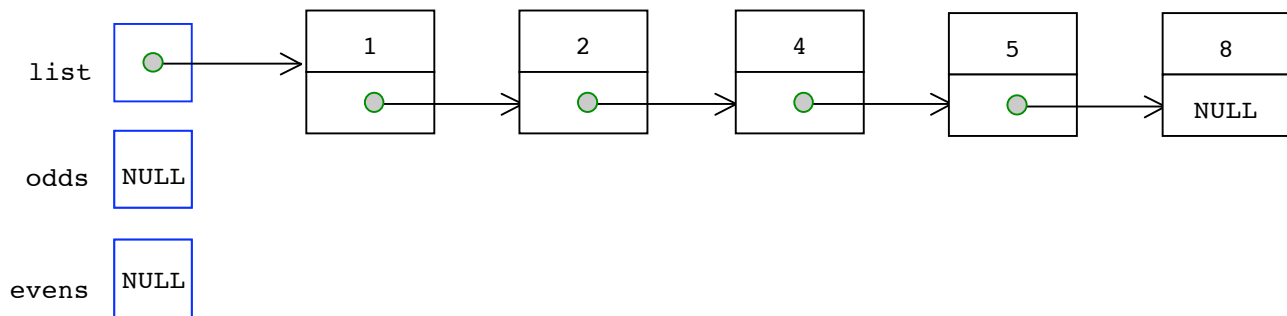
Write a function called `ExtractCodons`, which takes a string of nucleotides (represented by the characters `'A'`, `'G'`, `'U'`, and `'C'`) and returns a dynamically allocated array of four linked lists. The 0th, 1st, 2nd, and 3rd list contain all those codons of the specified RNA strand that begin with `'A'`, `'G'`, `'U'`, and `'C'`, respectively. In addition to building the lists, you should also place the number of individual `'A'`, `'G'`, `'U'`, and `'C'` nucleotides comprising the specified RNA strand in the supplied `counts` array. The codons in each of the four lists can be strung together in any order. (Biologists: forgive my simplification of your biochemistry. It's more about providing an interesting backdrop for linked lists. ☺)

```
struct node {
    string codon;
    node *next;
};

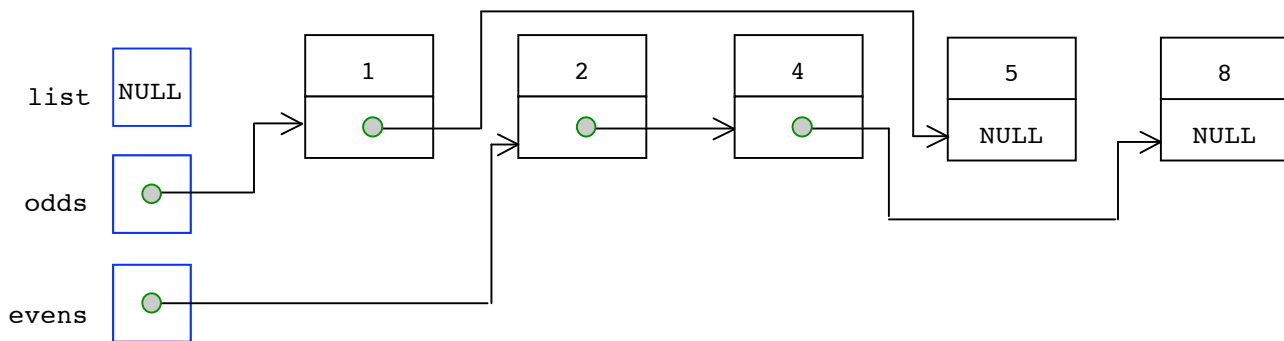
node **ExtractCodons(string rna, int counts[]) // counts is of length 4
```

Problem 3: Separating Odds And Evens,

Write a function `SeparateOddsAndEvens`, which destructively partitions the specified linked list of integers into two other lists—one containing all of the odd values, and the other containing all of the even values. Rather than allocating space for new nodes, you should simply reuse the nodes from the original and be content that the original will no longer be needed. As an illustration, assume that the original list is represented as follows:



Just after exit, `SeparateOddsAndEvens` would leave relevant memory in the following state:



The illustration emphasizes the fact that no new memory is dynamically allocated. Note that original `list` has been set to `NULL` in the process. Also note the prototype, which deals with references to pointers for all three parameters. Tricky!

```
struct node {
    int value;
    node *next;
};
```

```
void SeparateOddsAndEvens(node *& list, node *& odds, node *& evens)
```