

Section Solution

Problem 1: Removing Duplicates

```
void RemoveDuplicates(node *list)
{
    for (node *cur = list; cur != NULL; cur = cur->next) {
        if (cur->next != NULL && cur->value == cur->next->value) { // match?
            node *duplicate = cur->next;           // remember
            cur->next = cur->next->next;           // circumvent
            delete duplicate;                       // dispose
        }
    }
}
```

Problem 2: Ribonucleic Acid and Codons

```
node **ExtractCodons(string rna, int counts[]) // counts is of length 4
{
    node **lists = new node *[4];

    for (int i = 0; i < 4; i++) {
        counts[i] = 0;
        lists[i] = NULL;
    }

    for (int i = 0; i < rna.length(); i+=3) {
        string codon = rna.substr(i, 3);
        for (int j = 0; j < codon.length(); j++)
            counts[CharToIndex(codon[j])]++;
        int index = CharToIndex(codon[0]);
        node *n = new node;
        n->codon = codon;
        n->next = lists[index];
        lists[index] = n;
    }

    return lists;
}

int CharToIndex(char ch)
{
    switch (ch) {
        case 'A': return 0;
        case 'G': return 1;
        case 'U': return 2;
        case 'C': return 3;
        default: Error("Unexpected character: '%c'\n", ch);
    }
}
```

Problem 3: Separating Odds And Evens!

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

void SeparateOddsAndEvens(node *& list, node *& odds, node *& evens)
{
    node **oddsp = &odds;    // track the address of the original odds
    node **evensp = &evens; // same thing for the original evens

    for (node *curr = list; curr != NULL; curr = curr->next) {
        if (curr->value % 2 == 0) {
            *evensp = curr;
            evensp = &(curr->next);
        } else {
            *oddsp = curr;
            oddsp = &(curr->next);
        }
    }

    *oddsp = *evensp = list = NULL;
}
```