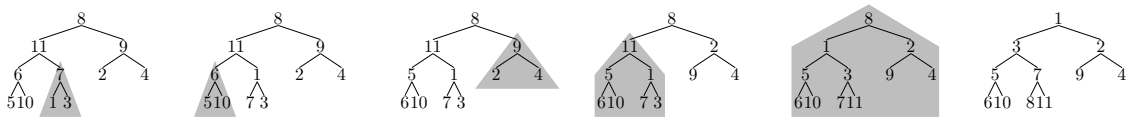
1. 

2. This solution assumes `s` is a null-terminated c-style string. If using the C++ string class, we would need one additional parameter indicating the current position in the string.

```
// nuop and nucp are the numbers of unmatched open and closed
// parentheses in the array of characters from str to s-1.
int cucp( char *s, int nuop, int nucp ) {
  if( *s == 0 ) return nucp;
  if( *s == '(' ) return cucp(s+1,nuop+1,nucp);
  if( *s == ')' ) {
    if( nuop>0 ) return cucp(s+1,nuop-1,nucp);
    else return cucp(s+1,nuop,nucp+1);
  }
  return cuop(s+1,nuop,nucp);
}
int countUnmatchedCloseParens( char *str ) {
  return cucp(str,0,0);
}
```

3. (a)
```
// Sort an array A of 0's and 1's so the 0's come before the 1's
// using swapping.  The size of the array is n.
int bitsort(int *A, int n) {
  int i=0;
  int j=n-1;
  while( i<j ) { // alternative solution: ( i<=j )
    if( A[i] == 0 ) {
      i++;
    } else if( A[j] == 1 ) {
      j--;
    } else {
      swap( A[i], A[j] );
    }
  }
  return j;
}
```

(b) We claim that the invariant: "`A[0...i-1]` is all zeros and `A[j+1...n-1]` is all ones" is true at the start of each iteration of the while-loop.

Proof. We prove the invariant by induction on the number of iterations of the while-loop. Base case: At the start of the first iteration, `i` is 0 and `j` is `n-1` so both `A[0...i-1]` and `A[j+1...n-1]` are empty and the invariant is true.

Assume as our inductive hypothesis (IH) that the invariant is true at the start of the w-th iteration of the while-loop, where $w \geq 0$. There are three cases to consider.

Case `A[i] == 0`: By the IH, `A[0...i-1]` is all zeros. Since `A[i]==0`, the program increases `i` by one and `A[0...i-1]` (for the new `i`) is all zeros. Also `j` does not change. Thus the invariant is preserved.

Case `A[j] == 1`: By the IH, `A[j+1...n-1]` is all ones. Since `A[j]==1`, the program decreases `j` by one and `A[j+1...n-1]` (for the new `j`) is all ones. Also `i` does not change. Thus the invariant is preserved.

Case `A[i]==1` and `A[j]==0`: In this case, the swap of `A[i]` and `A[j]` does not affect the invariant since neither lies in the range `A[0...i-1]` or `A[j+1...n-1]`, and neither `i` nor `j` changes. Thus the invariant is preserved. □

(c) If the loop terminates, then, since only `i` or `j` are changed during an iteration and that change is only by one, `i==j` at termination. The loop invariant then states that "`A[0...i-1]` is all zeros and `A[i+1...n-1]` is all ones", which implies that the array `A` is sorted (no matter if `A[i]==0` or `A[i]==1`). In the alternative solution, the loop terminates when `i==j+1` and so "`A[0...i-1]` is all zeros and `A[i...n-1]` is all ones", which again implies that the array `A` is sorted.

We need to prove that the loop terminates. In some iterations, `i` and `j` don't change, so this is not completely trivial. However, if `i` and `j` don't change, then the program must swap `A[i]==1` with `A[j]==0`, so in the next iteration `A[i]==0` and the program increments `i`. In other words, `j-i` decreases by at least one in every two iterations of the while-loop, which implies termination.

(d) (bonus) When the array is not all zeros (you can check that on termination, `A[i]` is zero only if the array is all zeroes, hence, the number of zeroes equals to `i`). In the alternative solution, when the loop terminates, the number of zeroes is exactly `i` and `j=i-1`, so the answer is "never".

4. (a) Alice plays to preserve the invariant that the number of remaining links is divisible by 3 at the end of her move.

Claim 1. *If n is not divisible by 3, then Alice can always win.*

Proof. (by induction on n)
Base case: If $n = 1$ or $n = 2$, then Alice wins by taking all the links.
Inductive Hypothesis: Assume the claim is true for $n \leq 3k$ for some $k \geq 1$.
Inductive Step: We want to show that the claim is true for $n \leq 3(k+1)$, which means we need to show that Alice can win if $n = 3k+1$ or $n = 3k+2$. Alice takes one link (if $n = 3k+1$) or two links (if $n = 3k+2$) so that the number of remaining links is $3k$. No matter if Bob takes one or two links, the number of links remaining after his turn is not divisible by 3 and is less than $3k$, which means by the inductive hypothesis that Alice wins. □

Alternatively, we can prove that Alice can maintain the invariant that the number of links after her turn is not divisible by 3; and that the invariant implies that she wins. (This is more like the type of proof that we've seen for establishing loop invariants in class.)

Claim 2. *If the initial number of links n is not divisible by 3, then Alice can maintain the invariant that the number of links after her turn is divisible by 3.*

Proof. (by induction on the number, t, of Alice's turns)
Base Case If $t = 0$, since Alice moves first, if n is not divisible by 3, then she can achieve the invariant by removing one or two links.
Inductive hypothesis: Assume the invariant is true after Alice's t-th turn.
Inductive step: We want to show that the invariant is true after her $(t+1)$-st turn. Prior to her $(t+1)$-st turn, Bob has just removed one or two links from a number of links that, by the inductive hypothesis, is divisible by 3. Thus the number of links remaining after Bob's move is not divisible by 3. Alice can therefore remove one or two links to make the invariant true. □

Since zero is divisible by 3 and the number of links decreases after every turn, if the initial number of links is not divisible by 3, then Alice wins.

(b) If $n = 3$ or 4, Bob can always win. If $n \geq 5$, Alice's first move will create a single strand of connected links of length at least 3. Bob takes one or two links to create two strands of equal length. He moves to maintain the invariant (after his turn) that there is a complete matching between remaining strands (a pairing of all the strands) that matches strands of equal length. This is initially true, since the matching that pairs the two strands of equal length is such a matching. Since every move of Alice is restricted to one strand, Bob can mimic her move on the matched strand. The matching can then be updated to match the strand(s) created by Alice with the strand(s) created by Bob. Alice can never win since Bob can always mimic her move on the matched strand.

5. (a) The total number of three-character strings is 256^3. No matter what hash function we use to map these strings into a table of size $2^{16}-1$, there must be some slot that receives $\lceil 256^3/(2^{16}-1)\rceil = 257$ strings. This is an application of the general pigeonhole principle.

(b) Any permutation that can be obtained by repeatedly swapping characters in positions i and j such that $j - i$ is even results in a string that collides with the original string.
Notice that $m = 2^{16} - 1 = 256^2 - 1$. Thus if i is even, $s_i \cdot 256^i \bmod m = s_i \bmod m$, while if i is odd, $s_i \cdot 256^i \bmod m = s_i \cdot 256 \bmod m$. One way to see this is:

$$\begin{aligned} s_i \cdot 256^i \bmod m &= (s_i \cdot 256^i - s_i \cdot 256^{i-2} \cdot m) \bmod m \\ &= (s_i \cdot 256^i - s_i \cdot 256^{i-2} \cdot (256^2 - 1)) \bmod m \\ &= s_i \cdot 256^{i-2} \bmod m. \end{aligned}$$

By repeatedly applying this, we decrease the power of 256 to either 0, if i is even, or 1, if i is odd. So,

$$\begin{aligned} &(s_0 + s_1 \cdot 256 + s_2 \cdot 256^2 + \cdots + s_k \cdot 256^k) \bmod m \\ &= ((s_0 + s_2 + s_4 + \cdots + s_k) + (s_1 + s_3 + s_5 + \cdots + s_{k-1}) \cdot 256) \bmod m \text{ (if } k \text{ is even)} \\ &= ((s_0 + s_2 + s_4 + \cdots + s_{k-1}) + (s_1 + s_3 + s_5 + \cdots + s_k) \cdot 256) \bmod m \text{ (if } k \text{ is odd)} \end{aligned}$$

From this, it follows that interchanging characters s_i and s_j such that $j - i$ is even doesn't alter the hash value. In other words, we can permute characters in even positions $(s_0, s_2, \dots)$ and/or permute characters in odd positions $(s_1, s_3, \dots)$ without changing the value of the hash function.