Here are the solutions to the given problems in C++:

1. **Reverse a Singly Linked List:**

```cpp
struct ListNode {
    int val;
    ListNode *next;
    ListNode(int x) : val(x), next(nullptr) {}
};

ListNode* reverseList(ListNode* head) {
    ListNode* prev = nullptr;
    ListNode* curr = head;
    while (curr) {
        ListNode* nextTemp = curr->next;
        curr->next = prev;
        prev = curr;
        curr = nextTemp;
    }
    return prev;
}
```

2. **Length of the Longest Substring Without Repeating Characters:**

```cpp
#include <unordered_map>
#include <string>
using namespace std;

int lengthOfLongestSubstring(string s) {
    unordered_map<char, int> charIndex;
    int maxLength = 0, start = 0;
    for (int end = 0; end < s.length(); end++) {
        if (charIndex.find(s[end]) != charIndex.end()) {
```

```
            start = max(start, charIndex[s[end]] + 1);
        }
        charIndex[s[end]] = end;
        maxLength = max(maxLength, end - start + 1);
    }
    return maxLength;
}
```

3. **Maximum Path Sum in a Binary Tree:**

```cpp
struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

int maxPathSumHelper(TreeNode* root, int& globalMax) {
    if (!root) return 0;
    int left = max(0, maxPathSumHelper(root->left, globalMax));
    int right = max(0, maxPathSumHelper(root->right, globalMax));
    globalMax = max(globalMax, root->val + left + right);
    return root->val + max(left, right);
}

int maxPathSum(TreeNode* root) {
    int globalMax = INT_MIN;
    maxPathSumHelper(root, globalMax);
    return globalMax;
}
```

4. **Serialize and Deserialize a Binary Tree:**

```cpp
#include <string>
#include <queue>
#include <sstream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
};

class Codec {
public:
    string serialize(TreeNode* root) {
        if (!root) return "#";
        return to_string(root->val) + "," + serialize(root->left) + "," + serialize(root->right);
    }

    TreeNode* deserialize(string data) {
        queue<string> nodes;
        stringstream ss(data);
        string item;
        while (getline(ss, item, ',')) {
            nodes.push(item);
        }
        return deserializeHelper(nodes);
    }

private:
    TreeNode* deserializeHelper(queue<string>& nodes) {
        string val = nodes.front();
        nodes.pop();
        if (val == "#") return nullptr;
        TreeNode* node = new TreeNode(stoi(val));
```

```
        node->left = deserializeHelper(nodes);
        node->right = deserializeHelper(nodes);
        return node;
    }
};
```

5. **Rotate an Array to the Right by k Steps:**

```cpp
#include <vector>
using namespace std;

void rotate(vector<int>& nums, int k) {
    int n = nums.size();
    k = k % n;
    reverse(nums.begin(), nums.end());
    reverse(nums.begin(), nums.begin() + k);
    reverse(nums.begin() + k, nums.end());
}
```

6. **Find the Factorial of a Given Number:**

```cpp
unsigned long long factorial(int n) {
    if (n <= 1) return 1;
    return n * factorial(n - 1);
}
```

7. **Compute the Sum of the Digits of a Given Number:**

```cpp
int sumOfDigits(int n) {
    int sum = 0;
```

```
    while (n != 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}
```

8. **Find the Greatest Common Divisor (GCD) of Two Numbers:**

```cpp
int gcd(int a, int b) {
    if (b == 0) return a;
    return gcd(b, a % b);
}
```

9. **Find the Maximum Difference Between Any Two Elements in an Array:**

```cpp
#include <vector>
#include <algorithm>
using namespace std;

int maxDifference(vector<int>& nums) {
    if (nums.size() < 2) return 0;
    int minElement = nums[0];
    int maxDiff = 0;
    for (int i = 1; i < nums.size(); i++) {
        maxDiff = max(maxDiff, nums[i] - minElement);
        minElement = min(minElement, nums[i]);
    }
    return maxDiff;
}
```

10. **Check if a Given String Contains Only Alphabetic Characters:**

```cpp
#include <string>
using namespace std;

bool isAlphabetic(const string& s) {
    for (char c : s) {
        if (!isalpha(c)) return false;
    }
    return true;
}
```