61. Rotate List

C++ solution:

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     ListNode *next;
 *     ListNode(int x) : val(x), next(NULL) {}
 * };
 */
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {

        if (!head)
        {
            return head;
        }

        ListNode* end = head;
        int n = 1;
        while (end->next)
        {
            ++n;
            end = end->next;
        }

        k %= n;

        if (k == 0)
        {
            return head;
        }

        ListNode* current = head;
        int position = 0;
        while (current)
        {
            ++position;
            if (position == n - k)
            {
                break;
            }
            current = current->next;
        }

        ListNode* result = current->next;
        current->next = nullptr;
        end->next = head;

        return result;
    }
};
```

## 62. Unique Paths

C++ solution:

```
class Solution {
public:
    int uniquePaths(int m, int n) {

        int total_steps = m + n - 2;
        int selected_steps = m > n ? n - 1 : m - 1;

        long long int result = 1;
        for (int i = 0; i < selected_steps; ++i)
        {
            result = result * (total_steps - i) / (1 + i);
        }

        return result;
    }
};
```

## 63. Unique Paths II

C++ solution:

```
class Solution {
public:
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {

        int rows = obstacleGrid.size();
        int cols = obstacleGrid[0].size();

        if (obstacleGrid[0][0] == 1 || obstacleGrid[rows - 1][cols - 1] == 1)
        {
            return 0;
        }

        vector<vector<unsigned long long int>> nums_paths(rows + 1, vector<unsigned lo
ng long int>(cols + 1));

        for (int r = rows; r >= 0; --r)
        {
            for (int c = cols; c >= 0; --c)
            {
                if (r == rows || c == cols || obstacleGrid[r][c] == 1)
                {
                    nums_paths[r][c] = 0;
                }
                else if (r < rows - 1 || c < cols - 1)
                {
                    nums_paths[r][c] = nums_paths[r + 1][c] + nums_paths[r][c + 1];
                }
                else
                {
                    nums_paths[r][c] = 1;
                }
            }
        }

        return nums_paths[0][0];
    }
};
```

## 64. Minimum Path Sum

C++ solution:

```
class Solution {
public:
    int minPathSum(vector<vector<int>>& grid) {

        int rows = grid.size();
        int cols = grid[0].size();

        vector<vector<int>> min_sums(rows + 1, vector<int>(cols + 1));

        for (int r = rows; r >= 0; --r)
        {
            for (int c = cols; c >= 0; --c)
            {
                if (r == rows - 1 && c == cols - 1)
                {
                    min_sums[r][c] = grid[r][c];
                    continue;
                }
                if (r == rows || c == cols)
                {
                    min_sums[r][c] = INT_MAX;
                }
                else
                {
                    min_sums[r][c] = min(min_sums[r + 1][c], min_sums[r][c + 1]) + grid[r][c];
                }
            }
        }

        return min_sums[0][0];
    }
};
```

65. Valid Number

C++ solution:

```
class Solution {
public:
    bool isNumber(string s) {

        int n = s.length();

        bool exp = false;
        bool flo = false;

        int start = 0;
        while (start < n && s[start] == ' ')
        {
            ++start;
        }

        int end = n - 1;
        while (end >= 0 && s[end] == ' ')
        {
            --end;
        }

        if (start > end)
        {
            return false;
        }

        for (int i = start; i <= end; ++i)
        {
            if (s[i] == '+' || s[i] == '-')
            {
                if (i == end)
                {
                    return false;
                }
                if (i > start && s[i - 1] != 'e')
                {
                    return false;
                }
                if ((s[i + 1] < '0' || s[i + 1] > '9') && s[i + 1] != '.')
                {
                    return false;
                }
            }
            else if (s[i] == 'e')
            {
                if (i == start || i == end || exp)
                {
                    return false;
                }
                exp = true;
            }
            else if (s[i] == '.')
            {
                if (exp || flo)
                {
                    return false;
                }
                if (start == end)
                {
                    return false;
                }
                if (i == start && (s[start + 1] < '0' || s[start + 1] > '9'))
                {
                    return false;
                }
                if (i == end && (s[end - 1] < '0' || s[end - 1] > '9'))
                {
                    return false;
                }
                if (i > start && i < end && (s[i - 1] < '0' || s[i - 1] > '9') && (s[i
 + 1] < '0' || s[i + 1] > '9'))
                {
                    return false;
                }
                flo = true;
            }
            else if (s[i] < '0' || s[i] > '9')
            {
                return false;
            }
        }

        return true;
    }
};
```

66. Plus One

C++ solution:

```
class Solution {
public:
    vector<int> plusOne(vector<int>& digits) {
        int n = digits.size();

        int i = n - 1;
        while (i >= 0 && digits[i] == 9)
        {
            --i;
        }

        if (i == -1)
        {
            vector<int> result(n + 1, 0);
            result[0] = 1;
            return result;
        }

        vector<int> result(n, 0);
        for (int j = 0; j < i; ++j)
        {
            result[j] = digits[j];
        }
        result[i] = digits[i] + 1;

        return result;
    }
};
```

67. Add Binary

C++ solution:

```
class Solution {
public:
    string addBinary(string a, string b) {

        int na = a.length();
        int nb = b.length();

        int ia = na - 1, ib = nb - 1;

        string sum_reverse;

        int current_sum = 0, extra = 0;
        char current_char;

        while (ia >= 0 && ib >= 0)
        {
            current_sum = (a[ia] - '0') + (b[ib] - '0') + extra;
            extra = current_sum / 2;
            current_char = current_sum % 2 + '0';
            sum_reverse.insert(sum_reverse.end(), current_char);
            --ia;
            --ib;
        }

        while (ia >= 0)
        {
            current_sum = (a[ia] - '0') + extra;
            extra = current_sum / 2;
            current_char = current_sum % 2 + '0';
            sum_reverse.insert(sum_reverse.end(), current_char);
            --ia;
        }

        while (ib >= 0)
        {
            current_sum = (b[ib] - '0') + extra;
            extra = current_sum / 2;
            current_char = current_sum % 2 + '0';
            sum_reverse.insert(sum_reverse.end(), current_char);
            --ib;
        }

        if (extra == 1)
        {
            sum_reverse.insert(sum_reverse.end(), '1');
        }

        reverse(sum_reverse.begin(), sum_reverse.end());

        return sum_reverse;
    }
};
```

68. Text Justification

C++ solution:

```
class Solution {
public:
    vector<string> fullJustify(vector<string>& words, int maxWidth) {

        int n = words.size();
        int start = 0, end = 0, length = 0;
        vector<string> result;

        while (start < n)
        {
            end = start;
            length = words[start].size();
            while (end < n - 1 && length + 1 + words[end + 1].size() <= maxWidth)
            {
                ++end;
                length = length + 1 + words[end].size();
            }

            string current_string = words[start];

            if (start == end)
            {
                for (int i = 0; i < maxWidth - length; ++i)
                {
                    current_string.insert(current_string.end(), ' ');
                }
            }
            else if (end == n - 1)
            {
                for (int i = 0; i < end - start; ++i)
                {
                    current_string.insert(current_string.end(), ' ');
                    current_string += words[start + i + 1];
                }
                while (current_string.length() < maxWidth)
                {
                    current_string.insert(current_string.end(), ' ');
                }
            }
            else
            {
                int space_number = maxWidth - length + end - start;
                int max_extra_index = space_number % (end - start);
                int average_space_number = space_number / (end - start);

                string space;
                for (int i = 0; i < average_space_number; ++i)
                {
                    space.insert(space.end(), ' ');
                }

                for (int i = 0; i < end - start; ++i)
                {
                    if (i < max_extra_index)
                    {
                        current_string.insert(current_string.end(), ' ');
                    }
                    current_string += space;
                    current_string += words[start + i + 1];
                }
            }

            result.insert(result.end(), current_string);

            start = end + 1;
        }

        return result;
    }
};
```

69. Sqrt(x)

C++ solution:

```
class Solution {
public:
    int mySqrt(int x) {

        if (x == 0)
        {
            return 0;
        }

        unsigned long long int result = 0, current = 0;

        while (true)
        {
            current = 1;

            while ((result + current) * (result + current) <= x)
            {
                current <<= 1;
            }

            if (current == 1)
            {
                break;
            }
            result += (current >> 1);

        }

        return result;
    }
};
```

70. Climbing Stairs

C++ solution:

```
class Solution {
public:
    int climbStairs(int n) {
        int ways[n + 1];

        for (int i = 0; i < n + 1; ++i)
        {
            if (i > 1)
            {
                ways[i] = ways[i - 2] + ways[i - 1];
            }
            else
            {
                ways[i] = 1;
            }
        }

        return ways[n];
    }
};
```