1. #include <stdio.h>

```c
#include <stdlib.h>

int* plusOne(int* digits, int digitsSize, int* returnSize) {
    // Iterate through the digits in reverse order
    for (int i = digitsSize - 1; i >= 0; --i) {
        // Increment the current digit
        digits[i]++;

        // If the digit becomes 10, set it to 0 and continue to the next digit
        if (digits[i] == 10) {
            digits[i] = 0;
        } else {
            // If the digit is less than 10, no need to carry over, break the loop
            break;
        }
    }

    // If the most significant digit became 10, add a new digit at the beginning
    if (digits[0] == 0) {
        *returnSize = digitsSize + 1;
        int* result = (int*)malloc((*returnSize) * sizeof(int));
        result[0] = 1;
        for (int i = 1; i < *returnSize; ++i) {
            result[i] = digits[i - 1];
        }
        return result;
    } else {
        *returnSize = digitsSize;
        return digits;
    }
```

```
}

int main() {
    // Example 1
    int digits1[] = {1, 2, 3};
    int size1;
    int* result1 = plusOne(digits1, 3, &size1);

    printf("Example 1:\nInput: [1, 2, 3]\nOutput: [");
    for (int i = 0; i < size1; ++i) {
        printf("%d", result1[i]);
        if (i < size1 - 1) {
            printf(", ");
        }
    }
    printf("]\n");

    free(result1);

    // Example 2
    int digits2[] = {9};
    int size2;
    int* result2 = plusOne(digits2, 1, &size2);

    printf("\nExample 2:\nInput: [9]\nOutput: [");
    for (int i = 0; i < size2; ++i) {
        printf("%d", result2[i]);
        if (i < size2 - 1) {
            printf(", ");
        }
    }
    printf("]\n");

    free(result2);

    return 0;
}
```

2.
```
#include <stdio.h>
#include <stdbool.h>

bool canJump(int* nums, int numsSize) {
    int maxReach = 0;

    for (int i = 0; i < numsSize; ++i) {
        // If the current index is beyond the maximum reach, return false
        if (i > maxReach) {
            return false;
        }

        // Update the maximum reach based on the current element and index
        maxReach = (i + nums[i]) > maxReach ? (i + nums[i]) : maxReach;

        // If the maximum reach is beyond or at the last index, return true
        if (maxReach >= numsSize - 1) {
            return true;
        }
    }

    return false;
}
```

```
int main() {
    // Example 1
    int nums1[] = {2, 3, 1, 1, 4};
    bool result1 = canJump(nums1, 5);
    printf("Example 1: %s\n", result1 ? "true" : "false");

    // Example 2
    int nums2[] = {3, 2, 1, 0, 4};
    bool result2 = canJump(nums2, 5);
    printf("Example 2: %s\n", result2 ? "true" : "false");

    return 0;
}
```

3. #include <stdio.h>

```
int maxSubArray(int* nums, int numsSize) {
    int maxSum = nums[0];
    int currentSum = nums[0];

    for (int i = 1; i < numsSize; ++i) {
        // If adding the current element increases the sum, add it to the current subarray
        currentSum = (currentSum + nums[i] > nums[i]) ? currentSum + nums[i] : nums[i];

        // Update the maximum sum if the current subarray has a larger sum
        maxSum = (currentSum > maxSum) ? currentSum : maxSum;
    }

    return maxSum;
}
```

```
int main() {
    // Example 1
    int nums1[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int result1 = maxSubArray(nums1, 9);
    printf("Example 1: %d\n", result1);

    // Example 2
    int nums2[] = {1};
    int result2 = maxSubArray(nums2, 1);
    printf("Example 2: %d\n", result2);

    // Example 3
    int nums3[] = {5, 4, -1, 7, 8};
    int result3 = maxSubArray(nums3, 5);
    printf("Example 3: %d\n", result3);

    return 0;
}
```