DAY-3
QUIZ-1

1. You are given a large integer represented as an integer array digits, where each digits[i] is the ith digit of the integer. The digits are ordered from most significant to least significant in left-to-right order. The large integer does not contain any leading 0's.

Increment the large integer by one and return the resulting array of digits.

Example 1:

Input: digits = [1,2,3]
Output: [1,2,4]
Explanation: The array represents the integer 123.
Incrementing by one gives 123 + 1 = 124.
Thus, the result should be [1,2,4].

Example 2:

Input: digits = [9]
Output: [1,0]
Explanation: The array represents the integer 9.
Incrementing by one gives 9 + 1 = 10.
Thus, the result should be [1,0].

CODE:

```
#include <stdio.h>
#include <stdlib.h>

// Function to increment an array of digits representing a large integer
void incrementDigits(int digits[], int size) {
    // Start from the least significant digit
    for (int i = size - 1; i >= 0; i--) {
        // Increment the current digit
        digits[i]++;

        // If there is no carry, we can stop
        if (digits[i] < 10) {
            break;
        } else {
            // Carry to the next higher digit
            digits[i] = 0;
        }
    }

    // If there is still a carry at the end, insert a new digit at the beginning
```

```
    if (digits[0] == 0) {
        printf("[");
        for (int i = 0; i <= size; i++) {
            printf("%d", digits[i]);
            if (i < size) {
                printf(", ");
            }
        }
        printf("]\n");
    } else {
        for (int i = 0; i < size; i++) {
            printf("%d", digits[i]);
            if (i < size - 1) {
                printf(", ");
            }
        }
        printf("\n");
    }
}

int main() {
    // Example 1
    int digits1[] = {1, 2, 3};
    int size1 = sizeof(digits1) / sizeof(digits1[0]);

    printf("Example 1:\n");
    printf("Input: [");
    for (int i = 0; i < size1; i++) {
        printf("%d", digits1[i]);
        if (i < size1 - 1) {
            printf(", ");
        }
    }
    printf("]\n");

    incrementDigits(digits1, size1);

    // Example 2
    int digits2[] = {9};
    int size2 = sizeof(digits2) / sizeof(digits2[0]);

    printf("\nExample 2:\n");
    printf("Input: [");
    for (int i = 0; i < size2; i++) {
        printf("%d", digits2[i]);
        if (i < size2 - 1) {
            printf(", ");
        }
    }
    printf("]\n");
```

```
    incrementDigits(digits2, size2);

    // Additional Example: Large integer
    int digits3[] = {9, 9, 9, 9, 9};
    int size3 = sizeof(digits3) / sizeof(digits3[0]);

    printf("\nAdditional Example:\n");
    printf("Input: [");
    for (int i = 0; i < size3; i++) {
        printf("%d", digits3[i]);
        if (i < size3 - 1) {
            printf(", ");
        }
    }
    printf("]\n");

    incrementDigits(digits3, size3);

    return 0;
}
```

OUTPUT:

```
Example 1:
Input: [1, 2, 3]
1, 2, 4

Example 2:
Input: [9]
[0, 1]
```

2. You are given an integer array nums. You are initially positioned at the array's first index, and each element in the array represents your maximum jump length at that position. Return true if you can reach the last index, or false otherwise.

Example 1:

Input: nums = [2,3,1,1,4]
Output: true
Explanation: Jump 1 step from index 0 to 1, then 3 steps to the last index.

Example 2:

Input: nums = [3,2,1,0,4]
Output: false
Explanation: You will always arrive at index 3 no matter what. Its maximum jump length is 0, which makes it impossible to reach the last index.

CODE:

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

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

    for (int i = 0; i < size; i++) {
        // If the current index is not reachable, return false
        if (i > maxReach) {
            return false;
        }

        // Update the maximum reachable index
        maxReach = (i + nums[i] > maxReach) ? i + nums[i] : maxReach;

        // If the last index is reachable, return true
        if (maxReach >= size - 1) {
            return true;
        }
    }

    return false;
}

int main() {
```

```
    // Example 1
    int nums1[] = {2, 3, 1, 1, 4};
    int size1 = sizeof(nums1) / sizeof(nums1[0]);
    printf("Example 1: %s\n", canJump(nums1, size1) ? "true" : "false");

    // Example 2
    int nums2[] = {3, 2, 1, 0, 4};
    int size2 = sizeof(nums2) / sizeof(nums2[0]);
    printf("Example 2: %s\n", canJump(nums2, size2) ? "true" : "false");

    return 0;
```

OUTPUT:

```
Example 1: true
Example 2: false
```

3. Given an integer array nums, find the subarray with the largest sum, and return its sum.

Example 1:

Input: nums = [-2,1,-3,4,-1,2,1,-5,4]
Output: 6
Explanation: The subarray [4,-1,2,1] has the largest sum 6.

Example 2:

Input: nums = [1]
Output: 1
Explanation: The subarray [1] has the largest sum 1.

Example 3:

Input: nums = [5,4,-1,7,8]
Output: 23
Explanation: The subarray [5,4,-1,7,8] has the largest sum 23.

CODE:

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

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

    for (int i = 1; i < size; i++) {
        // Update the current sum or start a new subarray
        currentSum = (currentSum + nums[i] > nums[i]) ? currentSum + nums[i] : nums[i];

        // Update the maximum sum
        maxSum = (currentSum > maxSum) ? currentSum : maxSum;
    }

    return maxSum;
}

int main() {
    // Example 1
    int nums1[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int size1 = sizeof(nums1) / sizeof(nums1[0]);
    printf("Example 1: %d\n", maxSubArray(nums1, size1));

    // Example 2
    int nums2[] = {1};
    int size2 = sizeof(nums2) / sizeof(nums2[0]);
    printf("Example 2: %d\n", maxSubArray(nums2, size2));

    // Example 3
    int nums3[] = {5, 4, -1, 7, 8};
    int size3 = sizeof(nums3) / sizeof(nums3[0]);
    printf("Example 3: %d\n", maxSubArray(nums3, size3));
```

```
    return 0;
}
```

OUTPUT: