**QUIZ 5**

**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].

#include <stdio.h>

#include <stdlib.h>

int\* increment(int\* digits, int n) {

int carry = 1;

for (int i = n - 1; i >= 0; i--) {

int total = digits[i] + carry;

digits[i] = total % 10;

carry = total / 10;

}

if (carry) {

int\* result = (int\*)malloc((n + 1) \* sizeof(int));

result[0] = carry;

for (int i = 1; i <= n; i++) {

result[i] = digits[i - 1];

}

return result;

}

return digits;

}

int main() {

int digits[] = {1, 2, 3};

int n = sizeof(digits) / sizeof(digits[0]);

int\* result = increment(digits, n);

for (int i = 0; i < n + (result != digits); i++) {

printf("%d ", result[i]);

}

if (result != digits) {

free(result);

}

return 0;

}

OUTPUT:

[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].

#include <stdio.h>

#include <stdlib.h>

int\* increment(int\* digits, int n) {

int carry = 1;

for (int i = n - 1; i >= 0; i--) {

int total = digits[i] + carry;

digits[i] = total % 10;

carry = total / 10;

}

if (carry) {

int\* result = (int\*)malloc((n + 1) \* sizeof(int));

result[0] = carry;

for (int i = 1; i <= n; i++) {

result[i] = digits[i - 1];

}

return result;

}

return digits;

}

int main() {

int digits[] = {9};

int n = sizeof(digits) / sizeof(digits[0]);

int\* result = increment(digits, n);

for (int i = 0; i < n + (result != digits); i++) {

printf("%d ", result[i]);

}

if (result != digits) {

free(result);

}

return 0;

}

OUTPUT:

[1,0]

**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.

#include <stdbool.h>

#include <stdio.h>

bool canJump(int\* nums, int numsSize) {

int maxReach = 0; // Keep track of the farthest index that can be reached

for (int i = 0; i < numsSize; i++) {

if (i > maxReach) {

return false;

}

maxReach = (i + nums[i]) > maxReach ? (i + nums[i]) : maxReach;

if (maxReach >= numsSize - 1) {

return true;

}

}

return false;

}

int main() {

int nums[] = {2, 3, 1, 1, 4};

int numsSize = sizeof(nums) / sizeof(nums[0]);

bool result = canJump(nums, numsSize);

printf(result ? "true\n" : "false\n");

return 0;

}

OUTPUT:

true

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.

#include <stdbool.h>

#include <stdio.h>

bool canJump(int\* nums, int numsSize) {

int maxReach = 0;

for (int i = 0; i < numsSize; i++) {

if (i > maxReach) {

return false;

}

maxReach = (i + nums[i]) > maxReach ? (i + nums[i]) : maxReach;

if (maxReach >= numsSize - 1) {

return true;

}

}

return false;

}

int main() {

int nums[] = {3, 2, 1, 0, 4};

int numsSize = sizeof(nums) / sizeof(nums[0]);

bool result = canJump(nums, numsSize);

printf(result ? "true\n" : "false\n");

return 0;

}

OUTPUT:

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.

#include <stdio.h>

int maxSubArray(int\* nums, int numsSize) {

if (numsSize == 0) {

return 0;

}

int currentMax = nums[0];

int overallMax = nums[0];

for (int i = 1; i < numsSize; i++)

currentMax = (nums[i] > currentMax + nums[i]) ? nums[i] : (currentMax + nums[i]);

overallMax = (currentMax > overallMax) ? currentMax : overallMax;

}

return overallMax;

}

int main() {

int nums[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};

int numsSize = sizeof(nums) / sizeof(nums[0]);

int result = maxSubArray(nums, numsSize);

printf("%d\n", result);

return 0;

}

OUTPUT:

6

Example 2:

Input: nums = [1]

Output: 1

Explanation: The subarray [1] has the largest sum 1.

#include <stdio.h>

int maxSubArray(int\* nums, int numsSize) {

if (numsSize == 0) {

return 0;

}

int currentMax = nums[0];

int overallMax = nums[0];

for (int i = 1; i < numsSize; i++) {

currentMax = (nums[i] > currentMax + nums[i]) ? nums[i] : (currentMax + nums[i]);

overallMax = (currentMax > overallMax) ? currentMax : overallMax;

}

return overallMax;

}

int main() {

int nums[] = {1};

int numsSize = sizeof(nums) / sizeof(nums[0]);

int result = maxSubArray(nums, numsSize);

printf("%d\n", result);

return 0;

}

OUTPUT:

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.

#include <stdio.h>

int maxSubArray(int\* nums, int numsSize) {

if (numsSize == 0) {

return 0;

}

int currentMax = nums[0];

int overallMax = nums[0];

for (int i = 1; i < numsSize; i++) {

currentMax = (nums[i] > currentMax + nums[i]) ? nums[i] : (currentMax + nums[i]);

overallMax = (currentMax > overallMax) ? currentMax : overallMax;

}

return overallMax;

}

int main() {

int nums[] = {5, 4, -1, 7, 8};

int numsSize = sizeof(nums) / sizeof(nums[0]);

int result = maxSubArray(nums, numsSize);

printf("%d\n", result);

return 0;

}

OUTPUT:

23