Problem 1: Climbing Stairs (Easy)
Problem Statement
You are climbing a staircase. It takes n steps to reach the top. Each time you can either climb 1 or 2 steps. In how many distinct ways can you climb to the top?

Solution
Dynamic Programming Solution
```
#include <stdio.h>

int climbStairs(int n) {
    if (n <= 1) {
        return 1;
    }
    int dp[n + 1];
    dp[0] = 1;
    dp[1] = 1;
    for (int i = 2; i <= n; i++) {
        dp[i] = dp[i - 1] + dp[i - 2];
    }
    return dp[n];
}

int main() {
    int n = 10;
    printf("Number of ways to climb %d stairs is %d\n", n,
climbStairs(n));
    return 0;
}
```

Problem 2: House Robber (Medium)
Problem Statement
You are a professional robber planning to rob houses along a street. Each house has a certain amount of money stashed, and you cannot rob two adjacent houses. Determine the maximum amount of money you can rob tonight without alerting the police.

Solution
Dynamic Programming Solution
```
#include <stdio.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}

int rob(int* nums, int numsSize) {
    if (numsSize == 0) return 0;
    if (numsSize == 1) return nums[0];

    int dp[numsSize];
    dp[0] = nums[0];
    dp[1] = max(nums[0], nums[1]);
```

```
    for (int i = 2; i < numsSize; i++) {
        dp[i] = max(dp[i - 1], nums[i] + dp[i - 2]);
    }

    return dp[numsSize - 1];
}

int main() {
    int nums[] = {1, 2, 3, 1};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    printf("Maximum amount of money that can be robbed is %d\n",
rob(nums, numsSize));
    return 0;
}
```

Problem 3: Coin Change (Medium)
Problem Statement
Given an integer array coins representing coins of different denominations and an integer amount representing a total amount of money, return the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Solution
Dynamic Programming Solution

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

int coinChange(int* coins, int coinsSize, int amount) {
    int dp[amount + 1];
    for (int i = 0; i <= amount; i++) {
        dp[i] = amount + 1;
    }
    dp[0] = 0;

    for (int i = 1; i <= amount; i++) {
        for (int j = 0; j < coinsSize; j++) {
            if (coins[j] <= i) {
                dp[i] = dp[i] < (dp[i - coins[j]] + 1) ? dp[i] : (dp[i -
coins[j]] + 1);
            }
        }
    }

    return dp[amount] > amount ? -1 : dp[amount];
}

int main() {
    int coins[] = {1, 2, 5};
    int coinsSize = sizeof(coins) / sizeof(coins[0]);
    int amount = 11;
```

```
    printf("Fewest number of coins needed to make up %d is %d\n", amount,
coinChange(coins, coinsSize, amount));
    return 0;
}
```

Problem 4: Longest Increasing Subsequence (Medium)
Problem Statement
Given an integer array nums, return the length of the longest strictly increasing subsequence.

Solution
Dynamic Programming Solution

```
#include <stdio.h>

int lengthOfLIS(int* nums, int numsSize) {
    if (numsSize == 0) return 0;

    int dp[numsSize];
    for (int i = 0; i < numsSize; i++) {
        dp[i] = 1;
    }

    int maxLength = 1;
    for (int i = 1; i < numsSize; i++) {
        for (int j = 0; j < i; j++) {
            if (nums[i] > nums[j]) {
                dp[i] = dp[i] > (dp[j] + 1) ? dp[i] : (dp[j] + 1);
            }
        }
        maxLength = maxLength > dp[i] ? maxLength : dp[i];
    }

    return maxLength;
}

int main() {
    int nums[] = {10, 9, 2, 5, 3, 7, 101, 18};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    printf("Length of the longest increasing subsequence is %d\n",
lengthOfLIS(nums, numsSize));
    return 0;
}
```

Problem 5: Subset Sum Problem (Medium)
Problem Statement
Given a set of positive integers and an integer sum, determine if there is a subset of the given set with a sum equal to the given sum.

Solution
Dynamic Programming Solution

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

bool isSubsetSum(int set[], int n, int sum) {
    bool subset[n + 1][sum + 1];

    for (int i = 0; i <= n; i++) {
        subset[i][0] = true;
    }

    for (int i = 1; i <= sum; i++) {
        subset[0][i] = false;
    }

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= sum; j++) {
            if (j < set[i - 1]) {
                subset[i][j] = subset[i - 1][j];
            } else {
                subset[i][j] = subset[i - 1][j] || subset[i - 1][j -
set[i - 1]];
            }
        }
    }

    return subset[n][sum];
}

int main() {
    int set[] = {3, 34, 4, 12, 5, 2};
    int sum = 9;
    int n = sizeof(set) / sizeof(set[0]);
    if (isSubsetSum(set, n, sum) == true)
        printf("Found a subset with given sum\n");
    else
        printf("No subset with given sum\n");
    return 0;
}
```

Problem 6: Maximum Subarray Sum (Easy)

Problem Statement

Given an integer array nums, find the contiguous subarray (containing at least one number) which has the largest sum and return its sum.

Solution

Dynamic Programming Solution

```c
#include <stdio.h>

int max(int a, int b) {
    return (a > b) ? a : b;
}
```

```
int maxSubArray(int* nums, int numsSize) {
    int max_so_far = nums[0];
    int curr_max = nums[0];

    for (int i = 1; i < numsSize; i++) {
        curr_max = max(nums[i], curr_max + nums[i]);
        max_so_far = max(max_so_far, curr_max);
    }

    return max_so

_far;
}

int main() {
    int nums[] = {-2, 1, -3, 4, -1, 2, 1, -5, 4};
    int numsSize = sizeof(nums) / sizeof(nums[0]);
    printf("Maximum subarray sum is %d\n", maxSubArray(nums, numsSize));
    return 0;
}
```