Problem 2240: Number of Ways to Buy Pens and Pencils

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

You are given an integer

total

indicating the amount of money you have. You are also given two integers

cost1

and

cost2

indicating the price of a pen and pencil respectively. You can spend

part or all

of your money to buy multiple quantities (or none) of each kind of writing utensil.

Return

the

number of distinct ways

you can buy some number of pens and pencils.

Example 1:

Input:

total = 20, cost1 = 10, cost2 = 5

Output:

9

Explanation:

The price of a pen is 10 and the price of a pencil is 5. - If you buy 0 pens, you can buy 0, 1, 2, 3, or 4 pencils. - If you buy 1 pen, you can buy 0, 1, or 2 pencils. - If you buy 2 pens, you cannot buy any pencils. The total number of ways to buy pens and pencils is 5 + 3 + 1 = 9.

Example 2:

Input:

total = 5, cost1 = 10, cost2 = 10

Output:

1

Explanation:

The price of both pens and pencils are 10, which cost more than total, so you cannot buy any writing utensils. Therefore, there is only 1 way: buy 0 pens and 0 pencils.

Constraints:

1 <= total, cost1, cost2 <= 10

6

Code Snippets

C++:

```
class Solution {
public:
long long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
};
```

Java:

```
class Solution {
public long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
}
```

Python3:

```
class Solution:
def waysToBuyPensPencils(self, total: int, cost1: int, cost2: int) -> int:
```

Python:

```
class Solution(object):
def waysToBuyPensPencils(self, total, cost1, cost2):
"""
:type total: int
:type cost1: int
:type cost2: int
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} total
* @param {number} cost1
* @param {number} cost2
* @return {number}
*/
```

```
var waysToBuyPensPencils = function(total, cost1, cost2) {

};
```

TypeScript:

```
function waysToBuyPensPencils(total: number, cost1: number, cost2: number):
number {

};
```

C#:

```
public class Solution {
public long WaysToBuyPensPencils(int total, int cost1, int cost2) {

}
}
```

C:

```
long long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
```

Go:

```
func waysToBuyPensPencils(total int, cost1 int, cost2 int) int64 {

}
```

Kotlin:

```
class Solution {
fun waysToBuyPensPencils(total: Int, cost1: Int, cost2: Int): Long {

}
}
```

Swift:

```
class Solution {
func waysToBuyPensPencils(_ total: Int, _ cost1: Int, _ cost2: Int) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn ways_to_buy_pens_pencils(total: i32, cost1: i32, cost2: i32) -> i64 {

}
}
```

Ruby:

```
# @param {Integer} total
# @param {Integer} cost1
# @param {Integer} cost2
# @return {Integer}
def ways_to_buy_pens_pencils(total, cost1, cost2)

end
```

PHP:

```
class Solution {

/**
* @param Integer $total
* @param Integer $cost1
* @param Integer $cost2
* @return Integer
*/
function waysToBuyPensPencils($total, $cost1, $cost2) {

}
}
```

Dart:

```
class Solution {
int waysToBuyPensPencils(int total, int cost1, int cost2) {
```

```
}
}
```

Scala:

```
object Solution {
def waysToBuyPensPencils(total: Int, cost1: Int, cost2: Int): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec ways_to_buy_pens_pencils(total :: integer, cost1 :: integer, cost2 ::
integer) :: integer
def ways_to_buy_pens_pencils(total, cost1, cost2) do

end
end
```

Erlang:

```
-spec ways_to_buy_pens_pencils(Total :: integer(), Cost1 :: integer(), Cost2
:: integer()) -> integer().
ways_to_buy_pens_pencils(Total, Cost1, Cost2) ->
.
```

Racket:

```
(define/contract (ways-to-buy-pens-pencils total cost1 cost2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Number of Ways to Buy Pens and Pencils
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public:
long long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
};
```

Java Solution:

```
/**
* Problem: Number of Ways to Buy Pens and Pencils
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
}
```

Python3 Solution:

```
"""
Problem: Number of Ways to Buy Pens and Pencils
Difficulty: Medium
Tags: math
```

```
Approach: Optimized algorithm based on problem constraints
Time Complexity: O(n) to O(n^2) depending on approach
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def waysToBuyPensPencils(self, total: int, cost1: int, cost2: int) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def waysToBuyPensPencils(self, total, cost1, cost2):
"""
:type total: int
:type cost1: int
:type cost2: int
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Number of Ways to Buy Pens and Pencils
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} total
* @param {number} cost1
* @param {number} cost2
* @return {number}
*/
var waysToBuyPensPencils = function(total, cost1, cost2) {
```

```
};
```

TypeScript Solution:

```
/**
* Problem: Number of Ways to Buy Pens and Pencils
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

function waysToBuyPensPencils(total: number, cost1: number, cost2: number):
number {

};
```

C# Solution:

```
/*
* Problem: Number of Ways to Buy Pens and Pencils
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public long WaysToBuyPensPencils(int total, int cost1, int cost2) {

}
}
```

C Solution:

```
/*
* Problem: Number of Ways to Buy Pens and Pencils
```

```
* Difficulty: Medium
* Tags: math
*
* Approach: Optimized algorithm based on problem constraints
* Time Complexity: O(n) to O(n^2) depending on approach
* Space Complexity: O(1) to O(n) depending on approach
*/

long long waysToBuyPensPencils(int total, int cost1, int cost2) {

}
```

Go Solution:

```
// Problem: Number of Ways to Buy Pens and Pencils
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

func waysToBuyPensPencils(total int, cost1 int, cost2 int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun waysToBuyPensPencils(total: Int, cost1: Int, cost2: Int): Long {

}
}
```

Swift Solution:

```
class Solution {
func waysToBuyPensPencils(_ total: Int, _ cost1: Int, _ cost2: Int) -> Int {

}
}
```

Rust Solution:

```
// Problem: Number of Ways to Buy Pens and Pencils
// Difficulty: Medium
// Tags: math
//
// Approach: Optimized algorithm based on problem constraints
// Time Complexity: O(n) to O(n^2) depending on approach
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn ways_to_buy_pens_pencils(total: i32, cost1: i32, cost2: i32) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer} total
# @param {Integer} cost1
# @param {Integer} cost2
# @return {Integer}
def ways_to_buy_pens_pencils(total, cost1, cost2)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $total
* @param Integer $cost1
* @param Integer $cost2
* @return Integer
*/
function waysToBuyPensPencils($total, $cost1, $cost2) {

}
}
```

Dart Solution:

```
class Solution {
int waysToBuyPensPencils(int total, int cost1, int cost2) {

}
}
```

Scala Solution:

```
object Solution {
def waysToBuyPensPencils(total: Int, cost1: Int, cost2: Int): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec ways_to_buy_pens_pencils(total :: integer, cost1 :: integer, cost2 ::
integer) :: integer
def ways_to_buy_pens_pencils(total, cost1, cost2) do

end
end
```

Erlang Solution:

```
-spec ways_to_buy_pens_pencils(Total :: integer(), Cost1 :: integer(), Cost2
:: integer()) -> integer().
ways_to_buy_pens_pencils(Total, Cost1, Cost2) ->
.
```

Racket Solution:

```
(define/contract (ways-to-buy-pens-pencils total cost1 cost2)
(-> exact-integer? exact-integer? exact-integer? exact-integer?)
)
```