Write a function that returns 1 if the input integer is a prime number, otherwise return 0.

Prototype: int is_prime_number(int n);

```
julien@ubuntu:~/0x08. Recursion$ cat 6-main.c
#include "main.h"
#include <stdio.h>

/**
* main - check the code
*
* Return: Always 0.
*/
int main(void)
{
int r;

r = is_prime_number(1);
printf("%d\n", r);
r = is_prime_number(1024);
printf("%d\n", r);
r = is_prime_number(16);
printf("%d\n", r);
r = is_prime_number(17);
printf("%d\n", r);
r = is_prime_number(25);
printf("%d\n", r);
r = is_prime_number(-1);
printf("%d\n", r);
r = is_prime_number(113);
printf("%d\n", r);
r = is_prime_number(7919);
printf("%d\n", r);
return (0);
}
julien@ubuntu:~/0x08. Recursion$ gcc -Wall -pedantic -Werror -Wextra -std=gnu89 6-main.c 6-is_prime_number.c -o 6-prime
julien@ubuntu:~/0x08. Recursion$ ./6-prime
0
0
0
1
0
0
1
1
```

my solution

```
#include "main.h"
/**
* is_prime - check for prime number
* @number: the number to check
* @test: the test number
* Return: 1 if the number is prime 0 else
*/
int is_prime(int number, int test)
{
if (number == test)
{
return (1);
}
if (number % 2 == 0)
{
return (0);
}

return (is_prime(number, test + 1));
}

/**
* is_prime_number - a function that returns 1
* if the input integer is a prime number, otherwise return 0
* @n: the number as input
* Return: a function that returns 1
* by ramzy
*/
int is_prime_number(int n)
{
if (n <= 1)
{
return (0);
}

return (is_prime(n, 2));
}
```

breack down the solution

Certainly! Let's break down the solution you provided into its main components and explain what each part does.

Function: `is_prime`

```
int is_prime(int number, int test)
{
    if (number == test)
    {
        return (1);
    }
    if (number % 2 == 0)
    {
        return (0);
    }

    return (is_prime(number, test + 1));
}
```

- **Purpose**: Determines if `number` is a prime number by testing divisibility with integers starting from `test`.
- **Arguments**:
  - `number`: The number being tested for primality.
  - `test`: The current divisor being tested.
- **Flow**:
  - If `number` is equal to `test`, this means that no numbers less than `number` other than 1 and `number` itself could divide `number`. Hence, `number` is prime, and the function returns 1.
  - If `number` is divisible by 2, it returns 0. However, this check seems to be redundant because by the time `test` has reached the value of `number`, the number would already have been identified as non-prime. Also, this function will incorrectly return 0 for the prime number 2.
  - Otherwise, the function recursively calls itself with `test + 1` as the next divisor.

Function: `is_prime_number`

```
int is_prime_number(int n)
{
    if (n <= 1)
    {
        return (0);
    }

    return (is_prime(n, 2));
}
```

- **Purpose**: Acts as the public-facing function. It sets up the recursive function to determine if `n` is a prime number.
- **Arguments**:
  - `n`: The number to be checked if it's prime.
- **Flow**:
  - If `n` is less than or equal to 1, the function immediately returns 0 (since neither 0 nor 1 are prime numbers).
  - Otherwise, the function calls the `is_prime` function with `n` and starts testing divisibility from 2.

Issues with the solution:

- The function will give incorrect results for the number 2. The function `is_prime` will return 0 for 2, which is incorrect.

- The function will not be efficient for larger prime numbers as it will test all numbers from 2 up to the number itself. A more efficient approach would be to only test divisors up to the square root of the number, as any non-prime number will have a divisor less than or equal to its square root.
- Also, once a divisor has been found, the function should immediately return 0 instead of continuing to check other divisors.

To improve the function:

1. Handle the case for the number 2 separately.
2. Check divisors only up to the square root of `n`.
3. Return 0 immediately if a divisor is found.

explain it with n = 17 , n = 16 , -1

Of course! Let's walk through the solution's process using the examples of $n = 17$n = 17, $n = 16$n = 16, and $n = -1$n = -1:

---

1. $n = 17$**n = 17:**

When `is_prime_number(17)` is called:

- First, the function checks if nn is less than or equal to 1. Since 17 is greater than 1, we proceed.
- Then, the function `is_prime(17, 2)` is called:
  1. The function checks if the `number` (17) is equal to `test` (2). Since it is not, we proceed.
  2. The function checks if the number 17 is divisible by 2. It isn't, so we proceed.
  3. The function then calls itself recursively with `test + 1`, making the next call `is_prime(17, 3)`.
  4. This process repeats, increasing the value of `test` each time until `test` becomes equal to `number` (17 in this case). No value from 2 to 16 divides 17, so when `test` is equal to 17, the function returns 1, indicating 17 is a prime number.

---

2. $n = 16$**n = 16:**

When `is_prime_number(16)` is called:

- The function checks if nn is less than or equal to 1. Since 16 is greater than 1, we proceed.
- The function `is_prime(16, 2)` is called:
  1. It checks if `number` (16) is equal to `test` (2). Since it is not, we proceed.
  2. The function checks if the number 16 is divisible by 2. Since 16 is divisible by 2, the function immediately returns 0, indicating 16 is not a prime number.

---

3. $n = -1$n = -1:

When `is_prime_number(-1)` is called:

- The function checks if nn is less than or equal to 1. Since -1 is less than 1, the function immediately returns 0 without any further checks, indicating that -1 is not a prime number.

In summary:

- For $n = 17$n = 17, the function correctly identifies it as a prime number and returns 1.
- For $n = 16$n = 16, the function correctly identifies it as not a prime number and returns 0.
- For $n = -1$n = -1, the function correctly identifies it as not a prime number and returns 0.