

Write a function that returns the natural square root of a number.

Prototype: `int _sqrt_recursion(int n);`

If `n` does not have a natural square root, the function should return -1

FYI: The standard library provides a different function: `sqrt`. Run `man sqrt` to learn more.

```
julien@ubuntu:~/0x08. Recursion$ cat 5-main.c
```

```
#include "main.h"
```

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

```
/**
```

```
 * main - check the code
```

```
 */
```

```
 * Return: Always 0.
```

```
 */
```

```
int main(void)
```

```
{
```

```
    int r;
```

```
    r = _sqrt_recursion(1);
```

```
    printf("%d\n", r);
```

```
    r = _sqrt_recursion(1024);
```

```
    printf("%d\n", r);
```

```
    r = _sqrt_recursion(16);
```

```
    printf("%d\n", r);
```

```
    r = _sqrt_recursion(17);
```

```
    printf("%d\n", r);
```

```
    r = _sqrt_recursion(25);
```

```
    printf("%d\n", r);
```

```
    r = _sqrt_recursion(-1);
```

```
    printf("%d\n", r);
```

```
    return (0);
```

```
}
```

```
julien@ubuntu:~/0x08. gcc -Wall -pedantic -Werror -Wextra -std=gnu89
```

```
5-main.c 5-sqrt_recursion.c -o 5-sqrt
```

```
julien@ubuntu:~/0x08. Recursion$ ./5-sqrt
```

```
1
```

```
32
```

```
4
```

```
-1
```

```
5
```

```
-1
```

don't gives any solution and I want to know what is the the natural square root of a number.

The natural square root of a number is the non-negative root of that number. If the square root of a number isn't an integer, then it doesn't have a "natural" square root (in the context you provided).

To find the natural square root of a number without using the built-in `sqrt` function or any external libraries, we can use a recursive approach:

- 1 Start with a guess for the square root.
- 2 Check if the square of the guess is equal to the number.
- 3 If the square of the guess is less than the number, then our guess is too low and we need to increase it.
- 4 If the square of the guess is greater than the number, then our guess is too high and we return `-1` since the number doesn't have a natural square root.

Here's the function to find the natural square root of a number recursively:

```
#include "main.h"

/**
 * helper - Recursive helper function to check if a number has a square root.
 * @n: The number to find the square root of.
 * @guess: Current guess for the square root.
 *
 * Return: The square root if it exists, otherwise -1.
 */
int helper(int n, int guess)
{
    if (guess * guess == n)
        return guess;
    if (guess * guess > n)
        return -1;
    return helper(n, guess + 1);
}

/**
 * _sqrt_recursion - Returns the natural square root of a number.
 * @n: The number to find the square root of.
 *
 * Return: The square root if it exists, otherwise -1.
 */
int _sqrt_recursion(int n)
{
    if (n < 0)
        return -1;
    return helper(n, 1);
}
```

This function uses a helper function to keep track of the current guess for the square root, and then checks if the guess squared is equal to `n` or not.

The output from the provided test cases will match the expected output. However, it's worth noting that this isn't the most efficient method to find the square root, especially for larger numbers, but it does work for the context you provided.

break down the code please

Of course! Let's break down the provided code step by step.

Overview:

We've defined two functions:

- 1 `_sqrt_recursion(int n)`: This is the main function that calls a helper function to recursively compute the natural square root.
- 2 `helper(int n, int guess)`: A recursive helper function to determine if the number has a natural square root.

1. The `helper` function:

```
int helper(int n, int guess)
{
    if (guess * guess == n)
        return guess;
    if (guess * guess > n)
        return -1;
    return helper(n, guess + 1);
}
```

Explanation:

- This function takes two parameters:
 - `n`: The number for which we want to determine the natural square root.
 - `guess`: Our current estimate or guess for the square root.
- The function uses a recursive approach to check if `n` has a natural square root:
 - 1 `if (guess * guess == n)`: We check if the square of our current guess equals `n`. If so, we've found the natural square root and we return `guess`.
 - 2 `if (guess * guess > n)`: We check if the square of our guess is greater than `n`. If it is, then it means `n` doesn't have a natural square root (since we've passed the point where it could have one), and we return `-1`.
 - 3 `return helper(n, guess + 1)`: If neither of the above conditions hold, it means we need to try a higher guess. So, we recursively call the helper function with an incremented guess (`guess + 1`).

2. The `_sqrt_recursion` function:

```
int _sqrt_recursion(int n)
{
    if (n < 0)
        return -1;
    return helper(n, 1);
}
```

Explanation:

- This function acts as the main interface for the user to find the natural square root.
- ❶ `if (n < 0)`: We first check if the input number is negative. If it is, we directly return `-1` because negative numbers do not have a natural square root.
- ❷ `return helper(n, 1)`: If `n` is not negative, we call our helper function starting with a guess of `1` (since `1` is the smallest valid natural square root for positive numbers).

Summary:

The idea behind the code is to iteratively guess the square root. We start with the smallest possible guess (`1`) and keep increasing our guess until we either find a number whose square is equal to the input number or we surpass the input number (indicating that it doesn't have a natural square root).

Using recursion for this task is a choice for educational purposes, as there are more efficient algorithms for finding the square root. However, this recursive method serves well to illustrate the basics of recursive problem-solving.

break down the all details `n = 64`

Sure! Let's walk through the code for `_sqrt_recursion(64)`.

When you call `_sqrt_recursion(64)`:

`_sqrt_recursion` function:

- ❶ The first check is `if (n < 0)`, which tests if the input number `n` is less than `0`. In our case, `n = 64`, so this condition is `false`.
- ❷ Next, the function calls `helper(n, 1)` - effectively becoming `helper(64, 1)`.

Now, we move to the `helper` function with parameters `n = 64` and `guess = 1`.

First iteration in `helper`:

- `if (guess * guess == n)`:
 - This translates to `if (1 * 1 == 64)`. This condition is `false`.
- `if (guess * guess > n)`:
 - This translates to `if (1 * 1 > 64)`. This condition is also `false`.
- So, the function proceeds to recursively call itself with an incremented guess: `helper(64, 2)`.

Second iteration with `guess = 2`:

- `if (2 * 2 == 64)`: `false`.
- `if (2 * 2 > 64)`: `false`.
- Next call is `helper(64, 3)`.

Iteration continues:

- With `guess = 3`: `3 * 3` is neither equal to nor greater than 64.
- With `guess = 4`: `4 * 4` is neither equal to nor greater than 64.
- With `guess = 5`: `5 * 5` is neither equal to nor greater than 64.
- With `guess = 6`: `6 * 6` is neither equal to nor greater than 64.
- With `guess = 7`: `7 * 7` is neither equal to nor greater than 64.

Finally, with `guess = 8`:

- `if (guess * guess == n):`
 - This translates to `if (8 * 8 == 64)`. This condition is `true`!
- So, the function returns `guess`, which is 8.

The final result returned from `_sqrt_recursion(64)` is 8, which is the correct natural square root of 64.

From ChatGPT, designed with  FancyGPT