

8. Inception. Now, before you bother telling me it's impossible...

#advanced

Write a function that compares two strings and returns 1 if the strings can be considered identical, otherwise return 0.

Prototype: int wildcmp(char *s1, char *s2);

s2 can contain the special character *.

The special char * can replace any string (including an empty string)

julien@ubuntu:~/0x08. Recursion\$ cat 101-main.c

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

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

```
/**
```

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

```
*
```

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

```
*/
```

```
int main(void)
```

```
{
```

```
    int r;
```

```
    r = wildcmp("main.c", "*.c");
```

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

```
    r = wildcmp("main.c", "m*a*i*n*.*c*");
```

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

```
    r = wildcmp("main.c", "main.c");
```

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

```
    r = wildcmp("main.c", "m*c");
```

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

```
    r = wildcmp("main.c", "ma*****c");
```

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

```
    r = wildcmp("main.c", "*");
```

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

```
    r = wildcmp("main.c", "***");
```

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

```
    r = wildcmp("main.c", "m.*c");
```

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

```
    r = wildcmp("main.c", "**.*c");
```

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

```
    r = wildcmp("main-main.c", "ma*in.c");
```

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

```
    r = wildcmp("main", "main*d");
```

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

```
    r = wildcmp("abc", "*b");
```

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

```
    return (0);
```

```

}
julien@ubuntu:~/0x08. Recursion$ gcc -Wall -pedantic -Werror -Wextra -
std=gnu89 101-main.c 101-wildcmp.c -o 101-wildcmp
julien@ubuntu:~/0x08. Recursion$ ./101-wildcmp
1
1
1
1
1
1
1
0
1
1
0
0
julien@ubuntu:~/0x08. Recursion$

```

The problem is to implement a recursive function `wildcmp` that compares two strings and returns 1 if they can be considered identical, taking into account that the second string `s2` may contain wildcard characters (*) that can represent any sequence of characters (including none).

Here's the function to achieve this:

```

int wildcmp(char *s1, char *s2)
{
    /* If both strings are at the end, return 1 */
    if (*s1 == '\0' && *s2 == '\0')
        return (1);

    /* If the character in s2 is a '*', it can represent zero or more characters */
    if (*s2 == '*')
    {
        /* Check if the string matches with zero characters represented by '*' */
        if (wildcmp(s1, s2 + 1))
            return (1);

        /* Check if the string matches with one or more characters represented by '*' */
        if (*s1 != '\0' && wildcmp(s1 + 1, s2))
            return (1);
    }

    /* If current characters match, move on to next character in both strings */
    if (*s1 == *s2)
        return (wildcmp(s1 + 1, s2 + 1));

    /* If none of the above conditions are met, strings are not identical */
    return (0);
}

```

When you include this function in your program and compile using the given commands, the expected output, as given by Julien, will be achieved.

break down this code in details of some ex

Sure, I'd be happy to break down the code for you!

First, let's restate the problem: The function `wildcmp` needs to determine if the string `s1` can be matched to the pattern given by `s2`, where `s2` can contain wildcards (*). The wildcard (*) means it can match any string (including an empty string).

Now, let's break down the code piece by piece:

1 Base Case:

```
if (*s1 == '\0' && *s2 == '\0')
    return (1);
```

This checks if we have reached the end of both strings. If we have, it means all previous characters have matched (since we didn't return early), and hence we return `1` indicating the two strings are identical with respect to the pattern.

1 Handling the Wildcard:

```
if (*s2 == '*')
{
    if (wildcmp(s1, s2 + 1))
        return (1);

    if (*s1 != '\0' && wildcmp(s1 + 1, s2))
        return (1);
}
```

Here, we handle the wildcard in `s2`.

- The first `if` checks if the current character in `s2` is a `*`.
- The inner `if` conditions check for two possibilities:
 - a. The `*` in `s2` matches zero characters in `s1`. In this case, we advance `s2` by one character (`s2 + 1`) but keep `s1` at the same position.
 - b. The `*` in `s2` matches one or more characters in `s1`. Here, we move `s1` forward by one character (`s1 + 1`) but keep `s2` at the current position (since the `*` could potentially match more characters).

1 Direct Match:

```
if (*s1 == *s2)
    return (wildcmp(s1 + 1, s2 + 1));
```

This is the straightforward case. If the current character in `s1` directly matches the current character in `s2`, then we simply move forward in both strings.

1 Non-Matching Case:

```
return (0);
```


If none of the above conditions are met, we return `0`, meaning `s1` doesn't match the pattern of `s2`.

Let's illustrate with an example:

Suppose `s1 = "abcd"` and `s2 = "a*c"`.

- 1 `*s1 = 'a'` and `*s2 = 'a'`, so we move forward in both strings.
- 2 Now, `*s1 = 'b'` and `*s2 = '*'`. Since `*s2` is a wildcard, we check: a. If wildcard matches zero characters. In this case, `s1` remains the same and `s2` moves to next character. So, we're comparing "abcd" with "c". This doesn't match. b. If wildcard matches one or more characters. We advance `s1` and keep `s2` same. So we're comparing "bcd" with "ac". *This recursive step will eventually lead to a match since ' can replace 'b' and 'd'.*

The return value is `1` indicating that `s1` matches the pattern in `s2`.

From ChatGPT, designed with  FancyGPT