Wildcard Matching

Statement

Given an input string (`s`) and a pattern (`p`), implement wildcard pattern matching with support for `'?'` and `'*'` where:

- `'?'` Matches any single character.
- `'*'` Matches any sequence of characters (including the empty sequence).

The matching should cover the **entire** input string (not partial).

Example 1:

```
Input: s = "aa", p = "a"
Output: false
Explanation: "a" does not match the entire string "aa".
```

Example 2:

```
Input: s = "aa", p = "*"
Output: true
Explanation: '*' matches any sequence.
```

Example 3:

```
Input: s = "cb", p = "?a"
Output: false
Explanation: '?' matches 'c', but the second letter is 'a', which does not match 'b'.
```

Constraints:

- `0 <= s.length, p.length <= 2000`
- `s` contains only lowercase English letters.
- `p` contains only lowercase English letters, `'?'` or `'*'`.

The best way to solve this problem

The best way is to use dynamic programming.

As we know, dynamic programming is a way to solve a problem by dividing it onto subproblems and using the solutions of these subproblems. Summary, it consists on stocking results of each “subtasks” and use it later.

So in this case, we are gonna to use an bool array. A two dimensional bool array. Let call it T.

So the len of T is T[len(s) + 1][len(p) + 1].

T[0][0] = true.

We will return T[len(s) + 1][len(p) + 1] as the result.

Let’s assume that we have to integer i, j to parkour our array. (i start to 1 and stop when it is greater than len(s); j start to 1 and stop when it is greater than len(p))

So to fill up each element the array (T[i][j]) we are gonna to follow these 3 cases

- If s[i - 1] == p[j - 1] || p[j] == ‘?’ ⇒ T[i - 1][j - 1]
- If p[j - 1] == ‘*’ ⇒ T[i - 1][j] || T[i][j - 1]
- else False

In fact we are looking for each substring of s if an associate substring on p could be equivalent.

For more explanations watch this video of Tushar.

My code

(I wrote it on C but the logic and some cases are here)

```
#include <string.h>

bool isMatch(char * s, char * p)
{
    int s_size = strlen(s);
    int p_size = strlen(p);
    bool array[s_size + 1][p_size + 1];
```

```
    //memset(array, false, sizeof(array));
    array[0][0] = 1;
    for (int j = 1; j <= p_size; j++) {
        if (p[j - 1] == '*')
            array[0][j] = array[0][j - 1];
        else
            array[0][j] = 0;
    }
    for (int i = 1; i <= s_size; i++)
        array[i][0] = 0;
    for (int i = 1; i <= s_size; i++) {
        for (int j = 1; j <= p_size; j++) {
            if (p[j - 1] == '?' || p[j - 1] == s [i - 1])
                array[i][j] = array[i - 1][j - 1];
            else if (p[j - 1] == '*')
                array[i][j] = array[i - 1][j] || array[i][j - 1];
            else
                array[i][j] = false;
        }
    }
    return (array[s_size][p_size]);
}

int main(int argc, char **argv)
{
    if (argc != 3) {
        std::cerr << "You have to enter two strings" << std::endl;
        return (84);
    }
    std::cout << "Real is match" << isMatch(argv[1], argv[2]) << std::endl;
    return (0);
}
```