Certainly! Let's break down the given solution and then explain `strcmp`.

Your solution:

```
int _strcmp(char *s1, char *s2)
{
int i;

for (i = 0; s1[i] != '\0' && s2[i] != '\0'; i++)
{
if (s1[i] != s2[i])
{
return (s1[i] - s2[i]);
}
}
return (s1[i] - s2[i]);
}
```

Breakdown:

1. Function Declaration:

```
int _strcmp(char *s1, char *s2)
```

You're defining a function named `_strcmp` that takes two string (char pointers) arguments, `s1` and `s2`.

1. Initialization:

```
int i;
```

You're declaring an integer `i` which you'll use as an index to traverse both strings simultaneously.

1. The Loop:

```
for (i = 0; s1[i] != '\0' && s2[i] != '\0'; i++)
```

This loop iterates through both strings until one of them ends (i.e., until it encounters the null terminator `\0` in either string).

- `s1[i] != '\0'`: ensures we're not at the end of string `s1`.
- `s2[i] != '\0'`: ensures we're not at the end of string `s2`.

1. Inside the Loop:

```
if (s1[i] != s2[i])
{
return (s1[i] - s2[i]);
}
```

While iterating, if you find any character that is not equal in both strings (`s1[i] != s2[i]`), you return the difference between the two characters (`s1[i] - s2[i]`). This difference will be positive if the character in `s1` is greater, negative if the character in `s2` is greater, and zero if they're the same.

1. After the Loop:

```
return (s1[i] - s2[i]);
```

If the loop completes without finding any differences, it means that either both strings are identical up to a point, or one string is shorter than the other. By performing `s1[i] - s2[i]` again, you're handling the case where one string is shorter. If `s1` is shorter, this will result in a negative value; if `s2` is shorter, this will result in a positive value. If both are of the same length and identical, it'll be 0, indicating that they're exactly the same.

Explanation of `strcmp`:

`strcmp` is a function from the C standard library which compares two strings.

- If the strings are identical, it returns `0`.
- If the first string is lexicographically less than the second one, it returns a negative number.
- If the first string is lexicographically greater than the second one, it returns a positive number.

The lexicographic order is based on the ASCII values of characters.

The behavior of your `_strcmp` function is mimicking the behavior of the standard `strcmp` function.

In essence, `strcmp` helps determine the relative order of two strings in "dictionary" order. So, "apple" comes before "banana" and `strcmp` will return a negative number for `strcmp("apple", "banana")`.

if (s1[i] != s2[i])
{
return (s1[i] - s2[i]);

}
why in this section when the condition is true it return the diff and break the loop is return can break the loop if it return value

Yes, the `return` statement effectively breaks out of the loop and exits the function, returning control to the caller of the function.

Let's break down the logic:

```
if (s1[i] != s2[i])
{
return (s1[i] - s2[i]);
}
```

When `s1[i]` and `s2[i]` are not equal, it means the two strings differ at the current index `i`. The function then calculates the difference `(s1[i] - s2[i])`.

1. If `s1[i]` has a higher ASCII value than `s2[i]`, the result is positive.
2. If `s2[i]` has a higher ASCII value than `s1[i]`, the result is negative.
3. If they are equal, the result is zero (but the condition ensures this scenario won't be reached).

By returning `(s1[i] - s2[i])`, the function is indicating how the two strings differ. The actual value of the difference isn't as important as the sign of the value (positive, negative, or zero), which indicates which string is lexicographically greater or if they are the same.

As soon as a difference is found and the function returns, no further processing in the function happens, and no more iterations of the loop are executed. This is because the `return` statement immediately exits the function and sends the return value back to the function's caller.