15.- ft_strchr.-

Function based on the definition given in the BSD man pages for “strchr(3)”.
The library associated is <string.h> (standard C library).

Synopsis:

```
char *strchr(const char *s, int c);
```

Purpose:

Locates the first occurrence of a character (`c`) within a string (`s`).

Parameters:

- `s`: The string to search within.
- `c`: The character to search for.

Return Value:

Returns a pointer to the first occurrence of `c` in `s`, or `NULL` if `c` is not found.

Description:

- Iterates through the characters of `s` until it finds `c` or reaches the null terminator.
- Returns a pointer to the matching character's position in `s`.

The strchr() function locates the first occurrence of c (converted to a char) in the string pointed to by s. The terminating null character is considered to be part of the string; therefore if c is `\0', the func- tions locate the terminating `\0'.

Code

```
#include "libft.h"

char *ft_strchr(const char *s, int c)
{
    char str = c;
    int i = 0;

    while (s[i] != str)
    {
        if (s[i] == 0)
            return (NULL);
        i++;
    }
    return ((char *)&s[i]);
}
```

Code Explanation

1. **Converts `c` to char:** Stores `c` as a character for comparison (`str`).
2. **Iterates through string:**
   - Checks each character of `s` against `str`.
   - Stops if a match is found or the null terminator is reached.
3. **Returns pointer or NULL:**
   - If a match is found, returns a pointer to the position of `c` in `s`.

        - If `c` is not found, returns `NULL`.

Main Function (Optional)

```
int main(void)
{
    char *str = "Hello, friend";
    char c = 'o';
    char *ptr;

    ptr = ft_strchr(str, c);
    if (ptr)
    {
        printf("Character '%c' found in string '%s', in position '%ld'\n",
               c, str, (ptr - str) + 1);
    }
    else
    {
        printf("Character '%c' not not found in string '%s'.\n", c, str);
    }
    return (0);
}
```

Key Points:

- **Null Terminator:** The null terminator ('\0') marks the end of a string.
- **Pointer Arithmetic:** Subtracting pointers (`ptr - str`) yields the character offset.
- **Character Comparison:** Uses == to compare characters directly.

Here's a breakdown of the line `return ((char *)&s[i]);` within the `ft_strchr` function:

1. `&s[i]`**:**

- `s[i]`**:** Accesses the character at index `i` within the string `s`.
- & **(address-of operator):** Takes the memory address of that character.

2. `(char *)`**:**

- Casts the address to a `char *` type, meaning it's now treated as a pointer to a character.

3. `return`**:**

- Returns this pointer from the function.

In essence, this line returns a pointer to the first occurrence of the character `c` within the string `s`.

Here's a more detailed explanation of what's happening:

1. **Searching for the Character:**
    - The `while` loop iterates through the string `s` until either:
        - The character `c` is found at index `i`.
        - The end of the string is reached (`s[i] == 0`).
2. **Returning the Pointer:**
    - If the character `c` is found, the loop breaks, and:

        - `&s[i]` takes the address of the character at index `i` (the first occurrence of `c`).
        - `(char *)` casts this address to a `char *` pointer.
        - `return` sends this pointer back to the calling code.
    - If the character `c` isn't found, the function returns `NULL`.

Key Points:

- The function mimics the behavior of the standard C library function `strchr`.
- It's designed to find the first occurrence of a character within a string efficiently.
- The return value is a pointer that can be used to access and manipulate the found character or the subsequent characters in the string.

Here's a breakdown of the `printf` line and its components:

1. `printf("Character '%c' found in string '%s', in position '%ld'\n", c, str, (ptr - str) + 1);`:

- **Format string:**
    - `"Character '%c' found in string '%s', in position '%ld'\n"`: This string contains placeholders for values to be inserted, marked by `%` and a format specifier.
    - `%c`: Prints a character.
    - `%s`: Prints a string.
    - `%ld`: Prints a long integer (more on this later).

2. Arguments:

- `c`: The character to be displayed.
- `str`: The original string.
- `(ptr - str) + 1`: The calculated position of the character within the string.

3. Calculation of Position:

- `ptr - str`: Subtracts the address of the beginning of the string (`str`) from the address of the found character (`ptr`). This yields the offset of the character within the string, but as a pointer difference, not a direct index.
- `+ 1`: Adds 1 to the result to align with human-readable indexing (where the first character is at position 1, not 0).

Why `%ld` and not `%d`:

- **Pointer arithmetic:** The result of `ptr - str` is a pointer difference, which is typically of type `ptrdiff_t`. This type is often larger than `int` to accommodate potential large offsets in memory.
- **Accurate representation:** Using `%ld` ensures that the pointer difference is printed correctly as a long integer, capable of handling larger values if needed. While `%d` could work for smaller offsets, it's safer to use `%ld` for compatibility and potential future extensions.

Key Points:

- The `printf` function is used to format and print output to the console.
- Format specifiers like `%c`, `%s`, and `%ld` control how different data types are displayed.
- Pointer arithmetic can be used to calculate offsets within strings, but the results need to be handled appropriately based on their data types.
- Using the correct format specifiers is essential for accurate and consistent output.