

The first code does not work, ChatGPT's code did not return a `*mut *mut`. The second request was helpful, but is only a function that takes a string. Unfortunately, I do not know enough about Rust to turn that into a list of 3 strings as a `*mut *mut i8` that I would be able to use that for the return type. The third time was a charm, the function

Can you write a rust function that returns the groups of the strings "Hi", "Hello", and "Hey" as a `*mut *mut i8`?

ChatGPT

Certainly! To achieve this, we'll create a Rust function that takes a string as input and returns a pointer to an array of pointers to strings. In this case, the input strings will be "Hi", "Hello", and "Hey". Here's the code:

```
fn main() {
```

```
    let s = "Hi, Hello, Hey";
```

```
    let mut arr = Vec::new();
```

```
    for s in s.split(", ") {
```

```
        let p = s.as_ptr();
```

```
        arr.push(p);
```

```
    }
```

```
}
```

```
fn main() {
```

```
    let s = "Hi, Hello, Hey";
```

```
    let mut arr = Vec::new();
```

```
    for s in s.split(", ") {
```

```
        let p = s.as_ptr();
```

```
fn main() {
```

```
    let s = "Hi, Hello, Hey";
```

```
    let mut arr = Vec::new();
```

```
    for s in s.split(", ") {
```

```
        let p = s.as_ptr();
```

```
        arr.push(p);
```

```
    }
```

```
fn main() {
```

```
    let s = "Hi, Hello, Hey";
```

```
    let mut arr = Vec::new();
```

```
    for s in s.split(", ") {
```

```
        let p = s.as_ptr();
```

```
fn main() {
```

```

}

struct StringGroups {
    *mut c_string;
    *mut c_string;
    *mut c_string;
}

fn get_string_groups() -> *mut StringGroups {
    let mut sg = StringGroups {
        0: to_c_string("Hi"),
        1: to_c_string("Hello"),
        2: to_c_string("Hey"),
    };
    &sg
}

```

This code defines a Rust function `get_string_groups` which creates three strings ("Hi", "Hello", and "Hey") and stores them in a struct called `StringGroups`. It then returns a pointer to this struct.

The `to_c_string` function is a helper function that converts a Rust string to a C-compatible string.

The `free_string_groups` function is used to free the memory allocated for the `StringGroups` struct when it's no longer needed.

Please note that when using this code, you'll need to be cautious about memory management, as you're dealing with raw pointers. Always ensure that you call `free_string_groups` when you're done using the returned pointer to avoid memory leaks.



How would one make a `*mut *mut i8` in Rust?

ChatGPT

In Rust, you can create a `*mut *mut i8` (a pointer to a mutable pointer to a mutable i8) using raw pointers. Here's an example of how you can do it:

```

fn create_double_mut_ptr() -> *mut *mut i8 {
    let mut value: i8 = 42;

    let mut ptr: *mut i8 = &mut value;

    let mut double_ptr: *mut *mut i8 = &mut ptr;

    double_ptr
}

fn main() {
    let double_mut_ptr = create_double_mut_ptr();
}

```

```

unsafe {

    let ptr_to_i8 = *double_mut_ptr;

    let value = *ptr_to_i8;

    println!("Value: {}", value);

}

```

Explanation:

1. The `create_double_mut_ptr` function creates an `i8` value and a mutable pointer to it (`ptr`). Then, it creates a mutable pointer to the mutable pointer (`double_ptr`).
2. The function returns `double_ptr`, which has the type `*mut *mut i8`.
3. In the `main` function, we call `create_double_mut_ptr` to get the double mutable pointer.
4. We use `unsafe` to dereference the double mutable pointer. First, we dereference it to get the inner mutable pointer (`ptr_to_i8`). Then, we dereference `ptr_to_i8` to get the `i8` value (`value`).
5. Finally, we print out the value, which will be `42` in this example.

Remember, using raw pointers in Rust is marked as `unsafe` because it's your responsibility to ensure memory safety. Always be careful when working with raw pointers to avoid common pitfalls like null pointer dereferencing, data races, and memory leaks.



How would one turn a group of strings into a `*mut *mut i8` in Rust?

ChatGPT

If you want to turn a group of strings into a `*mut *mut i8` in Rust, you would need to create an array of C-compatible strings (`*mut i8`) and then return a pointer to this array. Here's an example:

```

use std::os::raw::c_char;

fn to_c_string(s: &str) -> *mut i8 {

    let c_string = std::ffi::CString::new(s).expect("CString::new failed");

    c_string.into_raw()

}

```

```

fn strings_to_double_mut_ptr(strings: Vec<&str>) -> *mut *mut i8 {

    let mut c_strings: Vec<*mut i8> = strings.iter().map(|s| to_c_string(s)).collect();

    let double_mut_ptr = c_strings.as_mut_ptr();

    std::mem::forget(c_strings); // Prevents memory deallocation

    double_mut_ptr
}

fn main() {

    let strings = vec!["Hi", "Hello", "Hey"];

    let double_mut_ptr = strings_to_double_mut_ptr(strings);

    // Use double_mut_ptr as needed

    // Remember to deallocate the memory when done (if not used in C/C++ land)

    unsafe {

        for i in 0..3 {

            let c_string = *double_mut_ptr.offset(i as isize);

            if !c_string.is_null() {

                let _ = std::ffi::CString::from_raw(c_string);

            }

        }

        // Now that we've cleaned up the individual strings, we can deallocate the array

        Box::from_raw(double_mut_ptr);

    }

}

```

In this example:

1. The `to_c_string` function converts a Rust string to a C-compatible string.
2. The `strings_to_double_mut_ptr` function takes a vector of string references, converts them to C-compatible strings, and returns a pointer to an array of `*mut i8`.
3. In the `main` function, we call `strings_to_double_mut_ptr` with a vector of strings.
4. We then use `double_mut_ptr` as needed. Keep in mind that this is now a pointer to an array of `*mut i8`.
5. When you're done using the pointers, you'll need to deallocate the memory. In this example, we've used `std::mem::forget` to prevent the individual `CString` objects from being deallocated. This means that you need to manually deallocate them later using `CString::from_raw` when you're done with them. Finally, we deallocate the array using `Box::from_raw`.

Please note that working with raw pointers and memory management in Rust requires caution and is marked as `unsafe`. Always make sure to properly handle memory deallocation to avoid memory leaks.