

Forum:[Hack Forums](#)

Thread: [\[Help\] User Input Function returns Gibberish](#)

lockhead - January 16, 2019, 6:29 am

Hey so I wanted to write a function that reads user input (of unknown length) and returns it to the caller. Sounds simple, right?

Well I can't figure out why it (kinda) doesn't work. Here's the code

```
Code #include <stdio.h>
#include <string.h>
#include <stdlib.h>
void getUserInput(char *str){
    int CHUNK = 8;
    int ch;
    int len = 1;
    int allocsize = len * CHUNK;
    while((ch = getchar()) != '\n' && ch != EOF){
        str[len-1] = ch;
        if(len % (CHUNK-1) == 0){
            allocsize = allocsize + 8;
            str = (char *)realloc(str, sizeof(char) * allocsize);
        }
        ++len;
    }
    str[len-1] = '\0';
}

int main()
{
    char *str = malloc(sizeof(char) * 8);
    getUserInput(str);
    printf("str: %s\n", str);
    return 0;
}
```

The reason why i say "kinda" doesn't work is because it does work when I run it through gdb, but doesn't when I run it normally (terminal) and enter a string equal to or longer than seven characters. The reason for that is that I have 8 byte buffer, if that is filled str get's realloc'ed to an additional 8 bytes.

So if the input string is smaller than seven characters in length this works fine, but if it is larger or equal to seven this works only in gdb and return gibberish through normal program execution and I don't know why. Realloc shouldn't be messing with my string, right? And it get's passed as a function argument, so it's not a local variable either...

Sorry for the low quality post but I just can't figure it out.

Auriana - January 16, 2019, 7:20 am

If you're going to reallocate the pointer, you need to pass the pointer to the pointer into the function.

Or you could just make it easier and return an allocated pointer from the getUserInput function. This should work.

```

Code #include <stdio.h>
#include <string.h>
#include <stdlib.h>
char* getUserInput(){
    int CHUNK = 8;
    int ch;
    int len = 1;
    int allocsize = len * CHUNK;
    char* str = malloc(allocsize);

    while((ch = getchar()) != '\n' && ch != EOF){
        str[len-1] = ch;
        if(len % (CHUNK-1) == 0){
            allocsize = allocsize + 8;
            str = (char *)realloc(str, sizeof(char) * allocsize);
        }
        ++len;
    }
    str[len-1] = '\0';
    return str;
}

int main()
{
    printf("Enter your string:");
    char* str = getUserInput();

    printf("str: %s\n", str);
    free(str);
    return 0;
}

```

lockhead - January 16, 2019, 7:38 am

(Jan 16, 2019 08:20 AM)Auriana Wrote: [\[CITE\]](#) If you're going to reallocate the pointer, you need to pass the pointer to the pointer into the function.

Or you could just make it easier and return an allocated pointer from the getUserInput function. Why do I have to pass a pointer to the pointer? Does altering the pointer within a function somehow make it a local variable?

Anyway, thank you for your help! I appreciate it ?

Auriana - January 16, 2019, 8:49 am

(Jan 16, 2019 08:38 AM)lockhead Wrote: [\[CITE\]](#) Why do I have to pass a pointer to the pointer? Does altering the pointer within a function somehow make it a local variable?

Anyway, thank you for your help! I appreciate it ?

You're reassigning it's value in another function.

Code void reassignpointer(char** pointertobereassigned); What gets passed on the stack is a copy of the variable. But if you pass it's address on the stack then you can alter the value located at that memory address.

A pointer is just a memory address so when you pass a pointer, you're allowing the function to change whatever value is at that memory address. But you're not allowing the function to assign that pointer itself because it's just holding a copy. So in order to reassign a pointer you need to pass the address of the pointer itself on the stack, allowing the function to change the pointer, that the pointer

you passed on the stack, points to... If that makes any sense.

lockhead - January 16, 2019, 9:43 am

(Jan 16, 2019 09:49 AM)Auriana Wrote: [\[CITE\]](#) You're reassigning it's value in another function. Code void reassignpointer(char** pointertobereassigned);What gets passed on the stack is a copy of the variable. But if you pass it's address on the stack then you can alter the value located at that memory address.

A pointer is just a memory address so when you pass a pointer, you're allowing the function to change whatever value is at that memory address. But you're not allowing the function to assign that pointer itself because it's just holding a copy. So in order to reassign a pointer you need to pass the address of the pointer itself on the stack, allowing the function to change the pointer, that the pointer you passed on the stack, points to... If that makes any sense.

Ok, thank you again!

you_can_never_hack_me - January 16, 2019, 3:50 pm

I'm not goin' to help you with your idea, 'cause that is one of the most insecure ways in dealin' with inputs - in making functions.

Consider the following as your function's parameters:buffer (locations are filled with basic element - e.g. null)

Allocated buffer size - from (1 + (1+1)) to (...)

Number of elements to get - from (1) to (Allocated buffer size - (1+1))

Origin location in buffer - from (0) to (Number of elements to get - 1)

Termination element - e.g. new line

end-of-buffer element - e.g. new line

buffer (locations are filled with allowable elements - e.g. 1-9 A-Z a-z ...)

Allocated buffer size

Length of elements

Origin location in buffer

```
Code unsigned int getuserinput (char *buf1,
const unsigned int bufsiz1,
unsigned int lim1,
unsigned int org1,
const int ter,
const int eob,
const char *buf2,
const unsigned int bufsiz2,
const unsigned int lim2,
const unsigned int org2);
Good luck.
```

lockhead - January 17, 2019, 2:40 am

(Jan 16, 2019 04:50 PM)you_can_never_hack_me Wrote: [\[CITE\]](#) I'm not goin' to help you with your idea, 'cause that is one of the most insecure ways in dealin' with inputs - in making functions.

Consider the following as your function's parameters:buffer (locations are filled with basic element - e.g. null)

Allocated buffer size - from (1 + (1+1)) to (...)

Number of elements to get - from (1) to (Allocated buffer size - (1+1))

Origin location in buffer - from (0) to (Number of elements to get - 1)

Termination element - e.g. new line

end-of-buffer element - e.g. new line

buffer (locations are filled with allowable elements - e.g. 1-9 A-Z a-z ...)

Allocated buffer size

Length of elements

Origin location in buffer

```
Code unsigned int getuserinput (char *buf1,  
const unsigned int bufsiz1,  
unsigned int lim1,  
unsigned int org1,  
const int ter,  
const int eob,  
const char *buf2,  
const unsigned int bufsiz2,  
const unsigned int lim2,  
const unsigned int org2);
```

Good luck.

Security is not a factor. Also I really don't get the point you're trying to make

you_can_never_hack_me - January 17, 2019, 2:54 am

(Jan 17, 2019 03:40 AM)lockhead Wrote: [\[CITE\]](#) Security is not a factor.LOL! Sure - on your planet!

(Jan 17, 2019 03:40 AM)lockhead Wrote: [\[CITE\]](#) Also I really don't get the point you're trying to make I don't expect you to, kid.

Good luck!

lockhead - January 17, 2019, 3:49 am

(Jan 17, 2019 03:54 AM)you_can_never_hack_me Wrote: [\[CITE\]](#) LOL! Sure - on your planet!

I don't expect you to, kid.

Good luck!

Helpful member of the community! +1

you_can_never_hack_me - *January 17, 2019, 3:55 am*

(Jan 17, 2019 04:49 AM)lockhead Wrote: [\[CITE\]](#) Helpful member of the community! +1Thank you!