

Re: some unanswered questions on C

Source: <http://unix.derkeiler.com/Newsgroups/comp.unix.programmer/2003-06/0713.html>

From: Barry Margolin (barry.margolin_at_level3.com)

Date: 06/20/03

Date: Thu, 19 Jun 2003 22:03:25 GMT

In article <fd383985.0306191329.d732131@posting.google.com>, seemanta dutta <seemanta_18@yahoo.com> wrote:
>*greetings linux gurus...*

Why do you keep saying this? If you want Linux gurus, you should post in a Linux-specific newsgroup, not one that's for all varieties of Unix.

Also, it seems like your questions are about generic C, not specific to Unix. There are C newsgroups for that, like comp.lang.c.

>1. *i feel ashamed, but what exactly is an uninitialised pointer?*

A pointer variable that's never been given a value. A static pointer variable will be initialized to the null pointer, but automatic variables are not automatically initialized.

>*is*
> *int *localVar;*
> **localVar = 3 ;*
>*an uninitialised pointer?*
>
>*or is this is an *initialised* pointer?*
> *int *localVar;*
> *localVar = malloc(sizeof int);*
>
>*please clarify with examples of both initialised and uninitialised*
>*pointers.*

Assuming the declarations are within a function, the first one is an uninitialized pointer. When you declare "int *localVar", you're creating a pointer variable but not initializing it. "*localVar = 3" tries to update the memory that it points to, but you've never made it point to anything valid. As a result, you don't know what memory you're modifying. If you're lucky, it points to memory that isn't available to your process, and you'll get an immediate "Segmentation Fault (core dumped)"; if you're not that lucky, it will corrupt some random piece of data in your process's memory.

Your second example creates a pointer and then initializes it to point to the memory that malloc() provided. After that, it would be safe to do `*localVar = 3` to fill in that memory.

>2. *in some of my programs i have to read from a file and write to a different file using fgets and fprintf...*
>
>*what i want to ask is that when i declare my buffer for fgets as*
> `char *buff;`
>
>*my program compiles alright but while running it gives a seg fault..i*
>*can understand this is because of a buffer overflow, but if i use a*

No, it's another uninitialized pointer. `"char *buffer"` creates a pointer, but it doesn't create any memory for what it points to; you can call `malloc()` to allocate some memory for the buffer.

>*declaration like this for my buffer*
> `char buff[80];`
>*my seg fault disappears...*
>
>*but here i am implicitly assuming my data will never be larger than 80*
>*bytes...if my data exceeds 80 bytes there occurs another seg fault and*
>*my program exits...*
>*is it a good practice to use limited size buffers for file i/o?*
>
>*what i want to know is that is there any way to write code for file*
>*i/o ,or any i/o for that matter which shall be independent of the*
>*amount*
>*of data entered by the user or of that amount of characters present*
>*in a line of the file read by a call to fgets.*

Many Unix programs simply use a fixed-size buffer. If lines are too long, they usually report an error, but some either ignore the excess or they treat it as a separate line. I don't recommend any of these approaches. But if you're going to go this way, make the buffer very large, like several thousand characters, so that it's unlikely users will hit the limit.

`realloc()` is useful for this. When you start out, use `malloc(80)` to create an 80-character buffer for the lines. When you call `fgets()`, check whether the last character it read is `\n`; if it isn't, it means the line was too long. Call `realloc()` to increase the size of the buffer, and then call `fgets()` again to read the rest of the line. Keep doing this until you get the whole line.

>3. *in some of my programs like this..*
>
> `int main()`
>{
>
>

comp.unix.programmer: Re: some unanswered questions on C

```
> char deststr[20],srcstr1[20]="foo",srcstr2[20]="bar";
> deststr=strcat(srcstr1,srcstr2); // insertion of 4 given
>lines made here in place of this line...
> puts(deststr);
> return 0;
>}
>
>but the above program compiles with the following error: t.c: In
>function `main':
>t.c:8: incompatible types in assignment
>if the man page says that strcat returns a 'char *' and since the name
>of a char array implies a pointer to a char, why is this giving an
>error?
>
>if i replace the declaration of deststr with char *deststr;
>i no longer get any error, but on running i get a seg
>fault...obviously due to a buffer overflow and if i use deststr[20] i
>get the above
>error.
>what to do? it is a catch22 kind of situation for me... i cannot use
>*deststr because of buffer overflows,neither can i use deststr[20] for
>it does allow me to use strcat as i want to...
>and gives an error
```

You should not get an error when you change the declaration to "char *deststr". There's no buffer overflow; you're modifying srcstr1 by changing the string's length from 3 to 6, and it has room for 19 (not including the trailing 0 byte). Then you set deststr to the address of srcstr1 (because that's what strcat() returns).

```
>if i insert these lines in place of the second line above, i can use
>deststr[20] instead...but at the expense of a few more lines of
>code...
>and a limited buffer size of deststr[20]...
>
>char temp[30];
>
>strcpy(temp,srcstr1);
>strcat(temp,srcstr2);
>strcpy(deststr,temp);
>
>but is it ok to involve another variable like this and use an awkward
>strcpy and a limited buffer size?
>
>please advise where i am making a mistake...
```

If you want to be able to handle strings of any size, you should use malloc() to allocate memory of the size you need at that time. E.g.

```
char *deststr = malloc(strlen(srcstr1) + strlen(srcstr2) + 1);
strcpy(deststr, srcstr1);
```

```
srcat(deststr, srcstr2);
```

>4. please consider the following code:

```
>  
>#include<stdio.h>  
>#include<stdlib.h>  
>#include<string.h>  
>  
>char *foo(void);  
>  
>int main()  
>{  
>  
> char *ptr ;  
> ptr = foo();  
> puts(ptr);  
> return 0;  
>  
>}  
>  
>  
>char *foo()  
>{  
> char *retval; //using retval[10] causes warning...but using  
> *retval doesnot..
```

Not all error can be detected by the compiler.

```
> strcpy(retval,"hello"); //why is strcpy allowed here?  
> should'nt a seg fault occur here?
```

Since retval's contents are unpredictable, it may or may not point to a location that causes a segfault.

```
> return retval; //using strcat also does not cause a  
> seg fault...  
>  
> // question: is there a memory leak in this code if i add the line  
> 'retval=malloc(7);' after retval declaration?
```

Only if you never call free() to release the memory. You can do that in the main() function if you want.

However, all memory is reclaimed automatically when a program exits. Since your main() function returns immediately after using the pointer returned by foo(), it's not really a leak. But it's good practice to get into to free everything you allocate.

```
>  
>}  
>
```

comp.unix.programmer: Re: some unanswered questions on C

>
>*in the first line in foo(),if i replace *retval with retval[20] or*
>*something like it, i get a warning saying*
>*warning: function returns address of local variable*
>
>*is that an unacceptable practice to return address of an auto*
>*variable?*

Yes. Local variables no longer exist after the function returns. The memory that was used for them will be reused for other things. If you try to use that pointer, you won't get what you expect.

>*and in one program i was using such a variable by malloc'ig some*
>*bytes...*
> *char *retval;*
> *retval=malloc(5);*
> *strcpy(retval,"hello");*
> ...
> *return retval;*
>
>*using malloc call in place of retval[20] also removes the above*
>*warning.*
>
>*but this has put me into worries since i have read in books that*
>*malloc() calls that do not have any corresponding free() calls cause*
>*memory leaks...*
>*so in this case i ought to call free()*

In general, yes.

>*but if i call free before i call return my pointer that would return a*
>*garbage pointer, is n't it?*

Correct. You should call free after you're all done using the memory that was allocated.

>*but on the other hand i must free() my memory also...as well return my*
>*pointer to the calling function...*
>*how to write code that would prevent memory leaks as well as won't*
>*give any warnings as shown above...*

The free() can be done anywhere in the program, it doesn't have to be in the same function that called malloc().

>*5. in a program while debugging i got a 'program returned 62 error*
>*code' from gdb.*
>*how do i convert this error code into a meaningful string, like with*
>*errno using perror? i can collect the exit code of my program thru*
>*the wait syscall which matched with that returned by gdb, but how do i*
>*get some meaningful info from it in human readable form?*
>*please help....*

comp.unix.programmer: Re: some unanswered questions on C

There is no meaning to this number, it's whatever you put in your exit() function or the "return" statement in main(). The only value that has any meaning is 0, which is conventionally used to mean that the program completed successfully.

--

Barry Margolin, barry.margolin@level3.com

Level(3), Woburn, MA

*** DON'T SEND TECHNICAL QUESTIONS DIRECTLY TO ME, post them to newsgroups.

Please DON'T copy followups to me -- I'll assume it wasn't posted to the group.