Forum:Hack Forums

Thread: C - When to use malloc? Pointers and Struct confusion.

xx1ben99xx - *April 12, 2020, 5:46 pm*

Hi, I'm quite new to C and was wondering when people actually use malloc. It's for dynamically assigning memory, correct? So people use it for returning arrays in a function. Maybe some examples would help?

I also get really confused with pointers. So you use the (type)* to create a pointer to a type. Then to assign an address to that pointer you would use &var. Then to get the value stored at the pointer address you would just use *pointerVar?

Now when structs come in to play it gets even more complex. Because you can use . and -> to assign values to an instance of a structure. Is -> used for pointers? Probably would benefit with some more examples here.

I know there isn't really any specific question, just writing all my thoughts down atm to try and get a better understanding.

I have been asked to make a hash table, and all of the things talked about will be key when trying to code one, so wanted to get a better understanding beforehand.

0xDEAD10CC - *April 12, 2020, 8:42 pm*

If you allocate something on the stack, obviously you can't return a pointer to it, but you can if you allocate the data on the heap with malloc().

I haven't seen anything wrong with what you're saying, you seem to be just confusing yourself. Go look up some tutorials and it should clarify your confusion. You don't need examples of code, just google information on how pointers work.

-> (pointer dereference)

saltycracker - *April 12, 2020, 11:42 pm*

A pointer is either a variable or a literal that can represent a memory location and that's pretty much it. That's all a pointer is. A pointer is a value that says...Hey look over there!

Why do we required pointers and the ability to allocate memory? Programs need the ability to dynamically create data as needed and that's pretty much all you need to know about malloc. malloc returns(if successful) a pointer to a block of memory.

So now we have pointers which provide a 'Hey look over there!' and we have a method for providing ,dynamically , a 'over there'.

Why do we need this? As you become more familiar with programming languages, you'll note that they are filled with pointers and that includes languages like Python, Perl, Ruby, Java and C#... And you'll also note they(the languages mentioned) also have allocations of memory happening behind the scenes... Why do you think all modern languages have garbage collection systems?

xx1ben99xx - *April 13, 2020, 4:54 am*

Thanks for the reassurance guys.

saltycracker - *April 13, 2020, 8:27 am*

(Apr 13, 2020 04:54 AM)xx1ben99xx Wrote: [CITE] Thanks for the reassurance guys.

A way to look at pointers and malloc is to consider that pointer variables allow the program to point to new blocks of memory allocated in a program.

If you expand this concept. You can consider that a pointer variable may point to a block of memory allocated by malloc and that block may ,itself, have pointer(s) that may point to other blocks of memory.. Basically a recursive data structure.

xx1ben99xx - *April 13, 2020, 4:45 pm*

I'm a little stuck at the moment, I have updated a value in my struct: table->val = key; This happens inside a function.

Then when I go to look at the value inside another function it doesn't work. I even tried using strdup, but no.

If you want more information we could message on discord which would be easier?

saltycracker - *April 14, 2020, 12:24 am*

(Apr 13, 2020 04:45 PM)xx1ben99xx Wrote: [CITE] I'm a little stuck at the moment, I have updated a value in my struct: table->val = key; This happens inside a function.

Then when I go to look at the value inside another function it doesn't work. I even tried using strdup, but no.

If you want more information we could message on discord which would be easier?

Just Google C Hash Table and you will find several implementations.

xx1ben99xx - *April 14, 2020, 10:57 am*

I have my hash table struct.

Code typedef struct{

```
    int size;
    int count;
    char **names;
```

} HashTable;I then allocate memory to the hash table doing this:

Code HashTable *table = (HashTable*)malloc(sizeof(HashTable));I then allocate memory to the array of names doing this:

Code char **nameArray = (char**)calloc(size, sizeof(char*));

When I rehash the hashtable I first make a copy of the names, and then free the names doing this:

Code free(table->names) where table is a pointer to a hashtable.

Then I try to allocate memory to the new array of names but making it larger doing this:

Code char **newNamesArray = (char**)calloc(updatedSize, sizeof(char*));But it doesn't actually allocate me any memory, so when I do table->names[0] it crashes my program basically.

Why is this? Is it because I need to also allocate more memory to the hashtable or is there a simpler way to do this?

saltycracker - *April 14, 2020, 11:36 am*

(Apr 14, 2020 10:57 AM)xx1ben99xx Wrote: [CITE] I have my hash table struct.

Code typedef struct{

```
    int size;
    int count;
    char **names;
```

} HashTable;I then allocate memory to the hash table doing this:

Code HashTable *table = (HashTable*)malloc(sizeof(HashTable));I then allocate memory to the array of names doing this:

Code char **nameArray = (char**)calloc(size, sizeof(char*));
When I rehash the hashtable I first make a copy of the names, and then free the names doing this:
Code free(table->names) where table is a pointer to a hashtable.
Then I try to allocate memory to the new array of names but making it larger doing this:
Code char **newNamesArray = (char**)calloc(updatedSize, sizeof(char*));But it doesn't actually allocate me any memory, so when I do table->names[0] it crashes my program basically.
Why is this? Is it because I need to also allocate more memory to the hashtable or is there a simpler way to do this?
You'll have to post the complete code or a link to the complete code.

xx1ben99xx - *April 14, 2020, 11:39 am*
(Apr 14, 2020 11:36 AM)saltycracker Wrote: [CITE] You'll have to post the complete code or a link to the complete code.
https://ghostbin.co/paste/8hckpu7j
password: hackforums

Ignore my boolean struct and other methods like stringLength, my assignment was to try and not use any built in methods as much as possible.

saltycracker - *April 14, 2020, 11:44 am*
(Apr 14, 2020 11:39 AM)xx1ben99xx Wrote: [CITE] https://ghostbin.co/paste/8hckpu7j
password: hackforums

Ignore my boolean struct and other methods like stringLength, my assignment was to try and not use any built in methods as much as possible.
I'll look at it but its been years since I did any real C or C++ programming.

My compiler found this problem
Code if (newNamesArray = NULL){
    printf("Error allocating memory to the new hash table.");
You probably meant to use ==.

xx1ben99xx - *April 14, 2020, 11:54 am*
(Apr 14, 2020 11:44 AM)saltycracker Wrote: [CITE] I'll look at it but its been years since I did any real C or C++ programming.

My compiler found this problem
Code if (newNamesArray = NULL){
    printf("_error_ allocating memory to the new hash table.");
You probably meant to use ==.
Omg I had an error like that earlier, it was the issue, thanks. I don't understand why visual studio code can't just tell me that though. So annoying.

saltycracker - *April 14, 2020, 11:57 am*
Another point.. If you are reimplementing a function like strlen then you should use its types like below and embrace C's short cuts.

Code #include <stdio.h>
#include <stdlib.h>

size_t stringLength(const char *string){
  size_t length = 0;
  while (*string++){

```
      length++;
    }
    return length;
}

int main (int argc, char **argv) {

  fprintf(stdout, "%lu\n", stringLength("salty"));

  return 0;
}
```
You should do the same for you string compare function.

I never implemented a string compare function but if you are then you should make yours mimic the one that comes with string.h.

Something like this.
Code int compareStrings(const char *stringOne, const char *stringTwo){
```
    size_t strOneLen = stringLength(stringOne);
    size_t strTwoLen = stringLength(stringTwo);

    size_t shortest = strOneLen < strTwoLen ?
      strOneLen : strTwoLen;

    for (size_t i = 0; i < shortest; i++){
        if (stringOne[i] > stringTwo[i]){
            return 1;
        }else if (stringOne[i] < stringTwo[i]){
          return -1;
        }
    }

    if (strOneLen == strTwoLen) return 0;

    return strOneLen < strTwoLen ? -1 : 1;
```
}I think I covered all the conditions. Again note the function signature which I got from the manual pages of string,h

No sure what you are doing here:
Code const int tableSize = 10;
```
HashTable *myHashTable = initializeTable(tableSize);
```
and then you(in initializeTable) do this?
Code ...
```
HashTable *table = (HashTable*)malloc(sizeof(HashTable));
...
char **nameArray = (char**)calloc(size, sizeof(char*));
```
Shouldn't you be using size to create an array of HashTables or am I confused?

I stripped everything out of the program and used characters instead of strings to simplify the code...

It appears to work with characters. I think.. You'll be able to tell me if it works for my example with characters.

https://pastebin.com/UL9Svm2d

xx1ben99xx - *April 14, 2020, 3:09 pm*
size is how many names I can enter in the hashtable so I use it to allocate enough memory for the names character array. Also what does size_t actually mean? And why do you put const in the function parameters?

saltycracker - *April 14, 2020, 9:38 pm*
(Apr 14, 2020 03:09 PM)xx1ben99xx Wrote: *[CITE]* ... Also what does size_t actually mean? And why do you put const in the function parameters?
size_t is an integral type that's guaranteed to be big enough for any array size.

Why the const? Its not me, its the standard library.

Code size_t strlen(const char *s)
Code int strcmp(const char *s1, const char *s2)
If you are going to mimic the functionality in the string.h library then you should start by using their function signatures.

Why does the standard library use const in these cases? The character pointers don't change in the functions so you define them as constant so the compiler can apply optimizations for constants.

xx1ben99xx - *April 15, 2020, 5:30 am*
I see thanks for the explanations

saltycracker - *April 15, 2020, 11:15 am*
Had a couple of hours to look at this again.. I just used a hash function I found on Google for strings.

https://pastebin.com/66z5AEBm

The code is a bit sloppy but I did check with mem-leak software to be sure it wasn't leaking any memory. It checked out for all the operations I used.

The hash code is pretty simple. I just used a linked list for any collisions.

0xDEAD10CC - *April 15, 2020, 8:25 pm*
The general idea is there but why would you let a program continue if heap allocation fails?
Code struct htype *nt = (struct htype *)malloc(sizeof(struct htype));
if (nt == NULL) {
  fputs("malloc failed!\n", stderr);
}
struct htype *old = pos->next
I didn't look too thoroughly at the entire code since a hashtable is a common datastructure.

saltycracker - *April 15, 2020, 9:58 pm*
(Apr 15, 2020 08:25 PM)0xDEAD10CC Wrote: *[CITE]* The general idea is there but why would you let a program continue if heap allocation fails?
Code struct htype *nt = (struct htype *)malloc(sizeof(struct htype));
if (nt == NULL) {
  fputs("malloc failed!\n", stderr);
}
struct htype *old = pos->next

I didn't look too thoroughly at the entire code since a hashtable is a common datastructure.
Good point.. I guess speed kills..

jumprop - *April 29, 2020, 1:20 pm*

(Apr 12, 2020 08:42 PM)0xDEAD10CC Wrote: [CITE] If you allocate something on the stack, obviously you can't return a pointer to it, but you can if you allocate the data on the heap with malloc().
What about the case of the function alloca?

akali - *April 29, 2020, 2:37 pm*

(Apr 29, 2020 01:20 PM)jumprop Wrote: [CITE] What about the case of the function alloca?
The pointer still can not be returned since the function allocates within the caller's stack frame and destroyed outside.

jumprop - *April 29, 2020, 10:45 pm*

(Apr 29, 2020 02:37 PM)akali Wrote: [CITE] The pointer still can not be returned since the function allocates within the caller's stack frame and destroyed outside.
That's an interesting point. Thank you for the insight.

0xDEAD10CC - *April 30, 2020, 9:05 pm*

(Apr 29, 2020 01:20 PM)jumprop Wrote: [CITE] What about the case of the function alloca?
What about it?