

COMPUTER SYSTEMS AND ORGANIZATION

C Strings and More

Daniel G. Graham Ph.D



UNIVERSITY
of VIRGINIA

ENGINEERING



1. NULL and void *
2. Stack vs the heap
3. malloc / calloc / realloc
4. free
5. Implementing a binary tree with structs
6. Deleting a node in the tree
7. Next Time: header file and Style Guides

NULL POINTER



The person who introduced the idea,
Tony Hoare. Called the NULL pointer:

“my billion dollar mistake”.

Tony Hoare introduced Null references in
ALGOL W back in 1965 "simply because it was
so easy to implement",

<https://www.infoq.com/presentations/Null-References-The-Billion-Dollar-Mistake-Tony-Hoare/>

NULL POINTER

```
#define NULL 0
```

Defined by the library with a
#define

```
#include<stdio.h>
int main() {
    int * ptr = NULL;
    printf("%d",*ptr);
    return 0;
}
```

Dereference Null Can cause
the program to crash ☹️

EXAMPLE

UW PICO 5.09

File: pointerfun.c

danielgraham@Daniels-MacBook-Pro ~ %

```
#include <stdio.h>
```

```
int main(){
```

```
    int a = 7;
```

```
    char x = 'y';
```

```
    int *p;
```

```
    p = &a;
```

```
    p = &x;
```

```
}
```

VOID * POINTER

```
void* p;
```

void * is a pointer that is not associated with a data type

ADVANTAGE OF VOID * POINTER

```
int a = 7;  
char x = 'y';  
  
void *p = NULL;  
p = &a;  
p= &b;
```

```
#include <stdio.h>

int main(){
    int a = 7;
    char x = 'y';
    void *p = NULL;
    p = &a;
    p = &x;
}
```


WHAT ABOUT USING A VOID STAR POINTER

```
int a = 7;  
char x = 'y';
```

Will result in a type error
because types don't match

```
void *p = NULL;  
p = &a;  
p = &b;
```

Not a problem we can just
cast

```
char q = *p
```

But we need to be careful
how we cast

```
#include <stdio.h>

int main(){
    int a = 7;
    char x = 'y';
    void *p = NULL;
    p = &a;
    p = &x;
    char q = *p;
}
```

```
#include <stdio.h>
```

```
int main(){
```

```
    int a = 7;
```

```
    char x = 'y';
```

```
    void *p = NULL;
```

```
    p = &a;
```

```
    p = &x;
```

```
    char q = *p;
```

```
}
```

```
[ Wrote 11 lines ]
```

```
^G Get He^O WriteO^R Read F^Y Prev P^K Cut Te^C Cur Po  
^X Exit ^J Justif^W Where ^V Next P^U UnCut ^T To Spe
```

```
[0] 0:zsh*
```

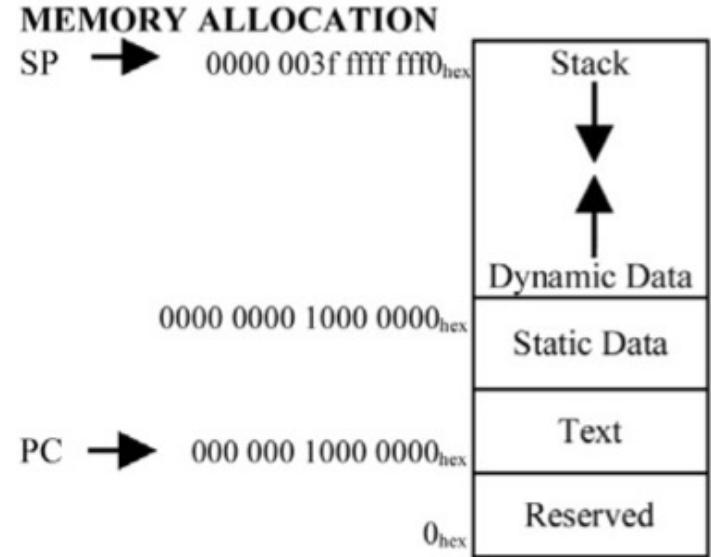
```
"Daniels-MacBook-Pro.1" 21:17 02-Nov-2
```

STACK VS HEAP

So all the variables that we look at so far have been stored on the stack.

Let's look at some examples of storing this on the heap. Here is an example from RISC-V data sheet
It should both starting memory address of the stack and heap (Dynamic Data)

https://www.elsevier.com/_data/assets/pdf_file/0011/297533/RISC-V-Reference-Data.pdf




MALLOC / CALLOC/ REALLOC

```
void* malloc(size_t size);
```



Notice that it returns a void pointer

So, we need to cast its result when we call it.



Number of
bytes to
allocate

MALLOC (IN <STDLIB.H>

```
int * a = (int *)malloc(sizeof(int) * 8);
```

This allocated 8*4 or 32 bytes in heap to store. Returns a pointer to that location on the heap. But what is stored in that location?

MALLOC

```
GNU nano 6.3      funwithalloc.c      Modified
#include <stdio.h>
#include <stdlib.h>
int main(){

    int * a = (int *) malloc(sizeof(int)*8);

    for(int i = 0; i < 8; i++){
        //some sugar
        printf("%d \n", a[i]);
    }
}
```

```
dgg6b@portal09:~$ clang -O3 funwithalloc.c
dgg6b@portal09:~$ ./a.out
343870184
16896672
16896672
16896672
16896672
16896672
16896672
16896672
dgg6b@portal09:~$
```

CALLOC

Number of elements

```
void* calloc(size_t nmemb, size_t size);
```

Notice that it returns a void pointer
So, we need to cast its result when we call it.

Size of each
member

special in that the memory is set to zero. 😊

GNU nano 6.3

funwithalloc.c

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

int main(){

    int * a = (int *) calloc(8, sizeof(int));

    for(int i = 0; i < 8; i++){
        //some sugar
        printf("%d \n", a[i]);
    }
}
```

```
dgg6b@portal09:~$ clang -O3 funwithalloc.c
dgg6b@portal09:~$ ./a.out
0
0
0
0
0
0
0
0
dgg6b@portal09:~$
```

MALLOC / CALLOC/ REALLOC

Pointer to memory
to resize

```
void* realloc(void *ptr, size_t size);
```

New size in bytes

Draw a picture of resize.

What is stored in the newly reserved space? Answer you never know.

```

GNU nano 6.3      funwithalloc.c
#include <stdio.h>
#include <stdlib.h>

int main(){

    int * a = (int *) calloc(8, sizeof(int));

    a = (int *) realloc(a, sizeof(int)*17);

    for(int i = 0; i < 17; i++){
        //some sugar
        printf("%d %d \n", i, a[i]);
    }
}

```

[Wrote 15 lines]

^G Help	^O Write Out	^W Where Is	^K Cut
^X Exit	^R Read File	^_\ Replace	^U Paste

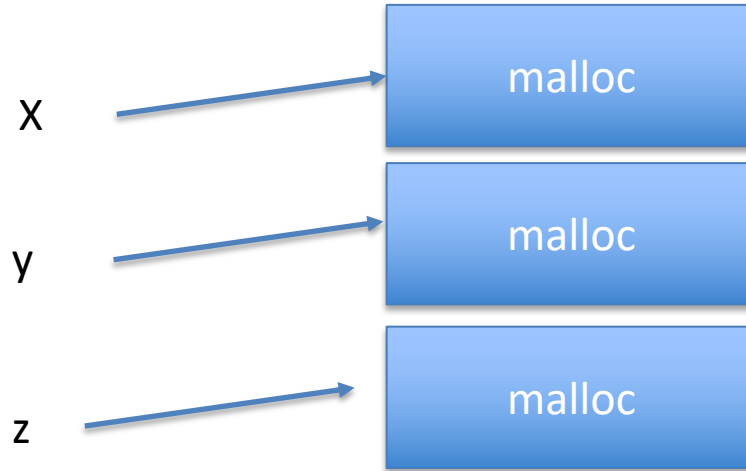
```

dgg6b@portal09:~$ clang -O3 funwithalloc.c
dgg6b@portal09:~$ ./a.out
0 0
1 0
2 0
3 0
4 0
5 0
6 0
7 0
8 0
9 0
10 134465
11 0
12 0
13 0
14 0
15 0
16 0
dgg6b@portal09:~$ 

```

FREE

If just keep allocating (reserving memory on our heap)
will run out of space.

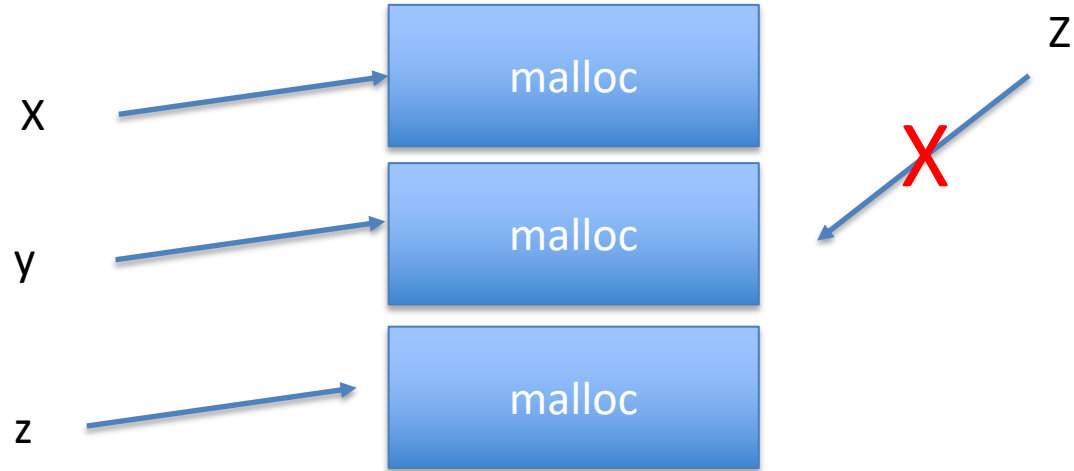


FREE

If just keep allocating (reserving memory on our heap)
will run out of space.

Because y has been allocated
The space Z can't be allocated
The same space.

But we can still make Z
Point to that address.
important difference

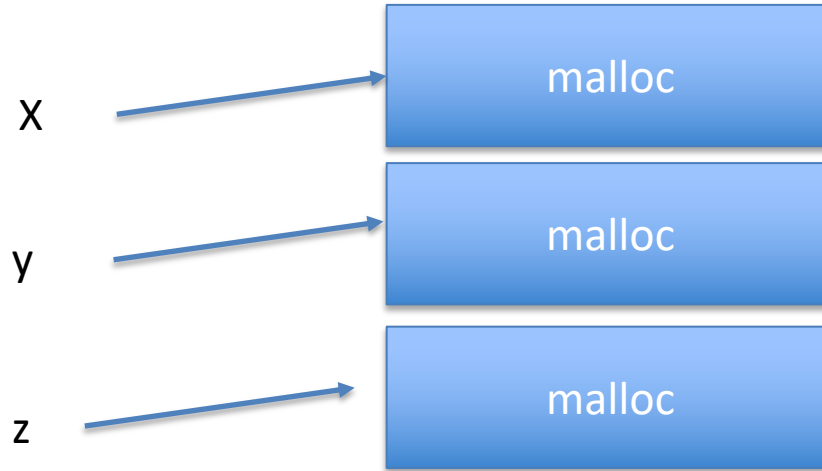


FREE

```
void* free(void *ptr);
```

FREE

`free(y)`

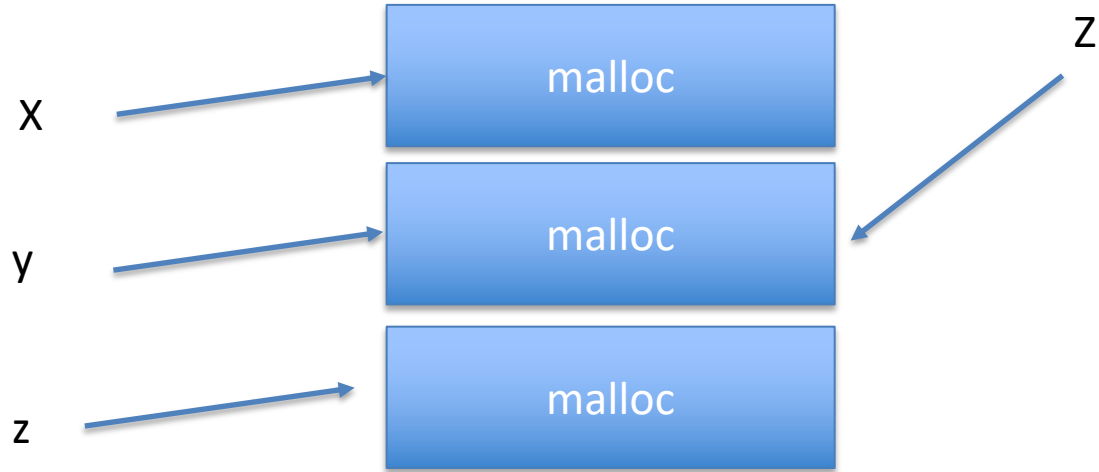


FREE

```
free(y);
```

Disassociates pointer y from the region of memory so that another variable can use it but does not zero out the memory location

y also still has its value
But now z can use the space
Because it has been freed




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

int main(){

    int * a = (int *) calloc(8, sizeof(int));

    for(int i = 0; i < 17; i++){
        //some sugar
        a[i] = i;
        printf("%d %d \n", i, a[i]);
    }

    free(a);

    for(int i = 0; i < 10; i++){
        printf("still there %d \n",a[i]);
    }
}
```

[Wrote 20 lines]

^G Help	^O Write Out	^W Where Is	^K Cut
^X Exit	^R Read File	^_\ Replace	^U Paste

[0] 0: bash*

"portal09" 22:38 02-Nov-23



UNIVERSITY OF VIRGINIA

ENGINEERING

SET POINTER TO NULL ONCE FREED

```
free(p);  
p = NULL;
```

YOU CAN'T DEALLOCATE SPACE ON STACK

```
int main() {  
    int* p1;  
    int m = 100;  
    p1 = &m;  
    free(p1);  
    return 0;  
}
```

CODING DEMO

Binary Tree in C with structs.

