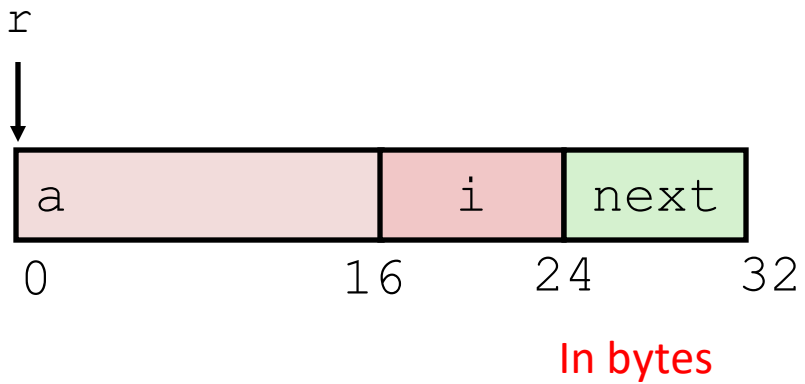
COMPUTER SYSTEMS AND ORGANIZATION

C Strings and More

Daniel G. Graham Ph.D

STRUCTS AREN'T REFERENCES

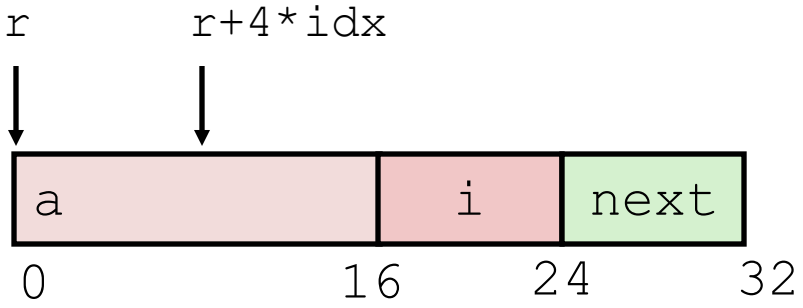
```
struct node {
    int a[4];
    size_t i;
    struct node *next;
};
```



Structs are represented as a block of memory big enough to hold all the fields
Fields are ordered according to order they appear in code
Machine level program has no understanding of structures, it just knows where the attributes associated with the struct are located. (The type is lost)

STRUCTS AREN'T REFERENCES

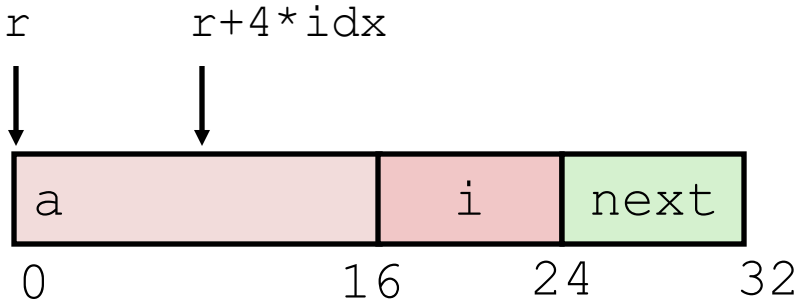
```
struct node {
    int a[4];
    size_t i;
    struct rec *next;
};
```




```
int *element(struct node *r, size_t idx)
{
  return &r->a[idx];
}
```

STRUCTS AREN'T REFERENCES

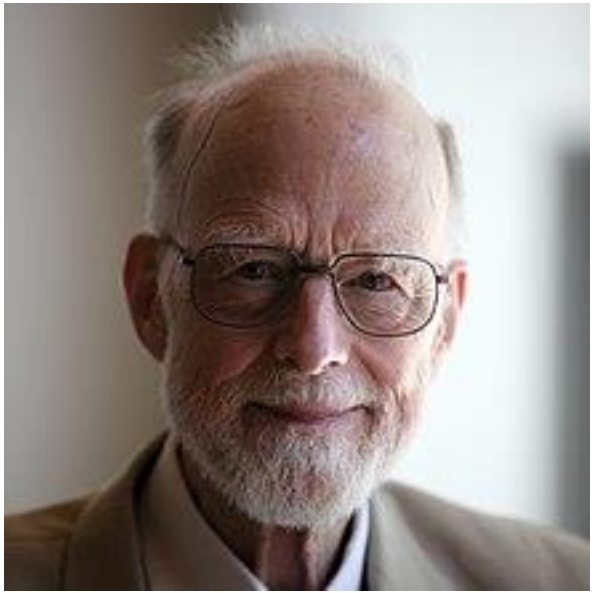
```
struct node{
    int a[4];
    size_t i;
    struct rec *next;
};
```




```
int *get_element(struct node *r, size_t idx)
{
  return &r->a[idx];
}
```

```
  # r in %rdi, idx in %rsi
  leaq  (%rdi,%rsi,4), %rax
  ret
```

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

```
#include <stdio.h>


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

```
danielgraham@Daniels-MacBook-Pro ~ %
```

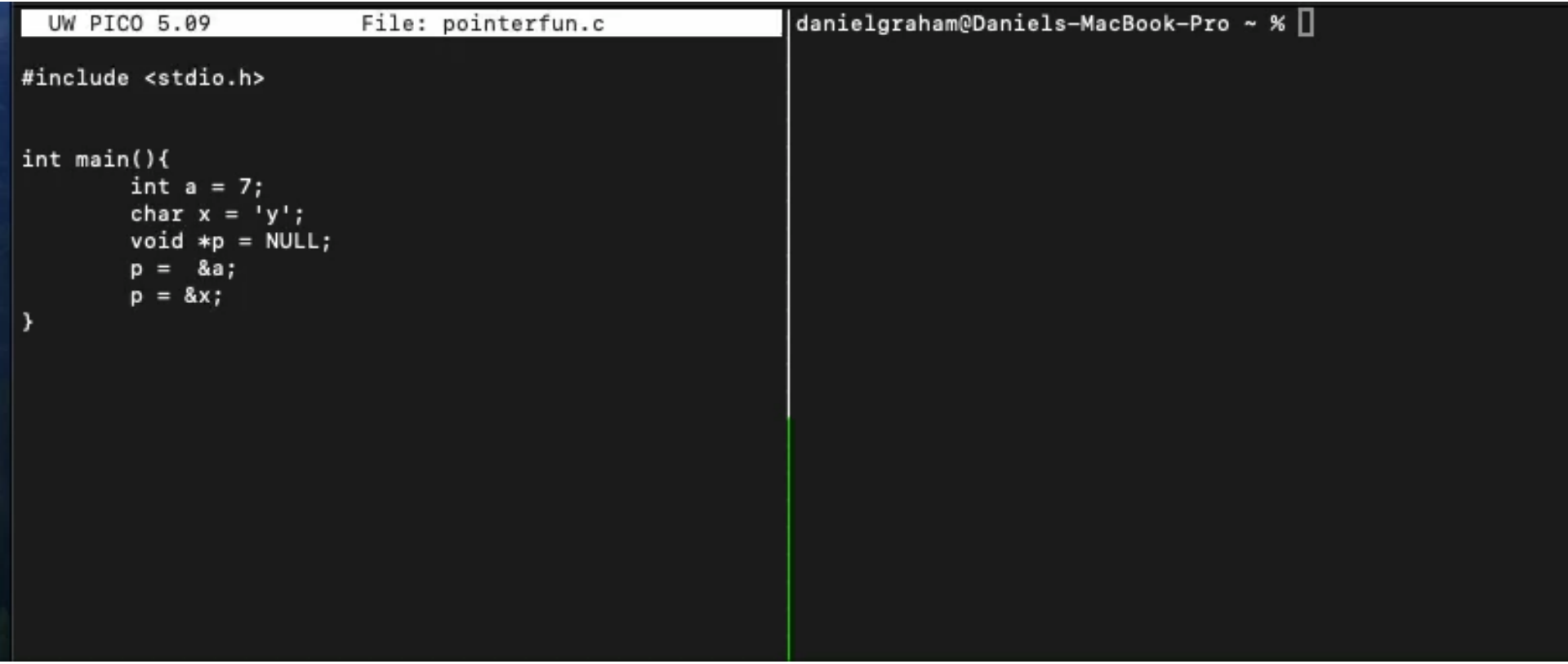
VOID * POINTER

```
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= &x;
```


UW PICO 5.09 File: pointerfun.c
#include <stdio.h>
int main(){
int a = 7;
char x = 'y';
void *p = NULL;
p = &a;
p = &x;
}
danielgraham@Daniels-MacBook-Pro ~ %

WHAT ABOUT USING A VOID STAR POINTER

```
int a = 7;
char x = ‘y’;

void *p = NULL;
p =  &a;
p= &x;

char q = *p;
```

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

Not a problem we can just cast

But we need to be careful how we cast

Can’t dereference void * pointer.
So we need to cast.

UW PICO 5.09 File: pointerfun.c

```
#include <stdio.h>


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

```
danielgraham@Daniels-MacBook-Pro ~ %
```

```
  UW PICO 5.09          File: pointerfun.c

#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
```

```
danielgraham@Daniels-MacBook-Pro ~ %
```

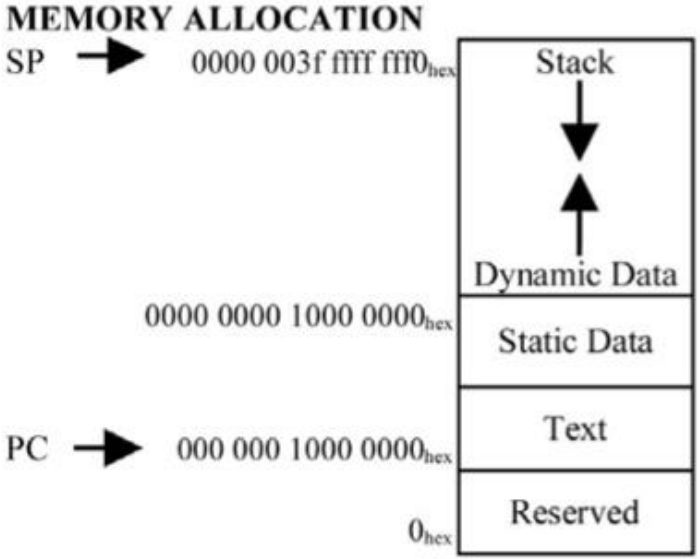
```
[0] 0:zsh*                                  "Daniels-MacBook-Pro.l" 21:17 02-Nov-2
```

STACK VS HEAP

So all the variables that we have looked 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 have both the starting memory address of the stack and the 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 it 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 it 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.

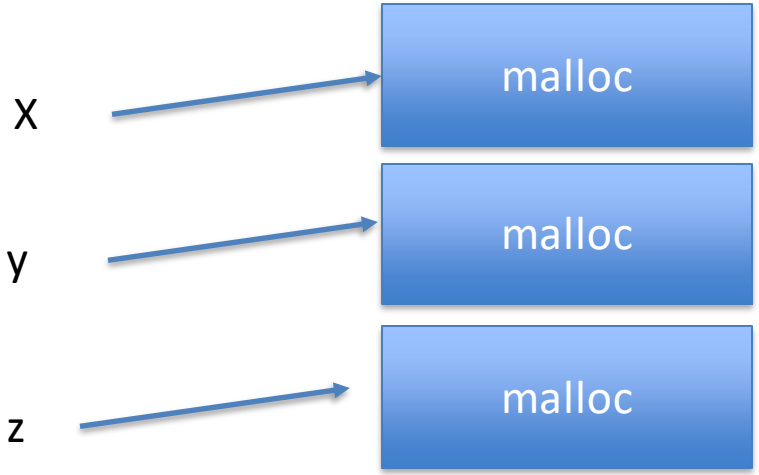
```
  GNU nano 6.3          funwithalloc.c
#include <stdio.h>
#include <stdlib.h>

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.

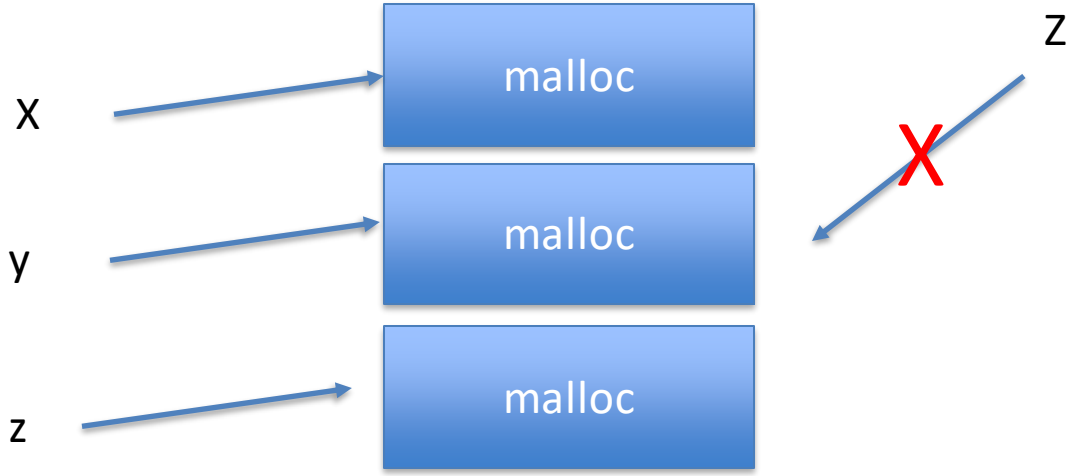


FREE

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



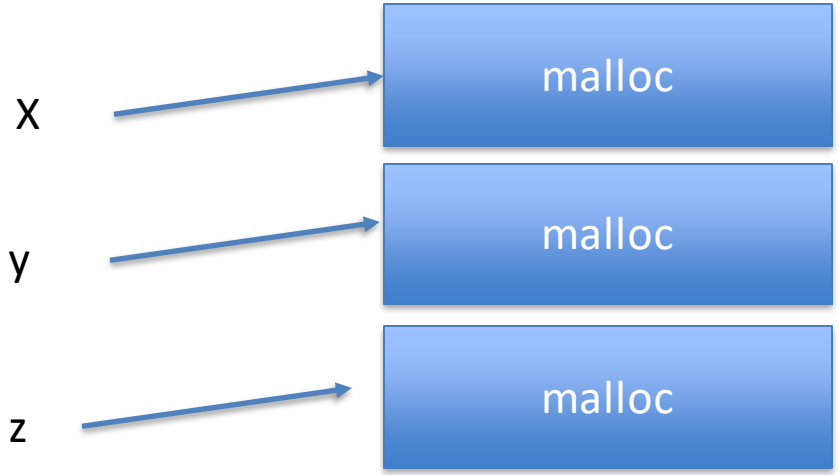
FREE

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

Correction from lecture. Free's return type is void not void *

FREE

```
free(y);
```



FREE

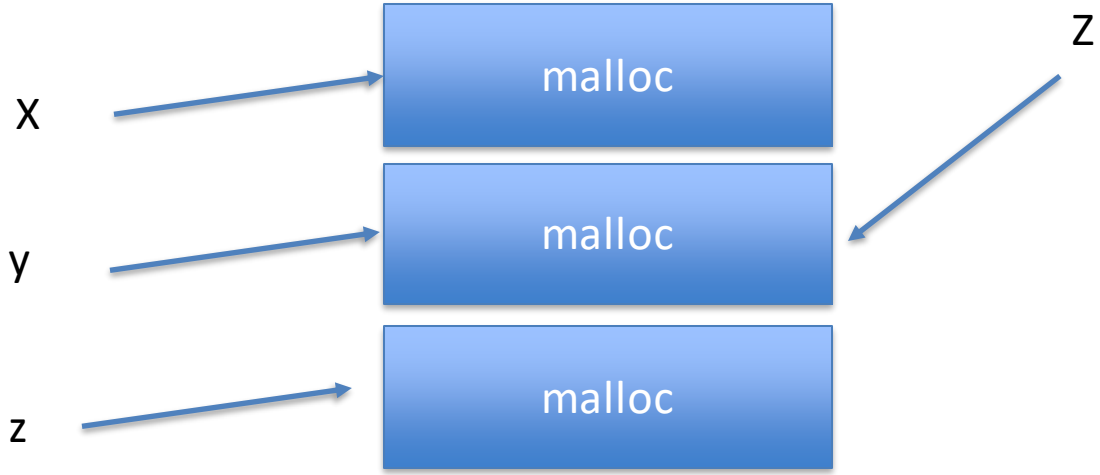
```
free(y);
```

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 it value

But now z can use the space

Because it has been freed



```
 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 < 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
```

```
dgg6b@portal09:~$
```

```
[0] 0:bash*                                  "portal09" 22:38 02-Nov-23
```

```
 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 < 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
```

```
dgg6b@portal09:~$ ./a.out
0 0
1 1
2 2
3 3
4 4
5 5
6 6
7 7
8 8
9 9
10 10
11 11
12 12
13 13
14 14
15 15
16 16
still there 4329
still there 0
still there 121498076
still there 179355929
still there 0
still there 0
still there 0
still there 0
still there 0
still there 0
dgg6b@portal09:~$
```

```
[0] 0:bash*                                   "portal09" 22:39 02-Nov-23
```

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.

BINARY TREE

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

struct TreeNode {
    int data;
    struct TreeNode* left;
    struct TreeNode* right;
};
```

BINARY TREE

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

struct TreeNode {
   int data;
   struct TreeNode* left;
   struct TreeNode* right;
};
```

```
struct TreeNode* createNode(int data) {
   struct TreeNode* newNode  =(struct TreeNode*)malloc(sizeof(struct TreeNode));
   if (newNode) {
      newNode->data = data;
      newNode->left = newNode->right = NULL;
   }
   return newNode;
}
```

BAD INSERT FUNCTION

```
void insert(struct TreeNode* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else {
        if (data < root->data) {
            insert(root->left, data);
        } else {
            insert(root->right, data);
        }
    }
}
```

```
int main() {
    struct TreeNode* root = NULL;
    insert(root, 50);
    insert(root, 30);
    insert(root, 70);
    insert(root, 20);
    insert(root, 40);
    return 0;
}
```

BAD INSERT FUNCTION

```
void insert(struct TreeNode* root, int data) {
    if (root == NULL) {
        root = createNode(data);
    } else {
        if (data < root->data) {
            insert(root->left, data);
        } else {
            insert(root->right, data);
        }
    }
}
```

```
int main() {
    struct TreeNode* root = NULL;
    insert(root, 50);
    ---SNIP---
}
```

Draw state of memory

FIXED INSERT NODE

```
void insert(struct TreeNode** root, int data) {
    if (*root == NULL) {
        *root = createNode(data);
    } else {
        if (data < (*root)->data) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}
```

```
int main() {
    struct TreeNode* root = NULL;

    // Insert some elements into the binary tree
    insert(&root, 50);
    insert(&root, 30);
    insert(&root, 70);
    insert(&root, 20);
    insert(&root, 40);
    return 0;
}
```

FIXED INSERT NODE

```
void insert(struct TreeNode** root, int data) {
    if (*root == NULL) {
        *root = createNode(data);
    } else {
        if (data < (*root)->data) {
            insert(&(*root)->left, data);
        } else {
            insert(&(*root)->right, data);
        }
    }
}
```

```
int main() {
    struct TreeNode* root = NULL;
    insert(&root, 50);
     ---SNIP---
}
```

Draw state of memory