Dynamic Memory Allocation

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

- allocates (reserves) size bytes of contiguous memory in the heap
- returns the location (memory address) of the start of this block
- returns NULL if it fails

Dynamic Memory Allocation

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

- deallocates (releases) the block of memory that ptr points to so that it can be allocated again
- if the block of memory has already been freed, does nothing

- the memory location of the block of memory to be freed should have been returned from a previous call to malloc/calloc/realloc
  ⇒ if it was not then the behaviour of free is undefined
- you can free a memory block using any pointer to it

Dynamic Memory Allocation

Potential Problems

a.) memory leaks

b.) dangling pointers

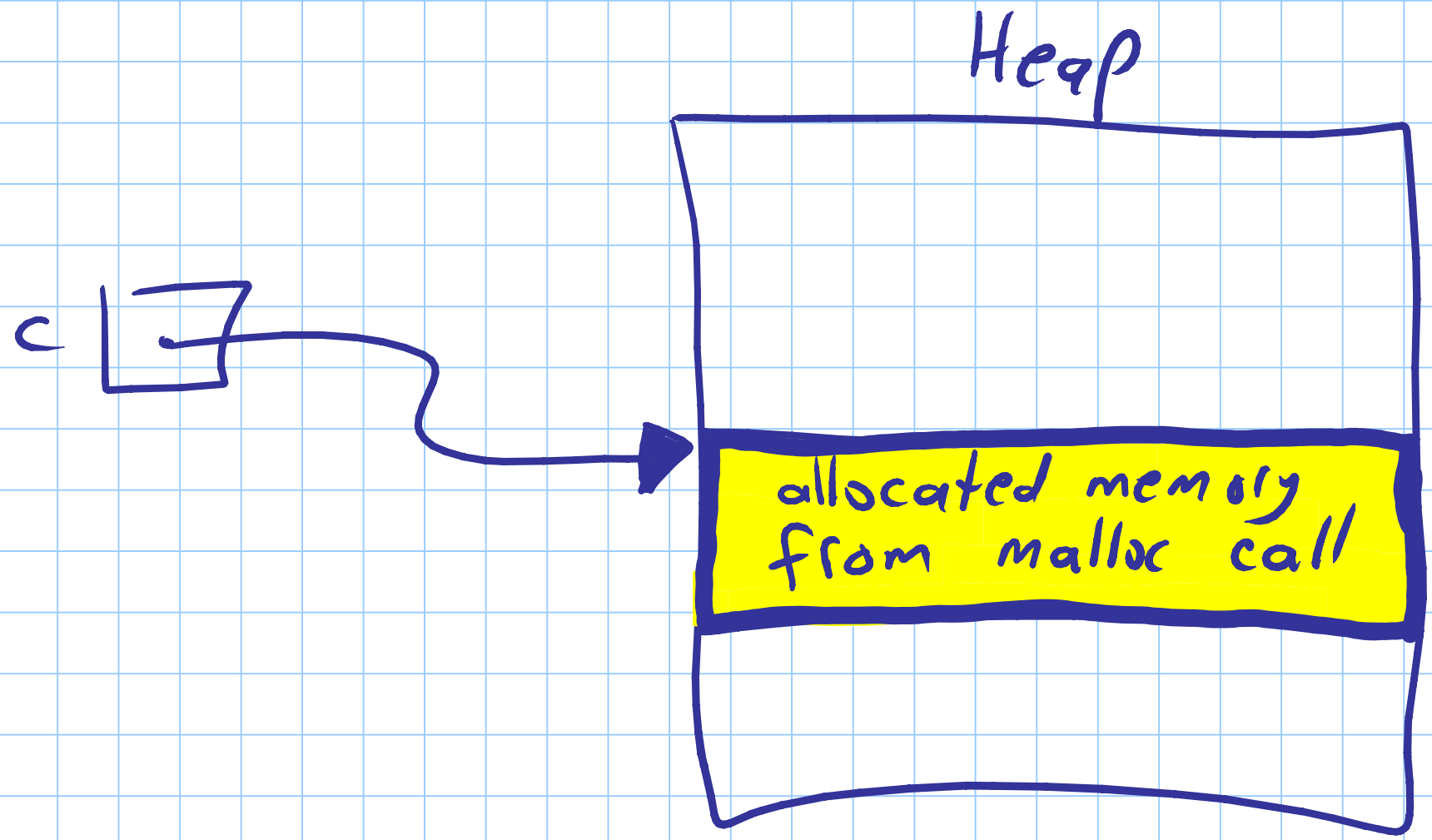
Dynamic Memory Allocation

A memory leak in C occurs when there is a block of memory that is no longer directly accessible to the program

That is, we have a block of memory (the we have allocated) but no longer have a pointer to it

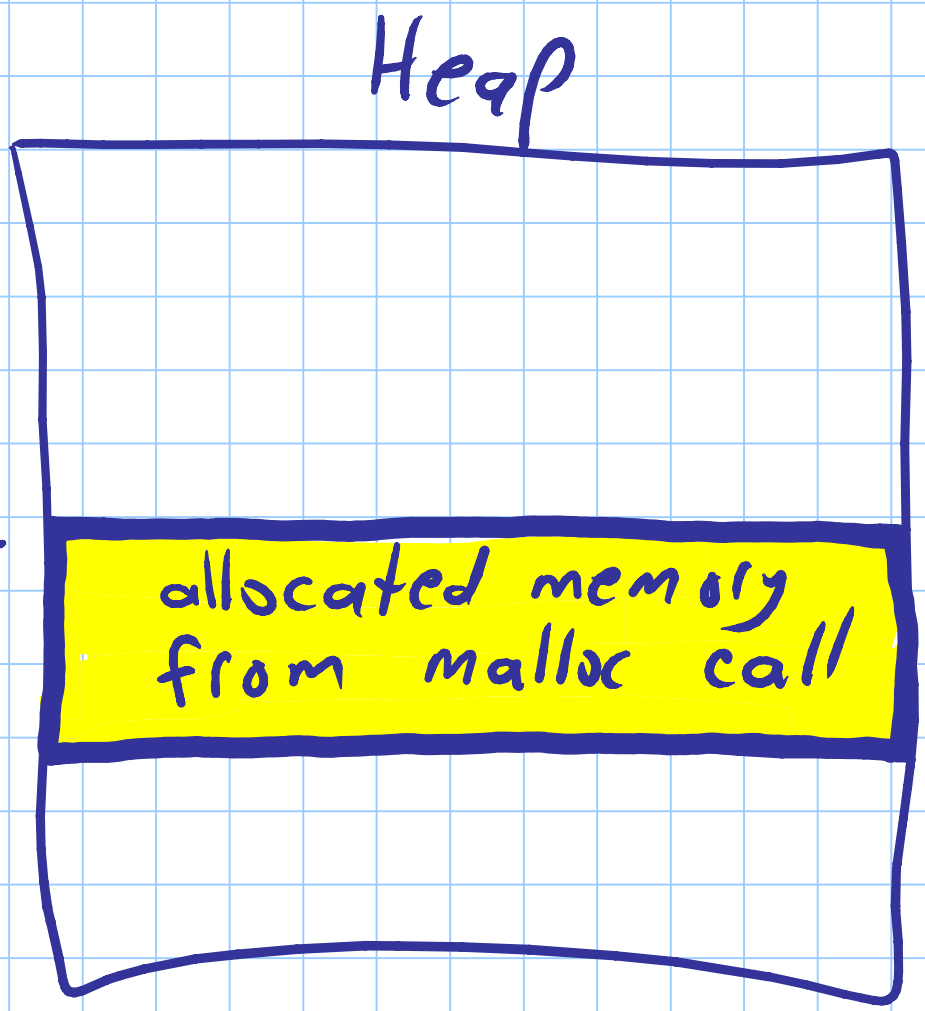
Memory Leak Example 1

```
char * c = (char *) malloc (100);
```



Memory Leak Example 1

```
char * c = (char *) malloc (100);
c = NULL;
```



memory is still allocated but we cannot access it any more (we can't find it)

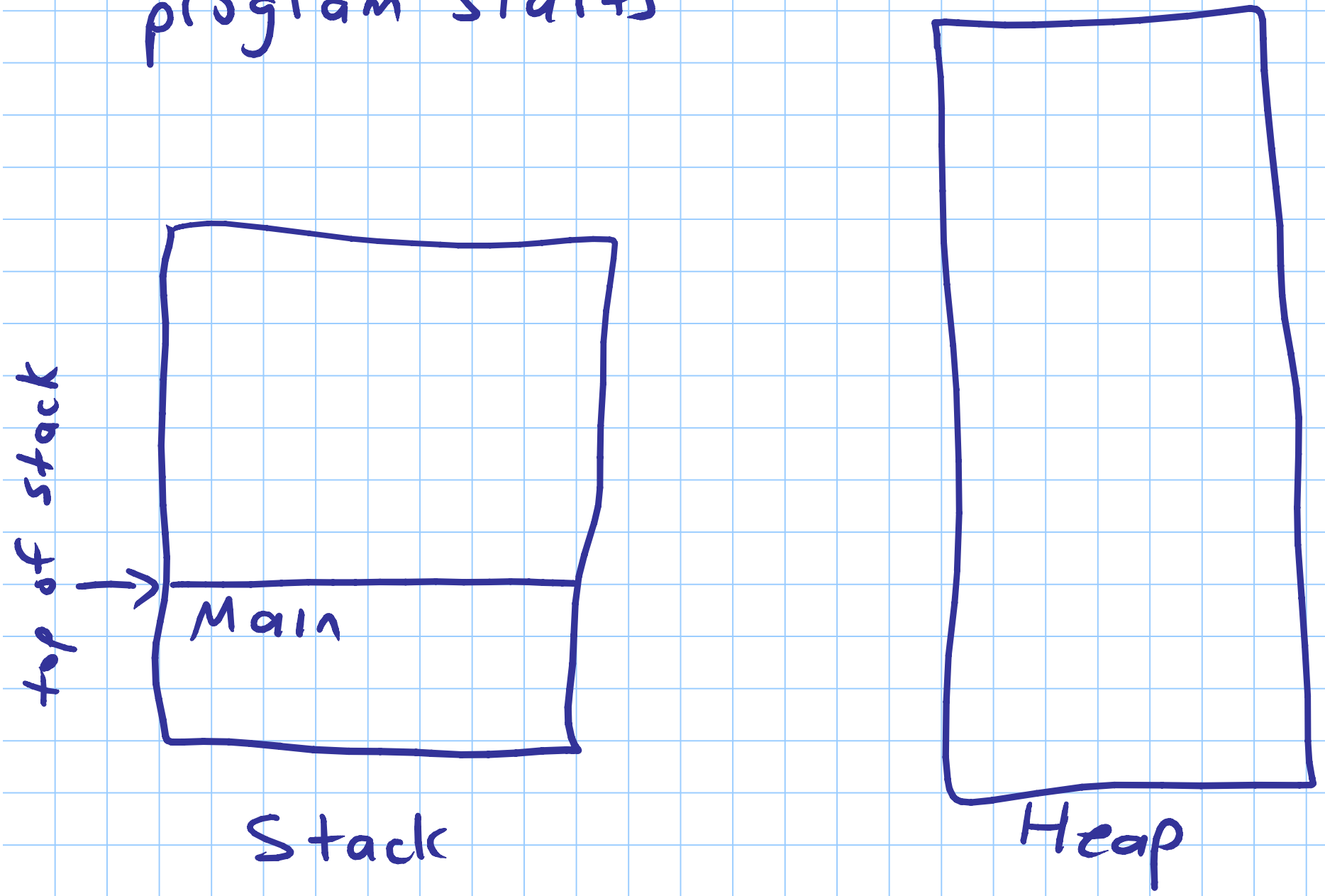
Memory Leak example 2

```
void foo() {
    int * z = (int *) malloc( sizeof(int));
    *z = 3;
}

int main() {
    foo();
    ...
}
```

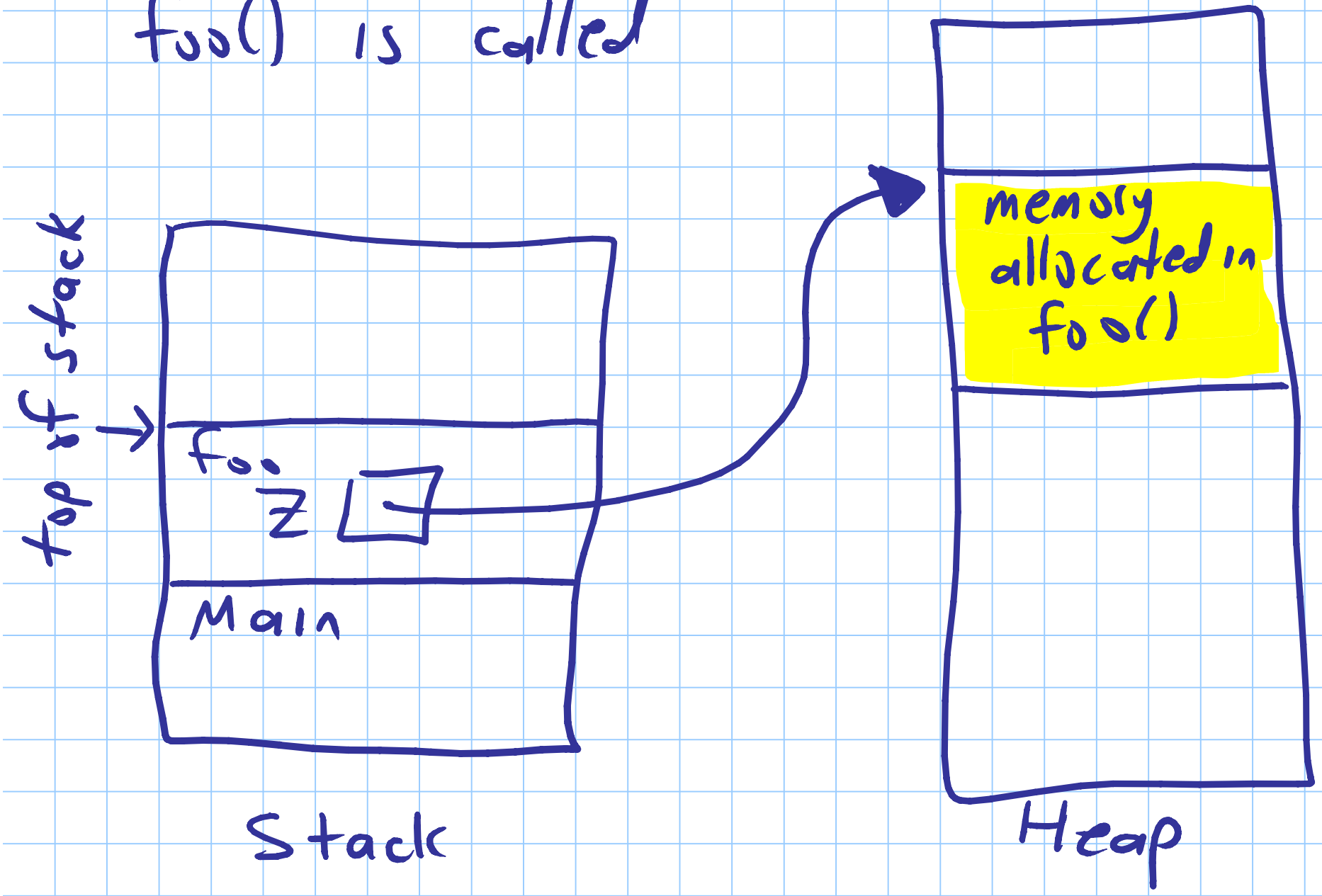
Memory Leak example 2

program starts



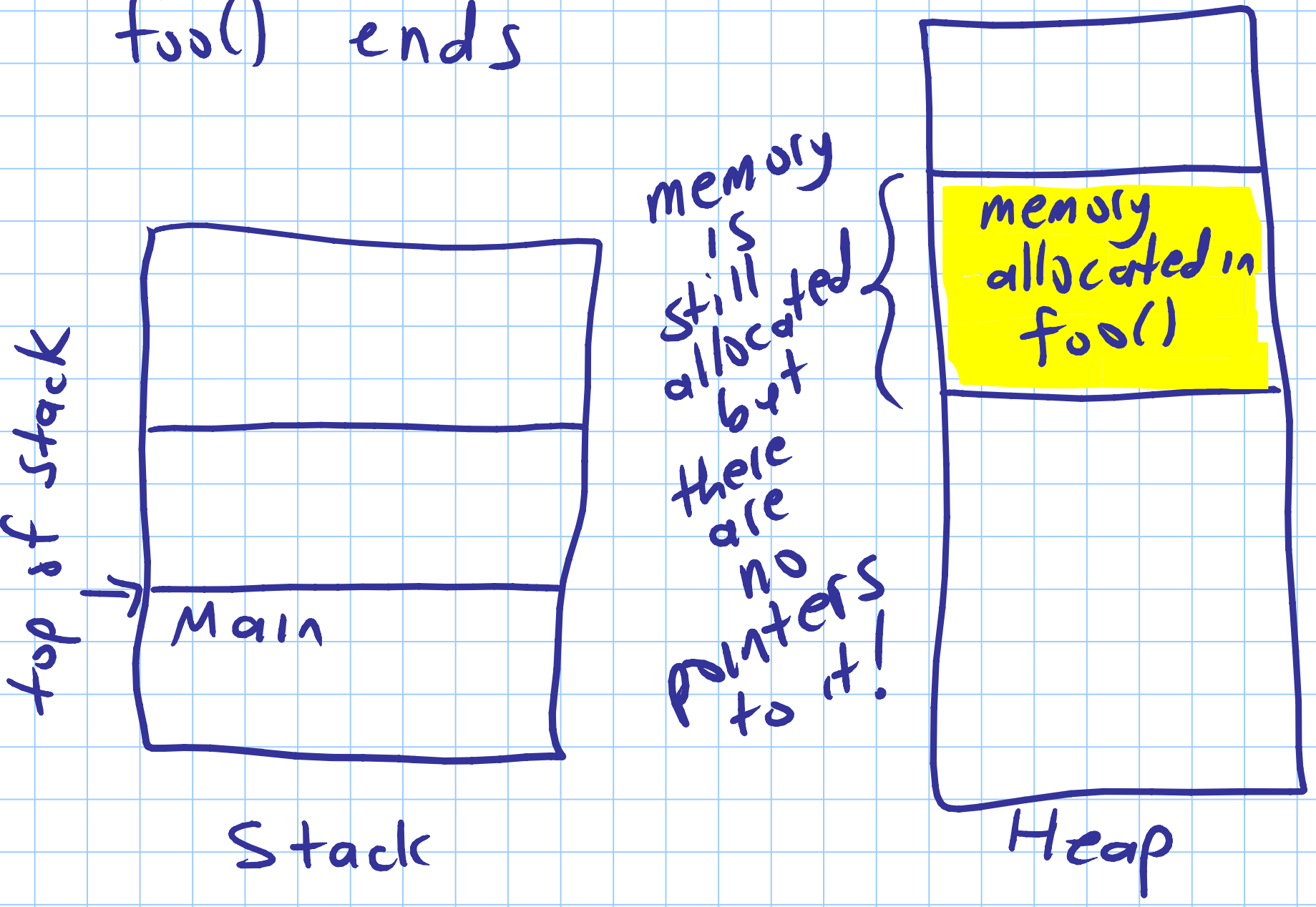
Memory Leak example 2

foo() is called



Memory Leak example 2

foo() ends



Dynamic Memory Allocation

when a block of memory is released (calling free()) all pointers to it become dangling pointers

Modifying the memory that a dangling pointer points to can cause serious (and subtle) errors

Dangling Pointer Example

```
char * c = (char *) malloc (100);
char * d = c
free (d);          // c/d are now both
                   // dangling pointers
...
c[0] = 'a';        // what does this
                   // do?

                   // we may have just
                   // overwritten an
                   // important piece of data
```

Dangling Pointers

whenever we free a block of allocated memory, we should set all pointers (pointing to this newly freed block) to NULL

```
int * y = (int *) malloc (3 * sizeof(int));
free(y) ;
y = NULL;
```

Dangling Pointers

Warning: dereferencing NULL gives a SEG FAULT ERROR.

We will have to check our pointers (against NULL) to see if it is safe to access/modify what a pointer points to

(this is not a big deal. We had to do this in Java too!)