Dynamic Mem Alloc Day 1 (5/13)

Wednesday, May 13, 2020 4:05 PM

Almost all the memory you allocate in intro to c is statically allocated
This means the memory requirement is known at compile time.

Examples

```
int x;
Int values[1000];
```

BAD

```
int n;
// set n to something
int values[n]; // n isn't known at compile time
```

Wouldn't this be a nice capability? YES!!!

Dynamic Mem Allocation is allocating memory
At run time w/o necessarily knowing much memory
You will need before you run the program.

Things that are true about statically allocated memory
---

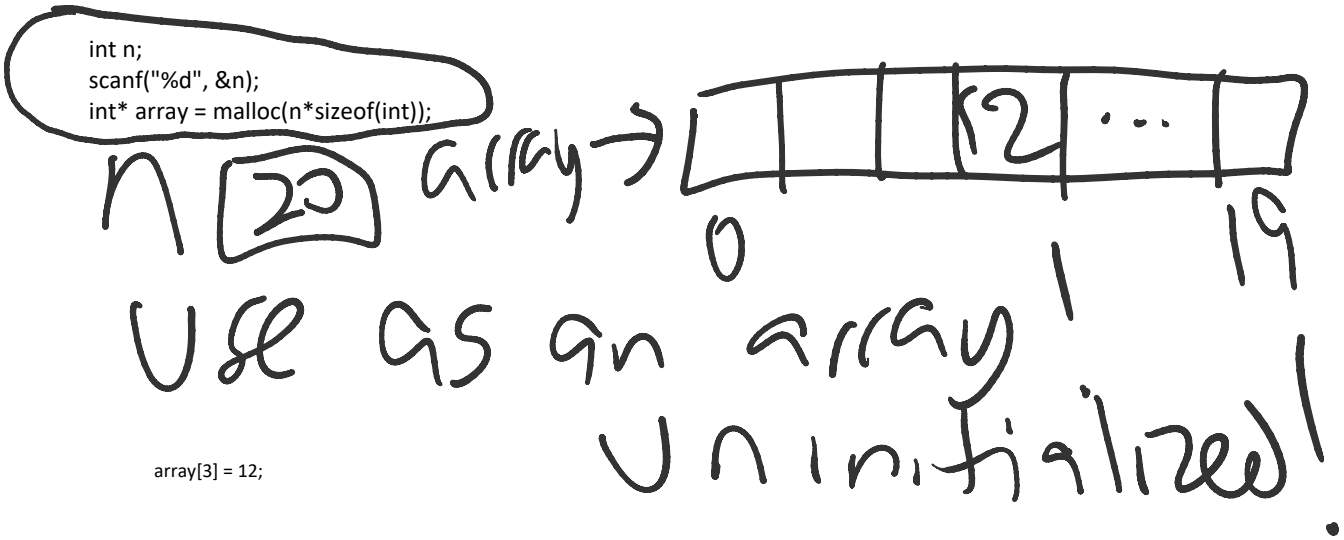
1. Size of it is fixed at compile time.
2. It "survives" (is in scope) from the time it is declared to the curly Brace that closes the nearest open that it came right after.
3. You are limited in how much you can declare by the stack size and the stack is smaller than the heap, where dynamically allocated memory comes from.

Why use dynamically allocated memory?

1. To have memory you allocate persist longer than the function within which it was declared.
2. When you want to have a lot of memory!
3. Dynamically allocated memory is always referred to by pointers and pointers are very quick to pass between functions.

Function: malloc - takes in a single integer - the number of bytes you want to allocate returns a pointer to that memory. If the malloc fails, null is returned.

```
int n;
scanf("%d", &n);
int* array = malloc(n*sizeof(int));
```



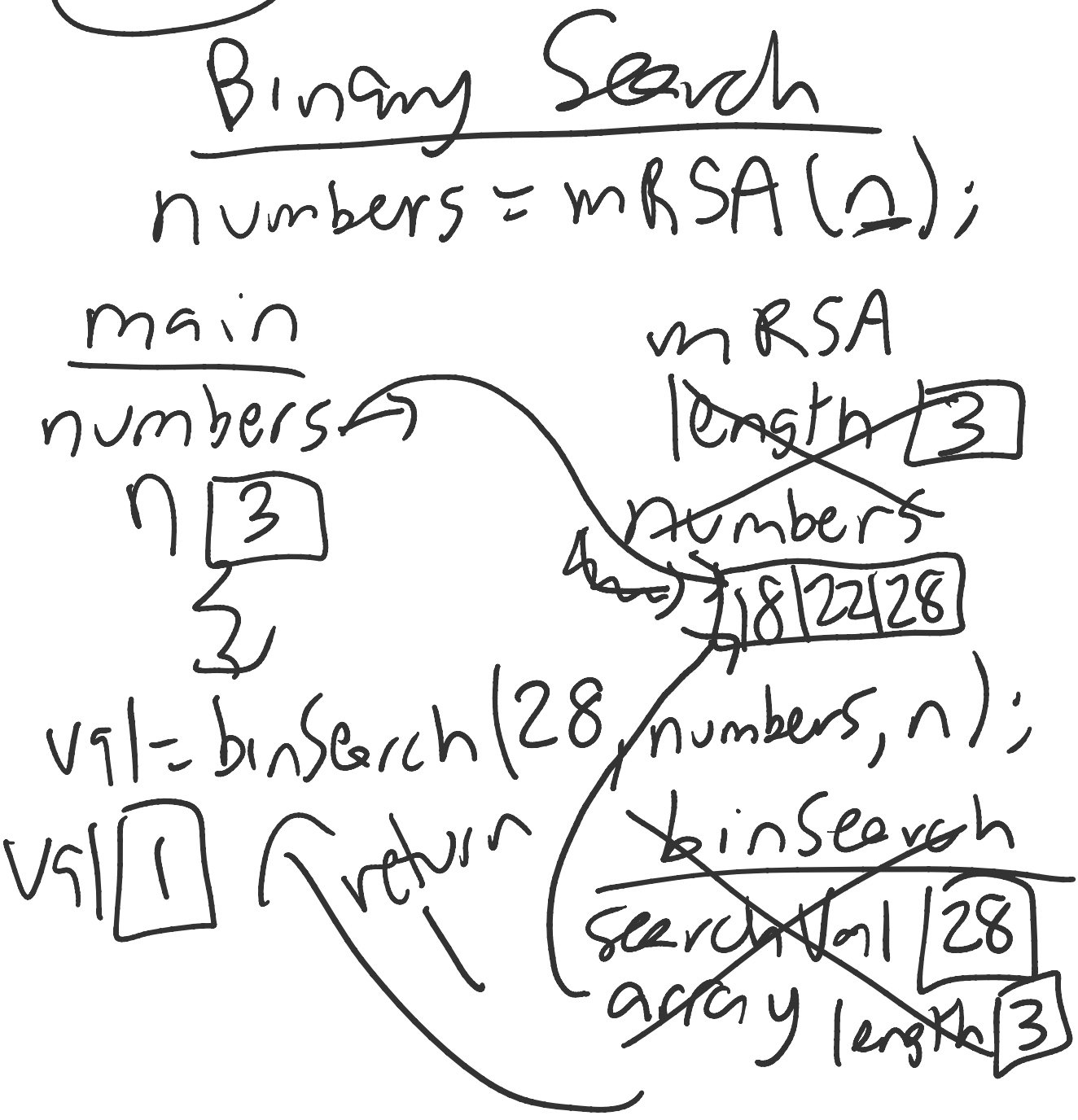

```
array[3] = 12;
```

When you are done using the memory you dynamically allocated, you must free it:

When you are done using the memory you dynamically allocated, you must free it:

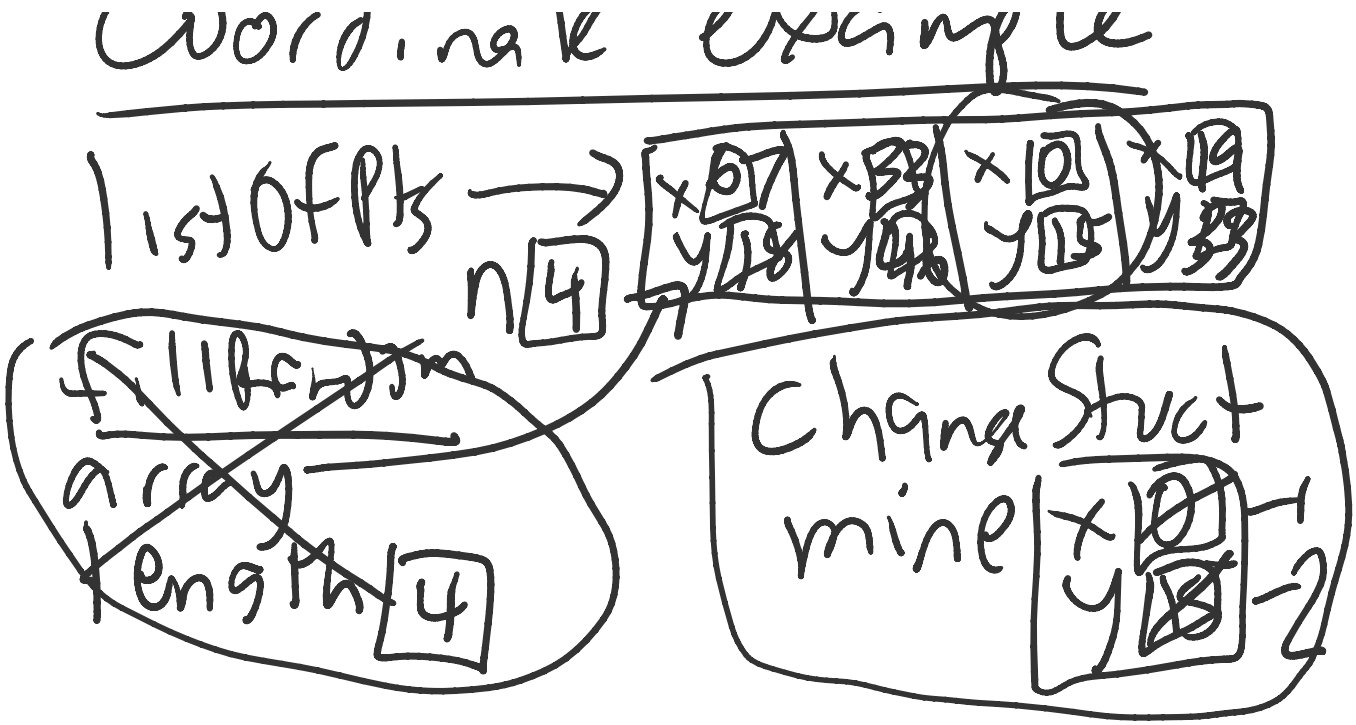
free(array)

Binary Search

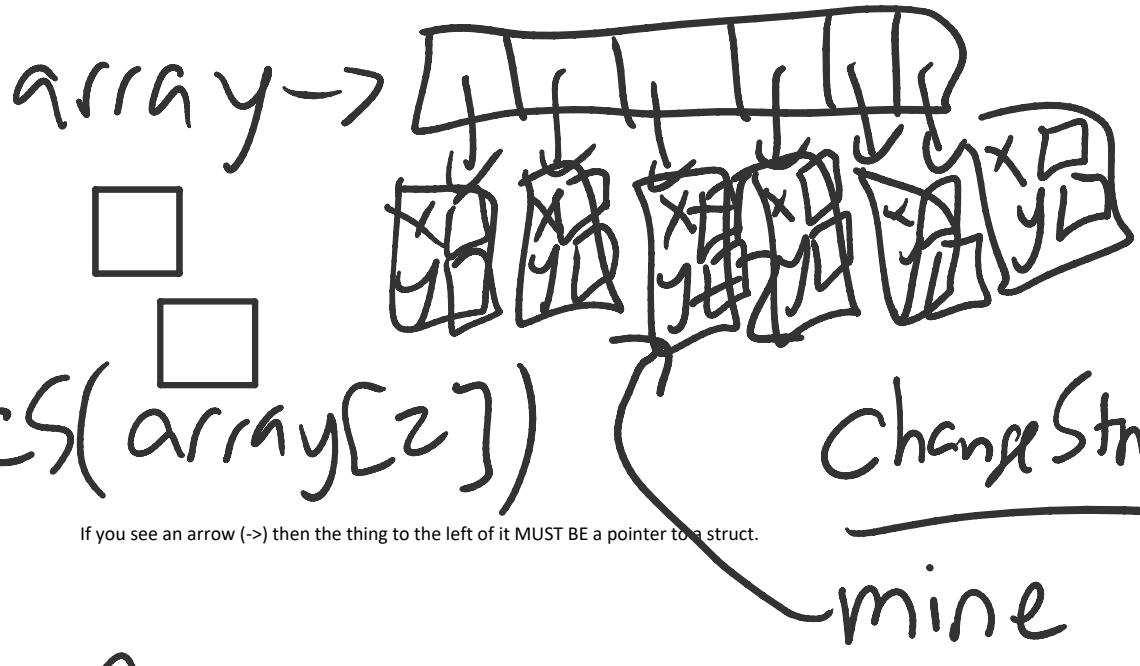


1. Array of Primitives (did this with binary search of ints)
2. Array of Structs
3. Array of Pointers to Structs
4. Array of Strings, where each string is the same size but dynamically allocated
5. Array of Strings, where each string is potentially a different size.

Coordinate example



Array of ptr to struct



If you see an arrow (->) then the thing to the left of it MUST BE a pointer to a struct.

Arrays of Strings

Wasted space!!!

char** words = malloc(n*sizeof(char*))

Wasted space!!!

```
char** words = malloc(n*sizeof(char*))
```

n (5) words →

```
for (int i=0; i<5; i++)
    words[i] = malloc(10*sizeof(char));
```

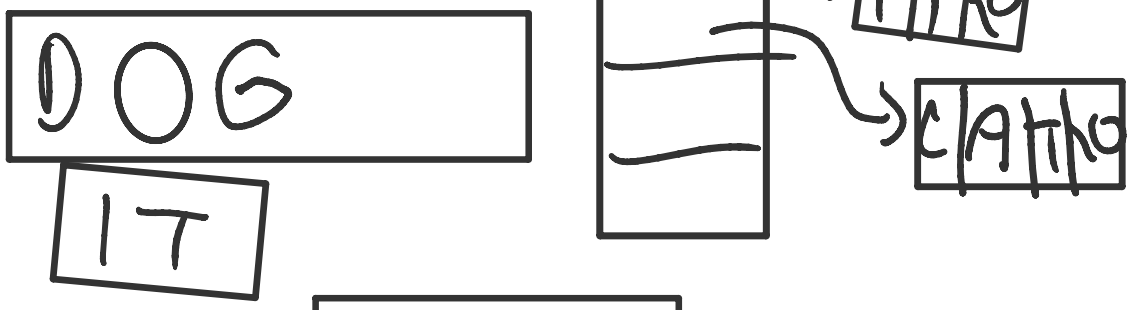
```
for (int i=0; i<5; i++)
    scanf("%s", words[i]);
```

SAME SIZE STR

Array of Strings DIFF Sizes

---

array —



word

array 257