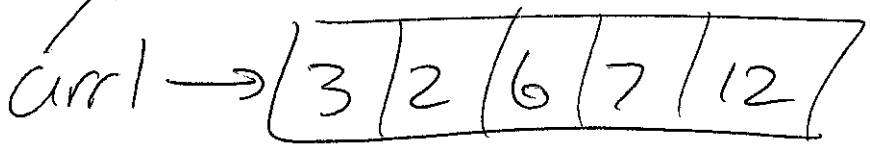
pointer
mem address



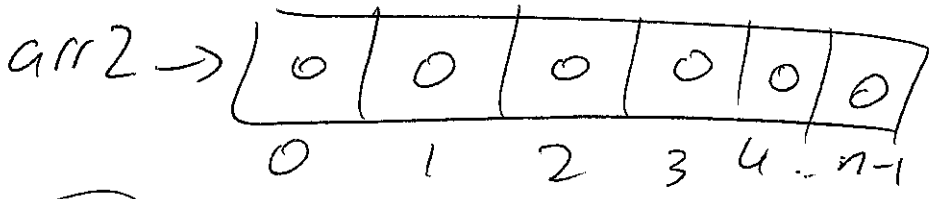

```
scanf("%d", &arr1[i]);   } equivelent
scanf("%d", arr1+i);     }
```

arr1[2] → eval to 6

How to allocate an array dynamically so everything is "zeroed out".

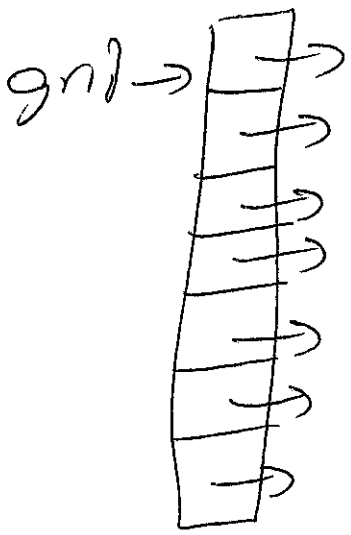
```
int* arr2 = calloc(n, sizeof(int));
```

n: # elements
sizeof(int): size of each elem.



```
arr1 = realloc(arr1, numbytesnew);
```

arr1: new ptr
arr1: orig ptr
numbytesnew: number of bytes we want now



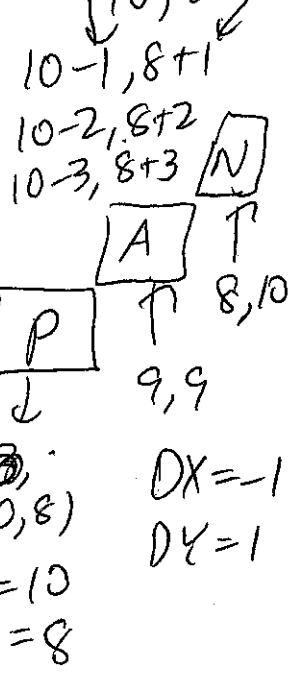

```
int** grid = malloc(r * sizeof(int*));
```

Why DX/DY arrays are nice

```
for (i=0; i<8; i++) {
    // Check word in direction i
    // Look at indexes
(x,y), (x+DX[i], y+DY[i]),
    (x+2DX[i], y+2DY[i]),
    (x+3DX[i], y+2DY[i]),
}
    ...
```

to move char to char

```
tempx += DX[i];
tempy += DY[i];
```



1) array of int/char ✓

2) array of 2D int/char ✓
(2D array of char is an array of strings)

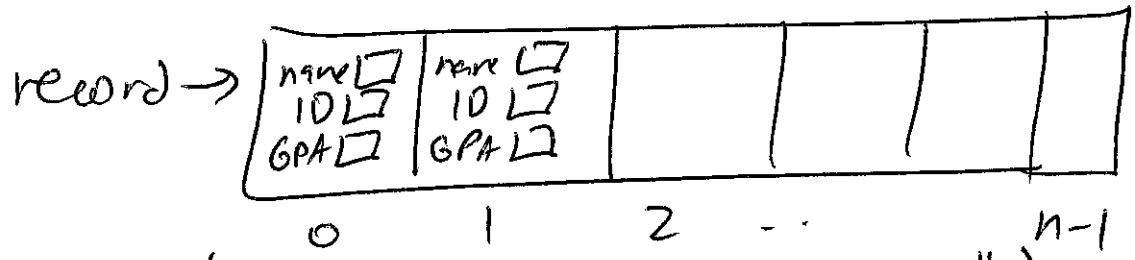
---

3) array of struct

4) array of ptr to struct

5) ptr to struct that has a ptr

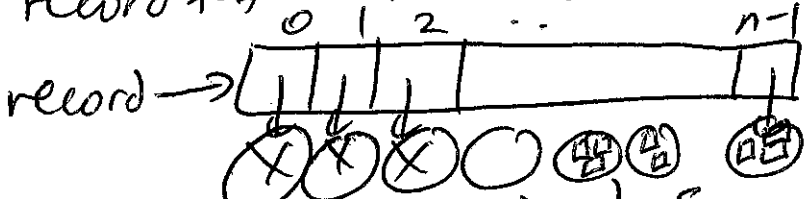
record* = malloc(n * sizeof(record))
= calloc(n, sizeof(record))



strcpy(record[i].name, "JohnDoe");
free(record);

Array of ptr to struct

record** = malloc(n * sizeof(record*));



for (i=0; i<n; i++)
record[i] = NULL;

for (i=0; i<n; i++) {
record[i] = malloc(sizeof(record));
strcpy(record[i]→name, "JohnDoe");
}

*record[i].name (equivalent)

To free

```
for (i=0; i<n; i++)
    free(record[i]);
free(record);
```

"Every malloc call has an equal and opposite free call"

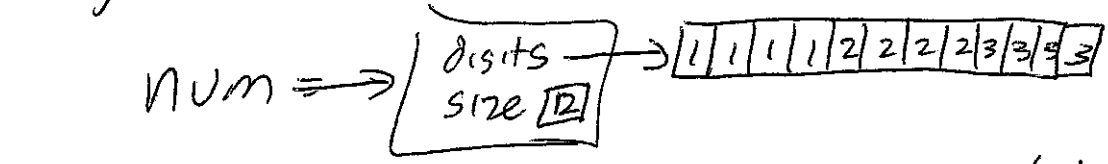
---

ptr to struct that has a ptr.

str 111122223333

~~intptr~~

bigint* num = malloc(sizeof(bigint))




```
//I want to copy contents of str (char*)
// into digits.
num->size = strlen(digits);
num->digits = calloc(num->size, sizeof(int));
for(i=0; i<num->size; i++)
    digits[i] = str[i] - '0';

} //JUST 2 FREES
free(num->digits);
free(num);
```

---

str C A T \0 @ $2\times26^2 + 0\times26^1 + 19\times26^0$

```
int val = 0;
for(i=0; i<strlen(str); i++)
    val = 26*val + str[i] - 'A';
```

main

values = create values

create

temp