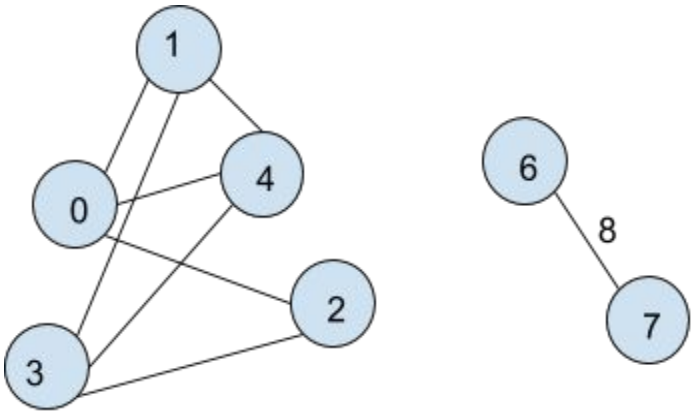
Arrays: int a[100] you can only use 99 of those slots cause the last one is the a null
a is a pointer.
a[0] = whatever
arrays are made of ints.
char a[100] is the same as a string but it is an array of chars
2d array int a[8][8] aka matrix
a[6][8] is same as a[0][48] when

```
int a[10]={1,2,3,4,5,6,7,8,9,10};???
int b[10];
int i, sum;
int *p;
p = b;
for(i = 0; i<100;i++){
        if(its_prime(a[i]){
                *(p++) ++ a[i];
        }
}num_of_primes = p-b;
```

p doesn't garbage collect
just enough space for a pointer

array of pointers,



Imagine there are numbers between each node listed below

nodes	0	1	2	3	4
0	0	7	8	100	2

1		0	100	1	100
2			0	7	100
3				0	5
4					0

Distance between individual nodes

```
#include <studio.h>
void
warshall (int distances[8][8])
{
        int i,j,k;
        for(k = 0; k < 8; k++){
                for(i = 0; i < 8; j++){
                        for(j = 0; j < 8; j++){
                                if(distances[i][j] > distances[i][k] + distances[k][j]){
                                        distances[i][j] = distances[i][k] + distances[k][j]
                                }
                        }
                }
        }
}
```

This will fill in all the shortest distances, though run time is a bastard

char *s;
char buf[20];
s = buf

s points at buf in memory if buf is changed or garbaged then we're screwed
s = strdup(buf)
how big is s? not as big as buf necessarily
if buf = "abcd"
s has room for abcd and also one more byte. (???)
    you can't use the last byte in a str.
strcmp(s1,s2) this can return -1(s1<s2),0(s1==s2),1(s1>s2)

In Java you allocate memory with New, in C you use malloc(64)
you can say p = malloc(64)
then later you can free(p)

char *s = malloc(100)

strcasecmp will case cmp two strings
strncmp will cmp the first n
strncasecmp will case cmp the first n
Use man pages
man -s3 is keyword search in the c stuff

Time to do linked lists

```
struct elem{
        char *s;
        int count;
        struct elem *next;
};

typedef struct elem{
        char *s;
        int count;
        foo *next;
}foo ;
struct elem a,b;
//or i could say foo c;
a.s
a.count
b.count
```

```
// if we want a pointer we'd
foo *e;
//we can;t do this because pointers don't have fields
*e.s
```

Don't use #define to try to typedef

foo has a pointer an int and a char that's 1 byte, 8 bytes, and 1 byte. That's 10 bytes right? nope, be careful, depending on compiler they may put ints on 8 byte border. You shouldn't be accessing stuff inside a structure by memory address. The inside of the structure should be treated like variables.

You can have nested structs but you can't have recursive structs because they will go on forever. You can have a pointer to another of the same stuct like *next because we know how big pointers are.

You can end a linked list several ways. You can point at itself or be a null pointer.
First elem in a linked list is the head

struct elem * head, cur;
head->count

strcmp(head->s, “the”)
//this moves to the second node
cur = head->next;
//this moves to the next nose
cur=cur->next;
when cur->next == NULL you are done

What do we do if we need to insert a new node.
new_elem = malloc(sizeof(struct elem));
new_elem->s = NULL;
new_elem->count = 0;
new_elem->next = cur->next;
cur->next = new_elem;

When you want to remove something you need to point at the one before.
del = cur->next;
cur->next = del->next;
free(del); //if you want to put it somewhere else you do that instead of killing it.

Doubly linked lists

```
struct dblylinklst{
        char *s;
        int count;
        struct elem *prev;
        struct elem *next;
};
```

When you delete from a doubly linked list you point at the one you’re deleting
cur->prev->next = cur->next
cur->next->prev = cur->prev

sentries are put at the front and/or back and usually have some ridiculous thing inside and then you don’t have to put in special cases to deal with adding or removing from the front or back.