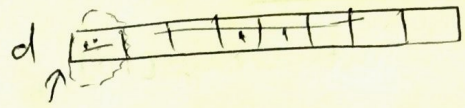
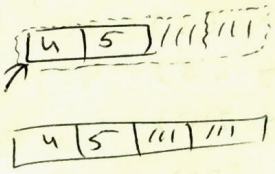
CS 107 Lecture 3

```
double d = 3.1416;
char ch = *(ch*) &d;
cout << ch << endl;
```



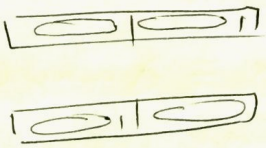
[redacted] dangerous recasting outside

```
double d = *(double*) &s;
```



big endian, little endian

in short



} big endian

} little endian

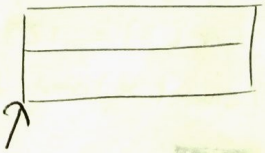
big: lowest byte stored the largest contribution

little: [redacted] bytes in the reverse order

Structs:

```
struct fraction {
    int num;
    int denum;
};
```

in memory
- two 4 byte blocks stacked on top of each other
- pointer to struct is points to the first field

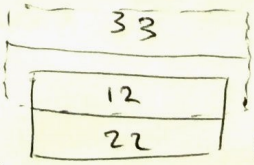



```
fraction pi;
pi.num = 22;
```
← store [redacted] 22 to the field that is at offset of 0 from the pointer to the struct

```
pi.denum = 7;
```
← based on definition, [redacted] stores 7 at offset of 4 bytes from the pointer to the struct

casting with struct pointer

```
fraction pi;
pi.num = 22;
pi.denum = 7;
((fraction*) &(pi.denum))->num = 12;
((fraction*) &(pi.denum))->denum = 33;
```



to access 33

(&pi)[1].num

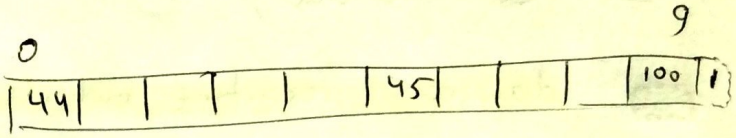
← [redacted] (fraction*) type

arrays :

```
int array[10];
```

allocates 40 Bytes of memory for 10 ints

address of 0th entry




```
array[0] = 44;
array[9] = 100;
array[5] = 45;
array[10] = 1;
array[25] = 25;
array[-4] = 77;
*(array - 4) = 77;
```

array ≡ &array[0]

in C/C++ no bounds checking on arrays; go 10 int quantums from &array[0] and store 1 ■ as int

pointer arithmetic

array + k ≡ &array[k]

type (int *) — integer automatically scaled by type

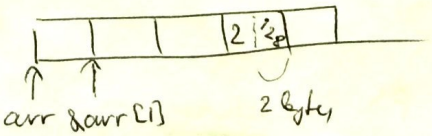
pointer arithmetic

*array ≡ array[0]

*(array + k) ≡ array[k]

ex

```
int arr[5];
arr[3] = 128;
((short *) arr)[6] = 2;
```




```
((short *)(((char *)(&arr[1])) + 8))[3] = 100;
```

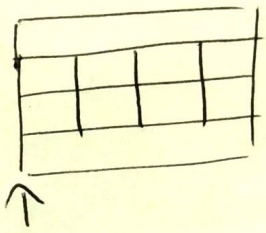
forward from &arr[1] by 8 chars

forward by 3 shorts

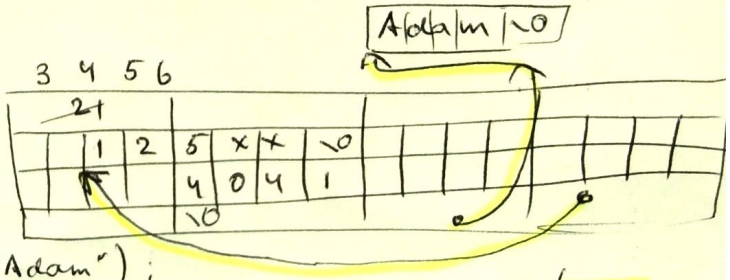
CS 107 Lecture 3

structs with arrays inside:

```
struct student {
    char *name;
    char suid[8];
    int numUnits;
};
```



```
student pupils[4];
pupils[0].numUnits = 21;
pupils[2].name = strdup("Adam");
pupils[3].name =
    pupils[0].suid + 6;
```

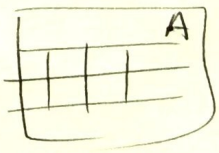


↖ dynamically allocates space for the string, writes it, and returns the address of the first character

```
strcpy(pupils[1].suid, "40415xx");
```

↖ no allocation, assumes the pointer to space is provided by the first argument

```
strcpy(pupils[3].name, "123456")
pupils[7].suid[11] = 'A'
```



generics (start)

```
void swap(int *ap, int *bp)
{ int temp = *ap;
  *ap = *bp;
  *bp = temp;
}
```