Introduction to **Pointers**

ESC101: Fundamentals of Computing

Nisheeth

The sizeof various variable types

8 bits □ make a byte □□□□□□□□

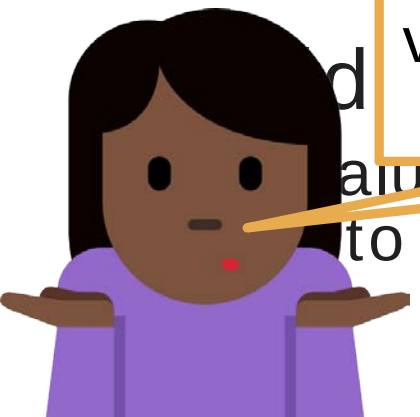
char takes 1 byte = 8 bits

Max value in a char is 127 = $2^{(8-1)}-1$

int/float takes 4 bytes = 32

Max value in int is 2,147,483,647
equal to $2^{(32-1)}$-1 – verify

Because, for signed variables, one bit is reserved for storing the sign



Why is max value for all these variables always $2^{(k-1)}$-1 and not 2^{k} -1 when there are k bits getting used?

d

alue in long is 9,223,372,036,854,775,807
to $2^{(64-1)}$-1 – verify

How Mr C stores variables

He has a very long chain of bytes

Each byte has a non-negative "address"

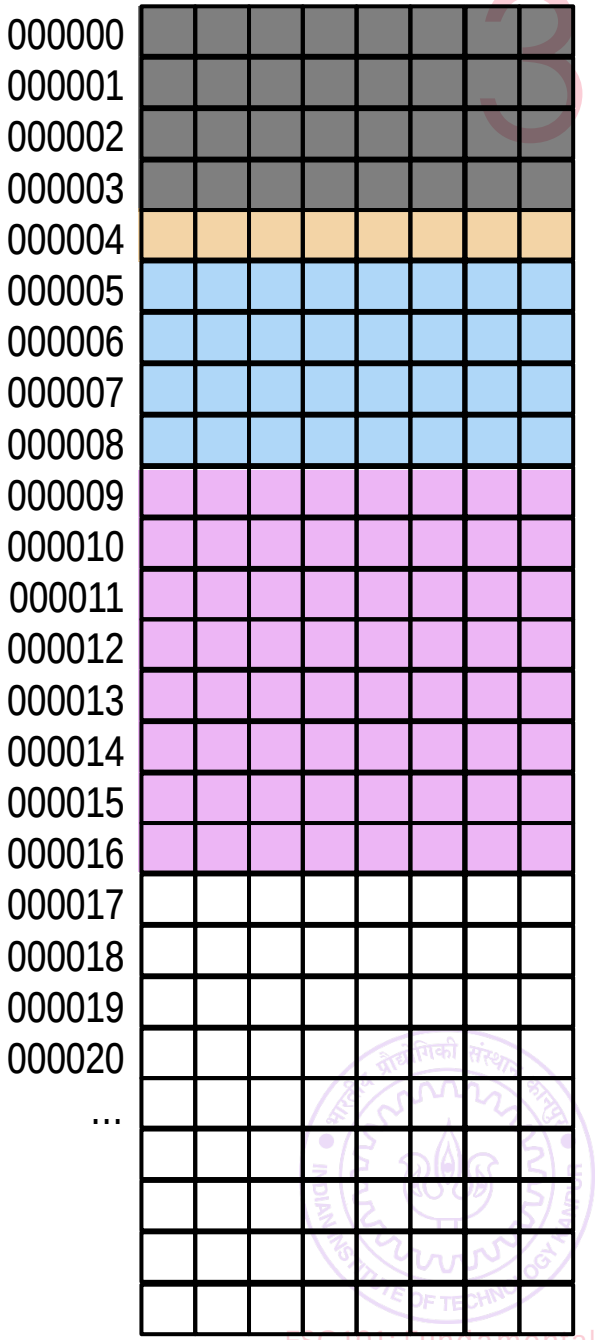
Each address (which is also a number) is stored using 8 bytes (=64 bits)

So there can be a total 2^{64}-1 possible addresses in memory

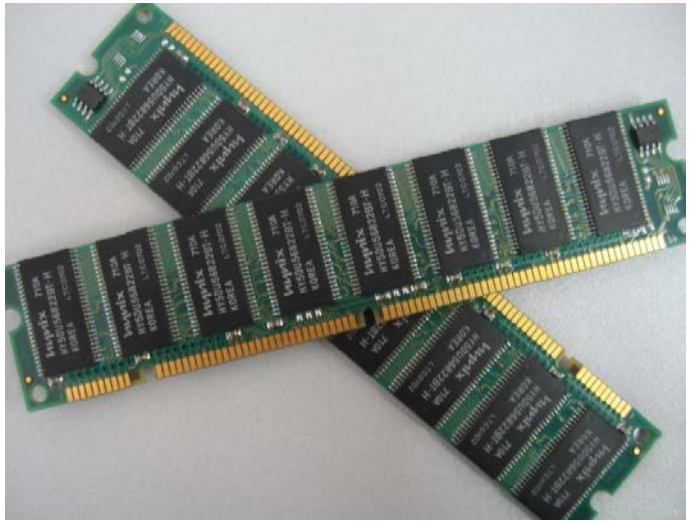
Some addresses are reserved for Mr. C

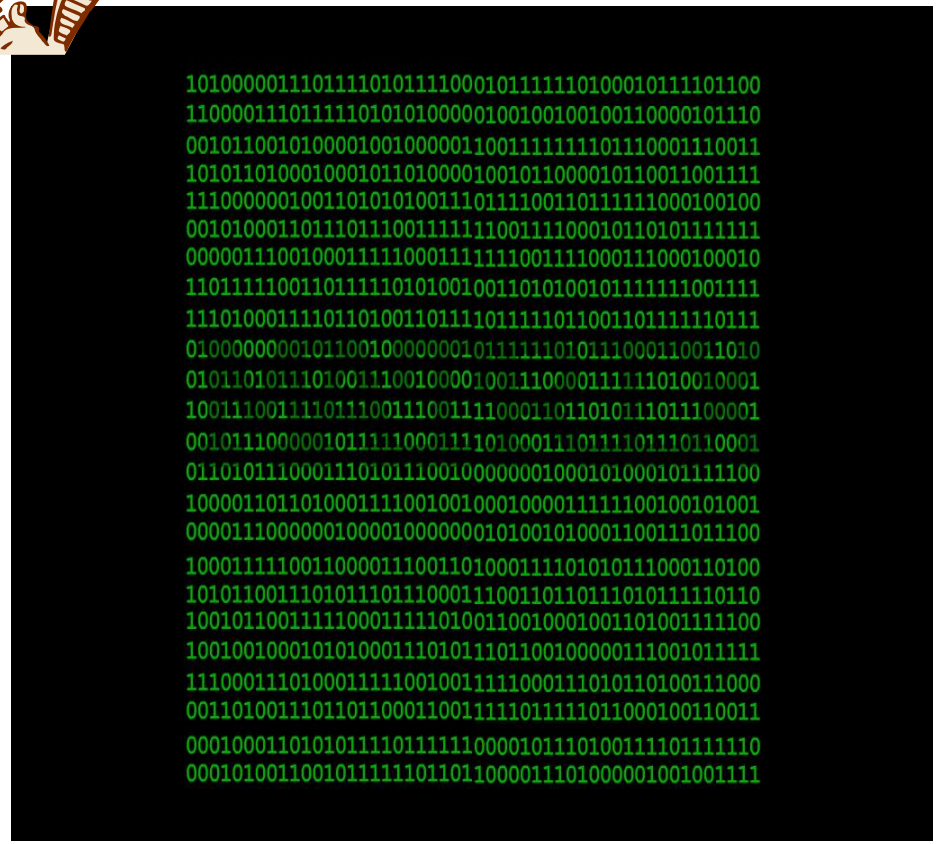
Others can be used by us for variables, e.g.,

```
char c;     // stored at 000004
int a;      // stored at 000005
double d;   // stored at 000009
```



Controlling/managing memory





Pointers

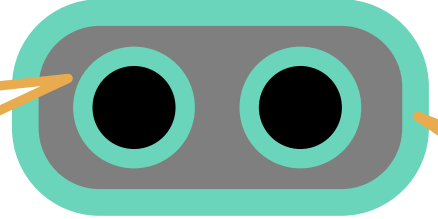
A pointer is just an address and needs 8 bytes to store

Pointers enable us to store/manage memory addresses

In some sense Mr C manages a ridiculously huge array!

Pointers can allow us to write very beautiful code but it is a very powerful tool – misuse it and you may suffer ☺

Our first pointer



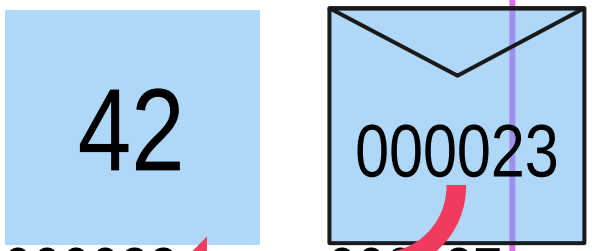
Whew!
Lets begin.

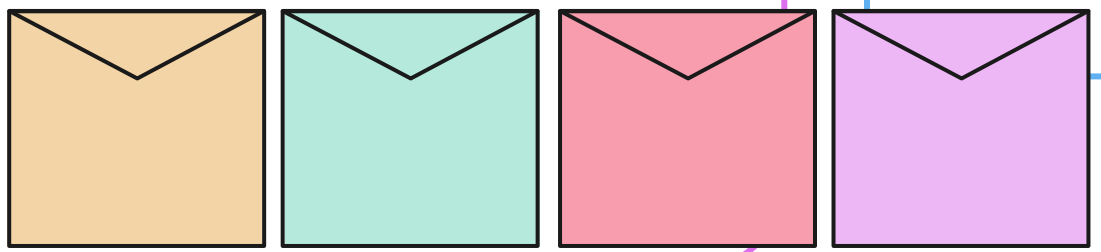
42

HOW WE SPEAK TO MR. COMP **HOW WE USUALLY SPEAK TO A HUMAN**

```
#include <stdio.h>
int main(){
  int a = 42;
  int *ptr;
  ptr = &a;
  printf("%d", *ptr);
  return 0;
}
```

int *ptr;
means ptr
is a pointer
to an
integer



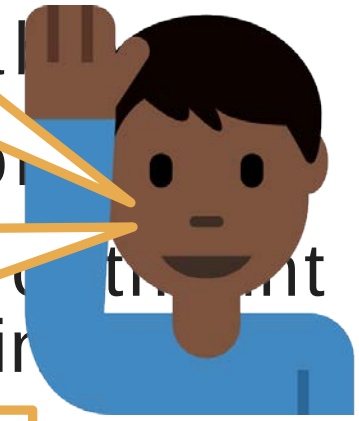


a is an int variable, value 42

ptr is a pointer that will store

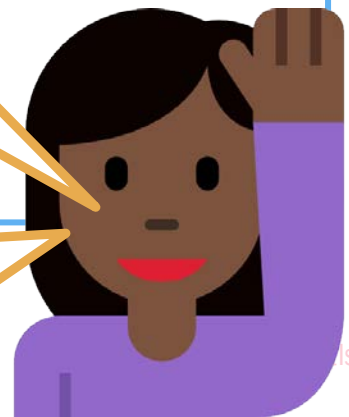
Can also have pointes to
char, long, float, double

All these envelope-like
boxes take 8 bytes



a is stored at internal
location 000023

int takes 4 bytes
to store



Pointers with printf and scanf

Pointers contain addresses, so to print the address itself, use the %ld format since addresses are 8 byte long

To print value at an address given by a pointer, first dereference the pointer using * operator

```
printf("%d", *ptr);
```

Scanf requires the address of the variable where input is to be stored. Can pass it the referenced address

```
scanf("%d", &a);
```

or else pass it a pointer

```
scanf("%d", ptr);
```

Pointers

Can have pointers to a char variable, int variable, long variable, float variable, double variable

Can have pointers to arrays of all kinds of variables

All pointers stored internally as 8 byte non-negative integers

NULL pointer – one that stores address 00000000

- Named constant NULL can be used to check if a pointer is NULL
- Do not confuse with NULL character '\0' – that has a valid ASCII value 0
- NULL character **is actually used** to indicate that string is over
- WARNING: NULL pointers may be returned by some string.h functions e.g. strstr

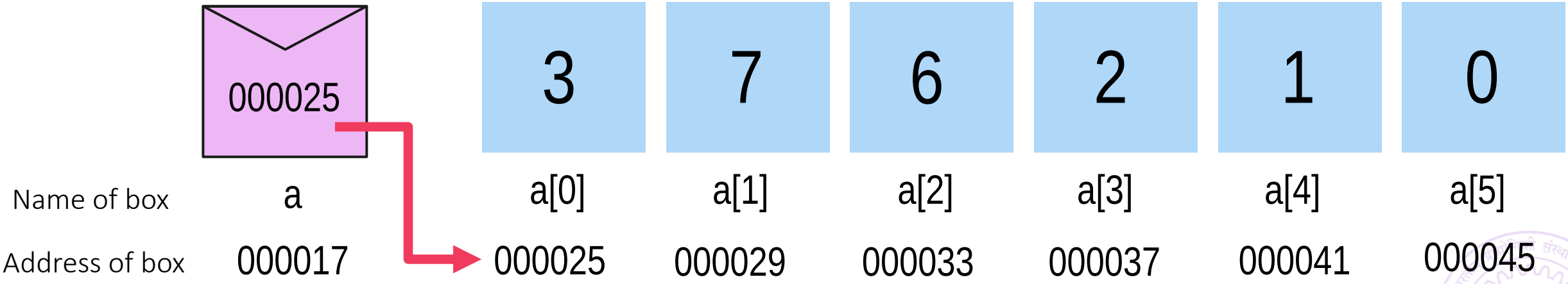
Do not try to read from/write to address 00000000

- Reserved by Mr C or else the operating system
- Doing so will cause a segfault and crash your program/even your computer

Pointers and Arrays

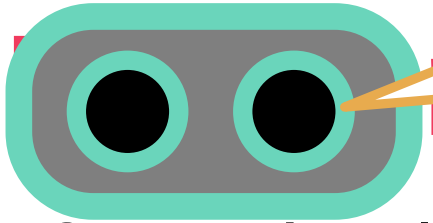

```
int a[6] = {3,7,6,2,1,0};
```

How many boxes in memory will be created for the above declaration + initialization? SEVEN



In case of arrays, the name of the array is the pointer to the first element of the array

Also note that a and a[0] need not be at adjacent addresses in memory (but often are)



Note: Though figure shows them as taking one byte, actually, being pointers (addresses), c and a each would take 8 bytes to store

If we declare an array, a sequence of addresses get allocated

```
char c[5];
int a[3];
```

c is pointer, the whole c[5] denotes the array
a is pointer, the whole a[3] denotes the array

Names c and a are actually pointers, c stores the address of c[0], a stores address of a[0]

c[0] is stored at address 000005, c[1] at address 000006, c[2] at 000007 and so on

a[0] is stored at address 000011, a[1] at address 000015 (int takes 4 bytes), a[2] at address 000019, and so on