What have we learned?

Pointers

- Declaration

```
int *pointer_to_x;
```

- Obtain the address (pointer) **The & operator**

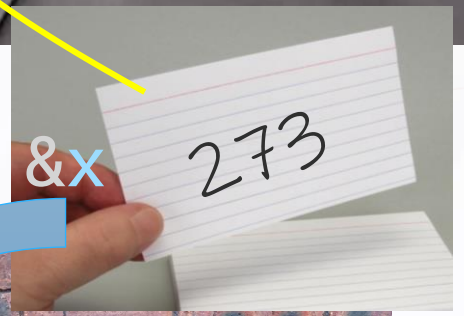
```
pointer_to_x = &x;
```

- Read the contents

```
int value_stored = *pointer_to_x;
```

- Change the contents **The * operator**

```
*pointer_to_x = 100;
```



How a function handles arguments

```c
#include <stdio.h>

void go_south_east(int lat, int lon)
{
    lat = lat - 1;
    lon = lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east(latitude, longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```

```
Avast! Now at: [32, -64]
```

C makes copies of the arguments, before passing them to a function

Passing pointers to a function

```c
#include <stdio.h>

void go_south_east( int *lat , int *lon )
{
    *lat = *lat - 1;
    *lon = *lon + 1;
}

int main()
{
    int latitude = 32;
    int longitude = -64;
    go_south_east( &latitude , &longitude);
    printf("Avast! Now at: [%i, %i]\n", latitude, longitude);
    return 0;
}
```

```
Avast! Now at: [31, -63]
```

To allow a function to change the original content, pass in a pointer!

Array variables

```c
#include <stdio.h>

void fortune_cookie(char msg[])
{
    printf("Message reads: %s\n", msg);
    printf("msg occupies %i bytes\n", sizeof(msg));
}

int main()
{
    char quote[] = "Cookies make you fat";
    fortune_cookie(quote);
    return(0);
}
```

sizeof() - a standard C operator to find how many bytes of space something takes in memory.

```
Message reads: Cookies make you fat
msg occupies 8 bytes
```

Really? Just 8 bytes?

Bit, byte, and word

A binary bit

2^1 = 2 different numbers

0, 1

A byte = 8 bits

1	0	1	0	0	0	1	0

2^8 = 256 different numbers

0, 1, 2, …, 255

char type in C takes 1 byte

Hexadecimal: A2

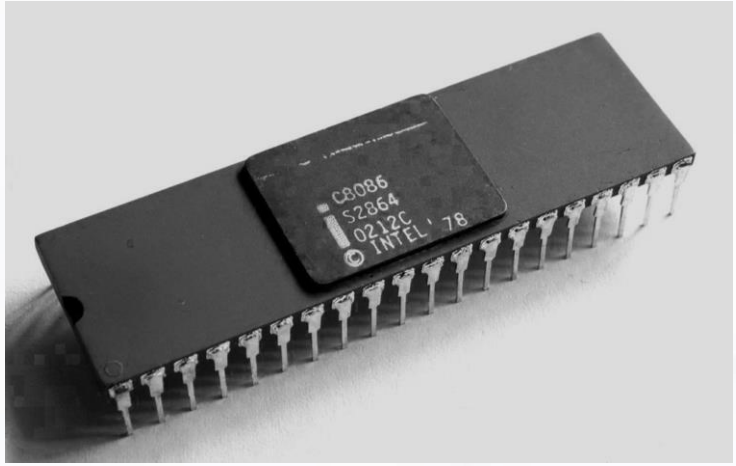
A word = 16, 32, 64… bits

0	1	1	0	1	0	0	0	1	0	1	0	0	0	1	0

A word in a 16-bit system

2^{16} = 65536 numbers

0, 1, 2, …, 65535



Bit, byte and word

A word in a 64-bit system = 8 bytes

2^{64} = 18,446,744,073,709,551,615 different numbers

0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 0 0 0	0 1 0 0	0 1 1 0
0	0	0	0	0	0	4	6
0 1 1 0	1 0 0 0	1 0 1 0	0 0 1 0	0 1 0 1	1 0 1 1	0 0 1 0	0 0 1 1
6	8	A	2	5	B	2	3

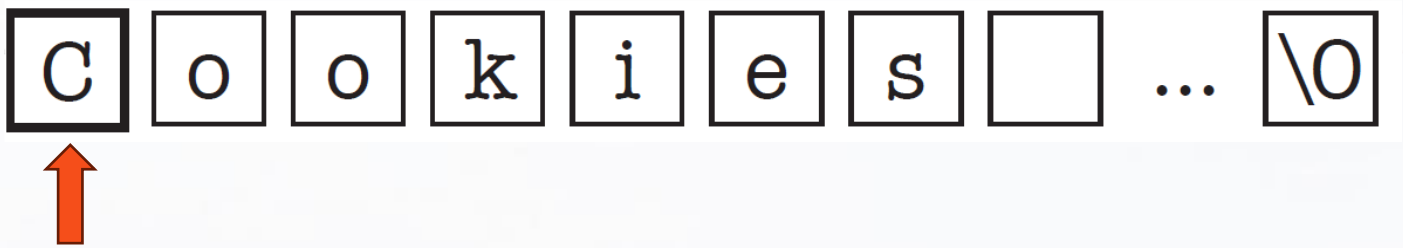
0x0000004668A25B23

64 bits = 8 bytes

long long int in C takes 8 byte

Array variables are like pointers

```
char quote[] = "Cookies make you fat";
```



quote is *like* a pointer, pointing to the first character (element) of the string (array).

```
printf("The quote string is stored at: %p\n", quote);
```

```
The quote string is stored at: 000000000061FE00
```

8 bytes

```
#include <stdio.h>

void fortune_cooki
{
    printf("Messag
    printf("msg occupies %i bytes\n", sizeof(msg));
}
```

'sizeof' on array function parameter 'msg' will return size of 'char *' [-Wsizeof-array-argument] gcc

(unsigned long long)8ULL

View Problem (Alt+F8) No quick fixes available

sizeof() is getting the size of a pointer (8 bytes I a 64 bit system)

Array vs. Pointer

```
char s[] = "How big is it?";
char *t = s;
```

sizeof() operator	Size of the entire array sizeof(s) == 15	Size of the memory address sizeof(t) == 8
& operator	Address of the array &s == s	Address of the pointer (pointer of a pointer) &t != t
Re-assign values	An array variable cannot be reassigned elsewhere s = t; ✗	Can point to elsewhere t = another_str; ✓

An array is a static pointer (constant), and the compiler knows its length

Pointer decay: the length information is lost when an array is assigned to a pointer (or passed to a function)

Why the index of an array starts at 0?

Array variable can be used as a pointer to the first element

```
int drinks[] = {4, 2, 3};
printf("1st order: %i drinks\n", drinks[0]);
printf("1st order: %i drinks\n", *drinks);
```

Equivalent

Can we address other elements in the same way?

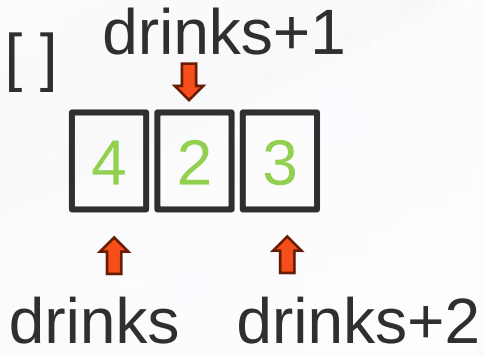
```
printf("3rd order: %i drinks\n", drinks[2]);
printf("3rd order: %i drinks\n", *(drinks+2));
```

Equivalent

You may offset the address by adding an integer value, or by [ ]

*(drinks + i) ⟷ drinks[i]

*drinks ⟷ drinks[0]



Memory offset is in terms of number of elements

Quiz

What should be passed to puts(), to print the later part of the string, starting from ‘c’?

```
void skip(char *msg)
{
    puts(    msg+6    );
}

char *msg_from_amy = "Don't call me";
skip(msg_from_amy);
```

Using pointers for data entry

```
char name[40];
printf("Enter your name: ");
scanf("%39s", name);
```

Awaiting a string of up to 39 characters.

To be stored in the memory starting from name

```
int age;
printf("Enter your age: ");
scanf("%i", &age);
```

Read an integer and store it to the memory addressed by pointer &age

```
char first_name[20];
char last_name[20];
printf("Enter first and last name: ");
scanf("%19s %19s", first_name, last_name);
```

You may collect more than one piece of information at a time.

Be careful with scanf()

```
char food[5];
printf("Enter favorite food: ");
scanf("%s", food);
printf("Favorite food: %s\n", food);
```

```
Thread 1 received signal SIGSEGV, Segmentation fault.
0x0000000000401599 in main () at C:\Teaching\CS111\careful_scanf.c:99 }
```

A safer alternative – **fgets()**

```
char food[5];
printf("Enter favorite food: ");
fgets(food, sizeof(food), stdin);
```

	scanf	vs fgets
limits	optional size descriptor in the format string	mandatory limit as a function argument
Multiple fields	Allow multiple fields of different types	One string at a time
Spaces	stops entering a field when a space is met	can read a string containing spaces

Three-card monte

```
#include <stdio.h>

int main()
{
    char *cards = "JQK";
    char a_card = cards[2];
    cards[2] = cards[1];
    cards[1] = cards[0];
    cards[0] = cards[2];
    cards[2] = cards[1];
    cards[1] = a_card;
    puts(cards);
    return 0;
}
```

String literals can't be updated!