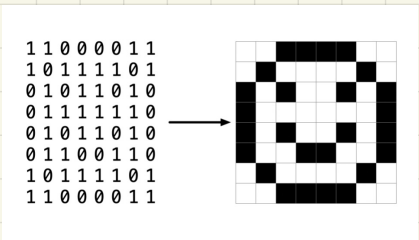
CS50-IV

Memory

If we keep zooming into an image, we will see pixels.

We can imagine an image as a map of bits, where zeros represent black and ones represent white.



RGB, or Red, Green, Blue, are numbers that represent the amount of each of this colors.

Hexadecimal

Hexadecimal is a system of counting that has 16 counting values.

0 1 2 3 4 5 6 7 8 9 a b c d e f

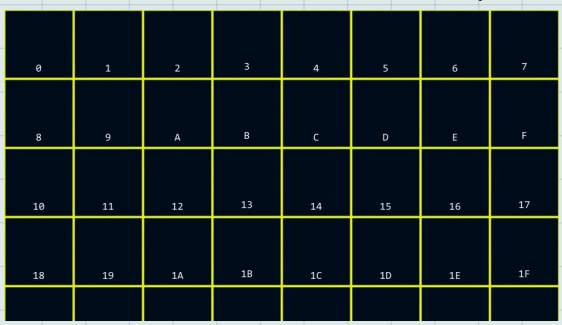
Hexadecimal is also known as base-16.
When counting in hexadecimal, each column is a power of 16.

The number 255 is represented has FF, because 16 x 15 (or F) is 240. Add 15 more to make 255.
This is the highest number you can count using a two-digit hexadecimal system

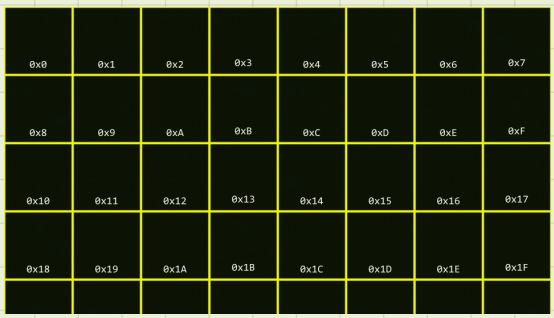
Hexadecimal is useful because it can be represented using fewer digits. Hexadecimal allows us to represent information more succintly.

Addresses

Applying hexadecimal numbering to each blocks of memory, it's possible to visualize these as follows:



We can imagine how there may be confusion regarding wether the 10 block may represent a location in memory or the value 10. Accordingly by convention, all hexadecimal numbers are often represented with the 0x prefix as follows:



The C language has two powerful operators that relate to memory:

```
& Provides the address of something stored in memory.
* Instructs the compiler to go to a location in memory.
```

For `int n = 50;` `n` returns `50`. And `&n` returns it's address: `0x123`

```
int n = 50;
printf("%p\n", &n);
```

→ `50` (`0x123`)

%p

%p allows us to view the address of a location in memory.

Pointers

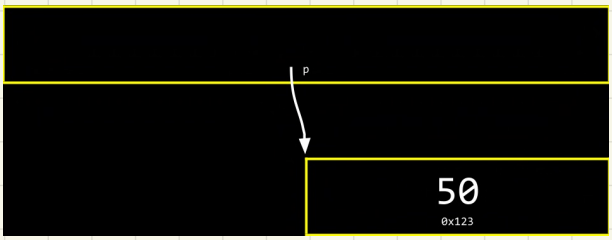
A pointer is a variable that contains the address of some value.
Most succinctly, a pointer is an address in your computer's memory.

```
int n = 50;

int *p = &n;
```

p is a pointer that contains a number that is the address of an integer n.

A pointer is usually stored as an 8-byte value.



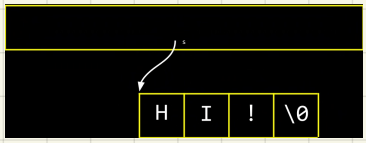
We can more accurately visualize a pointer as one address that points to another:

The * operator returns the value stored in the block of memory pointed by the pointer.

Strings

A string is simply an array of characters.

for a string s = "HI!", s is a pointer pointing to the first byte in memory.



Pointer Arithmetic

```
char *s = "HI!";
printf("%c\n", s[0]);
printf("%c\n", s[1]);
printf("%c\n", s[2]);
```

⟺

```
char *s = "HI!";
printf("%c\n", *s);
printf("%c\n", *(s + 1));
printf("%c\n", *(s + 2));
```

If you attempted to access something at location s+50, the program will likely quit as a safety precaution.

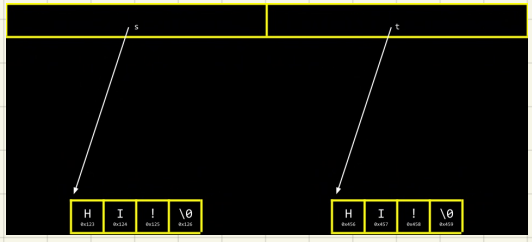
Comparing Strings

A string of characters is simply an array of characters identified by its first byte.

s == t : false

Using the == operator in an attempt to compare strings will attempt to compare the memory locations of the strings instead of the characters therein.

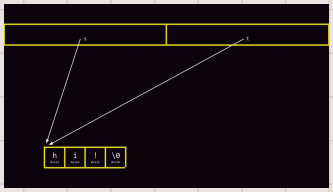
The following pointers point to two separate locations in memory.



Copying

A common need in programming is to copy one string to another.

String s = String t



that s and t are pointing at the same block of memory.
This is not an authentic copy of a string.
Instead, there are two pointers pointing at the same string.

To be able to make an authentic copy of a string,
we need two new building blocks;

Malloc; allows to allocate a block of specific size of memory.

Free; allows to tell the compiler to free up that block of memory you previously allocated.

```
// Get a string
char *s = get_string("s: ");

// Allocate memory for another string
char *t = malloc(strlen(s) + 1);

// Copy string into memory, including '\0'
for (int i = 0, n = strlen(s); i <= n; i++)
{
    t[i] = s[i];
}
```

malloc(strlen(s)+1) creates a block of memory that is the length of the string s plus one.
This allows for the inclusion of the null \0 character.

The C language has a built-in function to copy strings named strcpy(char *str1, char *str2)

malloc returns NULL, a special value in memory, in the event that something goes wrong.

Valgrind

Valgrind is a tool that can check to see if there are memory-related issues with a program wherein malloc is used.
Specifically, it checks to see if all the memory allocated has been freed.

make program_name, followed by valgrind ./program_name, will report where memory has been lost as a result of the program.
will return a report from valgrind that

Garbage Values

When asking the compiler for a block of memory, there is no guarantee that this memory will be empty.

It's very possible that this memory was previously utilized by the computer.
Accordingly, you may see junk or garbage values.
This is a result of getting a block of memory but not initializing it.

Swap

A common need in programming is to swap two values.
Naturally, it's hard to swap two variables without a temporary holding space.

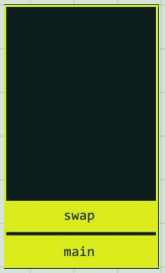
```
void swap(int a, int b)
{
    int tmp = a;
    a = b;
    b = tmp;
}
```

While this code runs, it does not work.
The values do not swap.

When values are passe to functions, only copies are actually provided. (Scope)

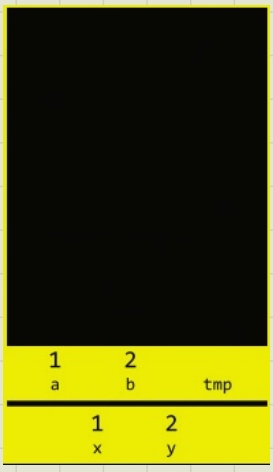
Global variables live in one place in memory.
Various functions are stored in the stack.

The main and swap have two seperate frames.
Therefore, we cannot simply pass the values from one function to another to change them.




```
void swap(int *a, int *b)
{
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```

The variables are not passed by value, but by reference.
The addresses of a and b are provided.



Overflow

A heap overflow is when you overflow the heap, touching areas of the memory your are not supposed to.
A stack overflow is when too many functions are called, overflowing the amount of memory available.
Both of these are considered buffer overflows.

scanf

scanf is a built-in function that can get user input.

```
int x;
printf("x: ");
scanf("%i", &x);
```

The value of x is stored at the location of x.

To read a string we need to pre-allocate the amount of memory required.
An input larger would create an error.

```
char s[4];
printf("s: ");
scanf("%s", s);
```

Files

You can read from and manipulate files.

```
// Open CSV file
FILE *file = fopen("phonebook.csv", "a");

// Get name and number
char *name = get_string("Name: ");
char *number = get_string("Number: ");

// Print to file
fprintf(file, "%s,%s\n", name, number);

// Close file
fclose(file);
```