How are variables stored in memory?

It is important to visualize the computer memory before we get started with pointers.

Because pointers are used to refer the memory location rather than the value stored in the memory.

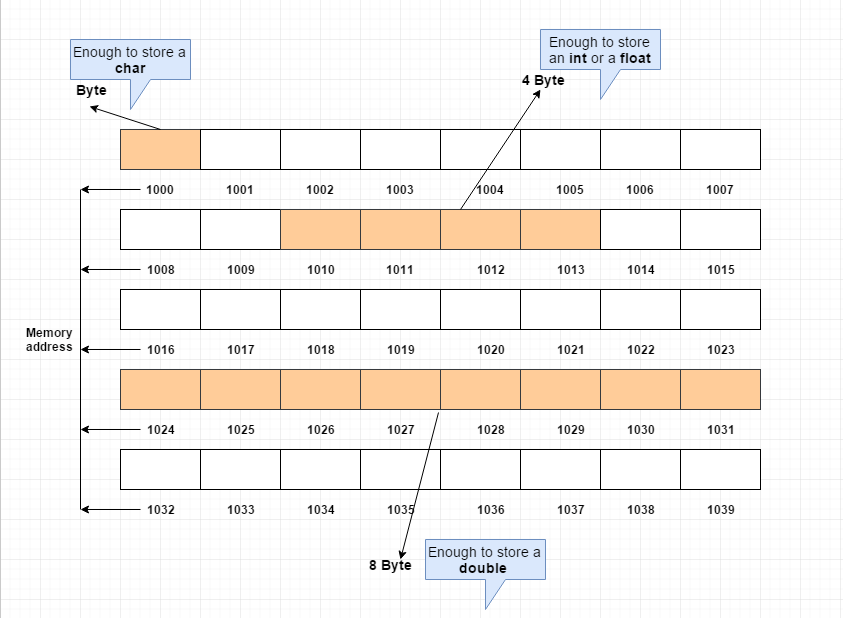
In other words, variables are used to manipulate the values stored in memory whereas pointers are used to manipulate the variable addresses(memory location).

Let's visualize the computer memory first.

In computer memory, the basic memory unit is a byte (8 bits).

Each byte will have unique memory address. For the sake of understanding, we will represent memory address using decimal value.

**Pictorial Explanation**



In the above diagram, each block is a byte which consists 8 bits.

And the memory address are linear and each byte will have a unique address.

To store a character, the computer will allocate 1 byte of memory which is 1 \* 8 = 8 bit.

To store an integer or a float, the computer will allocate 4 byte of memory which is 4 \* 8 = 32 bit.

To store a double value, the computer will allocate 8 byte of memory which is 8 \* 8 = 64 bit.

# [How are variable names stored in memory in C?](https://stackoverflow.com/questions/14612314/how-are-variable-names-stored-in-memory-in-c)

in C, let's say you have a variable called variable\_name. Let's say it's located at 0xaaaaaaaa, and at that memory address, you have the integer 123. So in other words, variable\_name contains 123.

I'm looking for clarification around the phrasing "variable\_name is located at 0xaaaaaaaa". How does the compiler recognize that the string "variable\_name" is associated with that particular memory address? Is the string "variable\_name" stored somewhere in memory? Does the compiler just substitute variable\_name for 0xaaaaaaaa whenever it sees it, and if so, wouldn't it have to use memory in order to make that substitution?

* Variable names don't exist anymore after the compiler runs (barring special cases like exported globals in shared libraries or debug symbols). The entire act of compilation is intended to take those symbolic names and algorithms represented by your source code and turn them into native machine instructions. So yes, if you have a global variable\_name, and compiler and linker decide to put it at 0xaaaaaaaa, then wherever it is used in the code, it will just be accessed via that address.

So to answer your literal questions:

How does the compiler recognize that the string "variable\_name" is associated with that particular memory address?

The toolchain (compiler & linker) work together to assign a memory location for the variable. It's the compiler's job to keep track of all the references, and linker puts in the right addresses later.

Is the string "variable\_name" stored somewhere in memory?

Only while the *compiler* is running.

🡪Great explanation for how C does it. Note that some other languages (particularly modern "dynamic" or "scripting" languages) may maintain symbolic names for data, and they do indeed use memory at runtime to keep that mapping information around.