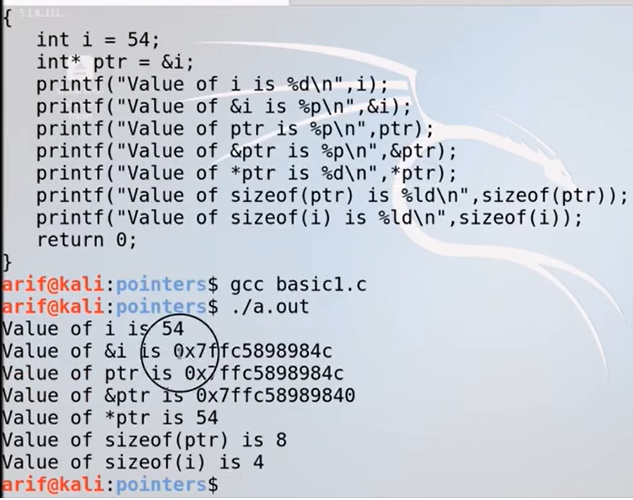
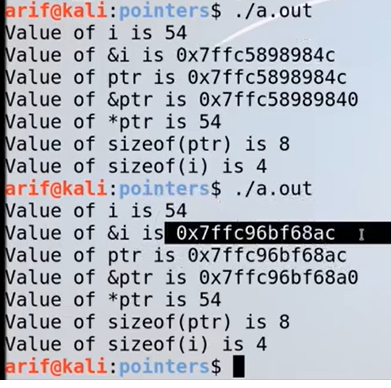
Why Pointers?

* Fast and more efficient code using pointers and arithmetic. They are more closer to hardware.
* They support dynamic memory allocation.
* We can protect the data that we pass to functions.



Every we run the program address are changed...



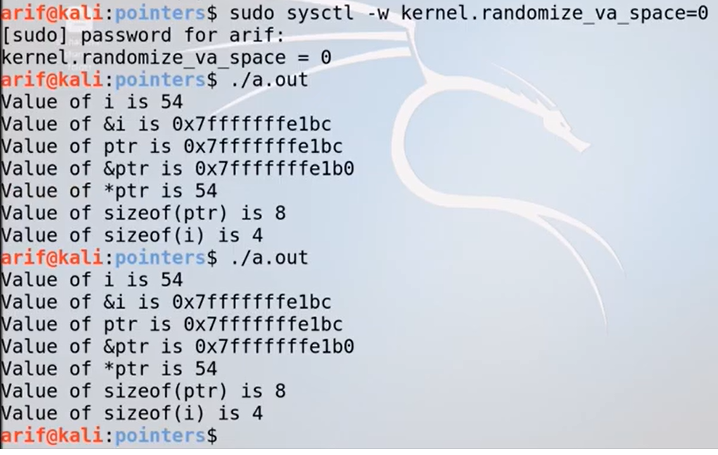
To fix it disable address space randomization.

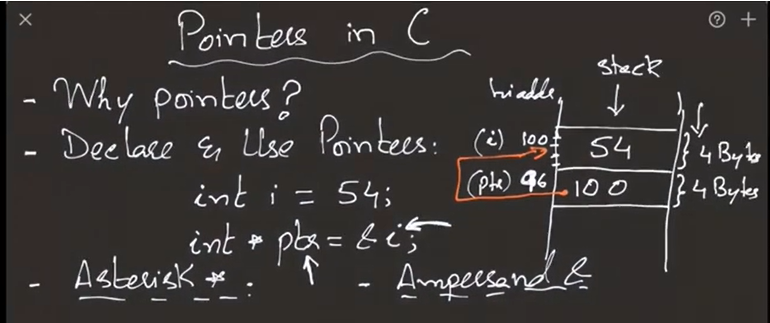
$ sudo sysctl -w kernel.randomize\_va\_space=0

For disabling

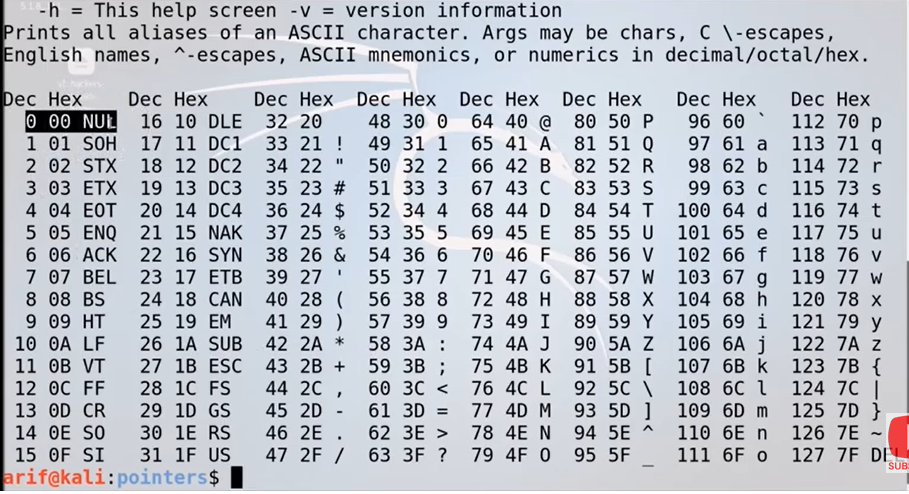
$ sudo sysctl -w kernel.randomize\_va\_space=1

For enabling





$ ascii



Int \* ptr;

It points to unknown location. Never deference an uninitialized pointer.

When ever we create an pointer try to initialize it in same line or initialize it with null.

Int \* prt = 10;

Int \* prt = NULL;

NULL is a macro defined in stdio.h. # define NULL((void\*)0).

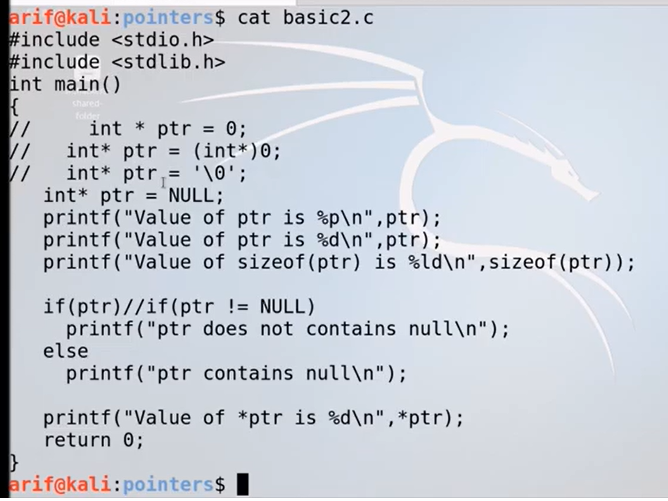
Int \* prt = NULL;

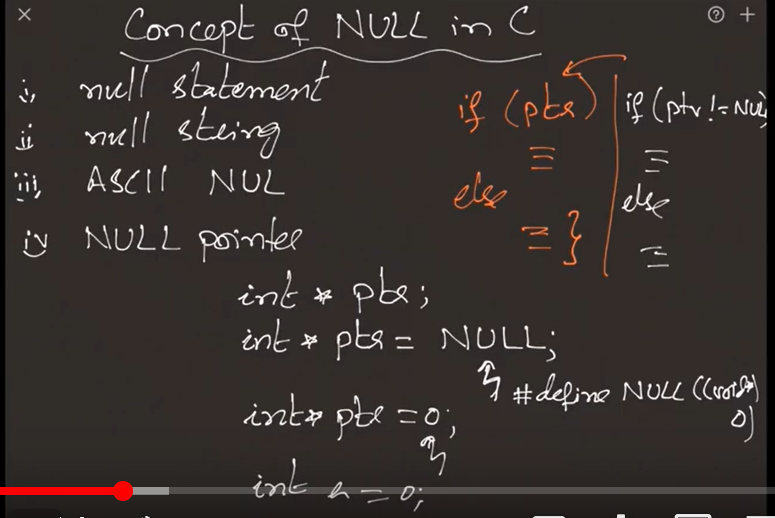
Int \* prt = 0;

Both NULL and 0 are same. Zero is an overloaded operand, it is automatically casted to null pointer.

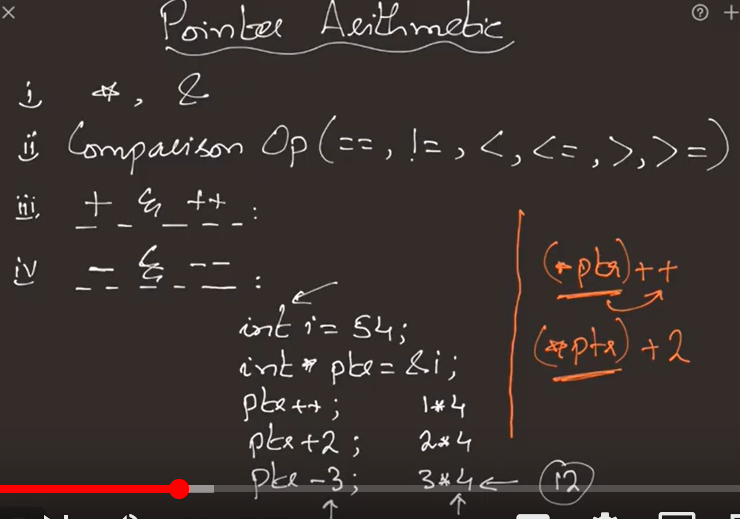
Int a = 0;

Not it’s a integer value.





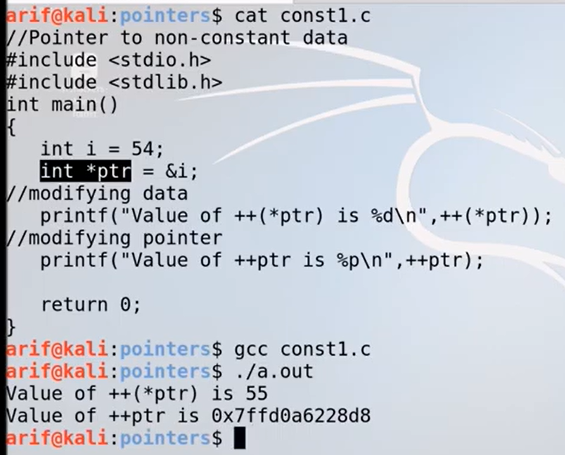
When ever we add or subtract an integer to an pointer the amount added or subtracted is product of the integer times the number of bytes of the underlying data type.

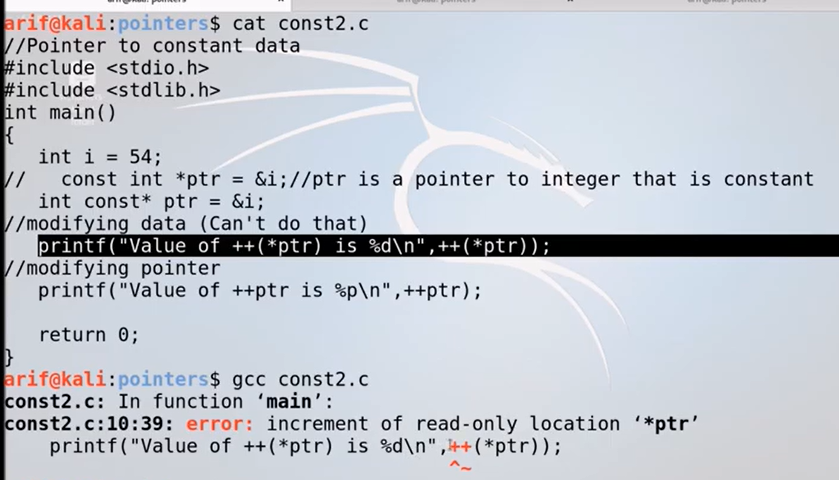


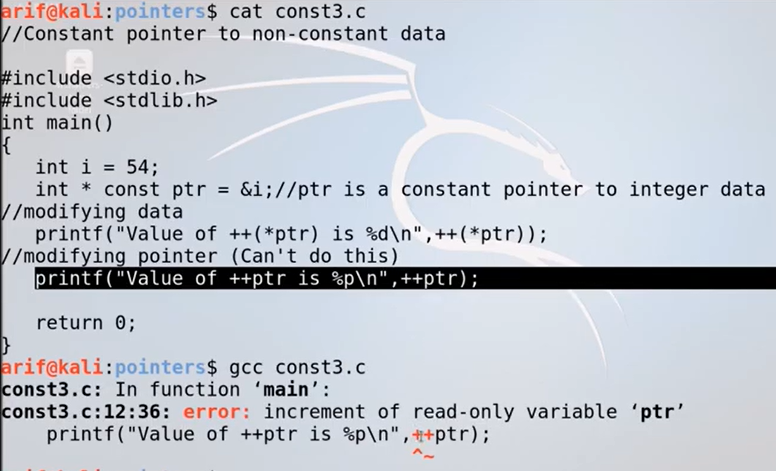
Int const a = 10;

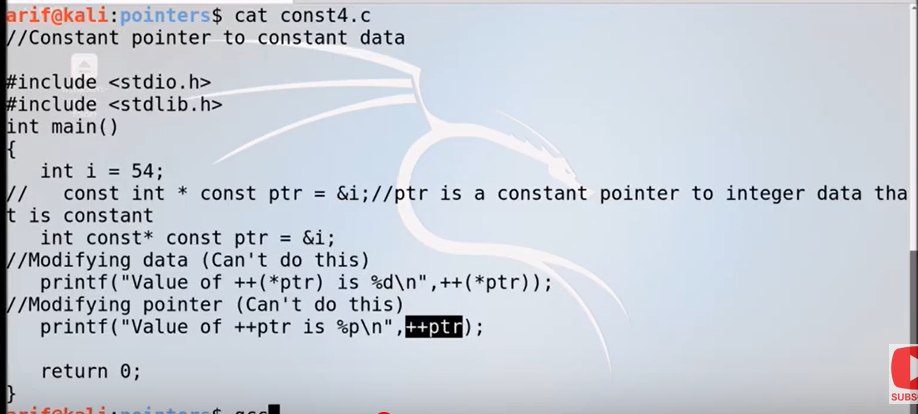
Const int a = 10;

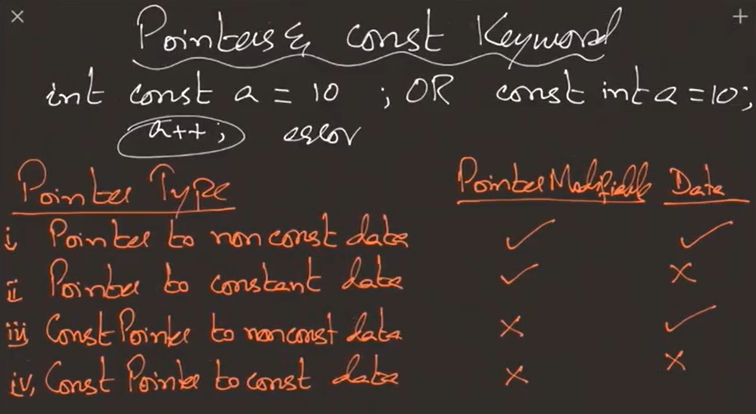
A variable will get stored in read only memory. If we do any update on it we get error.



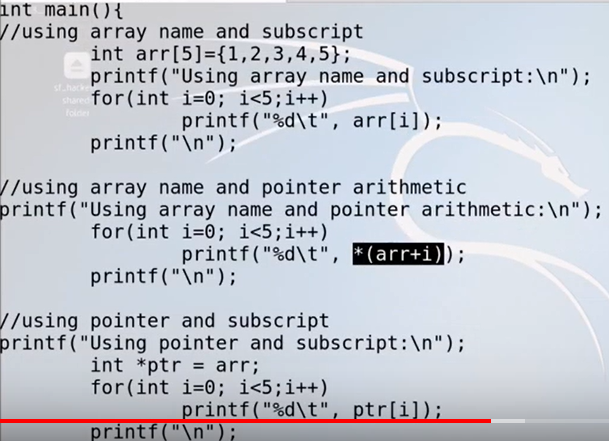


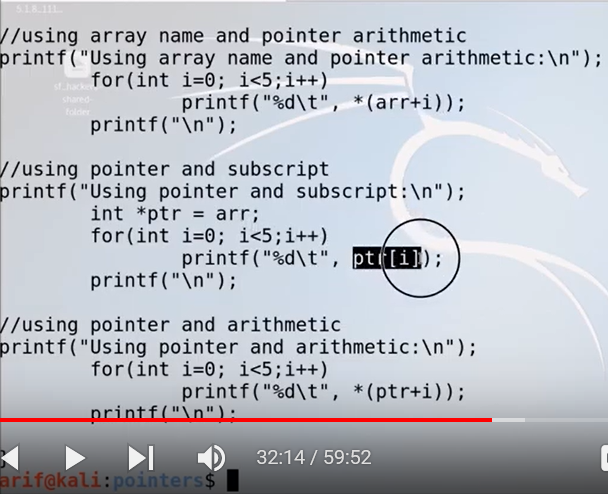


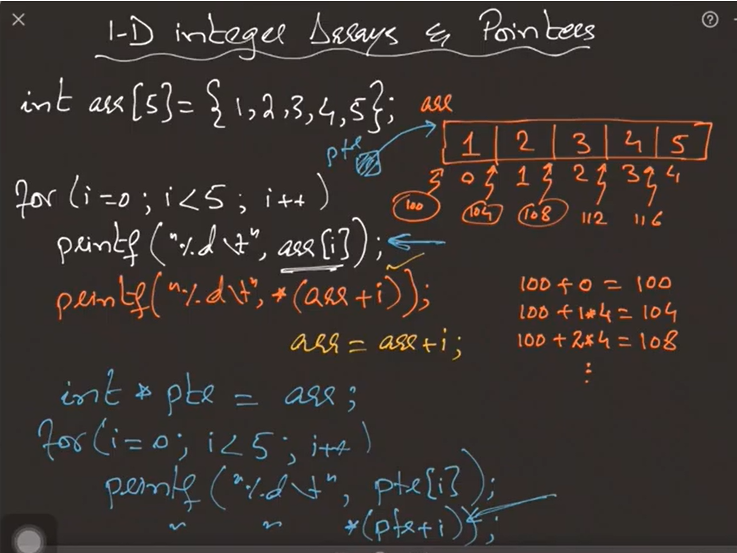


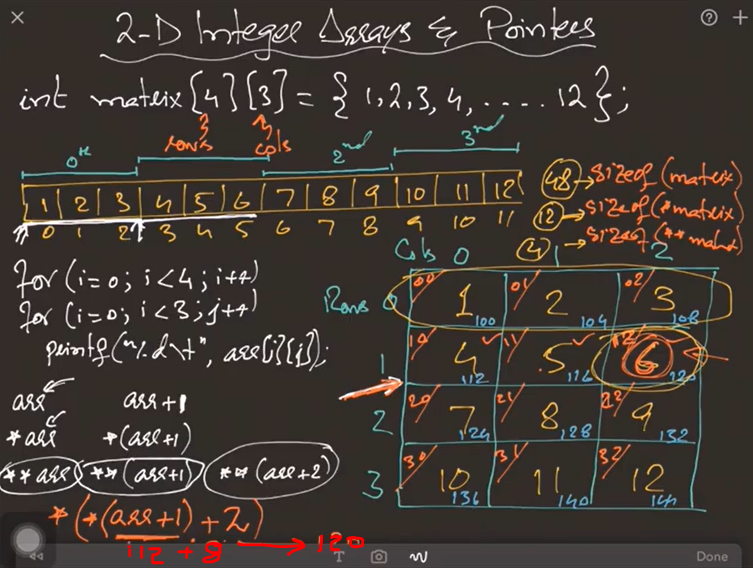


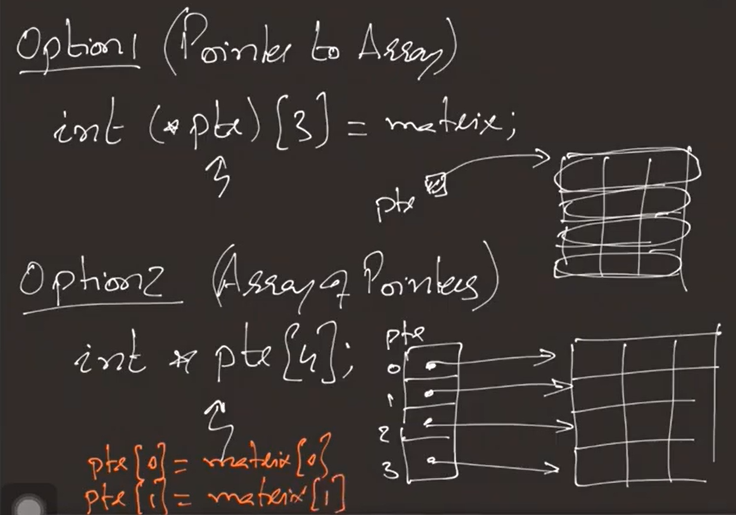
Arrays: is a finite collection of similar elements stored in adjacent memory location which can be accessed by a integer subscript.

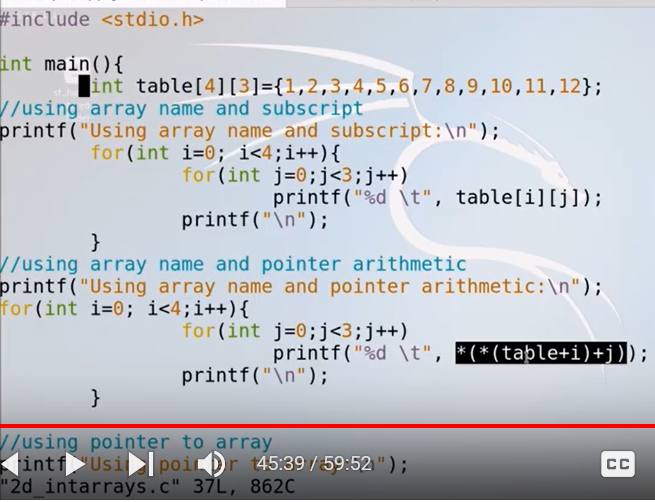


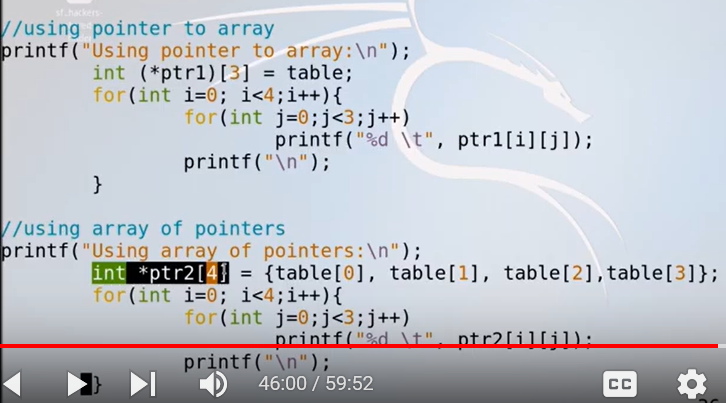












Note: String is a sequence of characters terminated having a ASCII NULL ‘\o’ character at end.

