Stack Vs Heap

1. $\boldsymbol{Like\ Stack, Heap\ is\ stored\ in\ RAM.}$

Suppose:

$$\boldsymbol{int * p = (int *) malloc\big(4 * sizeof(int)\big);}$$

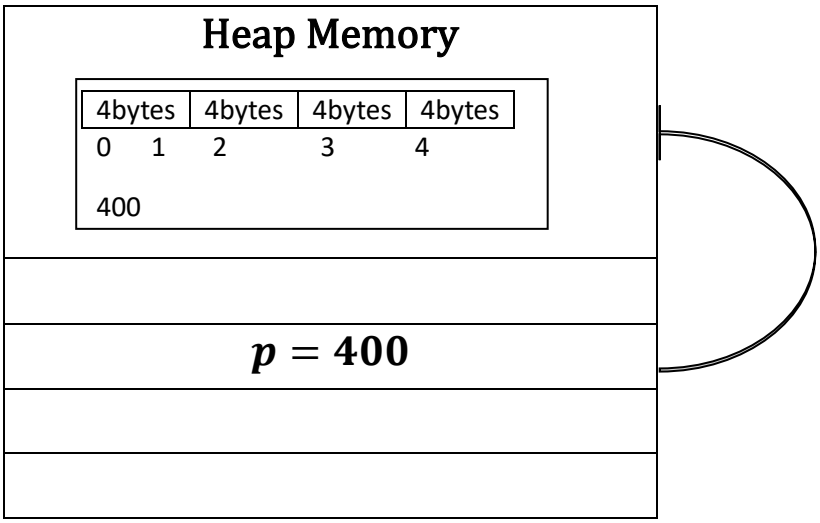
$\boldsymbol{It\ creates\ a\ space\ for\ an\ array\ of\ 4\ integers.}$

$\boldsymbol{But\ it\ get\ stored\ in\ Free\ Memory\ of\ RAM\ i.e.}$

$\boldsymbol{Heap\ Memory.}$

$\boldsymbol{Suppose\ `p`\ holds\ memory\ address: 400, it\ get\ stored\ in\ Stack.}$

$\boldsymbol{The\ memory\ address\ 400\ is\ the\ address\ of\ Heap\ Memory\ where\ array\ of\ 4\ inetegers\ is\ stored.}$

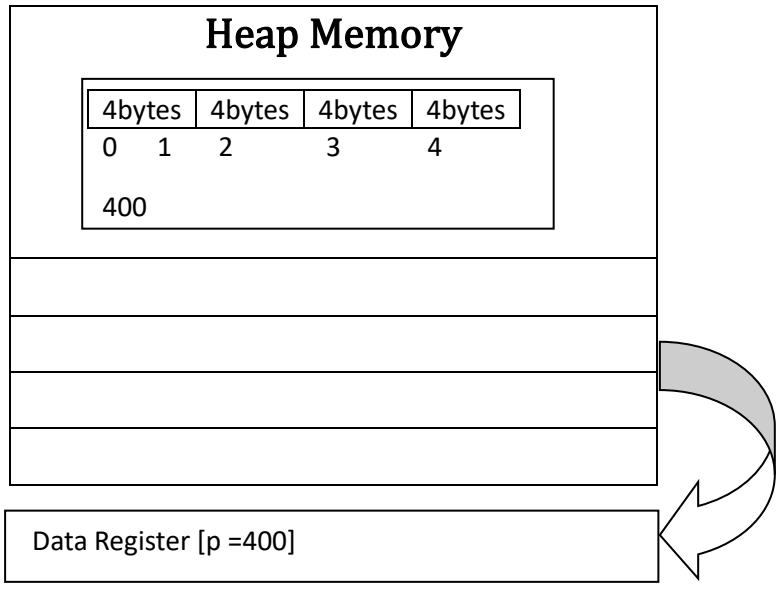


After operation in stack is finished:

- pointer p gets destroyed i.e., after Stack operation everything in stack gets popped out and stack gets empty.

- The data in heap memory remains at the location. Which we delete manually as discussed in arrays.

i.e.



i.e., free keyword for malloc and delete keyword for new operator.

What does Stack become Empty means ?:

Stack does NOT automatically become empty.

What actually happens:

- ***When the function returns,***
- ***Its stack frame is destroyed,***
- ***Only that frame is removed,***
- ***Not the entire program stack.***

Sometimes after deletion of data from stack we do is assign P to NULL after the operation gets over.

$$P = NULL\ .$$

This done to avoid dangling Pointers .

The pointer holds an address of a memory location, or a pointer that points to memory that is no longer valid are dangling pointers and hence they must be assigned to NULL.

***Note*: *Here popped out data showed it went to data register*.**

***Though , it can go to , General Purpose Registers such as, EAX , also in some architecture to Data Register*.**

***Also, depends upon instructions too such as POP EAX i.e. POP data will go to EAX*.**

***Here it is, just generalized*.**
