Dynamic Memory – memory is being used during runtime

Dynamic memory is accessed by pointers.

New – allocate space dynamically on the heap

New – returns the address of the allocated memory

Use pointer to reference the memory allocated on the heap.

Delete – to deallocate memory on the heap.

int \*p = new int // Allocates an int pointer on the stack

// Creates an int memory address on the heap

// Assigns pointer p points to int memory address on the heap

\*p = 5; // Assigns the value of 5 to the memory address associated with this pointer

// aka, it assigns 5 to the int that exists on the heap

delete p; // Deallocating the int located on the heap, frees up the memory address on heap

// doesn’t delete the pointer p.

// Pointer p will stay on the stack as long as this block of code that is being executed is

// on the stack. As soon as the function ends, the pointer will be deleted, as well as any

// other data/local variables that is located in that block.

// p is still pointing to the address that it was originally assigned to, but it is no longer a

// a valid location in memory. You wouldn’t want to make use of p (**dangling pointer**) by

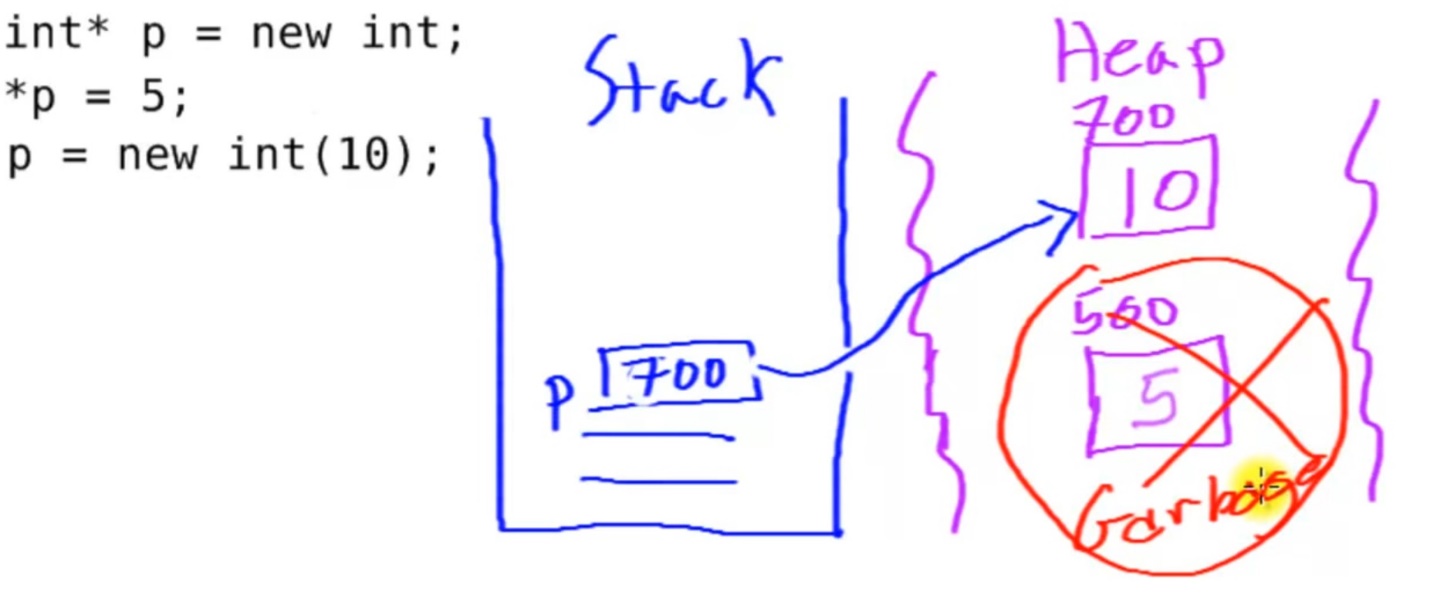
// assigning a value to p, or to try to de-reference it. It needs a new memory address!

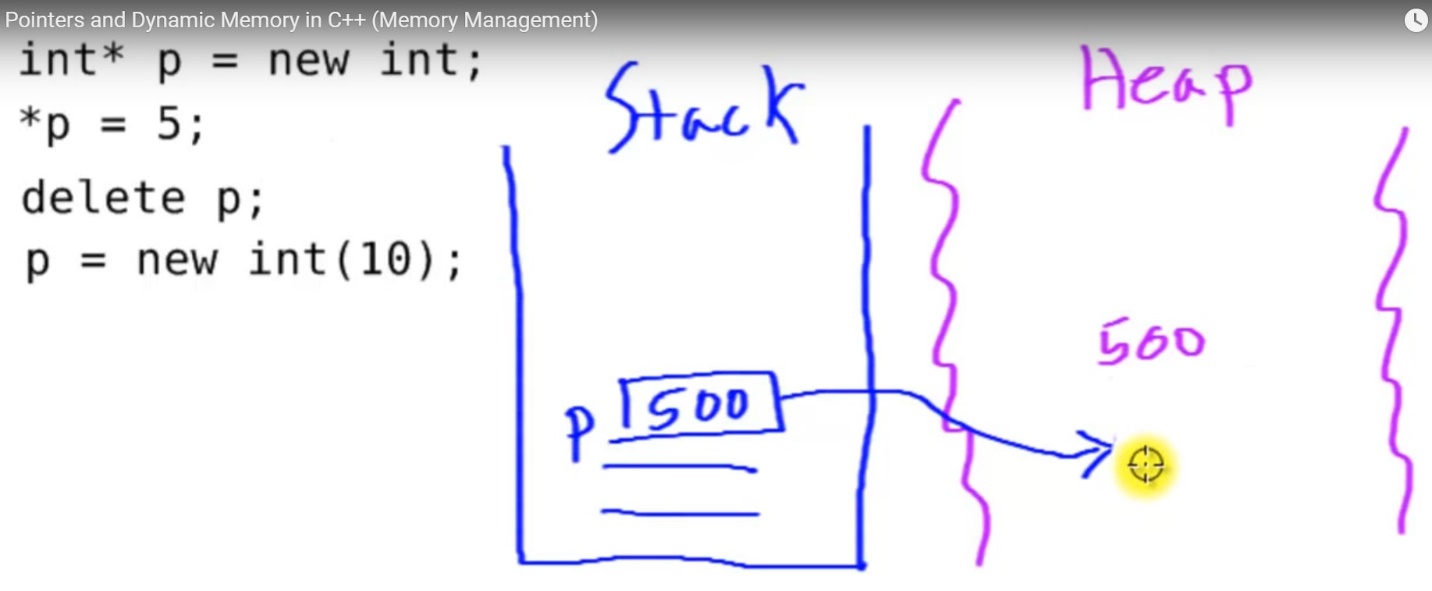
p = 0; // if you don’t have something you want to re-assign the pointer to straight away,

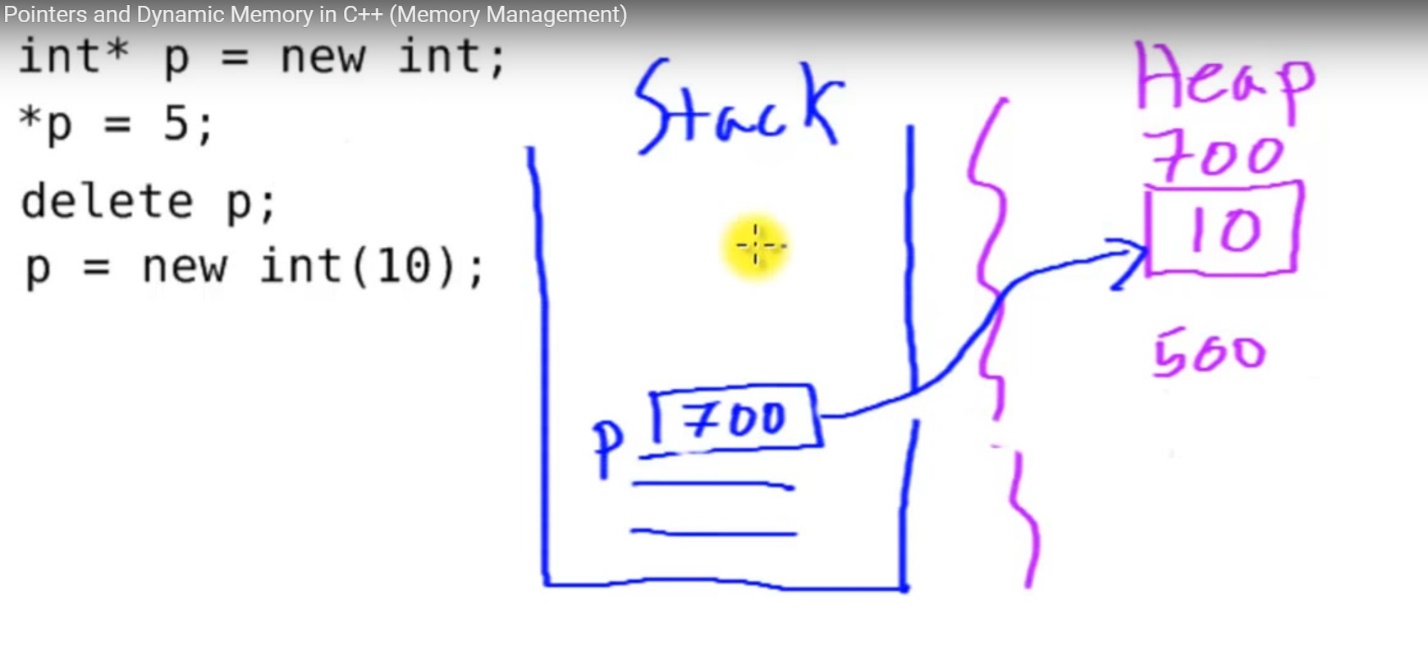
// you should always assign the pointer to null or 0. (prevents dangling pointer)

p = new int(10);

// reassigns the value of pointer p to a new address on the heap







Why do we want to dynamically allocate memory?

* To make arrays of variable length

