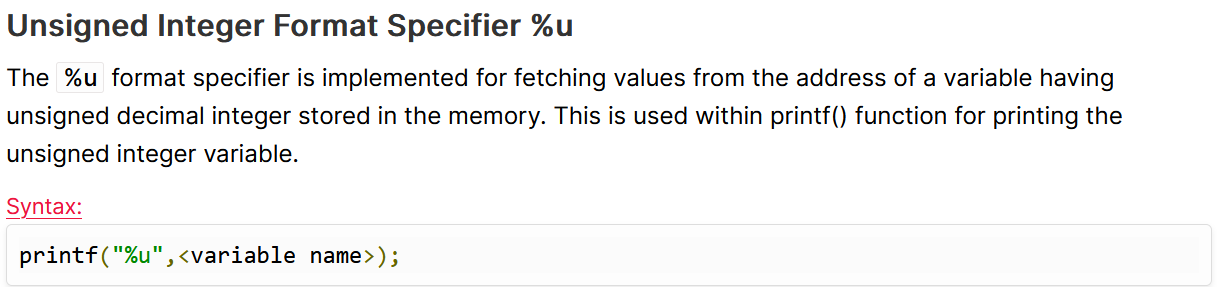
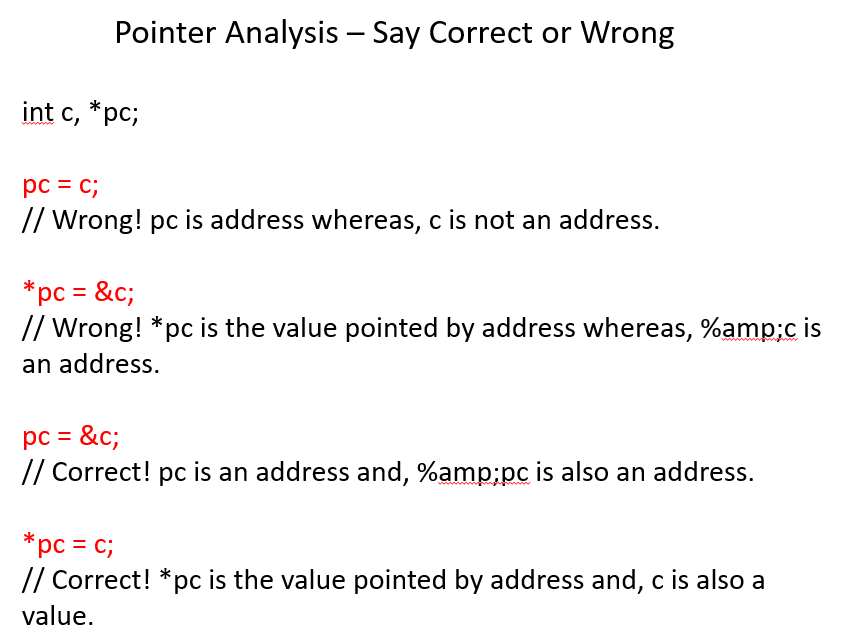
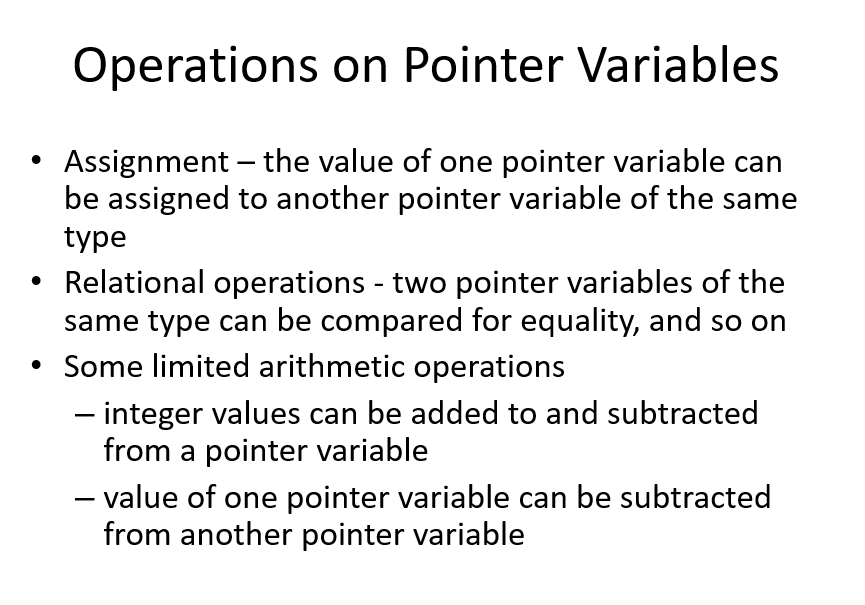
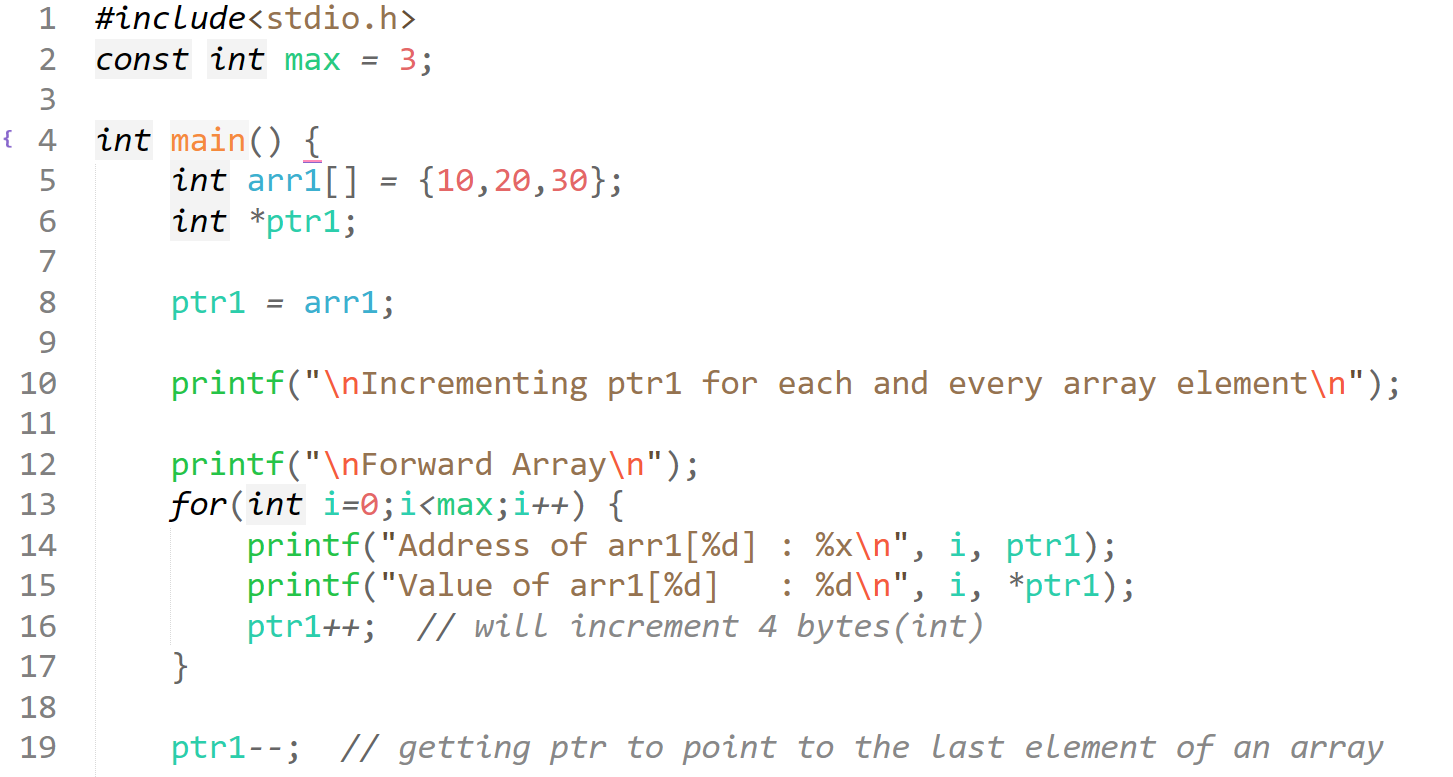
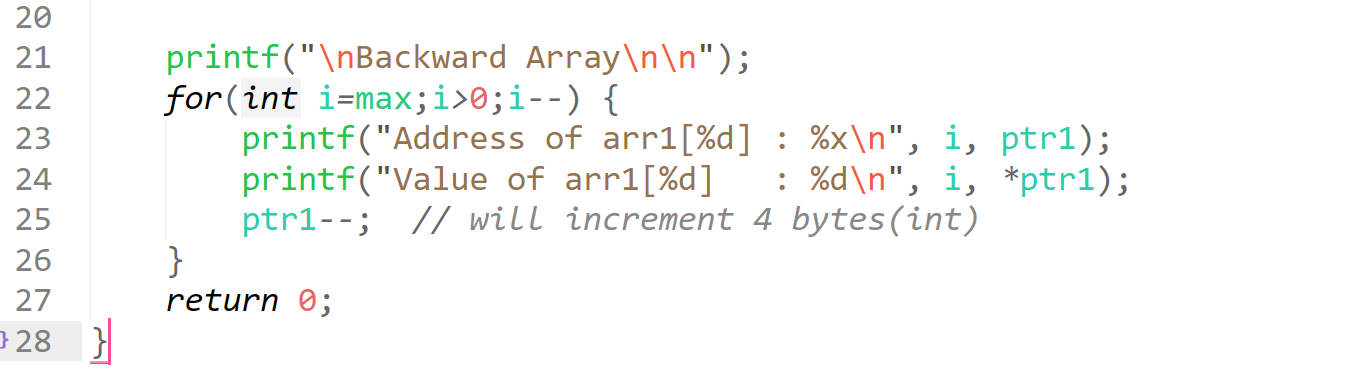
1. %u

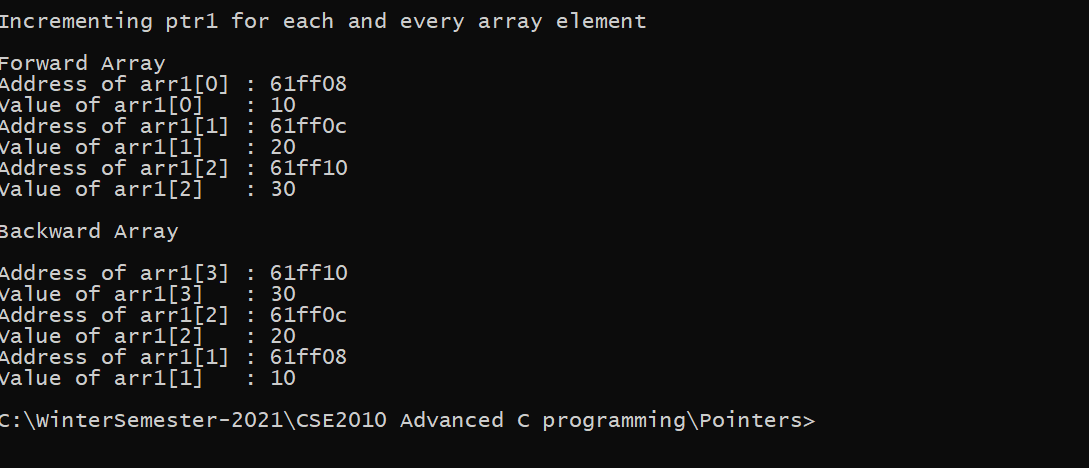




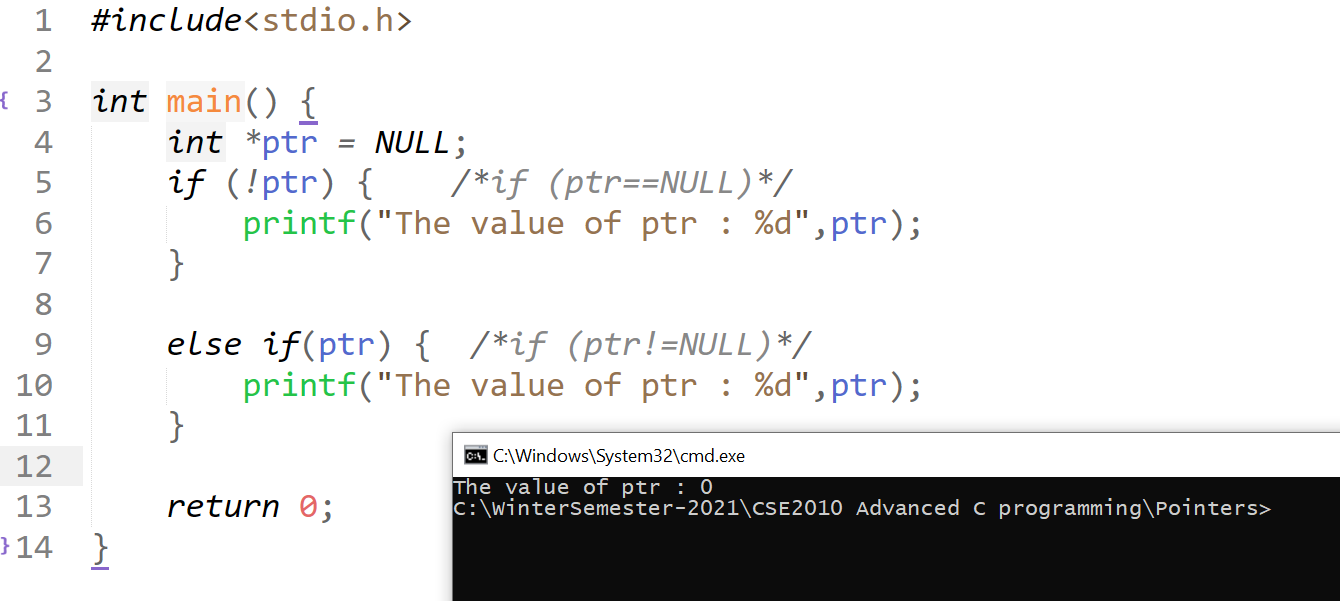


ptr 🡪 will increment and decrement 4 bytes (since int=4bytes)

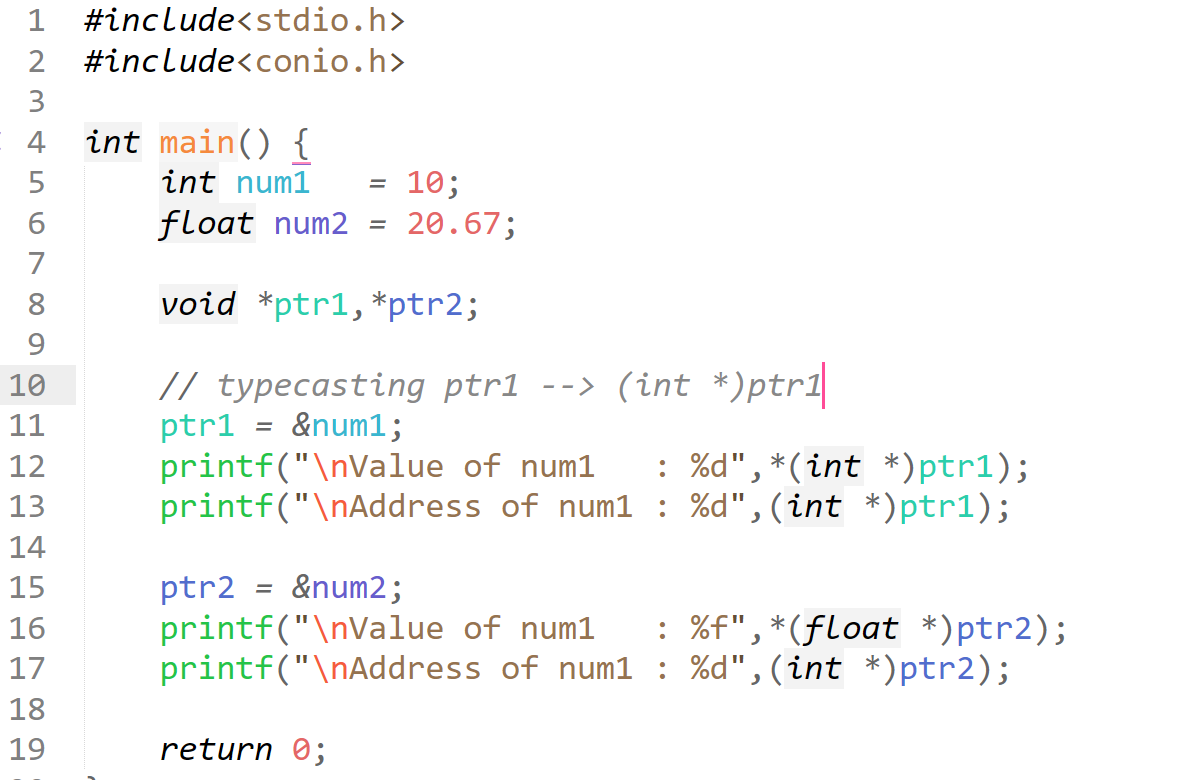
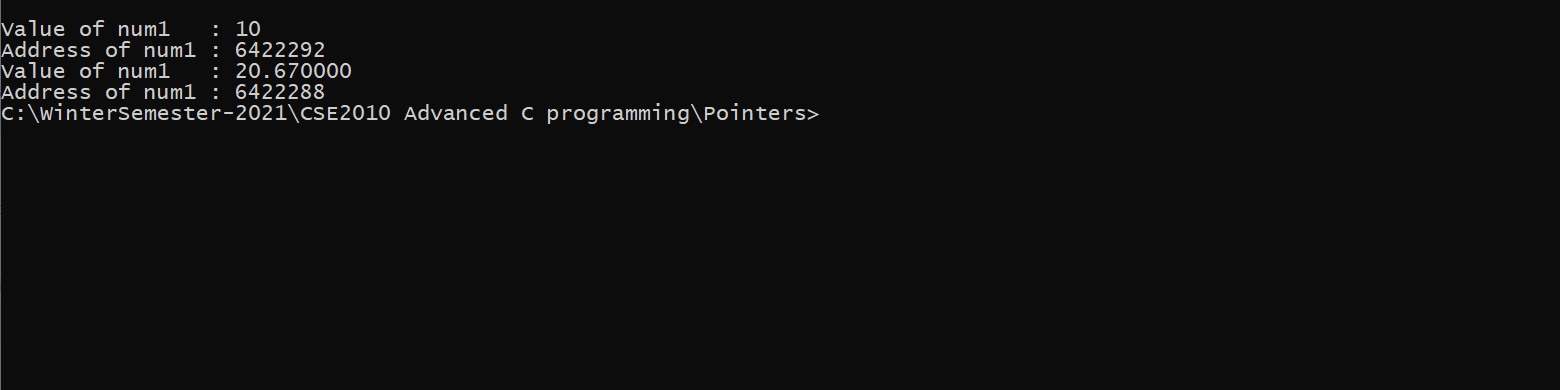


## **NULL pointer**

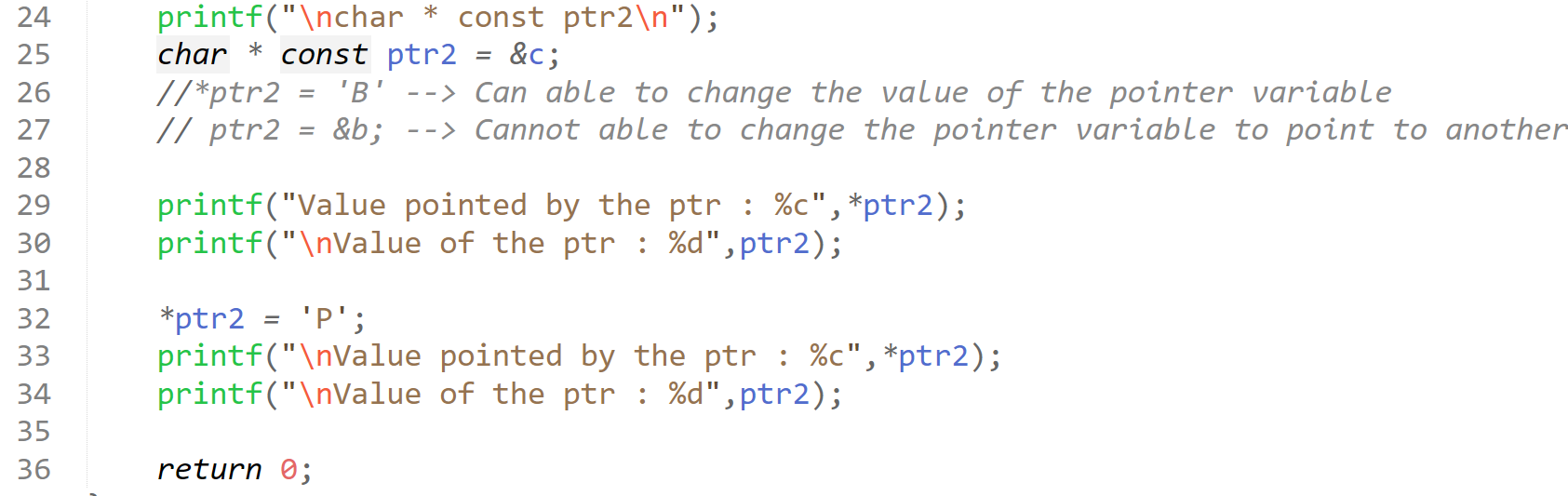


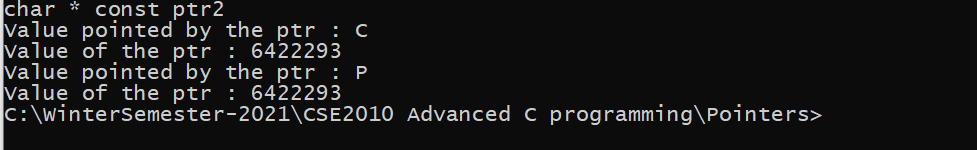
## **Void pointer / generic pointer**

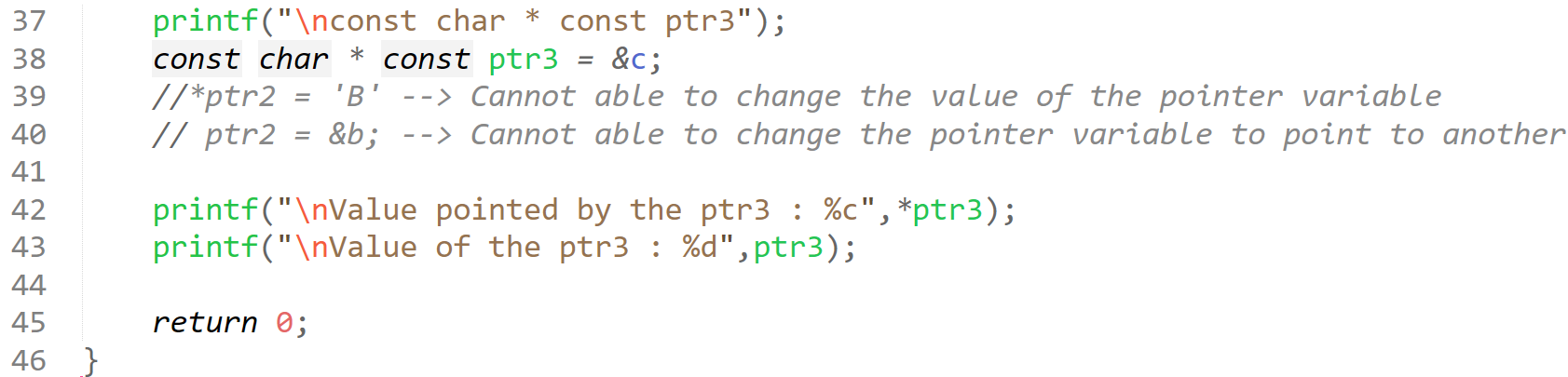
void pointers cannot be dereferenced

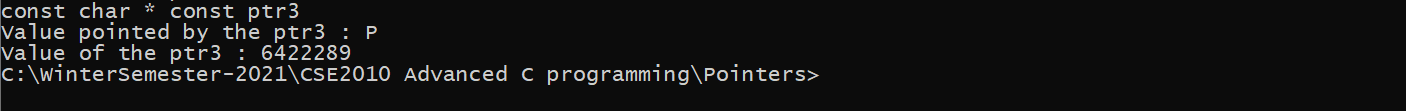
  


## **const pointer**



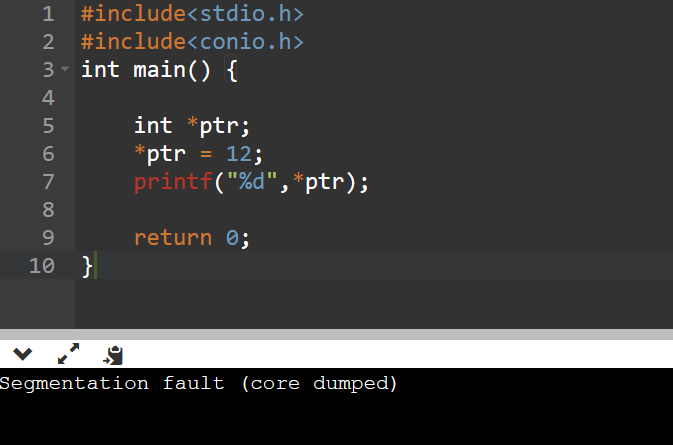




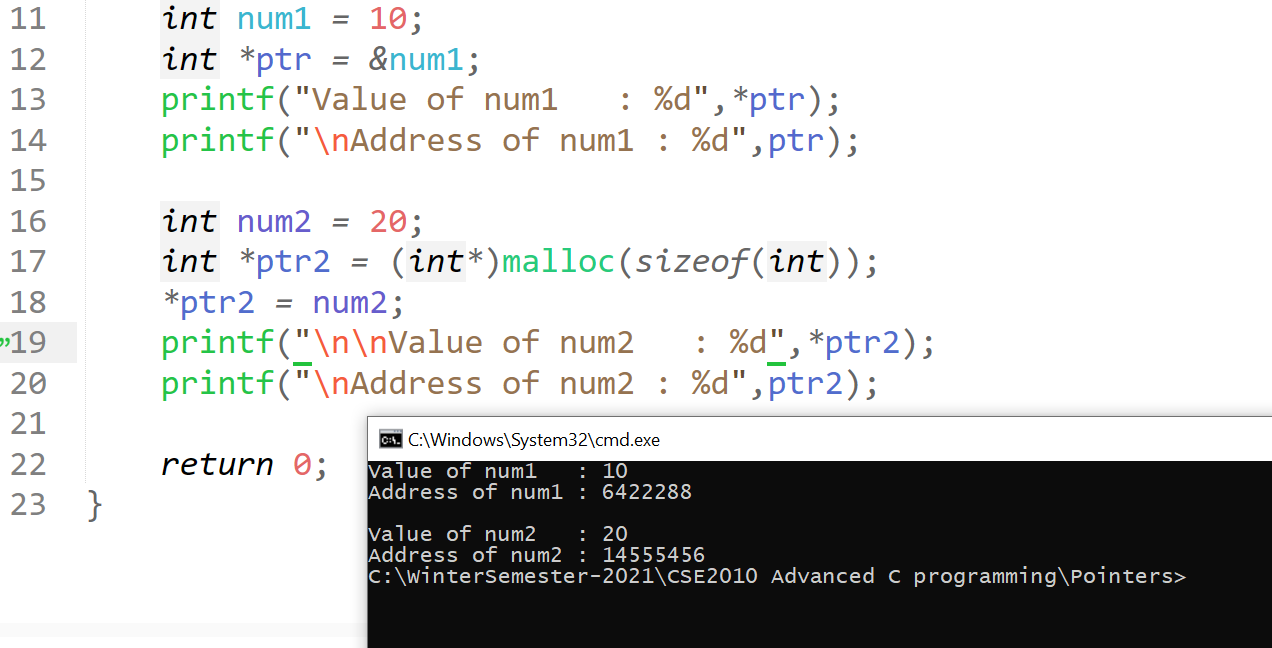


## **wild pointer**

Uninitialized pointers are called as wild pointers in C which points to arbitrary (random) memory location. This wild pointer may lead a program to behave wrongly or to crash.

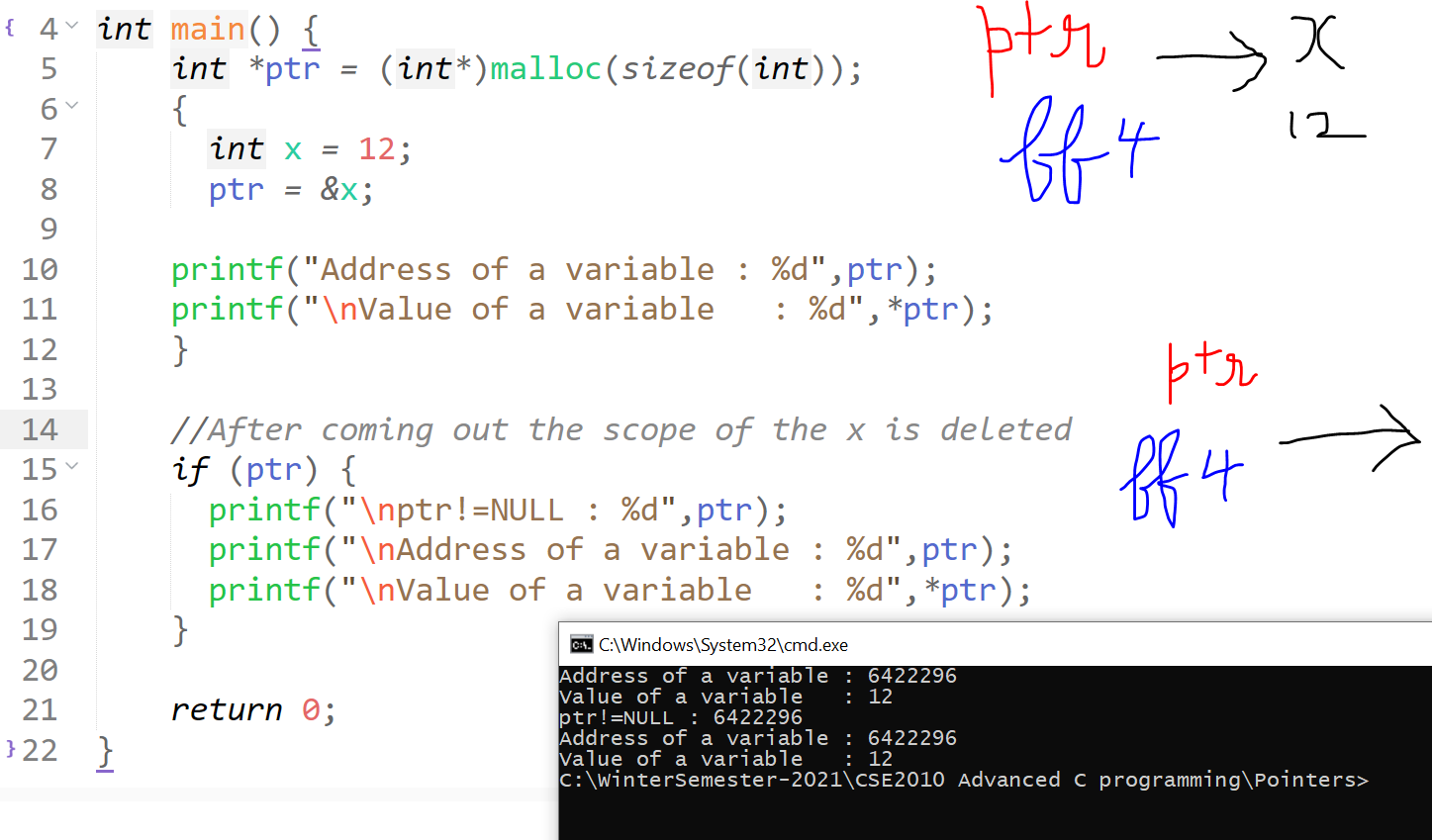


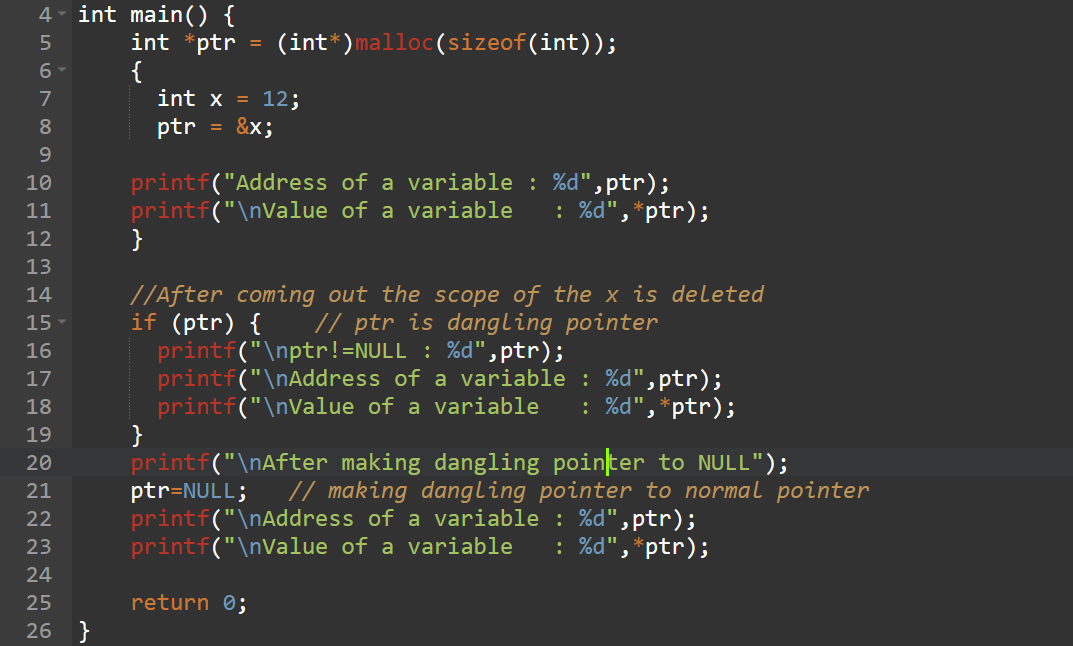
Since, it is not at all clear where the ptr is pointing we cannot able to change the ptr value. So only Segmentation fault(core dumped)

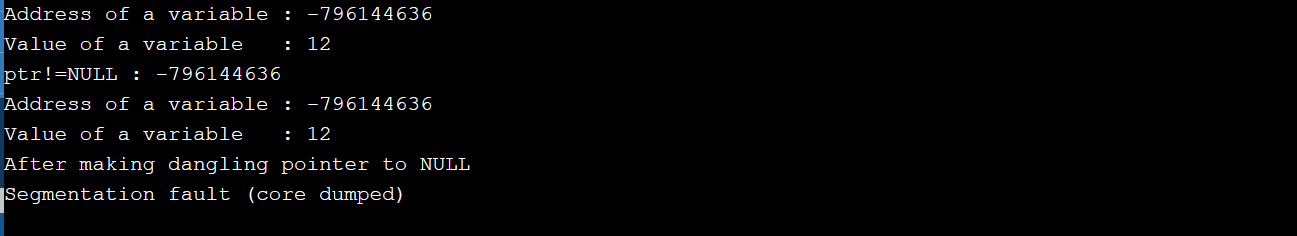
In-order to avoid Wild pointer,  


## **dangling pointer**

It is a pointer which points to the some non-existing memory location/ variable that has been deleted.  
It is a pointer which does not points to a valid memory location.

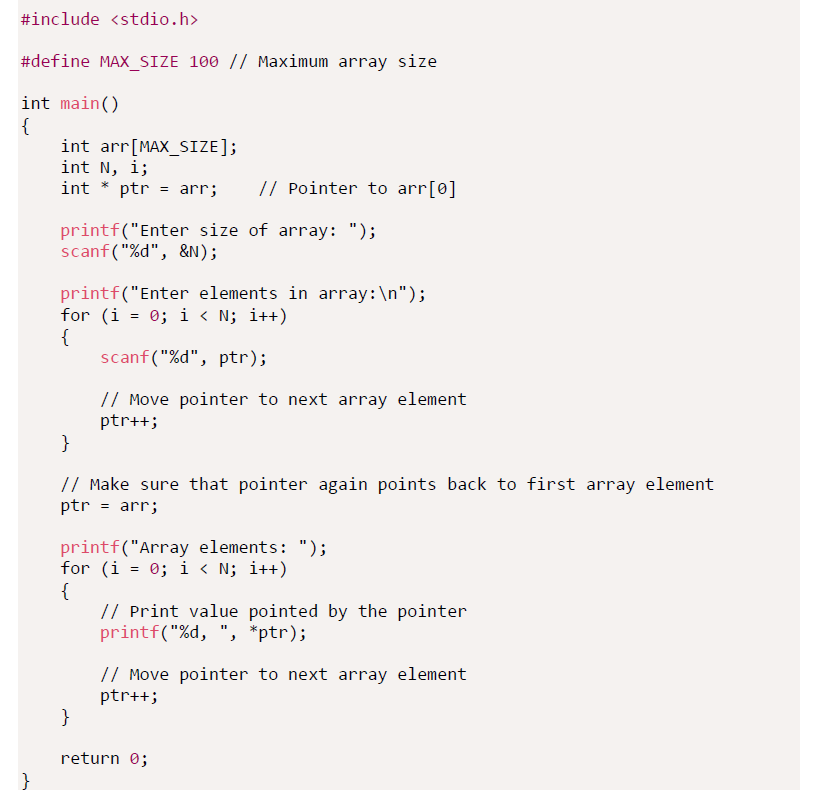




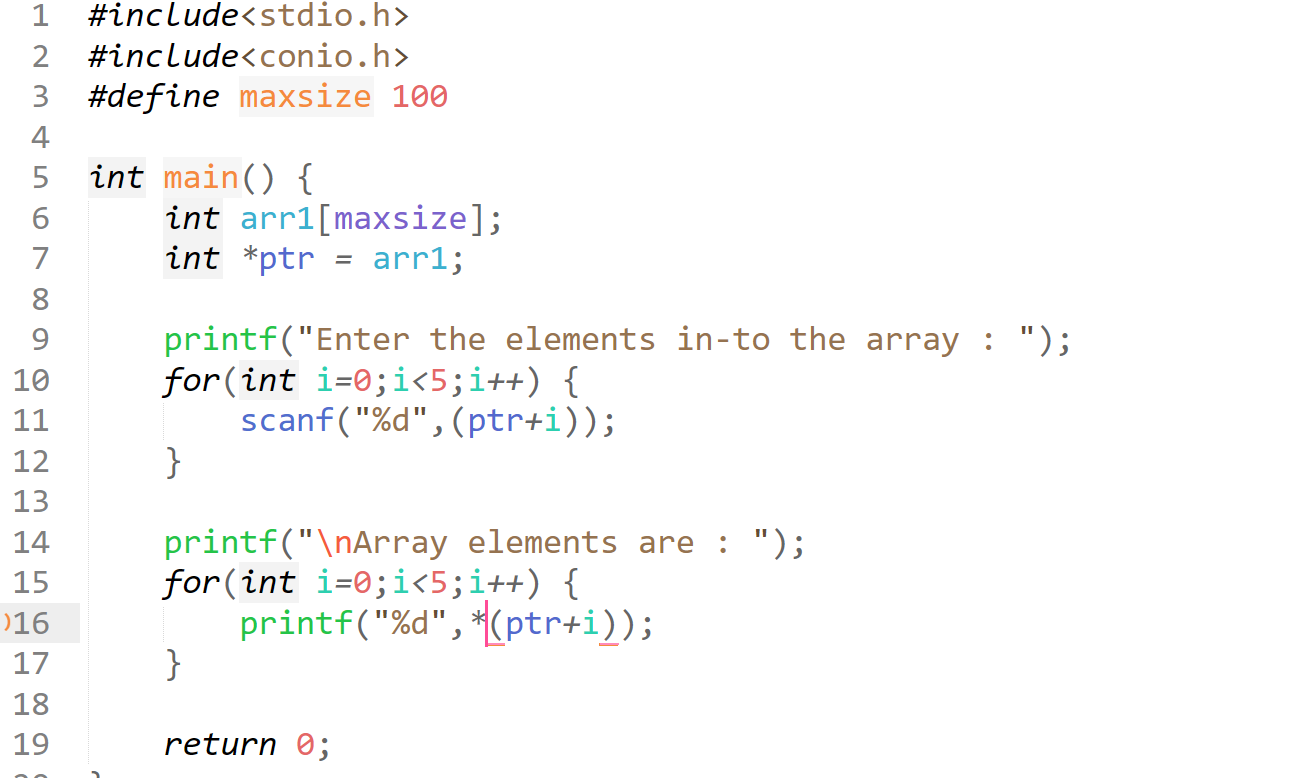
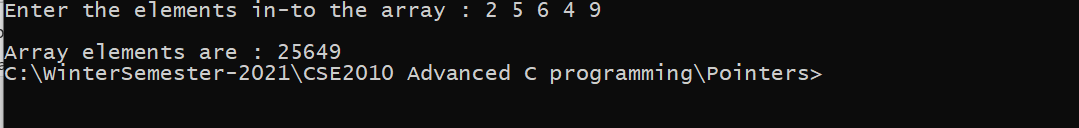


# **Arrays and pointers**

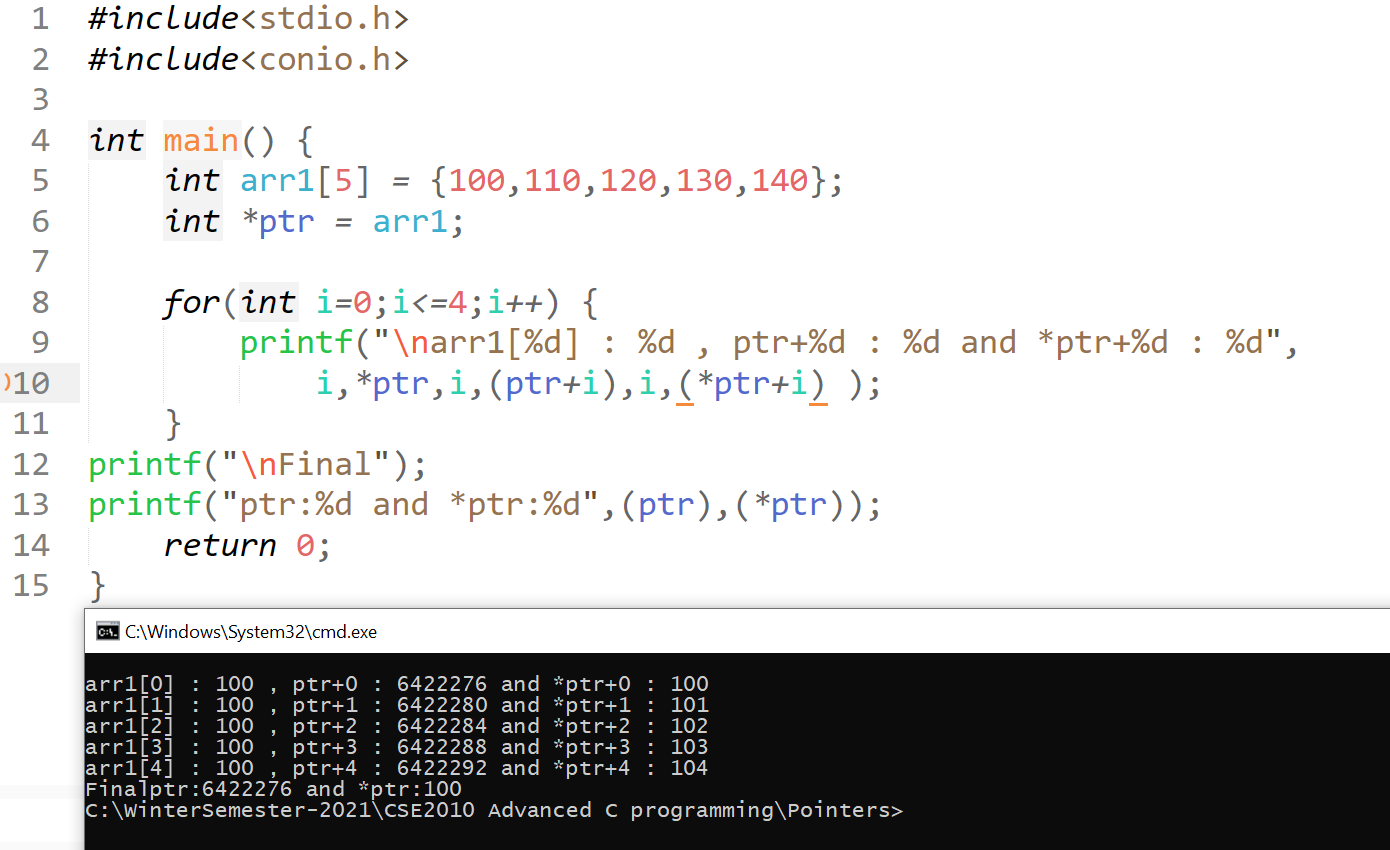
## **To input and print an array using pointers**



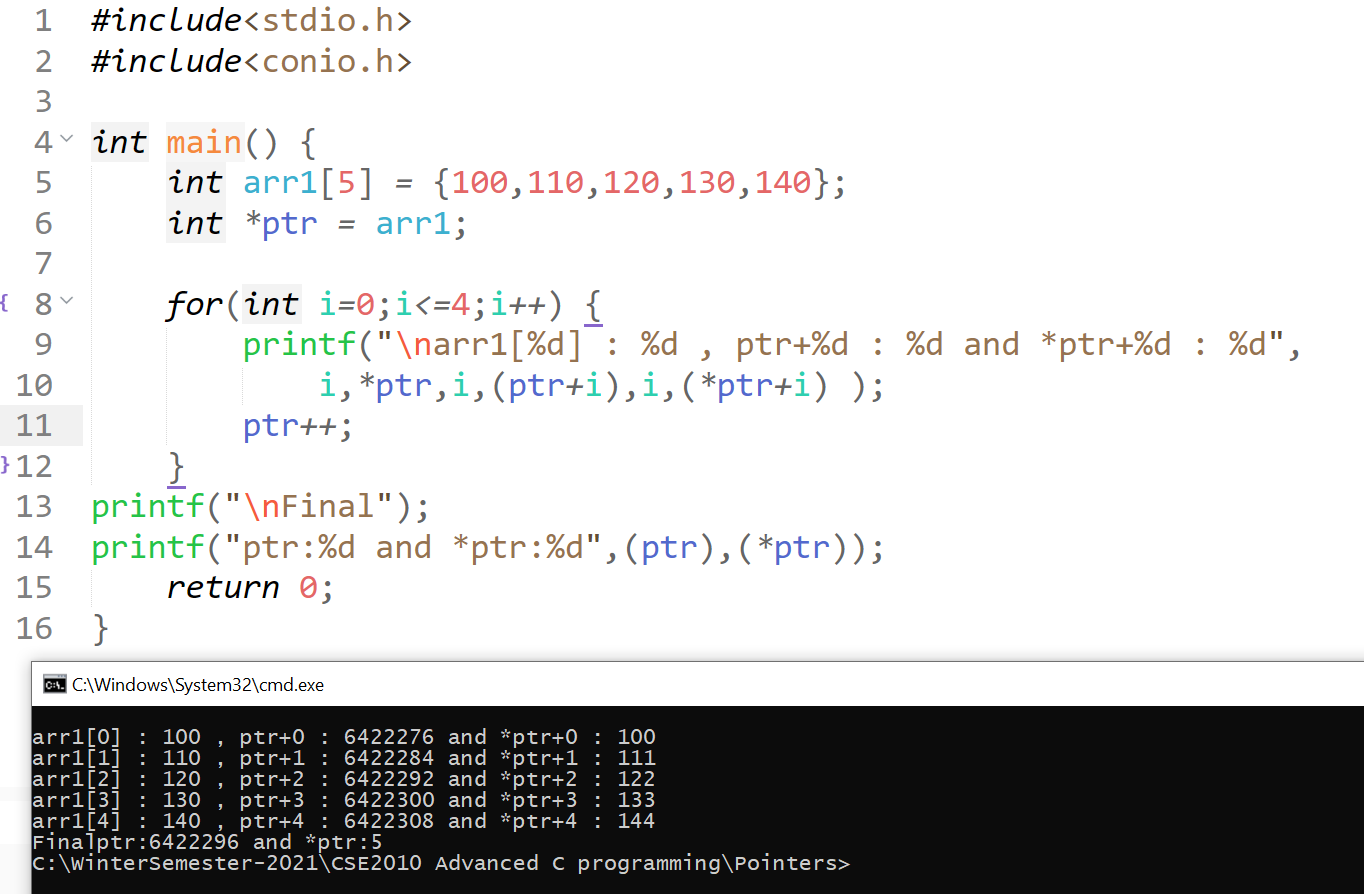
This is not the best approach, since we are moving the ptr to the last of the array, and then moving to the beginning.

  
  
(ptr + i) is equivalent to &ptr[i]  
\*(ptr + i) is equivalent to ptr[i] , (i + ptr) , i[ptr]

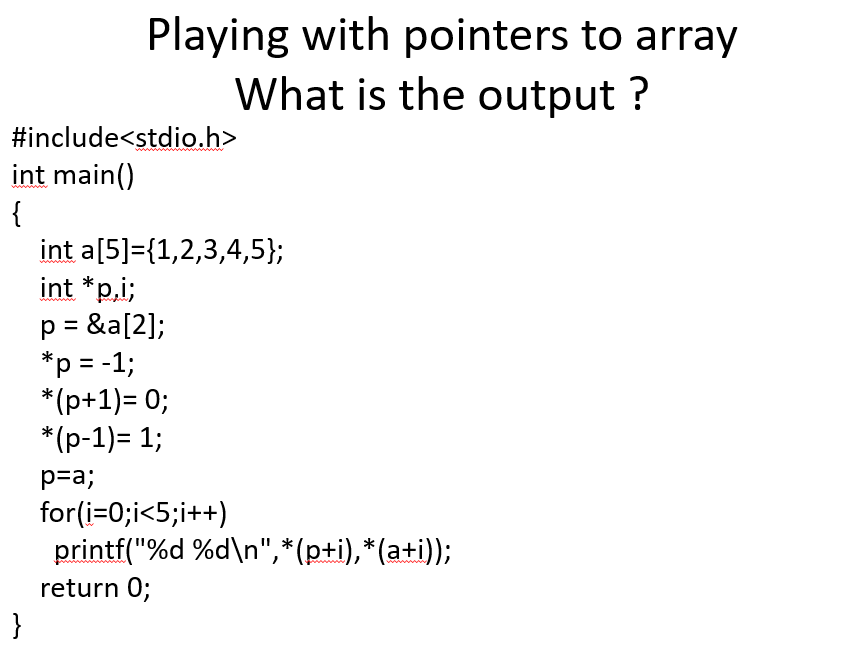
pointer\_variable + 1 🡪 doesn’t move the pointer  
pointer\_variable++ 🡪 moves the pointer

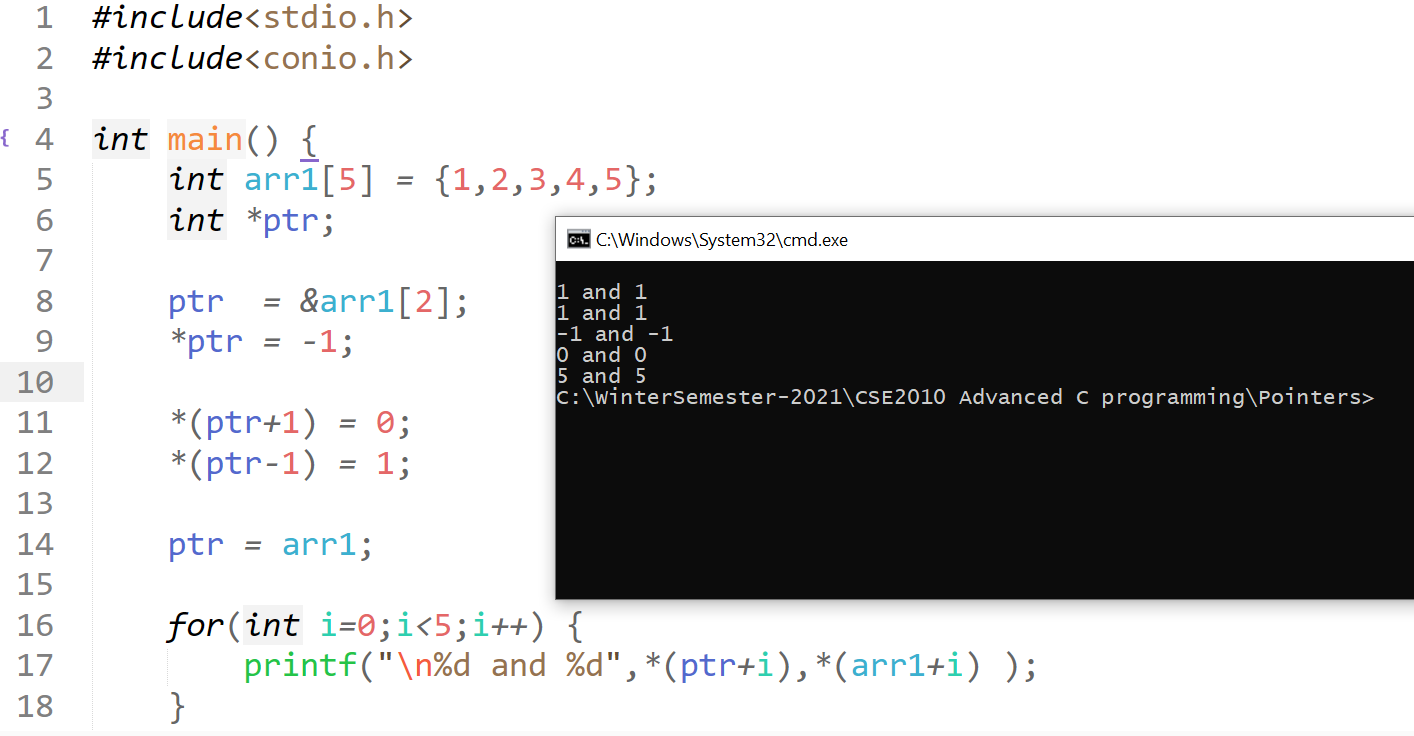












\*arr1 🡪 1 \*arr1 + 1 🡪 1 \*arr1 + 2 🡪 -1 \*arr1 + 3 🡪 0 \*arr1 + 4🡪 5  
\*ptr 🡪 1 \*ptr + 1 🡪 1 \*ptr + 2 🡪 -1 \*ptr + 3 🡪 0 \*ptr + 4 🡪 5

Both \*ptr and \*arr1 behaves in the same way like the above.

## **Difference between array name and pointer name**

var refers to array.  
ptr refers to pointer  
Array names cannot be used as an l-value.   
“l-value” refers to memory location which identifies an object.

Eg. var++;

* Array cannot be assigned to another array.

Eg. var2=var1;

* Return value of the address operator.

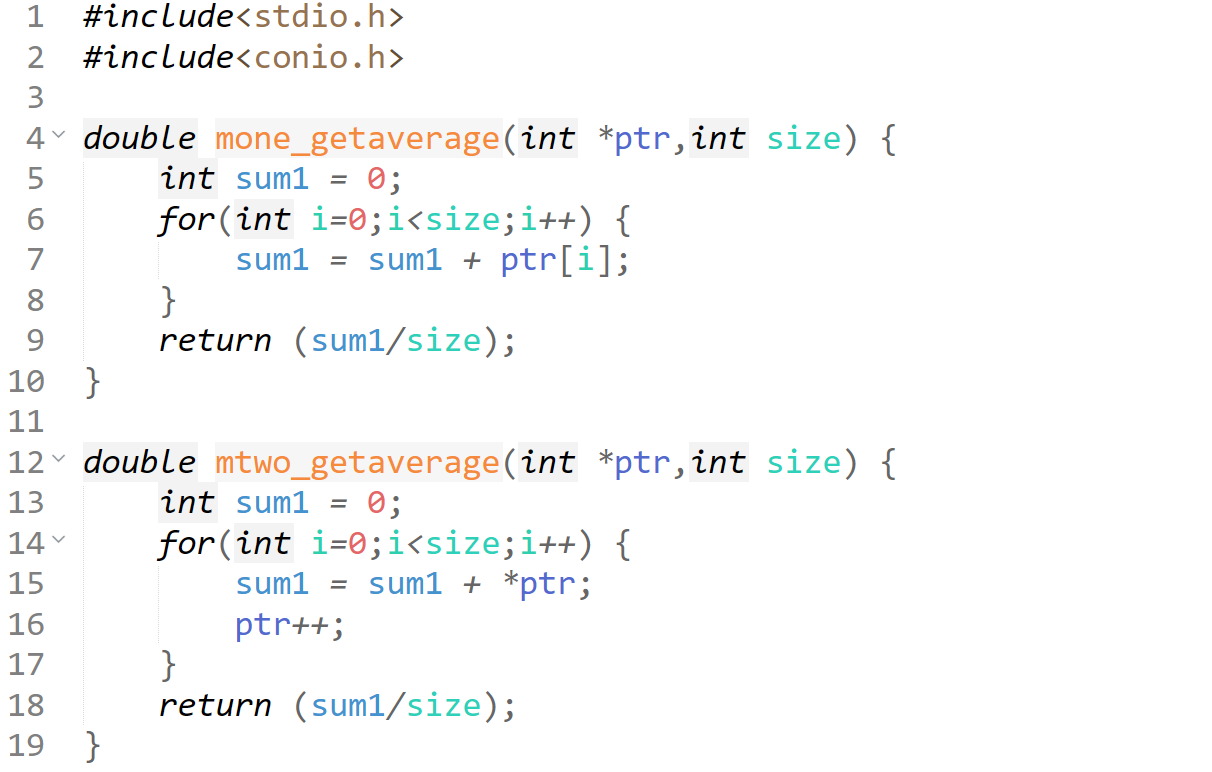
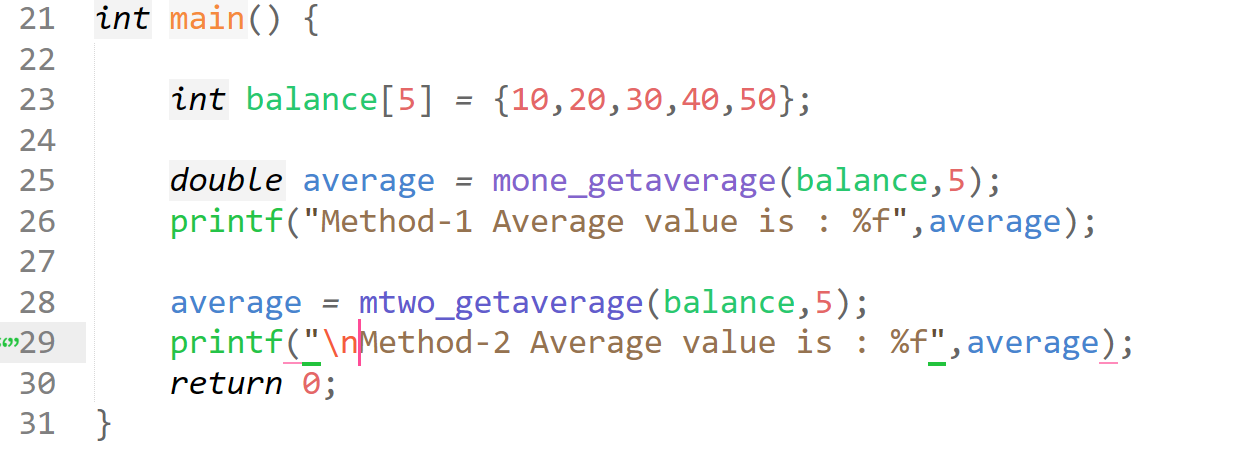
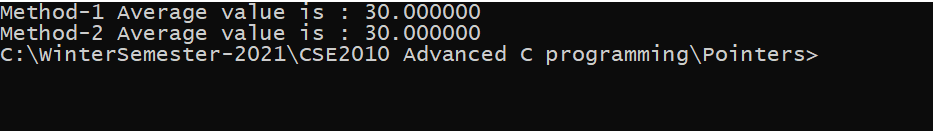
Eg. Var and &var are same, but ptr and &ptr are not same.

* Sizeof() operator

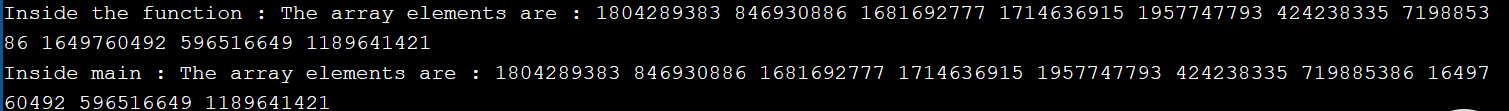
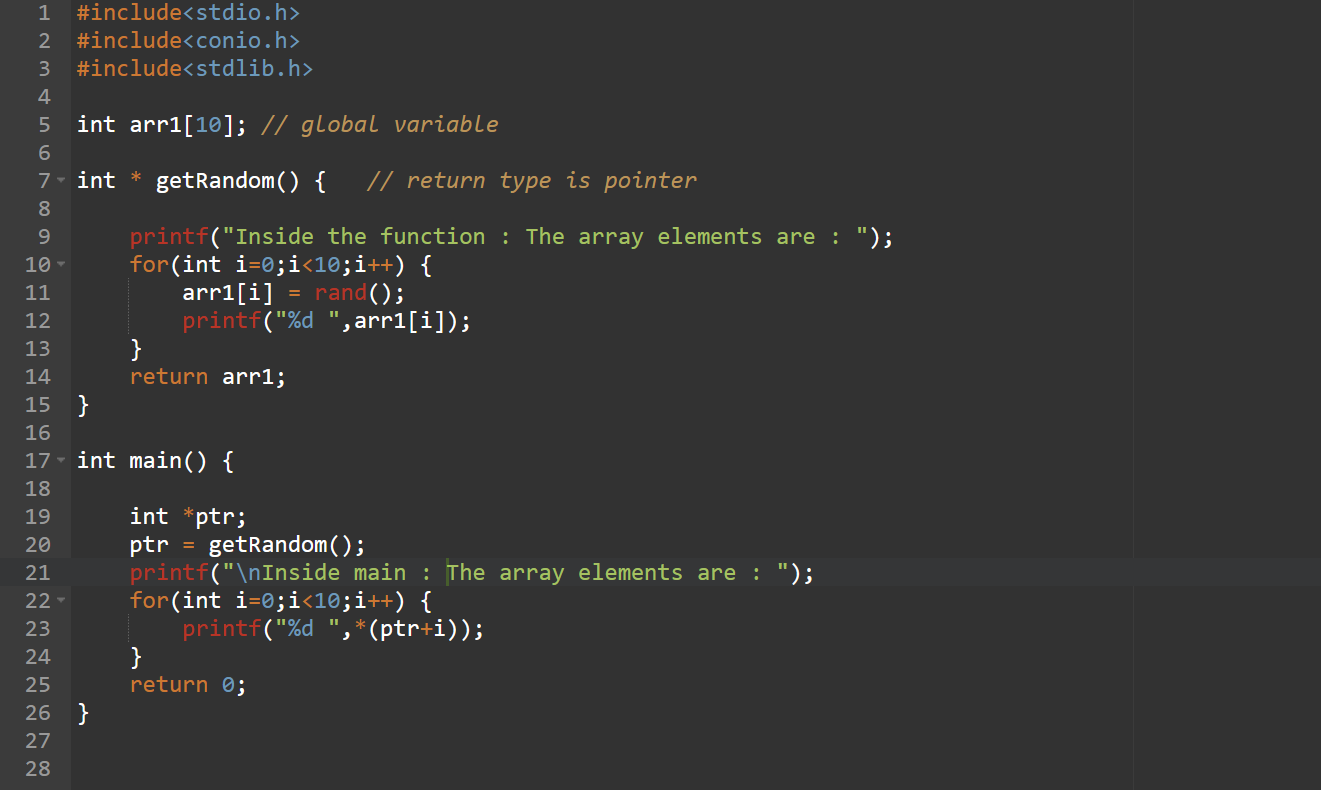
Sizeof(var) : returns the number of bytes allocated for the array.

Sizeof(ptr): returns the number of bytes used for the pointer variable.

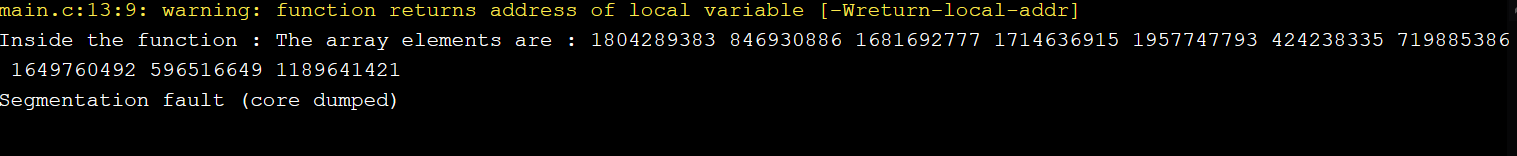
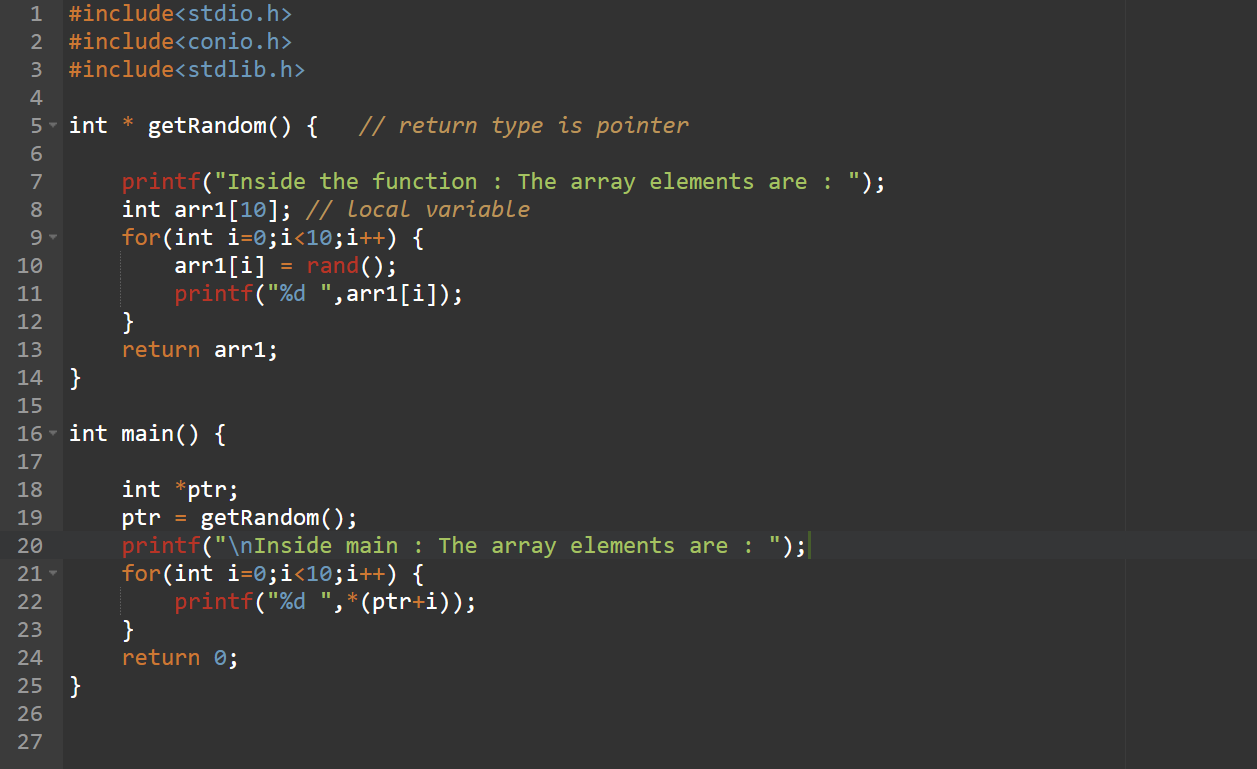
## **Passing an array to a function using pointers**

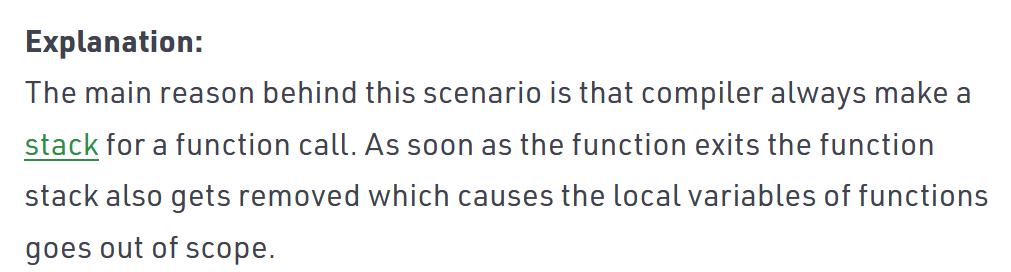
## **Return an array and receive it in a pointer**



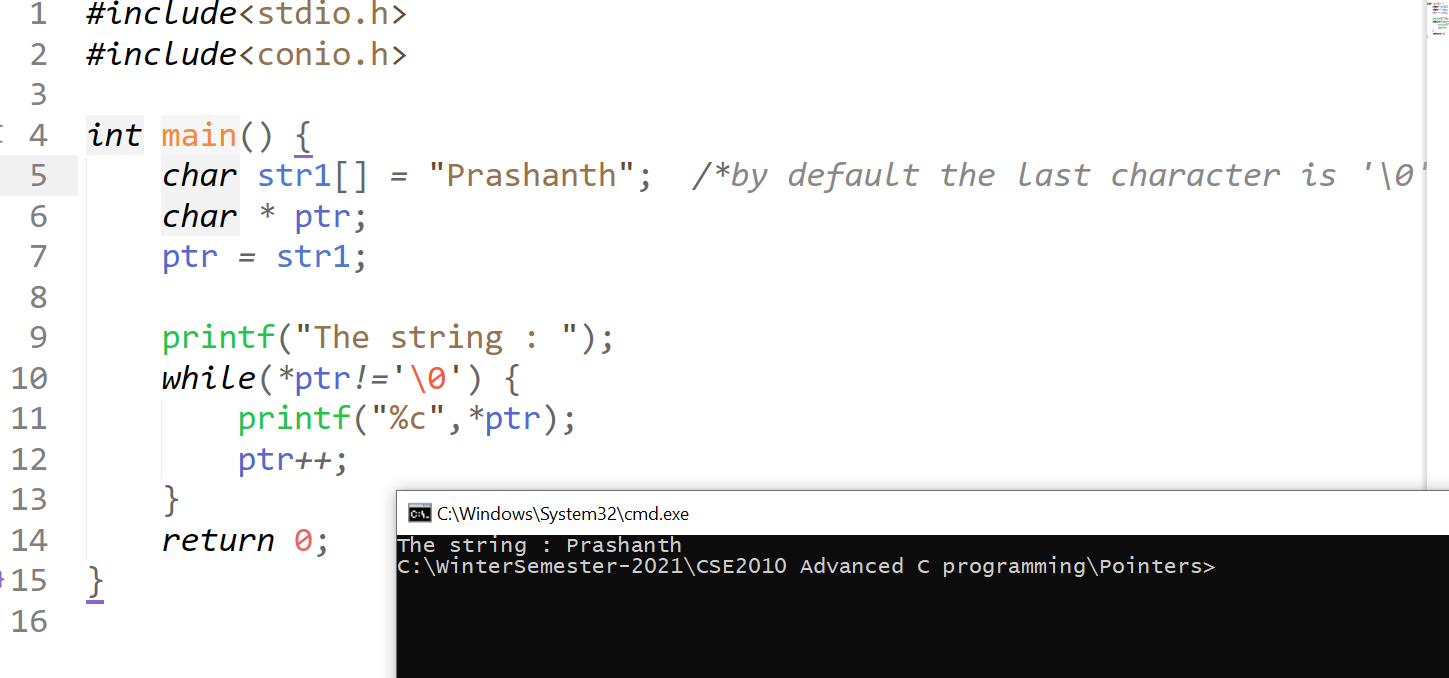








# **Strings and pointers**



## **Pointer to a function (Palindrome)**

