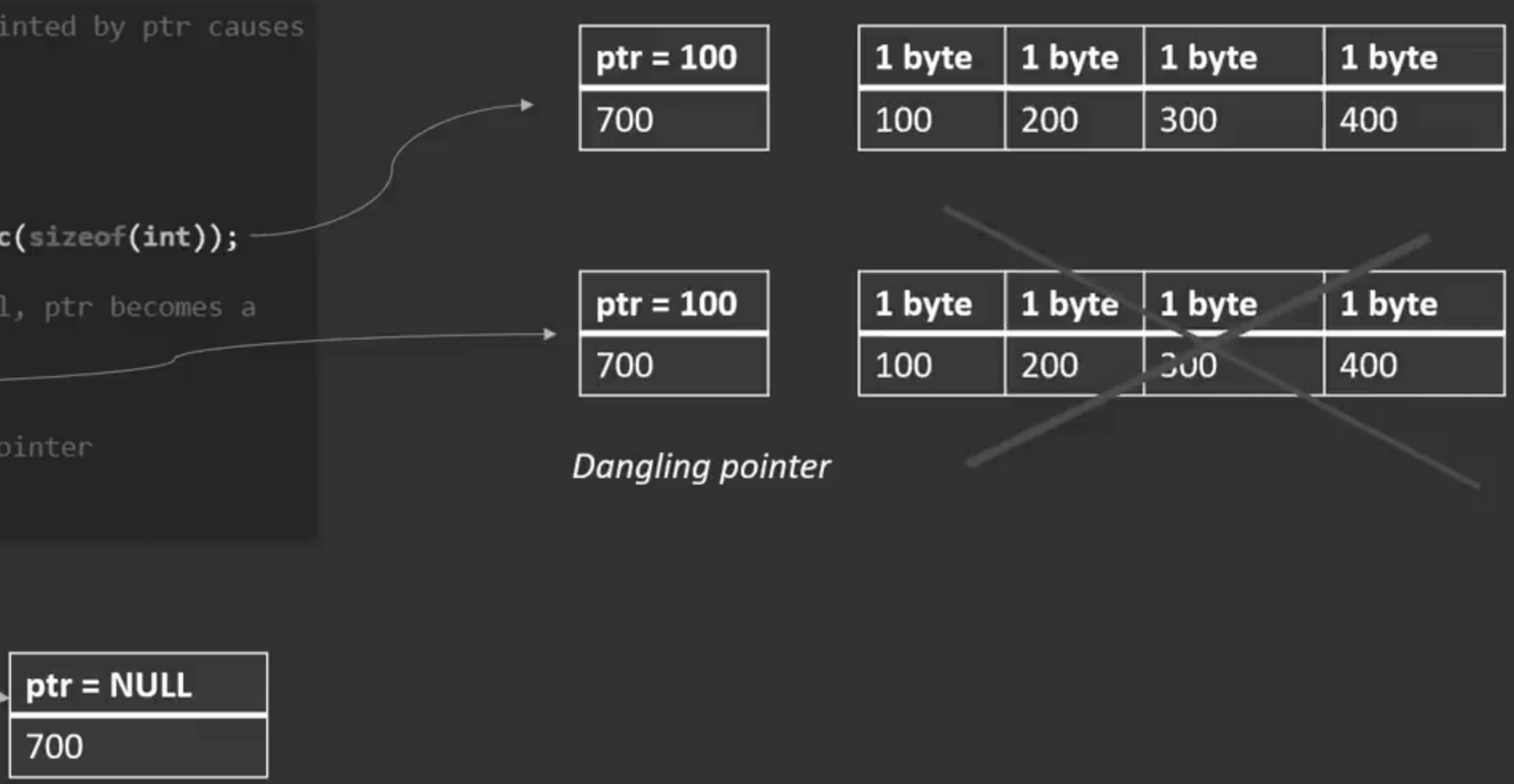
## Dangling pointer:

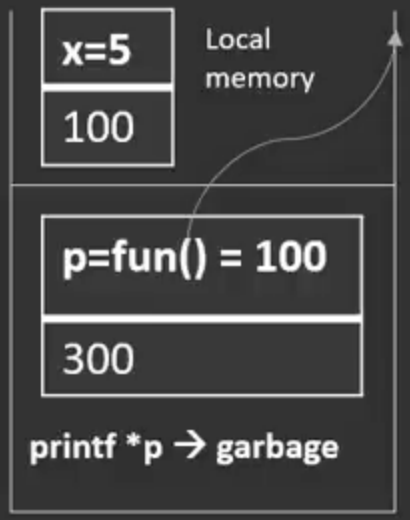
**A pointer pointing to a memory location that has been already deleted is called dangling pointer.**

```c
// Deallocating a memory pointed by ptr causes
// dangling pointer
#include <stdlib.h>
#include <stdio.h>
int main()
{
    int *ptr = (int *)malloc(sizeof(int));

    // After below free call, ptr becomes a
    // dangling pointer
    free(ptr);

    // No more a dangling pointer
    ptr = NULL;
}
```



## Dangling pointer ...

```c
// The pointer pointing to local variable becomes
// dangling when local variable is not static.
#include<stdio.h>

int *fun()
{
    // x is local variable and goes out of
    // scope after an execution of fun() is
    // over.
    int x = 5;

    return &x;
}

// Driver Code
int main()
{
    int *p = fun();
    fflush(stdin);

    // p points to something which is not
    // valid anymore
    printf("%d", *p);
    return 0;
}
```




Int *fun()

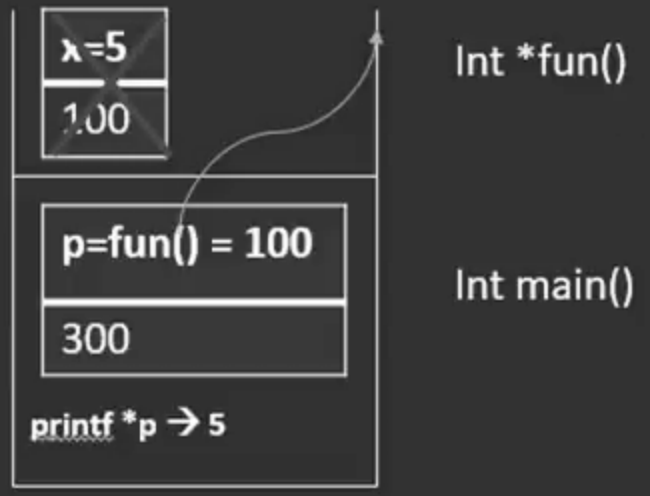
Int main()

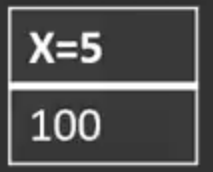
Output:
garbage

Dangling pointer …

```c
// The pointer pointing to local variable doesn't
// become dangling when local variable is static.
#include<stdio.h>

int *fun()
{
    // x now has scope throughout the program
    static int x = 5;

    return &x;
}

int main()
{
    int *p = fun();
    fflush(stdin);

    // Not a dangling pointer as it points
    // to static variable.
    printf("%d",*p);
}
```

**Now x is static, store in global memory, not in local or stack memory**




**Global memory, not local**

Output:
5

# Void pointer ...

## Void pointer – Is a type of pointer, not the value of pointer

```c
#include<stdlib.h>

int main()
{
    int x = 4;
    float y = 5.5;

    //A void pointer
    void *ptr;
    ptr = &x;

    // (int*)ptr - does type casting of void
    // *((int*)ptr) dereferences the typecasted
    // void pointer variable.
    printf("Integer variable is = %d", *( (int*) ptr) );

    // void pointer is now float
    ptr = &y;
    printf("\nFloat variable is= %f", *( (float*) ptr) );

    return 0;
}
```

**Void pointer can store address of any data type.**
**To fetch the value of that data type, we have to type cast it.**

**Output:**
**4**
**5.5**

Null Pointer - Is a value of a pointer

Wild pointer – Uninitialized value of a pointer

```c
#include <stdio.h>
int main()
{
    // Null Pointer
    int *ptr = NULL;

    printf("The value of ptr is %u", ptr);
    return 0;
}
```

```c
int main()
{
    int *p;  /* wild pointer */

    int x = 10;

    // p is not a wild pointer now
    p = &x;

    return 0;
}
```

# Thank you for watching!

Please leave us your comments.