**Codigo en C:**

```
char global_no_init[10];
char global_init[] = "mensaje global";

void foo(){
    int arr_no_ini[20];
    int arr_num_ini[20] = {1};
    char message_init[] = "mensaje";
    char message_init_changed_later[] = "menzaje";
    message_init_changed_later[3] = 's';
}
```

**Codigo asm generado:**

```
	.file	"test1.c"
	.intel_syntax noprefix
	.text
	.comm	global_no_init,10,4
	.globl	global_init
	.data
	.align 4
	.type	global_init, @object
	.size	global_init, 15
global_init:
	.string	"mensaje global"
	.text
	.globl	foo
	.type	foo, @function
foo:
	endbr32
	push	ebp
	mov	ebp, esp
	push	edi
	sub	esp, 116
	call	__x86.get_pc_thunk.ax
	add	eax, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_
	mov	eax, DWORD PTR gs:20
	mov	DWORD PTR -12[ebp], eax
	xor	eax, eax
	lea	edx, -108[ebp]
	mov	eax, 0
	mov	ecx, 20
	mov	edi, edx
	rep stosd
	mov	DWORD PTR -108[ebp], 1
	mov	DWORD PTR -28[ebp], 1936614765
	mov	DWORD PTR -24[ebp], 6646369
	mov	DWORD PTR -20[ebp], 2054055277
	mov	DWORD PTR -16[ebp], 6646369
	mov	BYTE PTR -17[ebp], 115
	nop
	mov	eax, DWORD PTR -12[ebp]
	xor	eax, DWORD PTR gs:20
	je	.L2
	call	__stack_chk_fail_local
```

```
.L2:
	add	esp, 116
	pop	edi
	pop	ebp
	ret
	.size	foo, .-foo
	.section
	.text.__x86.get_pc_thunk.ax,"axG",@progbits,__x86.get_pc_thunk.ax,comdat
	.globl __x86.get_pc_thunk.ax
	.hidden	__x86.get_pc_thunk.ax
	.type	__x86.get_pc_thunk.ax, @function
__x86.get_pc_thunk.ax:
	mov	eax, DWORD PTR [esp]
	ret
	.hidden	__stack_chk_fail_local
	.ident "GCC: (Ubuntu 9.4.0-1ubuntu1~20.04.2) 9.4.0"
	.section	.note.GNU-stack,"",@progbits
	.section	.note.gnu.property,"a"
	.align 4
	.long	1f - 0f
	.long	4f - 1f
	.long	5
0:
	.string	"GNU"
1:
	.align 4
	.long	0xc0000002
	.long	3f - 2f
2:
	.long	0x3
3:
	.align 4
4:
```

*Conclusiones:

Codigo de **char global_no_init[10]**;

```
	.comm	global_no_init,10,4
```

--> Va a la GOT

Codigo de **char global_init[]**;

```
.globl global_init
	.data
	.align 4
	.type	global_init, @object
	.size	global_init, 15
global_init:
	.string	"mensaje global"
	.text
	.globl foo
	.type	foo, @function
```

--> Va a la GOT

## Para int arr_no_ini[20]; se reserva espacio en el stack

---

Para array_ini[20] = {1}

```
void foo(){
    int arr_num_ini[20] = {1};
    for(int i=0; i<20; i++){
        printf("%d, ", arr_num_ini[i]);
    }
}
int main(){
    foo();
    return 0;
}
```

Imprime 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

```
foo:
      endbr32
      push   ebp
      mov    ebp, esp
      push   edi
      sub    esp, 100
      call   __x86.get_pc_thunk.ax
      add    eax, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_
      mov    eax, DWORD PTR gs:20
      mov    DWORD PTR -12[ebp], eax
      xor    eax, eax
      lea    edx, -92[ebp]
      mov    eax, 0
      mov    ecx, 20
      mov    edi, edx
      rep stosd
      mov    DWORD PTR -92[ebp], 1
      mov    DWORD PTR -88[ebp], 2
      mov    DWORD PTR -84[ebp], 3
      mov    DWORD PTR -80[ebp], 4
      mov    DWORD PTR -76[ebp], 5
      mov    DWORD PTR -72[ebp], 6
      nop
      mov    eax, DWORD PTR -12[ebp]
      xor    eax, DWORD PTR gs:20
      je     .L2
      call   __stack_chk_fail_local
.L2:
      add    esp, 100
      pop    edi
      pop    ebp
      ret
```

Concl: Se guarda en el stack. En este caso hice int ini[20] = {1,2,3,4,5,6} --> los mueve lugar por lugar.

---

Para un mensaje inicializado, que no se accede:

```
int foo(){
    char message_init[] = "mensaje";
}
```

```
.file  "test4.c"
       .intel_syntax noprefix
       .text
       .globl foo
       .type  foo, @function
foo:
       endbr32
       push   ebp
       mov    ebp, esp
       sub    esp, 24
       call   __x86.get_pc_thunk.ax
       add    eax, OFFSET FLAT:_GLOBAL_OFFSET_TABLE_
       mov    eax, DWORD PTR gs:20
       mov    DWORD PTR -12[ebp], eax
       xor    eax, eax
       mov    DWORD PTR -20[ebp], 1936614765
       mov    DWORD PTR -16[ebp], 6646369
       nop
       mov    edx, DWORD PTR -12[ebp]
       xor    edx, DWORD PTR gs:20
       je     .L2
       call   __stack_chk_fail_local
.L2:
       leave
       ret
       .size  foo, .-foo
```

Concl: hace cuentas locas y lo guarda en el stack. Pone byte por byte

EBP-16: 656A61  == 00 65 6A 61
EBP-20: 736E656D == 73 6E 65 6D
EBP-16: 00
EBP-17: 65 ('e')
EBP-18: 6A ('j')
EBP-19: 61 ('a')
EBP-20: 73 ('s')
EBP-21: 6E ('n')
EBP-22: 65 ('e')
EBP-23: 6D ('m')
<-- ESP apunta aca!

Si luego lo modificamos hace el mismo proceso, y luego hace el mov en la posicion adecuada.