Cuando se compila un programa en C se debe escoger un modelo de memoria.

TINY
SMALL
MEDIUM
COMPACT
LARGE
HUGE

LARGE

Programas grandes que usan muchos datos.
Varios segmentos físicos para código (hasta 1 MB) y para datos y pila (hasta 1 MB). En total no puede superarse 1MB.
Punteros largos (*far*) para código y datos.

El compilador de C siempre nombra de igual forma los segmentos lógicos que utiliza:

El segmento de código se llama **_TEXT**.
El segmento **_DATA** contiene las variables globales inicializadas.
El segmento **_BSS** contiene las variables globales NO inicializadas.

El segmento de pila lo define e inicializa el compilador de C en la función *main*.

En los modelos pequeños de datos (*tiny*, *small* y *medium*), todos los segmentos de datos están agrupados con el nombre **DGROUP**:
DGROUP **GROUP** _DATA, _BSS

-> no aplica en la práctica (modelo LARGE)

El compilador de C añade un **_** delante de todos los nombres de variables y procedimientos:

```
int a = 12345;
char b = ‘A’;
char c[] = ''Hola mundo'';
int d = 12;
```

```
main()
{
  funcion();
}
```

```
_DATA SEGMENT WORD PUBLIC ‘DATA’
PUBLIC _a, _b, _c, _d
_a DW 12345
_b DB ‘A’
_c DB ''Hola mundo'', 0
_d DW 12
_DATA ENDS
```

```
_TEXT SEGMENT BYTE PUBLIC ‘CODE’
_main PROC FAR
CALL _funcion
RET
_main ENDP
_TEXT ENDS
```

Paso de parámetros

En lenguaje C, un procedimiento que llama a otro apila sus parámetros antes de ejecutar el **CALL**.

Los procedimientos de ensamblador que llamen a funciones de C también han de apilar sus parámetros.

Los parámetros se apilan en orden inverso a como aparecen en la llamada de C: empezando por el último y acabando por el primero.

Tras retornar de la subrutina, se extraen de la pila los parámetros sumando al registro **SP** el tamaño en bytes de los parámetros.

Los parámetros de un byte (char) se apilan con dos bytes (el más significativo vale 0).

Los parámetros se apilan en formato *little endian*: palabra menos significativa en dirección menor y byte menos significativo en dirección menor.

Para pasar por parámetro punteros a funciones o a datos, es necesario saber en qué modelo de memoria se está compilando el programa en C, **para apilar el registro de segmento (modelo largo)** o no apilarlo (modelo corto).

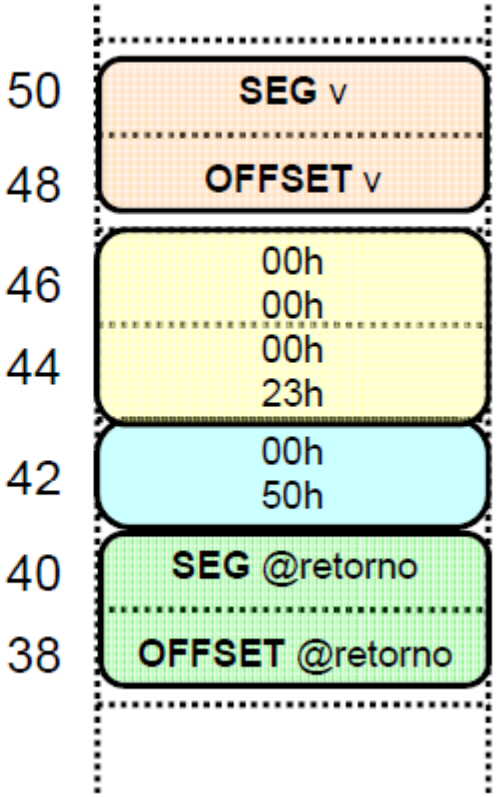
- Prototipo de la función:

  void funcion (**char**, **long int**, **void** *);

- Llamada en modelo largo (punteros **FAR** para datos y código)

  funcion ('P', 0x23, &v);

```
call _funcion
```



Acceso a parámetros (ejemplo)

void funcion (**char**, **long int**, **void** *);

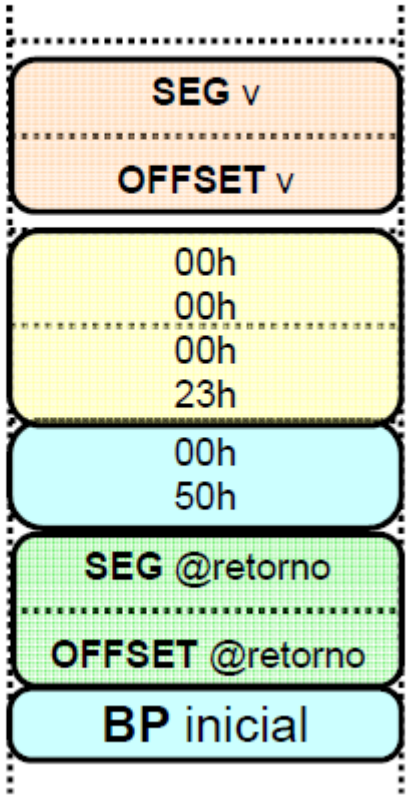
```
_funcion PROC FAR
        push bp
        mov bp, sp

        les bx, [ bp + 12 ]
        mov dx, [ bp + 10 ]
        mov ax, [ bp + 8 ]
        mov cx, [ bp + 6 ]
        …

        ret
_funcion ENDP
```

bx := **OFFSET** v
es := **SEG** v

SP = BP



Devolución de resultados

- Las variables de retorno de una función con una longitud de 16 bits se devuelven al procedimiento llamante en **AX** y las de 32 bits en **DX:AX**.

EJEMPLO A USAR / MODIFICAR

```
PRAC3A SEGMENT BYTE PUBLIC 'CODE'

PUBLIC _funcion1, _funcion2, _etc

ASSUME CS: PRAC3A

_funcion1 PROC FAR
                PUSH BP
                MOV BP, SP

                PUSH BX DX ; salvar registros a usar

                MOV AX, [BP + 6] ; recuperar datos
                MOV BX, [BP + 8] ; recuperar datos

                MOV AX, Resultado ; en caso de retorno de datos por AX.
                POP DX BX BP ;recuperar registros usados

                RET
_funcion1 ENDP


PRAC3A ENDS ; FIN DEL SEGMENTO DE CODIGO

END ; FIN DE pract3a.asm
```

Si necesitamos definir datos de manera global:

```
; DEFINICION DEL SEGMENTO DE DATOS
_DATOS SEGMENT WORD PUBLIC 'DATA'
                PUBLIC _VAR
                _VAR DW 2    (variable global)
                TABLA DW 4 DUP(?)
_DATOS ENDS


_pract3a SEGMENT BYTE PUBLIC 'CODE'
                ASSUME CS:_pract3a, DS:_DATOS
```

Etc..