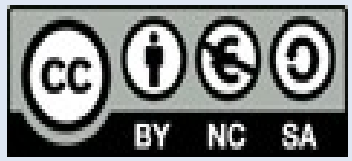
Memoria Dinámica

Jornadas de Marzo 2010

Grupo de Usuarios de Linux

Tania Pérez



Memoria Dinámica

1. PUNTEROS
2. MEMORIA DINÁMICA

Memoria Dinámica

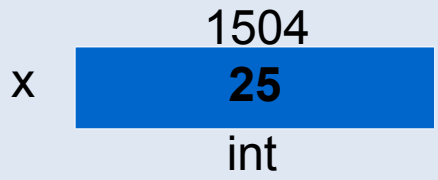
1. PUNTEROS

Memoria Dinámica

¿Qué es un puntero?

- Un tipo de variable cuyo valor es la dirección de memoria de otra variable.

int x = 25;



- La dirección de la variable x (&x) es 1504

- El contenido de la variable x es 25

Memoria Dinámica

- **Reglas por las que se rigen los punteros**
  - Un puntero es una variable como cualquier otra.
  - Una variable puntero contiene una dirección que apunta a otra posición en memoria.
  - En esa posición se almacenan los datos a los que apunta el puntero.
  - Un puntero apunta a otra variable de memoria.

Memoria Dinámica

- Declaración de punteros

<tipo_de_dato_apuntado> * <identificador_de_puntero>;

```
int    *p;
char  *p1;
float  *p2;
```

Memoria Dinámica

- Operadores de punteros

OPERADOR	PROPÓSITO
&	Obtiene la dirección de una variable
*	Define una variable como puntero
*	Obtiene el contenido de una variable puntero

Memoria Dinámica

- Inicialización de Punteros
  - **Estática**
  - **Dinámica**

Memoria Dinámica

- Inicialización Estática

  - Se asigna memoria estáticamente definiendo una variable.
  - Se declara el puntero.
  - El puntero apunta al valor de la variable.

```
Int x = 25;
int *p;
p = &x;
```

Memoria Dinámica

ERROR

```
int *p;

*p = 50;

/*error px no contiene
dirección */
```

SOLUCIÓN

```
int x = 50;

int *p;

p = &x;
```

Memoria Dinámica

ERROR

```
float *p;
char *c;
p = &c;
/* no es válido */
```

SOLUCIÓN

```
/* c y p tienen
   que ser del mismo
   tipo * /
```

Memoria Dinámica

TIPOS DE PUNTEROS

- Punteros Especiales (null,void)
- Punteros a punteros
- Punteros y arrays
- Arrays de punteros
- Punteros a cadenas
- Punteros como argumentos de funciones

Memoria Dinámica

TIPOS DE PUNTEROS

- Puntero a una función
- Puntero a una estructura

TIPOS DE PUNTEROS

- **Punteros Especiales (null,void)**
- Punteros a punteros
- Punteros y arrays
- Arrays de punteros
- Punteros a cadenas
- Punteros como argumentos de funciones

Memoria Dinámica

- **PUNTEROS ESPECIALES**
  - PUNTERO NULL
  - PUNTERO VOID

Memoria Dinámica

PUNTERO NULL

- No apunta a ninguna parte.
- No direcciona ningún dato válido en memoria.
- Lo podemos definir:
  - Macro → #DEFINE NULL 0
  - Librerias → stdef.h stdio.h stdlib.h string.h
- Ej: char *p = NULL;

PUNTERO VOID

- Apunta a cualquier tipo de dato
- Direcciona cualquier posición en memoria
- Se puede igualar a nulo si no direcciona ningún dato válido;
- Ej: void *p;

  void *ptr = NULL;

TIPOS DE PUNTEROS

- Punteros Especiales (null,void)
- **Punteros a punteros**
- Punteros y arrays
- Arrays de punteros
- Punteros a cadenas
- Punteros como argumentos de funciones

Memoria Dinámica

Punteros a punteros

- Puntero que apunta a otra variable puntero.

```
int a=20;

int * p1=&a;  /* Puntero a entero */
int ** p2=&p1; /* Puntero a puntero entero */

*p1=25;
**p2=50;
```

TIPOS DE PUNTEROS

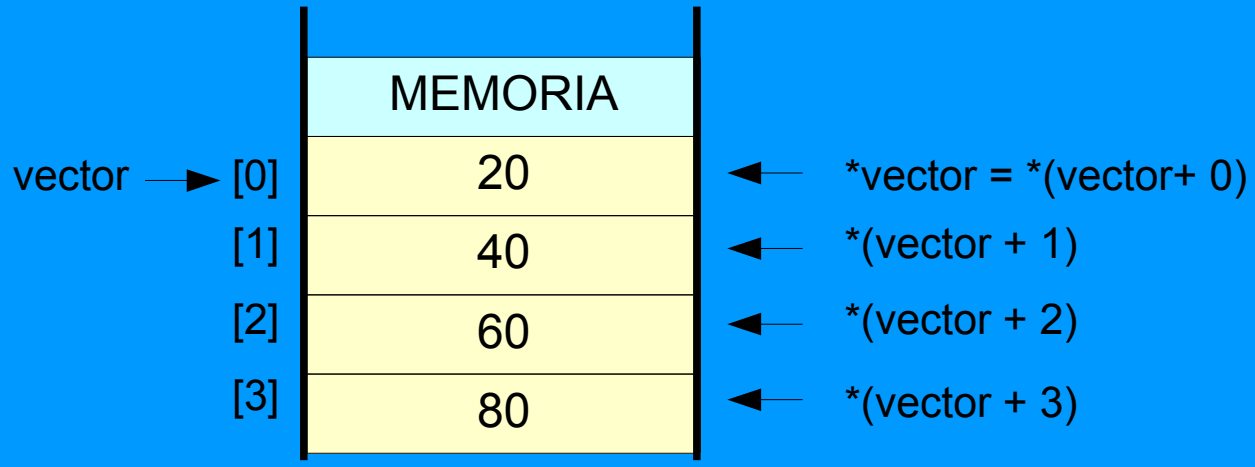
- Punteros Especiales (null,void)
- Punteros a punteros
- **Punteros y arrays**
- Arrays de punteros
- Punteros a cadenas
- Punteros como argumentos de funciones

Memoria Dinámica

Punteros y Arrays

- Un nombre de array es simplemente un puntero.

int vector[4] = {20,40,60,80} ;



Memoria Dinámica

ERROR

```
float v[10];
float x = 100.5;
v = &x;
/* ERROR
   intento modificar
   un puntero constante */
```

SOLUCIÓN

- $v[a_i] = x \mid a_i < 10$
- Asignar x a una variable puntero.

Memoria Dinámica

MORALEJA

No se puede cambiar el valor del nombre de un array durante la ejecución de un programa

NOMBRE DE ARRAY = CONSTANTE PUNTERO
!=
VARIABLE PUNTERO

Memoria Dinámica

ERROR

```
char *cadena;
..
cadena= ”Hola”;
/* ERROR*/
```

SOLUCIÓN

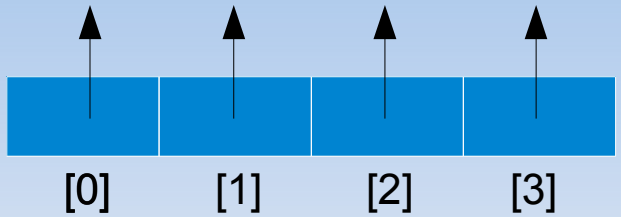
```
char * cadena = ”Hola”;
ó
malloc() + strcpy()
```

TIPOS DE PUNTEROS

- Punteros Especiales (null,void)
- Punteros a punteros
- Punteros y arrays
- **Arrays de punteros**
- Punteros a cadenas
- Punteros como argumentos de funciones

Memoria Dinámica

Arrays de punteros



- Array que contiene como elementos punteros, cada uno apunta a un tipo de datos específico.

```
int *p[10]; /* Array de 10 punteros a enteros*/
char *vector[25]; /*Array de 25 punteros a carácter*/
int (*p1)[]; → /*Puntero a un array de enteros*/
```

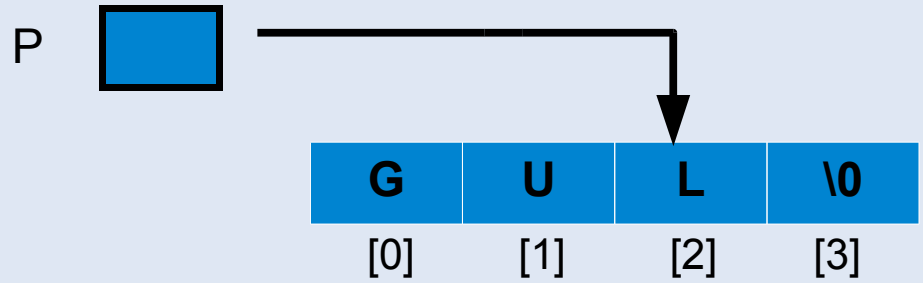
TIPOS DE PUNTEROS

- Punteros Especiales (null,void)
- Punteros a punteros
- Punteros y arrays
- Arrays de punteros
- **Punteros a cadenas**
- Punteros como argumentos de funciones

Memoria Dinámica

Punteros de cadenas

- Los punteros se pueden utilizar en lugar de índices de arrays.



Memoria Dinámica

ERROR

```
char cadena[4]=”gul”;
char * p;
p = &cadena[0];  /* p= cadena */
printf(”%c \n”,*p);
p=&cadena;      /* ERROR tipo de asignacion incompatible */
```

TIPOS DE PUNTEROS

- Punteros Especiales (null,void)
- Punteros a punteros
- Punteros y arrays
- Arrays de punteros
- Punteros a cadenas
- **Punteros como argumentos de funciones**

Memoria Dinámica

Punteros como argumento de funciones

- **Paso por valor** – no se puede cambiar el valor de la variable.
- **Paso por referencia** - Se puede cambiar el valor de la variable.

Memoria Dinámica

- En C por defecto, el paso de parámetros se hace por valor

Memoria Dinámica

- **C no tiene parámetros por referencia**

Solución → PUNTEROS

(pasando la dirección de una variable)

Memoria Dinámica

Ejemplo

```
void funcion ( int * a, int b){
    *a = 5;
    b = 25;
    };
```

```
/* invocación a la función*/
int x =1;
int y = 2;
funcion (&x,y);
```

TIPOS DE PUNTEROS

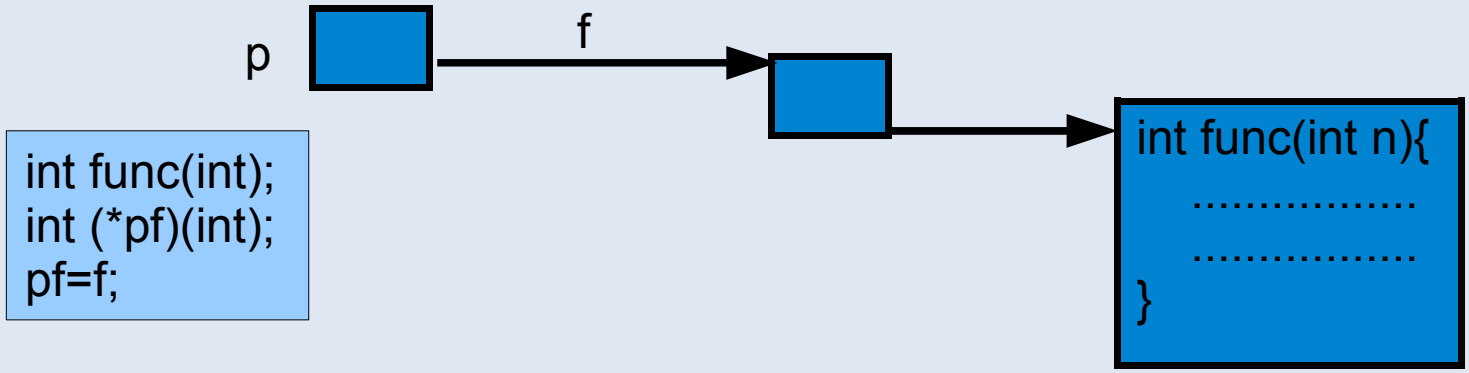
- **Puntero a una función**
- Puntero a una estructura

Memoria Dinámica

Punteros a funciones

- Apuntan a código ejecutable, en lugar de direccionar datos.

```
<tipo_de_retorno> (*PunteroaFuncion) (<lista_parametros>);
```



Memoria Dinámica

Utilidad

- Pasar una función como argumento a otra función.

```
int fa(int i, int j){
    return i+j;
);
```

```
/* decalaración */
void mifuncion( int (*fa)(int,int) );

/* llamada */
mifuncion(fa);
```

¡Cuidado!

- La función debe tener el mismo tipo de retorno y lista de parámetros.

TIPOS DE PUNTEROS

- Puntero a una función
- **Puntero a una estructura**

Memoria Dinámica

Punteros a estructuras

- Se declara igual que declaramos un puntero a cualquier otro tipo de objeto

```
struct persona {
        char nombre [30];
        int edad;
}
struct persona p1 = {"Elena",20};
```

Acceso a través de la estructura

p1.nombre

```
struct persona * p;
p = &p1;
```

Acceso a través del puntero a estructura

p1-> nombre

*(p1).nombre

2. MEMORIA DINÁMICA

Memoria Dinámica

¿Punteros o Memoria Dinámica?

- Los punteros y la memoria dinámica están muy relacionados... pero no siempre van juntos.
- Podemos trabajar con punteros sin memoria dinámica.
- ¿Ejemplo?
  - Parámetros por referencia
  - Arrays estáticos

Memoria Dinámica

¿Qué es la Memoria Dinámica?

- Memoria que reservamos durante la ejecución de nuestro programa.

Memoria Dinámica

¿Y dónde se almacena esa MEMORIA?

En una zona del mapa de memoria llamada...

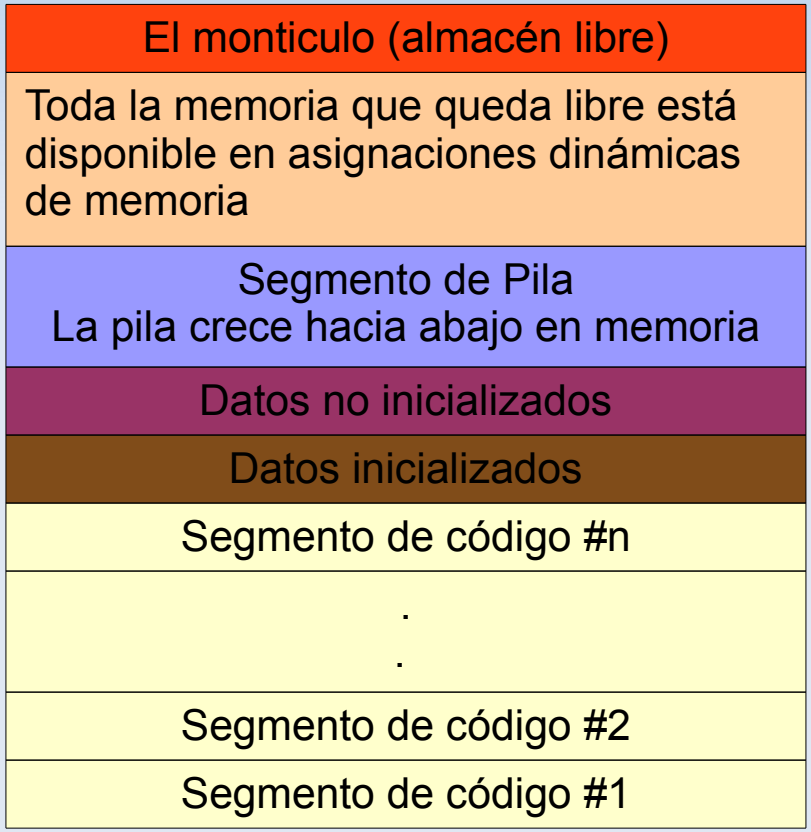
ALMACÉN LIBRE

Memoria Dinámica

Mapa de memoria de un programa

Memoria alta

Cada segmento de código, dato o pila se limita a 64 k



Segmento de datos

Memoria baja

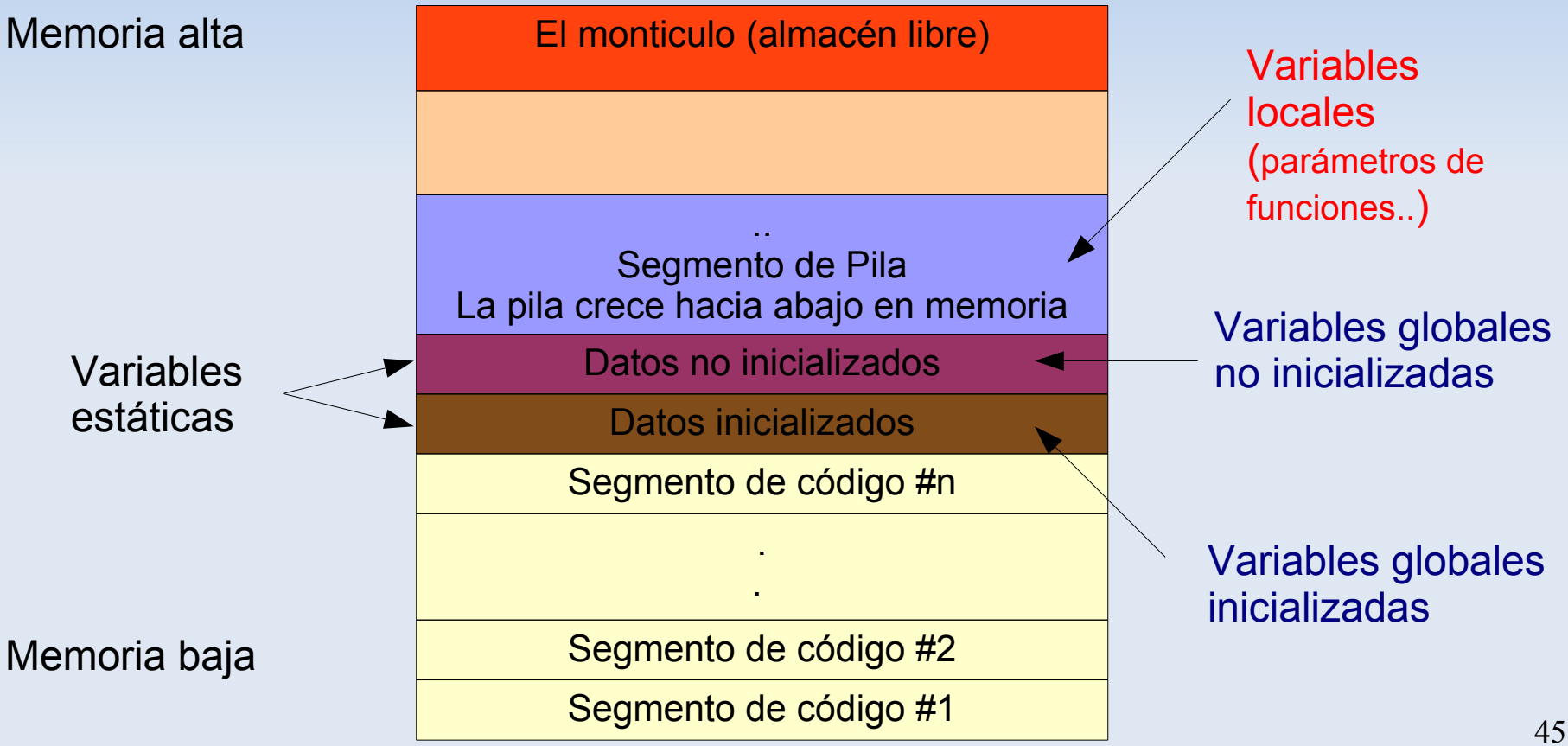
Memoria Dinámica

¿Y las variables?

- **Hay que distinguir..**
  - **Variables globales**
  - **Variables locales**
  - **Variables estáticas**

Memoria Dinámica

Mapa de memoria de un programa



Memoria Dinámica

¿Cómo gestionar esa MEMORIA?

- Asignar memoria
- Liberar memoria

Memoria Dinámica

Funciones de Gestión de Memoria

ASIGNACIÓN

- **Función MALLOC ()**
- **Función CALLOC ()**
- **Función REALLOC ()**

LIBERACIÓN

- **Función FREE ()**

Memoria Dinámica

Funciones de Gestión de Memoria

- **Función malloc()**
- Función malloc()
- Función malloc()
- Función free()

Memoria Dinámica

- Función malloc()
  - Asigna bloques de memoria
  - ¿Qué devuelve esta función?

    La dirección de memoria donde comienza la reserva realizada ó NULL.
  - ¿Qué parámetros necesita?
  - El tamaño de la memoria a reservar en bytes
    - Muy útil → SIZEOF()

```
#include <stdlib.h>

void *malloc( size_t size );
```

Memoria Dinámica

Ejemplo

```
/*  Reserva memoria para un entero */
int *p;
p = (int *)malloc(sizeof(int));
(*p)=5;
```

```
/* Reserva memoria para una array de 5 enteros */
int *p;
p = (int *)malloc(sizeof(int) *5 );
(*p)=5;
```

Memoria Dinámica

RECOMENDACIÓN

```
/* COMPROBAR EL VALOR DEVUELTO POR MALLOC*/
int *p;
p = (int *)malloc(sizeof(int) *5 );
If ( p == NULL ) {
      puts ( ”Error en la asignación de memoria”);
      return (-1);
}
(*p)=5;
```

Memoria Dinámica

ERROR

```
int num;
scanf (”%d”,&num);
int vector[num];
/* ERROR
el valor de num es
desconocido en tiempo de
compilación*/
```

SOLUCIÓN

```
int num;
int * vector;
scanf (”%d”,&num);

vector = (int*)
    malloc(num*sizeof(int);
```

Memoria Dinámica

Funciones de Gestión de Memoria

- Función malloc()
- **Función calloc()**
- Función realloc()
- Función free()

Memoria Dinámica

- Función calloc()
  - Reserva memoria
  - Inicializa la memoria a 0.
  - ¿Qué devuelve esta función?

    La dirección de memoria donde comienza la reserva realizada ó NULL.
  - ¿Qué parámetros necesita?
  - Numero de elementos a reservar
  - Tamaño en bytes del tipo de elemento.

```
#include <stdlib.h>

void *calloc( num_eltos,size_elto );
```

Memoria Dinámica

Funciones de Gestión de Memoria

- Función malloc()
- Función calloc()
- **Función realloc()**
- Función free()

Memoria Dinámica

- Función realloc()
  - Permite reservar un bloque de memoria reservado anteriormente.

```
#include <stdlib.h>

void *realloc(ptr_a_bloque, tam_total nuevo bloque)
```

  - Aconsejable hacer conversión a tipo puntero:

Tipo * puntero;

Puntero = (tipo *) realloc (ptr_a_bloque, tam_total nuevo bloque)

Memoria Dinámica

Funciones de Gestión de Memoria

- Función malloc()
- Función calloc()
- Función realloc()
- **Función free()**

Memoria Dinámica

- Función free()
  - Liberar memoria previamente asignada.

```
#include <stdlib.h>

void free( void *ptr );
```

  - El bloque de memoria suprimido se devuelve al montículo o almacén libre.

Memoria Dinámica


```
/* COMO RESERVAR  Y LIBERAR MEMORIA CON CADENAS */
char cad[100], *p;
puts("Introduce una frase");
gets(cad);
p = (char*) malloc ( (strlen(cad)+1) * sizeof(char) );
strcpy(p,cad);
free(p);
```

Memoria Dinámica

RECOMENDACIÓN

- **Siempre que se llama a malloc hay que llamar a..**

free(p);

Memoria Dinámica

¿Y qué es ese famoso SEGMENTATION FAULT?

- Violación de segmento de memoria.
- Se produce cuando accedemos a una posición de memoria indebida.
- **¿Cómo corregirlo?** Gestionando la memoria adecuadamente.

Memoria Dinámica

Segmentation Fault

```
int *p;
int x = 1;
p = &x;
free (p);
```

No podemos liberar memoria que previamente no fue asignada con malloc.

Memoria Dinámica

Segmentation Fault

```
int *p1,p2;
p1 = (int*) malloc(int);
*p1 = 1;
p2 = p1;
free(p1);
```

No podemos dejar punteros colgados (dangling)

Memoria Dinámica

Segmentation Fault

```
int *p1 = (int*) malloc(sizeof(int));
int *p2 = (int*) malloc(sizeof(int));
*p1 = 5;
*p2 = 10;
p2 = p1;
```

Debemos evitar la pérdida de memoria asignada

Memoria Dinámica

Dudas y sugerencias, consultar a:

Grupo de Usarios de Linux – Carlos III de Madrid (Leganés):

- Lista correo GUL UC3M:
  gul@gul.uc3m.es
- DESPACHO GUL: 2.3C05 (Ed. Sabatini)
  (Segundo viernes de cada mes)

Memoria Dinámica

FIN