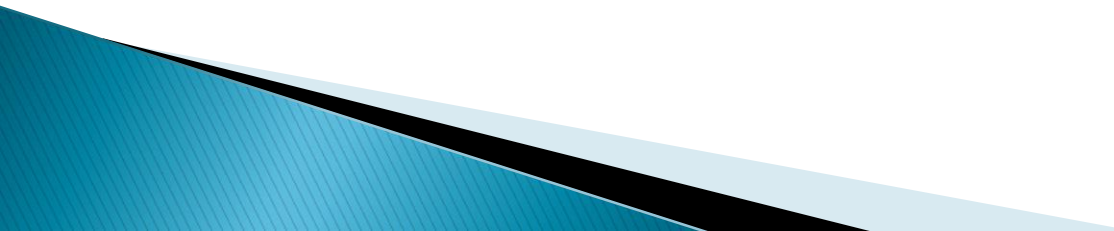


# Asignación dinámica de memoria

(1 ra. Parte)

# Asignación dinámica de memoria

- ▶ Es una característica de C.
  - ▶ Permite crear estructuras de cualquier tamaño de acuerdo a las necesidades del programa.
  - ▶ Funciones a utilizar
    - **malloc y calloc** : permite reservar cierta cantidad de bytes de memoria.
    - **free** : Libera la memoria reservada .
- 

# Función malloc( )

<stdlib.h>

- ▶ Es empleada para intentar alocar una porción contigua de memoria.

- ▶ Sintaxis

**void \* malloc( size\_t size );**

- Retorna un puntero del tipo **void \*** el cual es el inicio en memoria de la porción reservada de tamaño **size**.
- Si no puede reservar esa cantidad de memoria la función regresa un puntero nulo o **NULL**.
- El tipo **size\_t** está definido como un entero sin signo.

# Función free( )

<stdlib.h>

## ► Sintaxis

```
void free( void * ptr );
```

- Libera el espacio apuntado por **ptr**.
- Si **ptr** es un puntero nulo, no se realizará ninguna acción.
- Si el puntero no es válido o si el espacio ha sido liberado previamente, el comportamiento de la función no está definido.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
int main()
```

```
{  char * palabra;
```

```
    palabra = (char *) malloc(15);
```

```
    strcpy(palabra, "Texto dinamico");
```

```
    printf("%s", palabra);
```

```
    free(palabra);
```

```
    return 0;
```

```
}
```

Reserva 15 bytes en  
la heap.  
Retorna un void\*



```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
```

```
int main()
```

```
{  char * palabra;
```

```
    palabra = (char *) malloc(15);
```

```
    strcpy(palabra, "Texto dinamico");
```

Convierte el **void\***  
retornado en un **char\***



## Sintaxis

```
char *strcpy(char *s1, const char *s2);
```

Copia la cadena apuntada por **s2** (incluyendo el carácter nulo) a la cadena apuntada por **s1**.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main()
{   char * palabra;

    palabra = (char *) malloc(15);

    strcpy(palabra, "Texto dinamico");

    printf("%s", palabra);

    free(palabra); ←
    return 0;
}
```

Libera la memoria  
reservada con **malloc**


# malloc() y sizeof()

- ▶ Es usual usar la función `sizeof()` para indicar el número de bytes a alojar.

- ▶ **Ejemplo**

```
int *ptr;
```

```
ptr = (int *) malloc( sizeof(int) );
```



Retorna un  
**void\***

- ▶ Usando **`sizeof( )`** el código se hace independiente del dispositivo (portabilidad).
- ▶ Note la conversión del puntero **`void`** retornado por `malloc` mediante **`(int *)`**



# Arreglos dinámicos

Cuándo se usan?

- ▶ Ejemplo

```
int *ptr;
```

```
ptr = (int *) malloc(100 * sizeof(int) );
```

- ▶ El ejemplo anterior indica la manera de generar un arreglo de enteros de forma dinámica.
- ▶ Para acceder al 4to. elemento del arreglo puede utilizar **ptr[3]** o bien **\*(ptr+3)**

# Ejemplo

VectorDinamico.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
```

```
{
```

```
    double *pd;
```

```
    int i;
```



Puntero para guardar la  
dirección inicial del vector

```
    pd = (double*) malloc(3*sizeof(double));
```

```
    pd[0] = 1.2;
```

```
    *(pd+1) = 5.6;
```

```
    pd[2] = 3.0;
```

```
    for (i=0; i<3; i++)
```

```
        printf("%d %g\n", i, pd[i]);
```

```
    free(pd);
```

```
    return(0);
```

```
}
```

# Ejemplo

VectorDinamico.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
```

```
{
    double *pd;
    int i;
```

```
pd = (double*) malloc(3*sizeof(double));
```



```
pd[0] = 1.2;
*(pd+1) = 5.6;
pd[2] = 3.0;
```

```
for (i=0; i<3; i++)
    printf("%d %g\n", i, pd[i]);
```

```
free(pd);
return(0);
```

```
}
```

Reservar memoria  
para los elementos  
del arreglo

# Ejemplo

VectorDinamico.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
```

```
{
```

```
    double *pd;
    int i;
```

```
    pd = (double*) malloc(3*sizeof(double));
```

```
    pd[0] = 1.2;
```

```
    *(pd+1) = 5.6;
```

```
    pd[2] = 3.0;
```



Se puede acceder al 2do.  
elemento del vector  
usando `p[1]` o `*(p+1)`

```
    for (i=0; i<3; i++)
        printf("%d %g\n", i, pd[i]);
```

```
    free(pd);
```

```
    return(0);
```

```
}
```

# Ejemplo

VectorDinamico.c

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    double *pd;
    int i;

    pd = (double*) malloc(3*sizeof(double));

    pd[0] = 1.2;
    *(pd+1) = 5.6;
    pd[2] = 3.0;

    for (i=0; i<3; i++)
        printf("%d %g\n", i, pd[i]);

    free(pd);
    return(0);
}
```



Liberar cuando no se necesita

# Función calloc( )

## ► Sintaxis

**void \***calloc(**size\_t** N, **size\_t** size);

- Reserva espacio para un arreglo de N objetos, cada cual tiene un tamaño **size** de bytes. El espacio es inicializado a cero todos los bits.
- Retorna un puntero del tipo **void \*** con la dirección inicial del bloque de memoria reservado.
- Si no puede reservar esa cantidad de memoria la función regresa un puntero nulo o **NULL**.

# Ejemplo

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    double *pd;
    int i;
```

```
    pd = (double*) calloc(3, sizeof(double));
```



```
    pd[0] = 1.2;
    pd[1] = 5.6;
    pd[2] = 3.0;
```

```
    for (i=0; i<3; i++)
        printf("%d %g\n", i, pd[i]);
```

```
    free(pd);
    return(0);
```

```
}
```

Reserva de memoria usando  
**calloc()**

Diferencias con **malloc( )**?

# Indique los errores ...

```
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
    double *pd;
    int i;


    pd[0] = 1.2;
    pd[1] = 5.6;

    for (i=0; i<3; i++){
        printf("%d  %g\n", i, *pd);
        pd++;
    }

    free(pd);
    return(0);
}
```



# Ejercicio 1

- ▶ Escriba una función que reciba un vector de enteros y la cantidad de elementos que contiene e imprima su contenido en pantalla.
  - ▶ Escriba un programa C que lea una cierta cantidad de enteros y los almacene en un vector. La cantidad de números a leer se ingresa antes que los datos.
  - ▶ Una vez ingresados, utilice la función anterior para imprimirlos en pantalla.
- 

# Función realloc( )

## ► Sintaxis

**void \*realloc(void \*ptr, size\_t size);**

- Cambia el tamaño del objeto apuntado por **ptr** al tamaño especificado por **size**.
- El contenido del objeto no cambiará hasta el menor de los tamaños nuevo y viejo.
- Si el tamaño nuevo es mayor, el valor de la porción nuevamente adjudicada del objeto es indeterminado.
- Retorna o bien un puntero nulo o bien un puntero posiblemente al espacio adjudicado mudado.


# Función `realloc()`

## ► Sintaxis

**`void *realloc(void *ptr, size_t size);`**

- Si **`ptr`** es un puntero nulo, *realloc* se comporta a igual que la función `malloc` para el tamaño especificado.
- De lo contrario, si **`ptr`** no es igual a un puntero previamente retornado por `calloc`, `malloc` o *realloc*, o si el espacio ha sido liberado usando la función `free` o *realloc*, el comportamiento no está definido.
- Si **`size`** es cero y **`ptr`** no es nulo, el objeto al que apunta es liberado.

# Ejercicio 2

- ▶ Se leen números correspondientes al peso de una persona y se los almacena en un vector.
  - ▶ El proceso de lectura termina al encontrar un valor incorrecto. El intervalo de valores válidos es  $(0, 250]$
  - ▶ Como se desconoce la cantidad de números a leer, se irán reservando de a 10 elementos por vez.
  - ▶ El tamaño del vector será modificado utilizando `realloc( )`
- 

# Ejemplo

- ▶ Escriba un programa que lea el nombre (20 caracteres) y la edad de 5 empleados de una empresa.
- ▶ Almacénelos en memoria utilizando un **puntero** a un área de memoria reservada con `malloc()`.
- ▶ Al finalizar debe imprimir los nombres de quienes tengan más de 21 años.

## EjStructPersona.c

```
#include <stdio.h>
int main()
{
    struct Persona{
        char nombre[30];
        int edad;
    };

    struct Persona X;

    strcpy(X.nombre, "Alguien");
    X.edad = 24;

    printf("%s tiene %d años",
           X.nombre, X.edad);
    return 0;
}
```

Utilice esta estructura para leer los datos de las 5 personas



```

#include <stdio.h>
#define N 5
int main()
{
    int i;
    struct Persona{
        char nombre[30];
        int edad;
    } *V;

    V=(struct Persona*) malloc (N*sizeof(struct Persona));

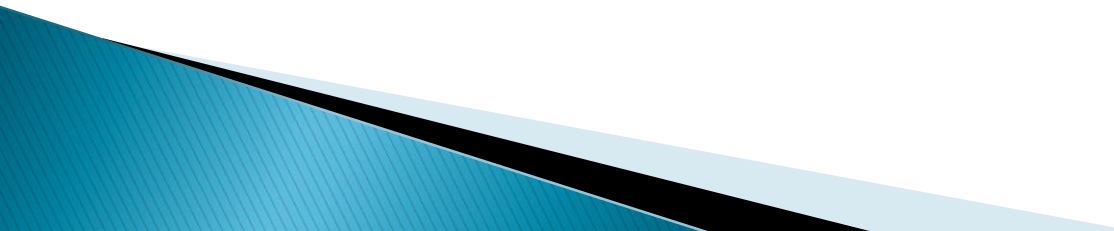
    for (i=0;i<N; i++){
        printf("%d - Nombre :",i);
        scanf("%s", V[i].nombre);
        printf("      edad      :");
        scanf("%d", &(V[i].edad));
    }
    for (i=0; i<N; i++)
        printf("%d - %s tiene %d años\n",i,
            V[i].nombre,V[i].edad);
    return 0;
}

```

También puede usarse  
`&( (V+i) ->edad)`

Pruebe usar `*(V+i).nombre`  
 ¿Por qué no compila?

# Ejercicio 1

- ▶ Escriba un programa C que lea de teclado una secuencia de números enteros terminada en 0 y la imprima en el orden inverso al que fue ingresada.
  - ▶ Utilice una lista para realizar esta tarea. Note que los elementos se agregan siempre al comienzo de la lista.
- 



# Declaración de la pila

```
#include <stdio.h>
int main()
{
    struct nodo {
        int valor;
        struct nodo * ptr;
    };

    struct nodo *Pila = NULL, *aux;
```

# Cargando los elem.en la pila

```
int nro;  
//leer nros. hasta que se ingrese cero  
printf("Ingrese un nro :");  
scanf("%d",&nro);  
  
while (nro != 0){  
    // agregar el nro a la pila  
    aux = (struct nodo *) malloc(sizeof(struct nodo));  
    aux->valor = nro;  
    (*aux).ptr = Pila;  
    Pila = aux;  
    // leer otro número  
    printf("Ingrese un nro :");  
    scanf("%d",&nro);  
}
```

# Visualización

```
// recorrer la lista visualizando
// los valores
aux=Pila;
while (aux != NULL) {
    printf("%d\n", (*aux).valor);
    aux = aux->ptr;
}

return 0;
}
```

# Ejercicio 2

- ▶ Escriba un programa C que lea de teclado una secuencia de números enteros terminada en 0 y la imprima en el mismo orden en que fue ingresada.
- ▶ Resuelva este problema de dos formas distintas utilizando
  - ▶ Una lista.
  - ▶ Un vector dinámico.