


# Punteros a funciones



Eva Lucrecia Gibaja Galindo  
Dpto. Informática y Análisis Numérico

# Concepto

- Hasta ahora se han analizado punteros a datos
    - Variables, estructuras, *arrays*
    - Paso de parámetros por referencia
  - La dirección de una función es la del segmento de código donde comienza el código de dicha función, esto es la dirección de memoria a la que se transfiere el control cuando se invoca
- 
- Es posible crear punteros que referencien a funciones (a su dirección de comienzo)
  - **Apuntan a código ejecutable en vez de direccionar datos**
  - En C, estos punteros a funciones pueden:
    - Asignarse
    - Ser pasados a funciones y ser devueltos
    - Utilizarse para llamar a la función a la que apuntan
    - Ser colocados en vectores
    - NO se les puede aplicar ninguna operación aritmética

# Utilidad de los punteros a funciones

- La utilidad de las funciones a punteros se ve cuando trabajamos con programas grandes. Permiten personalizar funciones
- Al principio necesitamos elegir una función entre un conjunto y después utilizarla muchas veces
  - Elegimos la función una vez (asignándosela a un puntero)
  - Llamamos a la función utilizando el puntero
- Ejemplos
  - Selección del método de ordenación o búsqueda.
  - Selección del criterio de ordenación (ascendente, descendente)
  - Selección de la clave de ordenación o búsqueda:
    - dni
    - nombre
    - código de cliente
    - edad

# Declaración de punteros a función

## Tipo retorno (\*punteroFuncion) (<lista parametros>)

- El formato indica al compilador que *punteroFuncion* es un puntero a una función que devuelve el tipo *Tipo retorno* y tiene una lista de parámetros
  - Puntero a una función de tipo *int* que recibe un *int*
    - `int (*pf) (int);`
  - Puntero a una función de tipo *double* que recibe un *int*
    - `double (*fp) (int);`
  - Puntero a una función de tipo *void* que recibe un vector de enteros y un entero
    - `void (*sort) (int *, int);`
  - Puntero a una función de tipo *int\** que recibe un vector de enteros y dos enteros
    - `int* (*search) (int, int *, int );`
  - Función de tipo *int\** que recibe un vector de enteros y un puntero a función
    - `int* func(int *, int*(*nombre)(int, int));`
  - Puntero a función de tipo *int\*\** que acepta como parámetro un puntero a función
    - `int** (*func)(int* (*nombre)(int*, int));`



# Asignación y llamada al puntero

- La sintaxis general para asignar un puntero a función:
  - *PunteroFuncion = &unaFuncion*
- La función asignada debe tener el mismo tipo de retorno y lista de parámetros que el puntero a función. Si no, error de compilación

```
double calculo (int *v, int n);
double (*qf) (int *, int);
int r[11]={3,5,6,7,1,7,3,34,5,11,44};
double x;
qf=&calculo; //asigna dirección de la función
x=(*qf)(r,11); //llamada a la función
```

# Ejemplo básico I

```
int main()
{
    int (*orden) (int, int); //Puntero a funcion
    int ord;
    int a=7, b=2;

    printf("Que orden vas a seguir: \n");
    printf("0 Ascendente\n");
    printf("1 Descendente\n");
    scanf("%d",&ord);

    if (ord==0)
        orden=&menor;
    else
        orden=&mayor;

    printf("El elemento que va primero es %d\n", (*orden)(a,b));
    printf("El mayor es %d\n", mayor(a,b));
}
```

# Ejemplo básico I

```
int mayor(int a, int b)
{
    if (a>b)
        return a;
    else
        return b;
}
```

```
int menor(int a, int b)
{
    if (a<b)
        return a;
    else
        return b;
}
```

## Ejemplo básico II

```
#include <stdio.h>
int suma(int a, int b)
{
    return a +b;
}

int resta(int a, int b)
{
    return a - b;
}

int producto(int a, int b)
{
    return a*b;
}
```



## Ejemplo básico II

```
main()
{
    int a=3, b=5, resultado;
```

*Tipo devuelto*

*Tipo de los parámetros*

```
    int (*punteroFuncion)(int, int);
    punteroFuncion = &suma; //Asignación
    resultado = (*punteroFuncion)(a, b); //Llamada
    printf("La suma es : %d\n", resultado);

    punteroFuncion = &resta;
    resultado = (*punteroFuncion)(a, b);
    printf("La resta es : %d\n", resultado);

    punteroFuncion = &producto;
    printf("El producto es : %d\n", (*punteroFuncion)(a, b) );
}
```

# Utilización como argumentos de una función

- Ejemplo: Función general que calcule la suma de algunos valores , es decir,  $f(1)+f(2)+ \dots + f(n)$  para cualquier función  $f$  de tipo *double* y con un argumento *int*.

```
void main()
{
    double (*pf)(int);

    //A) pasamos el puntero
    pf=&inversos;
    printf("Suma 2 inversos:%lf\n", funcSuma(2,pf));
    pf=&cuadrados;
    printf("Suma 3 cuadrados:%lf\n", funcSuma(3,pf));

    //B) pasamos la direccion de la funcion
    printf("Suma 4 inversos:%lf\n", funcSuma(4, &inversos));
    printf("Suma 5 cuadrados:%lf\n", funcSuma(5, &cuadrados));
}
```

# Utilización como argumentos de una función

```
//Definimos la función que recibe el puntero
double funcSuma(int n, double (*f)(int))
{
    double s=0;
    int i;
    for (i=1;i<=n;i++)
        s+=(*f)(i);
    return(s);
}
double inversos (int k)
{
    return 1.0/k;
}
double cuadrados (int k)
{
    return (double)k*k;
}
```

# orden.h

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

/*****
/*          CABECERAS DE LAS FUNCIONES DEL FICHERO ORDEN.C          */
*****/

int es_mayor(int a, int b);
int es_menor(int a, int b);
void burbuja(int* V, int izda, int dcha, int (*comparacion)(int,
    int));
void seleccion(int* V, int izda, int dcha, int (*comparacion)(int,
    int));
```

# orden.c

```
void burbuja(int* V, int izda, int dcha, int (*comparacion)(int,
int))
{
    int i, j, aux;
    for(i=izda+1; i<=dcha; i++)
    {
        for(j=dcha; j>=i; j--)
        {
            if( (*comparacion)(V[j-1],V[j]))
            {
                aux=V[j];
                V[j]=V[j-1];
                V[j-1]=aux;
            }
        }
    }
}
```



# orden.c

## *Funciones de comparación*

```
int es_mayor(int a, int b)
{
    if( a > b )
        return(1);
    else
        return(0);
}
int es_menor(int a, int b)
{
    if ( a < b )
        return(1);
    else
        return(0);
}
```

# main.c

```
#include <stdio.h>
#include <stdlib.h>
#include "orden.h"
#define MAX_ELE 15
int main()
{
    int* V;

    //Reserva de memoria para el vector
    ...
    //Inicialización del vector
    ...
    //Llamada al método de ordenación
    burbuja(V, 0 ,MAX_ELE-1, &es_mayor);
    //Liberamos la memoria reservada al vector
    ...
}
```

# Punteros void



Eva Lucrecia Gibaja Galindo  
Dpto. Informática y Análisis Numérico

# Punteros void en funciones

- Permiten pasar la dirección de tipos de datos diferentes en una llamada a función cuando no se conoce por anticipado que tipo de dato se pasa
- Cualquier puntero se puede convertir a tipo *void\** sin pérdida de información. Si el resultado se regresa al tipo de puntero original, éste es recuperado
- Con *void\**, es obligatorio hacer un *casting* para asignar un puntero de un tipo a un puntero de otro tipo

# Punteros void en funciones

```
#include <stdio.h>

void ver(void *,int tipo);

void main()
{
    char a='b';
    int x=3;
    double y=4.5;
    char *cad="hola";
    ver(&a,1);
    ver(&x,2);
    ver(&y,3);
    ver(cad,4);
}

void ver( void *p, int tipo)
{
    switch(tipo)
    {
        case 1: printf("%c\n",*(char *)p);
                break;
        case 2: printf("%d\n",*(int *)p);
                break;
        case 3: printf("%ld\n",*(double *)p);
                break;
        case 4: printf("%s\n", (char *)p);
                break;
    }
}
```



# Quicksort de stdlib.h

- stdlib.h incluye una función que implementa el algoritmo de ordenación *quicksort*

- Ordenación *in situ* del vector
- Dos elementos que son iguales pueden aparecer ordenados en cualquier orden, método no estable

```
void qsort(void *base, size_t nelem, size_t size,  
int (*cmp)(const void *e1, const void *e2))
```

- void \* **base**: el vector a ordenar
- size\_t **nelem**: número de elementos del vector
- size\_t **size**: tamaño de los elementos del vector
- int (\*cmp)(const void \*e1, const void \*e2): función de comparación de dos elementos. Devuelve:
  - Un valor negativo si *e1* va antes que *e2*
  - Cero si son iguales
  - Un valor positivo si *e1* va después que *e2*

# Quicksort de stdlib.h

*función de comparación*

```
int comparaEnteros(const void* e1 , const void* e2)
{
    //Los argumentos de la funcion de tienen que ser de tipo const void*
    int* a, *b;
    //Para trabajar, copiamos los argumentos en variables del tipo correcto
    a = (int*)e1;
    b = (int*)e2;
    if(*a<*b)
        return (-1);
    else if(*a==*b)
        return(0);
    else
        return(1);
}
```

*llamada*

```
qsort((int*)v, n, sizeof(int), &comparaEnteros);
```

# Búsqueda binaria de stdlib.h

- stdlib.h incluye una función para hacer búsqueda binaria
  - Presupone que los elementos están en orden ascendente de acuerdo con la función de comparación
  - Devuelve la dirección del elemento con la clave buscada

```
void *bsearch(const void *key, const void *base, size_t nelem,
              size_t size, int (*cmp)(const void *ck, const void *ce))
```

- const void \* **key**: clave de búsqueda
- const void \* **base**: vector
- size\_t **nelem**: número de elementos del vector
- size\_t **size**: tamaño de los elementos del vector
- int (\*cmp)(const void \*e1, const void \*e2): función de comparación de dos elementos. Devuelve:
  - Un valor negativo si *e1* va antes que *e2*
  - Cero si son iguales
  - Un valor positivo si *e1* va después que *e2*

# Búsqueda binaria de stdlib.h

```
int comparaEstructuras(const void* e1 , const void* e2)
{ struct miEstructura *a, *b;

  a = (struct miEstructura *)e1;
  b = (struct miEstructura*)e2;

  if(a->campo<b->campo)
    return (-1);
  else if(a->campo==b->campo)
    return(0);
  else
    return(1);
}
```

*función de comparación*

*llamada*

```
struct miEstructura estrAux={9};
posicionEstr = (struct miEstructura*)bsearch((struct
miEstructura*)&estrAux, (struct miEstructura*)vEstructuras, MAX_ELEM,
sizeof(struct miEstructura), &comparaEstructuras);
```



# Punteros void en funciones

```
void escribeInt (void *p)
{
    int *entero = (int *)p;
    printf ("\n%d", *entero);
}
```

```
void escribeVector (void *p, int n, void (*funcptr)(void *),
int size)
{
    int tipo, i;

    for (i=0; i<n; i++)
    {
        (*funcptr)(p+size*i);
    }
}
```



# Punteros void en funciones

```
int comparaInt(void* a, void* b)
{
    int* a_aux, *b_aux;

    a_aux = (int*) a;
    b_aux = (int*) b;

    if (*a_aux > *b_aux)
        return(-1);
    else
        if (*a_aux==*b_aux)
            return (0);
        else
            return(1);
}
```

# Punteros void en funciones

```
void * busquedaLineal(void* V, void * clave, int n, int
(*comparacion)(void*, void*), int size)
{
    int i;
    void* res=NULL;

    for (i=0; i<n; i++)
    {
        if ((*comparacion)(clave, V+i*size)==0)
        {
            res = V+i*size;
            return res;
        }
    }

    return NULL;
}
```

No se puede pasar directamente V[i]  
porque no se sabe cuanto ocupa cada  
elemento

# Punteros void en funciones

```
void burbuja2(void* V,int n,int(*comparacion)(void*, void*),int size)
{
    int i, j;
    void* aux;
    aux = malloc(size);
    for(i=1; i<n; i++)
    {
        for(j=n-1; j>=i; j--)
        {
            if( (*comparacion)(V+(j-1)*size,V+j*size)<0)
            {
                memcpy(aux, V+(j-1)*size, size);
                memcpy(V+(j-1)*size, V+j*size, size);
                memcpy(V+j*size, aux, size);
            }
        }
    }
    free(aux);
}
```

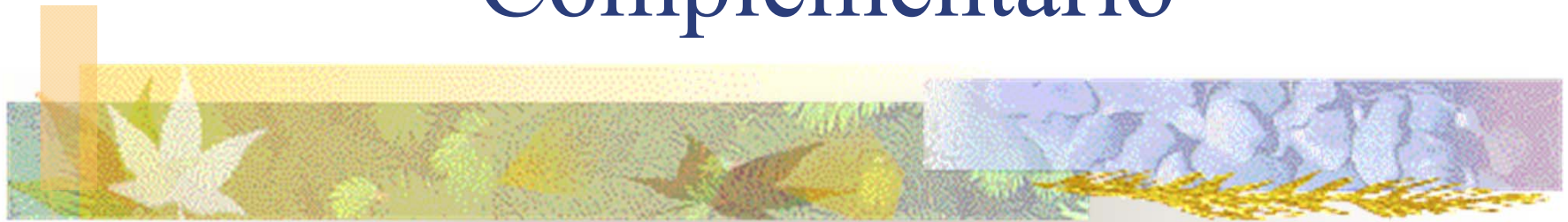
# Punteros void en funciones

```
void main(int argc, char ** argv)
{
    int V[]={1, 9, 4, 6, 2, 5, 3, 7, 8};
    int a=1, b=2, clave;
    int* res;

    escribeVector (V, 9, &escribeInt, sizeof(int));
    clave=6;
    res = busquedaLineal((int*)V, (int*)&clave, 9, &comparaInt,
sizeof(int));
    printf("\nEl elemento %d esta en posicion: %d valor: %d res:%p",
clave, res-V, *res, res);

    burbuja2((int*)V, 9, &comparaInt, sizeof(int));
    escribeVector (V, 9, &escribeInt, sizeof(int));
}
```

# Punteros a funciones. Complementario



Eva Lucrecia Gibaja Galindo  
Dpto. Informática y Análisis Numérico



# Arrays de punteros a funciones

- Ciertas aplicaciones requieren disponer de numerosas funciones, basadas en el cumplimiento de ciertas condiciones.
- Utilizar un array de punteros a función
  - Se selecciona una función de la lista y se llama
- Sintaxis
- ***TipoR(\*punteroFunc[longArray])(lista parm.)***
  - TipoR. Tipo de retorno de las funciones
  - PunteroFunc. Nombre del array de punteros
  - LongArray. N° elementos del vector
  - Lista parm. Parámetros de las funciones

# Arrays de punteros a funciones

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

```
int suma(int a, int b)
{
    return a +b;
}
```

```
int resta(int a, int b)
{
    return a - b;
}
```

```
int producto(int a, int b)
{
    return a*b;
}
```

# Arrays de punteros a funciones

```
main()
{
    int a=3, b=5, resultado;
    int (*punteroFuncion[3])(int, int) = {&suma, &resta};

    resultado = (*punteroFuncion[0])(a, b);
    printf("La suma es : %d\n", resultado);

    resultado = (*punteroFuncion)[1](a, b);
    printf("La resta es : %d\n", resultado);

    punteroFuncion[2]=&producto;
    resultado = (*punteroFuncion[2])(a, b);
    printf("El producto es : %d\n", resultado);
}
```

# Vectores dinámicos de punteros a funciones

```
int mayor(int a, int b);
int menor(int a, int b);
typedef int (*t_orden) (int, int);
int main()
{
    t_orden * V;
    int a=7, b=2;
    int ord;

    printf("Que orden vas a seguir: \n");
    printf("0 Ascendente\n");
    printf("1 Descendente\n");
    scanf("%d",&ord);

    V=(t_orden *) malloc (7*sizeof(t_orden));
    V[0]=&menor;
    V[1]=&mayor;

    if (ord==0)
        printf("El elemento que va primero es %d\n", (*V[0])(a,b));
    else
        printf("El elemento que va primero es %d\n", (*V[1])(a,b));
}
```

# Devolver punteros a funciones

```
typedef float (*t_function) (float, float);
int main()
{
    float a=3,b=5;
    float (*pfun)(float, float);
    pfun=escogerOperacion2();
    printf("%f\n", (*pfun)(a,b));
}
t_function escogerOperacion2()
{
    char opcion;
    do
    {
        puts("Selecciona operacion (*,/):");
        scanf("%c",&opcion);
        getchar();
    }while (opcion!='*' && opcion!=' /');
    switch (opcion)
    {
        case '*':
            return(&multiplicar);
            break;
        case '/':
            return(&dividir);
            break;
    }
}
```



# Punteros a funciones por referencia

```
int main()
{
    float a=3,b=5;
    float (*pfun)(float, float);
    escogerOperacion(&pfun);
    printf("%f\n", (*pfun)(a,b));
}

void escogerOperacion(float (**pun) (float, float))
{
    char opcion;
    do
    {
        puts("Selecciona operacion (*, /):");
        scanf("%c",&opcion);
        getchar();
    }while (opcion!='*' && opcion!=' /');

    switch (opcion)
    {
        case '*':
            *pun=&multiplicar;
            break;
        case '/':
            *pun=&dividir;
            break;
    };
}
```

# Notaciones equivalentes

```
#include <stdio.h>
int suma(int, int);
int funcion(int, int, int (*ptrf)(int, int));
main(){
    int (*ptrf)(int, int);
    int (*ptrV[1])(int, int);
    int res;
    ptrf=&suma;
    //ptrf=suma;

    res = (*ptrf)(1,2);
    // res = ptrf(1,2);
    printf("\nRes: %d", res);

    //res=funcion(3,4, &suma);
    //res=funcion(3,4, suma);
    //res=funcion(3,4, ptrf);
    //res=funcion(3,4, *ptrf);
    printf("\nRes: %d", res);
```

# Notaciones equivalentes

```

//ptrV[0]=&suma;
ptrV[0]=suma;
//res=ptrV[0](5,6);
res=(*ptrV[0])(5,6);
printf("\nRes: %d", res);

}

int suma(int a, int b)
{
    return a+b;
}

int funcion(int a, int b, int (*ptrf)(int, int ))
{
    //return(*ptrf)(a,b);
    return(ptrf(a,b));
}

```