APUNTADORES

(Introducción)

Las estructuras de datos hasta ahora vistas se almacenan estáticamente en la memoria física del ordenador.

– El espacio de memoria se reserva con anticipación y no cambia durante la ejecución del programa*.

– Esto permite una comprobación de tipos en tiempo de compilación.

• Inconvenientes de la configuración estática:

– Su rigidez, ya que estas estructuras no pueden crecer o cambiar durante la ejecución del programa.

* Esto no implica que la cantidad de memoria de ejecución de un programa sea constante, ya que dependerá del numero de subprogramas recursivos invocados por el programa.

APUNTADORES

(Introducción)

La definición y manipulación de estos objetos se realiza en «C» mediante los **punteros o apuntadores** (variables cuyo contenido son posiciones de memoria).

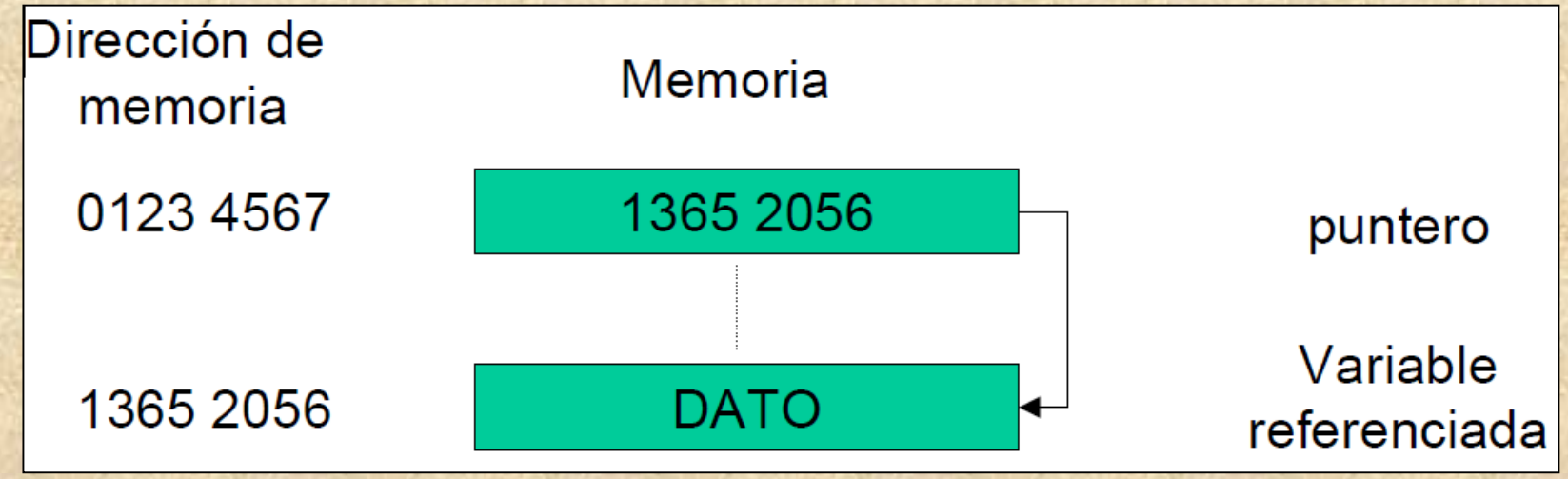
- Ventaja frente a las estructuras estática:
  - La **flexibilidad** que poseen las estructuras dinámicas en cuanto a las formas que pueden adoptar: árboles, listas, redes, etc...
- Inconvenientes:
  - **Aliasing**: Doble direccionamiento sobre una misma variable lo que implica efectos laterales.
  - **Gestión de la memoria**: Su uso requiere una especial atención de la memoria disponible así como de la que ya no queremos utilizar.

APUNTADORES

(Conceptos)

Un puntero es una variable que contiene la dirección de memoria donde se encuentra almacenado un dato.

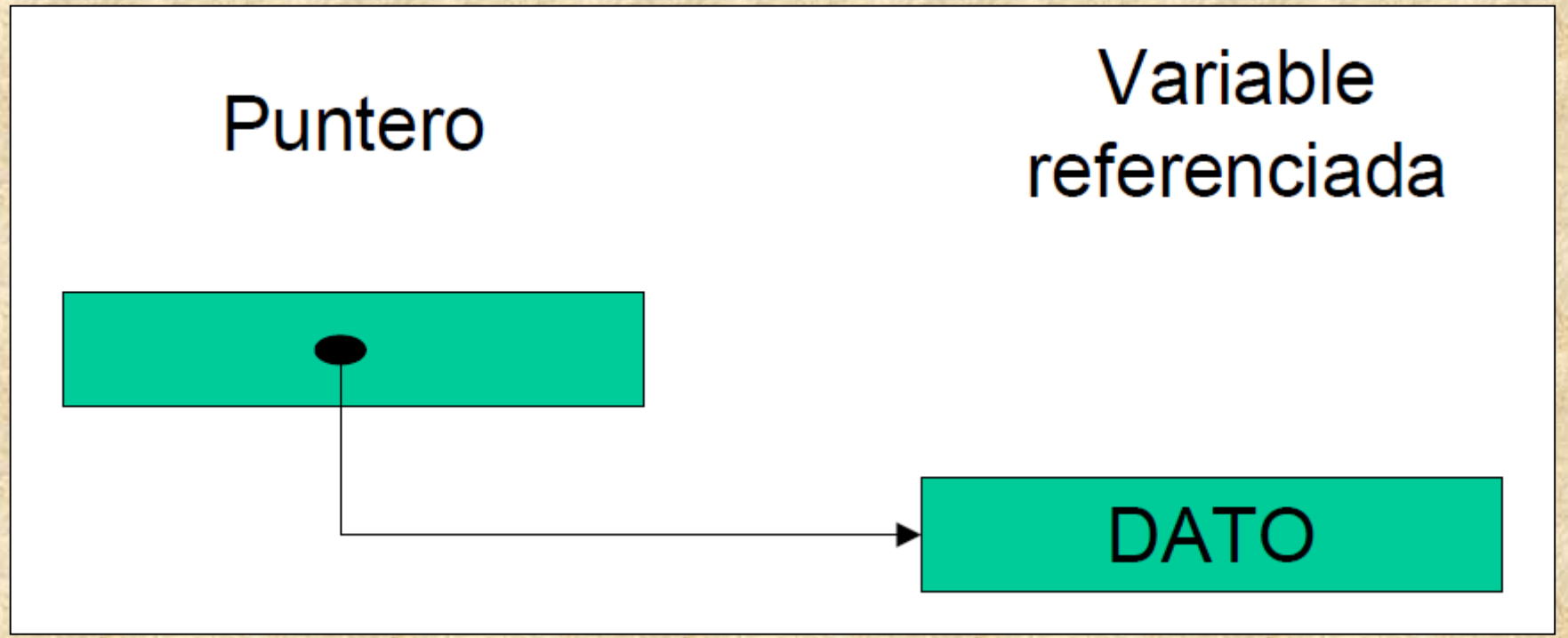
- **Una variable referenciada o dato apuntado** es el dato cuya posición en memoria está contenida en un determinado puntero (variable dinámica).



APUNTADORES

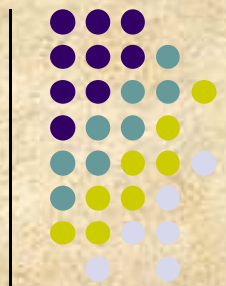
(Gráficamente)

La representación grafica tradicional de un puntero es la siguiente:



APUNTADORES

(Definición y Declaración)



Dentro de «C» hay 2 tipos de variables puntero:

1. Las definidas por el usuario
2. Los punteros de propósito general (void).

Caso 1

Para poder usar una variable puntero es necesario:

– Definir el tipo de dato (o estructura) al que se apunta.

– Declarar las variables punteros que sean necesarias.

En este caso un puntero sólo puede señalar a objetos de un mismo tipo, el establecido en la declaración.

Caso 2

Existe un tipo de dato genérico que permite almacenar una dirección de memoria (un puntero) que se llama «void».

Esto permite poder apuntar a un tipo de dato NO declarado de antemano.

APUNTADORES

(Ejemplos)

```
struct puntero {
    int id;
    char* desc;
};

int main()  {
            struct puntero *p;
            p = malloc(sizeof(struct puntero)); // Crear el puntero
            p->id = 1;  //uso de la variable puntero
            p->desc = malloc(100);
            strcpy(p->desc, "Descripcion de 1");
            printf("ID del Puntero: %d \n", p->id);
            printf("Desc. del Puntero: %s \n", p->desc);
            return 0;
}
```

APUNTADORES

(Tener en cuenta)

- Una variable de tipo puntero ocupa una cantidad de memoria fija, independiente del tipo de dato al que apunta (dirección de memoria).

- Un dato referenciado, como el del ejemplo, no posee existencia inicial, o lo que es lo mismo no existe inicialmente espacio reservado en memoria para el.

- Para poder emplear variables dinámicas es necesario emplear un tipo de dato que permita referenciar nuevas posiciones de memoria que no han sido declaradas a priori y que se van a crear y destruir en tiempo de ejecución.

- Estas variables son los punteros que en “C” es un tipo de dato simple.

APUNTADORES

(Instrucciones)

• Las variables dinámicas son por definición aquellas que se crean cuando se necesitan y se destruyen cuando ya han cumplido con su cometido.

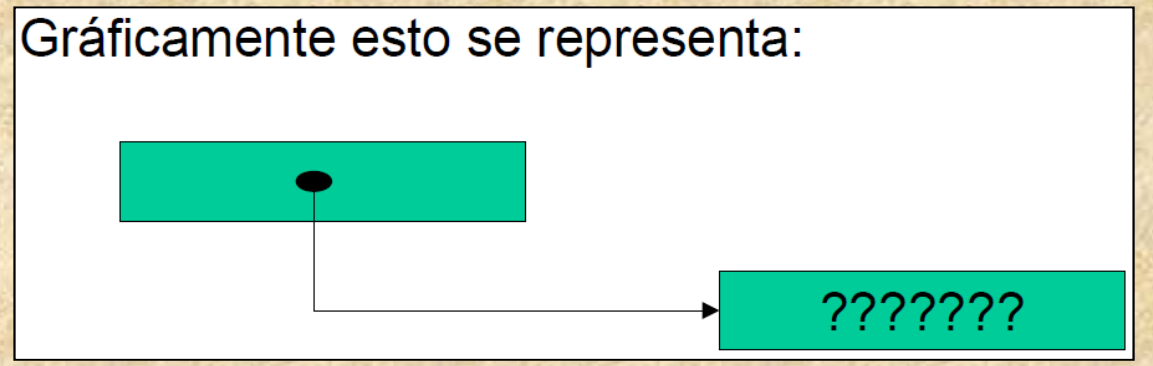
• En “C” la creación y destrucción de variables dinámicas se realiza mediante los siguientes procedimientos:

– **malloc (variable_de_tipo_puntero)** **// Crea el puntero**
– **free (variable_de_tipo_puntero)** **// Destruye el puntero**

Puntero = malloc (sizeof(tipo de dato))

– Reserva la memoria necesaria para un dato del tipo apropiado.
– Coloca la dirección de memoria de esta nueva variable en «puntero».

Gráficamente esto se representa:



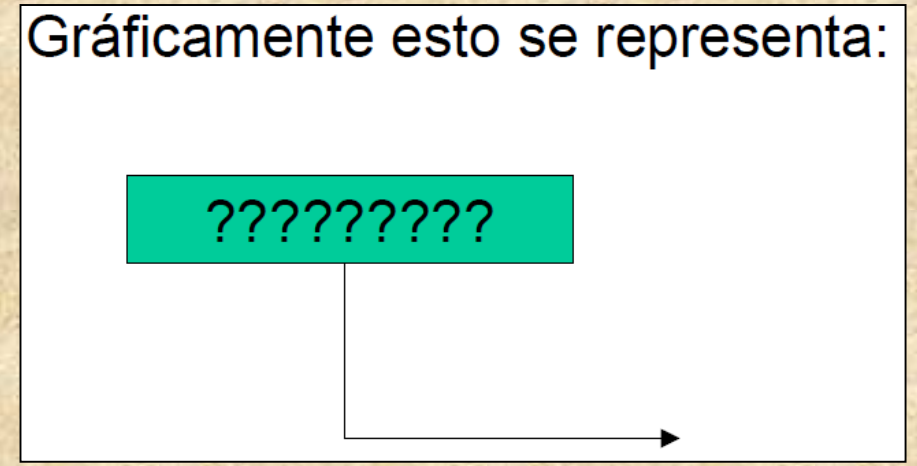
APUNTADORES

(Instrucciones)

free (puntero)

– Libera la memoria asociada a la variable referida (dejándola libre para otros fines).

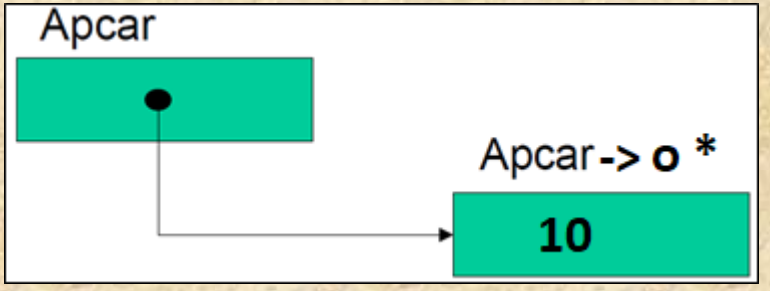
– Deja indefinido el valor del puntero.



APUNTADORES

(Operaciones)

- El contenido de la variable referenciada por el puntero se denota:
  puntero-> (o <*variable> para un tipo de dato simple)
- Las operaciones permitidas para esta nueva variables son:
  - Asignación
  - Lectura
  - Escritura
  - Todas las operaciones legales que se puedan realizar con dicho tipo.
- Ejemplo:

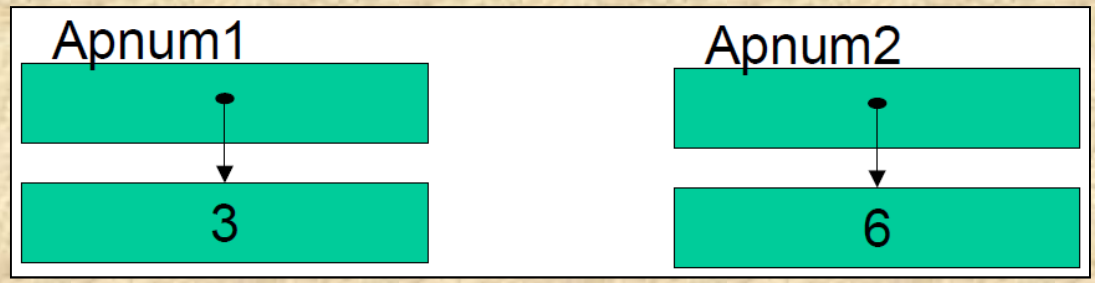



```
int *apcar;
   // Crear el puntero
   apcar = malloc(sizeof(int));
   *apcar = 10;
   printf("Contenido del puntero <apcar>: %d \n", *apcar);
```

APUNTADORES

(Operaciones - Ejemplos)

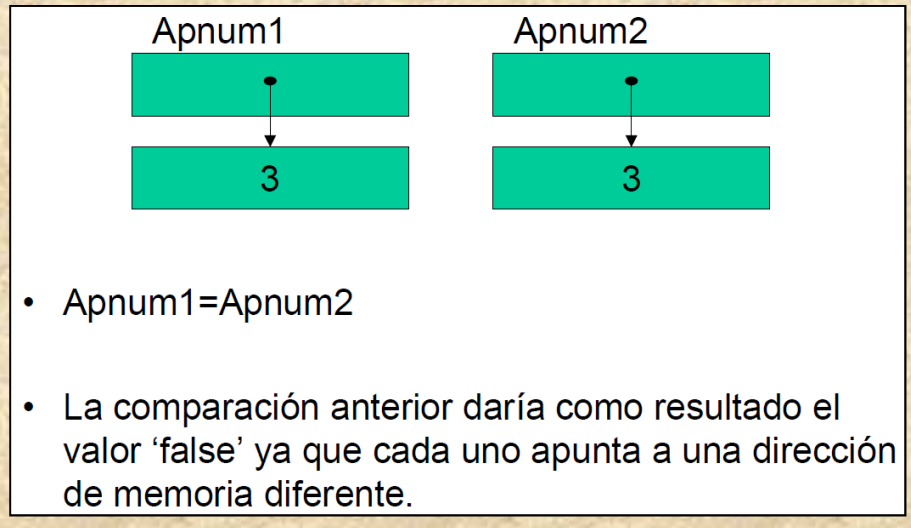
```
int*  Apnum1;
int* Apnum2;

Apnum1 = malloc(sizeof(int));
Apnum2 = malloc(sizeof(int));

*Apnum1 = 2;
*Apnum2 = 4;
*Apnum2 = *Apnum1 + *Apnum2;
*Apnum1 = (*Apnum2 / 2);
```

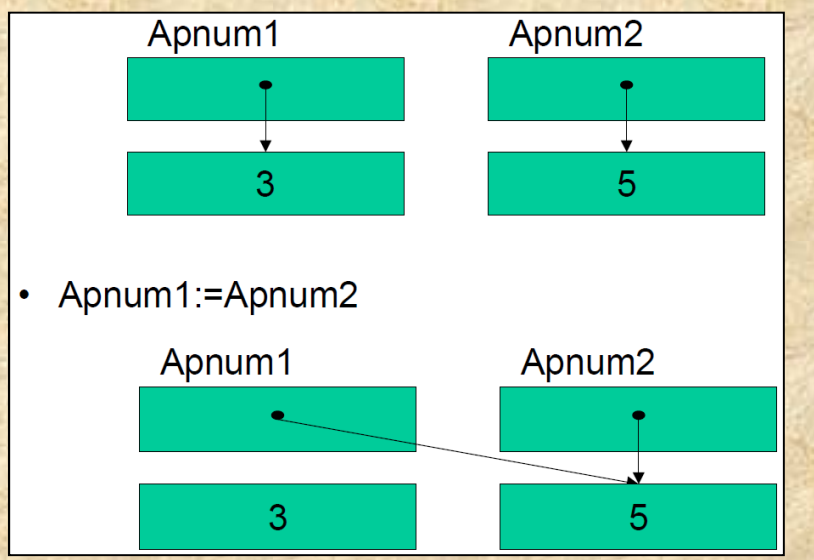


APUNTADORES

(Operaciones Básicas)

- Las únicas operaciones válidas son:
  - **La comparación** (se comparan las direcciones, no los contenidos de los datos apuntados).
    Apnum1==Apnum2
  - **La asignación** (se asignan las direcciones entre sí, no los contenidos de los datos apuntados).
    Apnum1=Apnum2



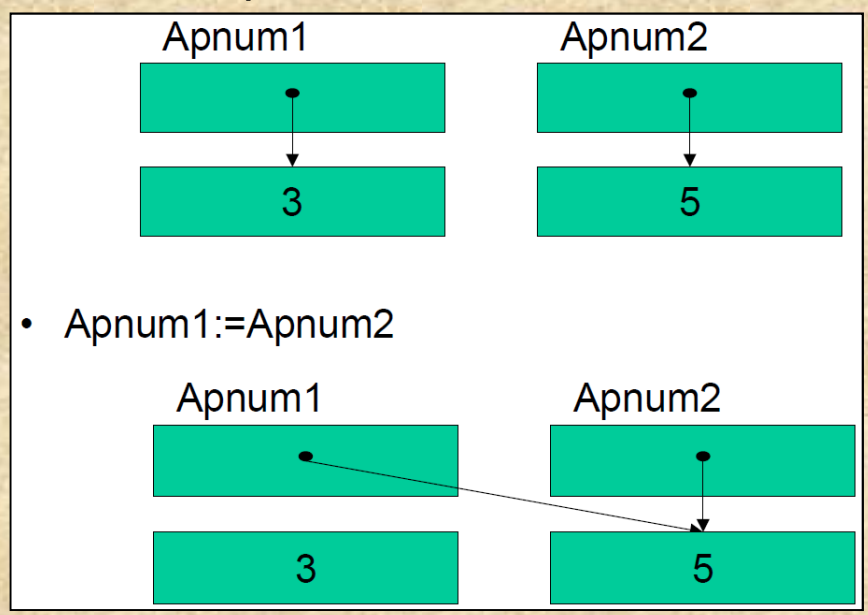


APUNTADORES

(Asignación)

• Los cambios efectuados sobre Apnum1 afectan a la variable Apnum2 (son indistintas) (Aliasing).

• El espacio de memoria reservado inicialmente por el puntero Apnum1 sigue situado en memoria. Una adecuada **gestión de la memoria** hubiera exigido la liberación de ese espacio antes de efectuar la asignación.



APUNTADORES

(La Constante NULL)

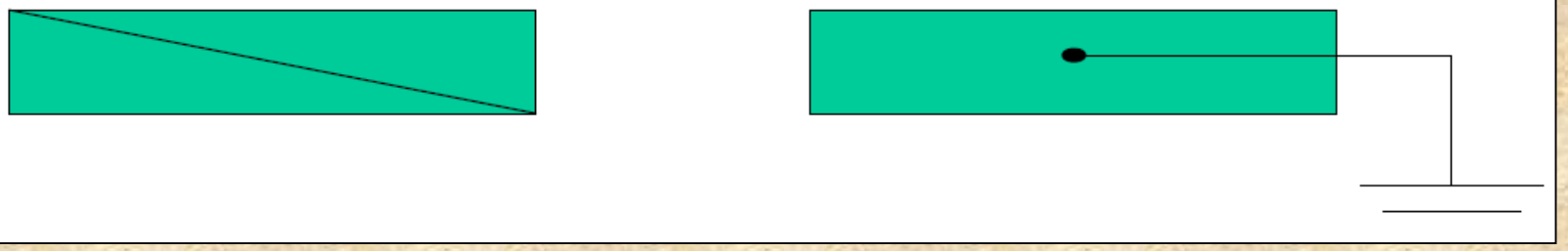
• Un modo alternativo de asignar un valor a un puntero es indicar que no apunta a ningún dato. Esto se lleva a cabo mediante la constante predefinida “NULL”

• “**NULL**” es independiente del tipo del dato apuntado por lo que puede ser utilizado por cualquier puntero.

– Apcar = NULL; // Asignación de NULL

– Apcar == NULL;// Comparación contra NULL

Representación gráfica:



APUNTADORES

(Usos de los punteros)

- Caso 1: Asignación de datos compuestos en un solo paso

  Esta aplicación es de utilidad cuando se manejan variables o estructuras de datos de gran tamaño (por el elevado coste que supone el realizar la copia de todas sus componentes).

  – Asignación de variables de gran tamaño

  – Ordenación de vectores con elementos de gran tamaño.

```
struct tFicha {;
        char* nombre;
        char* string;
} // tFicha

struct tFicha pers1;
struct tFicha pers2;

Pers1 = pers2; //operación muy costosa en espacio y tiempo
```

```
struct tFicha {;
        char* nombre;
        char* string;
} // tFicha

struct tFicha *pers1;
struct tFicha *pers2;

Pers1 = pers2; // asignación instantánea
```

APUNTADORES

(Usos de los punteros)

- Caso 2: Definición de funciones que devuelven datos compuestos.
  Como en el caso anterior se cambia el objeto por el puntero que lo apunta.
  Ejemplo: Definir un subprograma que a partir de un punto del plano, un ángulo y una distancia calcule la posición de un nuevo punto.

```
struct      tPunto {
    double x;
    double y;
}
```

```
struct tPunto *Destino (struct tPunto orig, double ang, double dist) {

        struct tPunto *pPun;

        pPun = malloc(sizeof(struct tPunto));

        pPun->x = orig.x + (dist * cos(ang));
        pPun->y = orig.y + (dist * sin(ang));

        return pPun;
};
```

APUNTADORES

(Usos de los punteros en recusividad)

Son de gran importancia para implementar estructuras de datos dinámicas de índole netamente recursiva como es el caso de Arboles, Grafos, etc.

Las estructuras desde la definición de las mismas son recursivas.

Los procesos o funciones que manejan esas estructuras también son recursivas.

Los apuntadores nos permiten manejar de forma dinámica el uso de la memoria para almacenar la información de estas estructuras, es decir solo se ocupa el espacio que se necesita y puede crecer acorde a la complejidad del problema a tratar.

APUNTADORES

(El tipo de dato «void pointer»)

Los punteros de propósito general «void» son usados para guardar en tiempo de ejecución estructuras sin importar su contenido.

Son de gran flexibilidad porque permiten asignarle una dirección de memoria que la misma puede contener cualquier cosa dentro, desde un dato simple como un «integer» hasta una estructura compleja como «un registro» de registros por ejemplo.

El tipo de dato «void» necesita ser creado en ejecución asociado el tipo de dato que en ese momento será almacenado en el.

Ejemplo:

```
void*  ptr;                                  // puntero genérico
struct      tPunto *origen;                  // Defino un puntero especifico

ptr = malloc(sizeof(struct tPunto));         // reservo espacio para el puntero
ptr = (struct tPunto*)origen;                // asigno el puntero especifico al genérico
```