ESTRUCTURAS DE DATOS DINÁMICAS
- PUNTEROS -

Algoritmos y Estructuras de Datos II

Lic. Ana María Company

HASTA AHORA…

- Las estructuras de datos estudiadas hasta ahora se almacenan estáticamente en la memoria física de la computadora.

- **Asignación estática de memoria:** Cuando se ejecuta un subprograma, se destina memoria para cada variable global del programa y ese espacio de memoria permanecerá reservado durante toda su ejecución, se usen o no esas variables.

INCONVENIENTES DE LAS ESTRUCTURAS DE DATOS ESTÁTICAS

1. No pueden crecer o decrecer durante la ejecución de un programa.
2. La representación de ciertas construcciones (ej listas) usando las estructuras conocidas (ej. arrays) se realizan situando elementos consecutivos en componentes contiguas, por lo tanto las operaciones de inserción de un elemento nuevo o eliminación de uno ya existente requieren el desplazamiento de todos los posteriores para cubrir el vacío producido, o para abrir espacio para el nuevo.

- Estos dos aspectos, **tamaño** y **disposición rígidos**, se superan con las llamadas **Estructuras De Datos Dinámicas**.

ESTRUCTURAS DE DATOS DINÁMICAS

- Son estructuras extremadamente flexibles que "crecen a medida que se ejecuta un programa".
- Una estructura de datos dinámica está formada por una colección de elementos llamados **nodos**.
  - Se puede **ampliar** y **contraer** durante la ejecución del programa.

    → Ej: Lista de pasajeros de una línea aérea.
- La definición y manipulación de estos objetos se efectúa mediante un mecanismo conocido como puntero o apuntador, que permite al programador referirse directamente a la memoria.

PUNTEROS

- Cuando una variable se declara, se asocian tres atributos fundamentales:
  - nombre
  - tipo
  - dirección en memoria

- Cada variable que se declara tiene una dirección asociada con ella.

- Todas las variables conocidas hasta ahora contienen valores de datos, sin embargo las variables **punteros** contienen valores que son **direcciones de memoria** donde se almacenan datos.

PUNTEROS

- Un puntero es una variable que sirve para señalar la posición de la memoria en que se encuentra otro dato almacenando como valor la dirección de ese dato.

- Tiene dos componentes:
  - la **dirección de memoria a la que apunta**
    → contenido del puntero

  - el **elemento referido**
    → contenido de la celda de memoria cuya dirección está almacenada en el puntero)

IMPORTANTE: la variable puntero y la variable a la que apunta son dos variables distintas

→ sus valores son también distintos

PUNTEROS - EJEMPLOS DE LA VIDA COTIDIANA

- Enviar una carta por correo
  - Su información se entrega basada en la dirección de la carta

- Realizar una llamada telefónica
  - Se debe marcar el número de teléfono de la persona que se quiere contactar

La dirección de correo y el número telefónico tienen común que:

- ✓ Ambos indican dónde encontrar algo
- ✓ Son punteros a edificios y teléfonos respectivamente

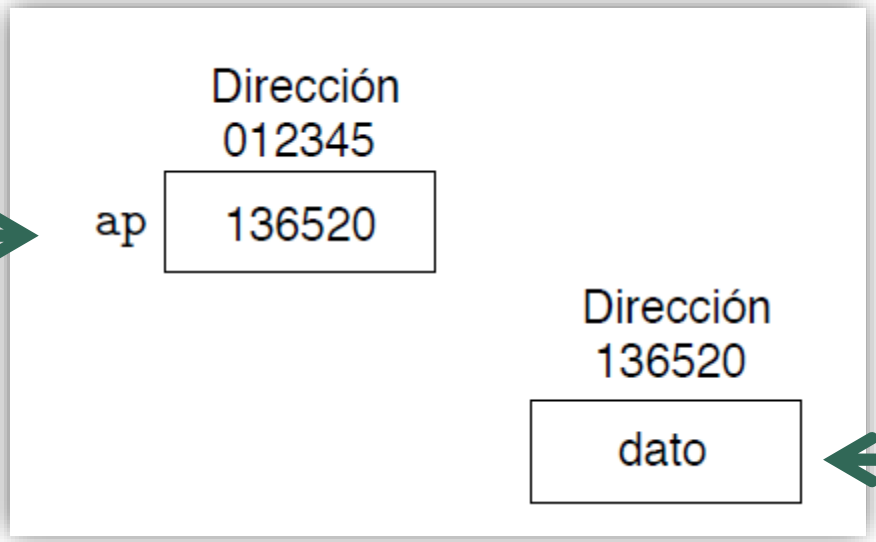
PUNTEROS

- Un **puntero** también indica **dónde encontrar algo**
  - dónde encontrar los datos que están asociados con una variable
  - un puntero es la dirección de una variable

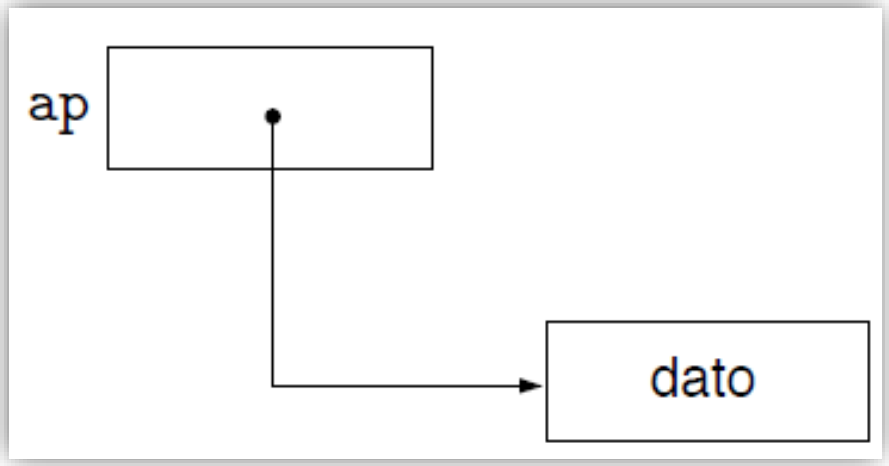
- **Resumiendo:**
  - un **puntero** es una variable como cualquier otra
  - una variable puntero contiene una dirección que apunta a otra posición en memoria
  - en esa posición se almacenan los datos a los que apunta el puntero
  - un puntero apunta a una variable de memoria

GRÁFICAMENTE

Variable de tipo puntero



Variable a la que apunta



DECLARACIÓN DE PUNTEROS

- Como todas las variables, las variables punteros deben declararse antes de ser utilizadas.

- La declaración de una variable puntero debe:
  - indicar al compilador el tipo de dato al que apunta el puntero
  - para esto se hace preceder a su nombre con un asterisco (*)

→mediante el siguiente formato:

<tipo de dato apuntado> *<identificador de puntero>

EJEMPLOS DE PUNTEROS

```
int *apuntaInt;        // Puntero a un tipo de dato int
char *apuntaChar;      // Puntero a un tipo de dato char
float *apuntaFloat;    // Puntero a un tipo de dato float
```

- Un operador * en una declaración indica que la variable declarada almacenará una dirección de un tipo de dato especificado.
  - La variable *apuntaInt* almacenará la dirección de un entero, la variable *apuntaChar* almacenará la dirección de un dato tipo char, etc.

OPERADOR DE DIRECCIÓN

- **Operador de dirección (&)**
  - Da la dirección de un objeto, de modo que la proposición:

    → p = &c;

  - asigna la dirección de c a la variable p
  - se dice que p "apunta a" c.
  - El operador & sólo se aplica a objetos que están en memoria: variables y elementos de arreglos.
  - No puede aplicarse a expresiones, constantes o variables tipo registro.

OPERADOR DE INDIRECCIÓN

- **Operador de indirección o desreferencia (*)**
  - Cuando se aplica a un apuntador (puntero), da acceso al objeto al que señala el apuntador.
  - Supóngase que x e y son enteros e ip es un apuntador a int. La siguiente secuencia muestra cómo declarar un apuntador y cómo emplear & y *:

```
int x = 1, y = 2, z[10];
int *ip;   // ip es un apuntador a int
ip = &x;  // ip ahora apunta a x
y = *ip;   // y es ahora 1
*ip = 0;  // x es ahora 0
ip = &z[0]; // ip ahora apunta a z[0]
```

INICIALIZACIÓN

- AI igual que otras variables, C no inicializa los punteros cuando se declaran y es preciso inicializarlos antes de su uso.
- La inicialización de un puntero proporciona a ese puntero la dirección del dato correspondiente.
- Después de la inicialización, se puede utilizar el puntero para referenciar los datos direccionados.
- Para asignar una dirección de memoria a un puntero se utiliza el operador de referencia **&**.
  - Por ejemplo → &valor
    significa "la dirección de *valor*"

INICIALIZACIÓN - MÉTODO ESTÁTICO

1. Asignar memoria (estáticamente) definiendo una variable y a continuación hacer que el puntero apunte al valor de la variable.

```
int edad;              // define una variable edad
int *apuntaEdad;       // define un puntero a un entero
apuntaEdad = &edad;    // asigna la dirección de edad a apuntaEdad
```

2. Asignar un valor a la dirección de memoria

```
*apuntaEdad = 25;
```

El asterisco delante de la variable puntero indica <el contenido de> la memoria apuntada por el puntero y será del tipo dado

INICIALIZACIÓN - MÉTODO ESTÁTICO

- La asignación de memoria utilizada para almacenar el valor es fija y no puede desaparecer.
- Una vez que la variable se define
  - el compilador establece suficiente memoria para almacenar un valor del tipo de dato dado.
- La memoria permanece reservada para esta variable y no se puede utilizar para otra cosa durante la ejecución del programa.

INICIALIZACIÓN - MÉTODO ESTÁTICO

- No se puede liberar la memoria reservada para una variable.
- El puntero a esa variable se puede cambiar, pero permanecerá la cantidad de memoria reservada.
- El operador **&** devuelve la dirección de la variable a la cual se aplica.

Es un error asignar un valor a un contenido de una variable puntero si previamente no se ha inicializado con la dirección de una variable, o se le ha asignado dinámicamente memoria.

INICIALIZACIÓN - MÉTODO DINÁMICO

- Mediante técnicas de asignación dinámica de memoria.
- Este método utiliza las funciones de asignación de memoria malloc ( ) , calloc ( ) , realloc ( ) y free ( )

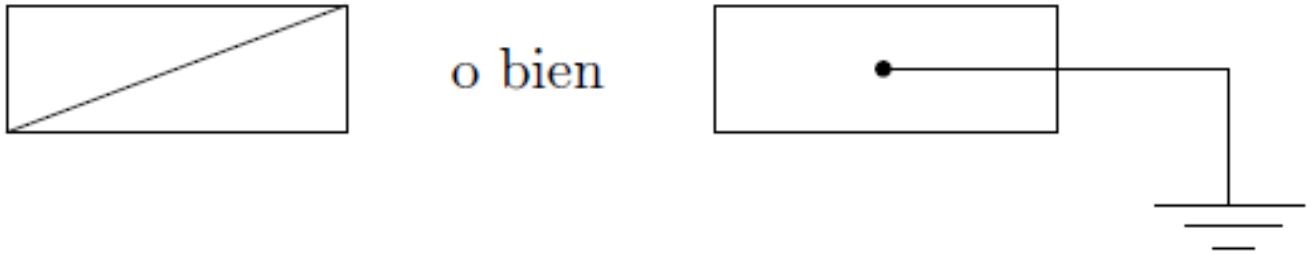
INICIALIZACIÓN

- Un puntero inicializado adecuadamente apunta a alguna posición específica de la memoria.
- Un puntero no inicializado, como cualquier variable, tiene un valor aleatorio hasta que se inicializa el puntero.
- Por esto, será preciso asegurarse que las variables puntero utilicen direcciones de memoria válidas.

PUNTERO NULL

- Un **puntero nulo** no apunta a ninguna parte -dato válido- en particular
  - un **puntero nulo** no direcciona ningún dato válido en memoria.
  - Un **puntero nulo** se utiliza para proporcionar a un programa un medio de conocer cuando una variable puntero no direcciona a un dato válido.

Gráficamente



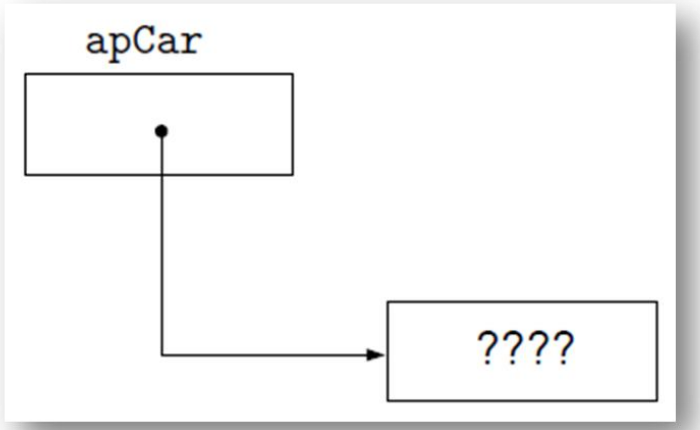
ASIGNACIÓN DINÁMICA DE MEMORIA

- El espacio de la variable asignada dinámicamente se crea durante la ejecución del programa, al contrario que en el caso de una variable local cuyo espacio se asigna en tiempo de compilación.
- El programa puede crear o destruir la asignación dinámica en cualquier momento durante la ejecución.
- Se puede determinar la cantidad de memoria necesaria en el momento en que se haga la asignación.

FUNCIÓN MALLOC()

- Asigna un bloque de memoria que es el número de bytes pasados como argumento.
- Devuelve un puntero, que es la dirección del bloque asignado de memoria.
  - El puntero se utiliza para referenciar el bloque de memoria y devuelve un puntero del tipo void*.

Reserva la memoria para un dato del tipo apropiado. Coloca la dirección de esta nueva variable en el puntero.



FUNCIÓN MALLOC - SINTAXIS

- La forma de llamar a la función malloc ( ) es:

  puntero = malloc(tamaño en bytes);

- Generalmente se hará una conversión al tipo del puntero:

  tipo *puntero;
  puntero =(tipo *)malloc(tamaño en bytes);

  - puntero es el nombre de la variable puntero a la que se asigna la dirección del objeto dato

- La función malloc( ) está declarada en el archivo de cabecera
  - stdlib.h

FUNCIÓN MALLOC

- El operador unario **sizeof** se utiliza con mucha frecuencia en las funciones de asignación de memoria.
  - Se aplica a un tipo de dato (o una variable), el valor resultante es el número de bytes que ocupa.

- Si se quiere reservar memoria para un buffer de 10 enteros:

```
int *r;
r = (int*) malloc(10*sizeof(int) );
```

- Al llamar a la función malloc ( ) puede ocurrir que no haya memoria disponible, en ese caso malloc ( ) devuelve **NULL**.

FUNCIÓN MALLOC - Ejemplo

- El siguiente código utiliza malloc( ) para asignar espacio para un valor entero:

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

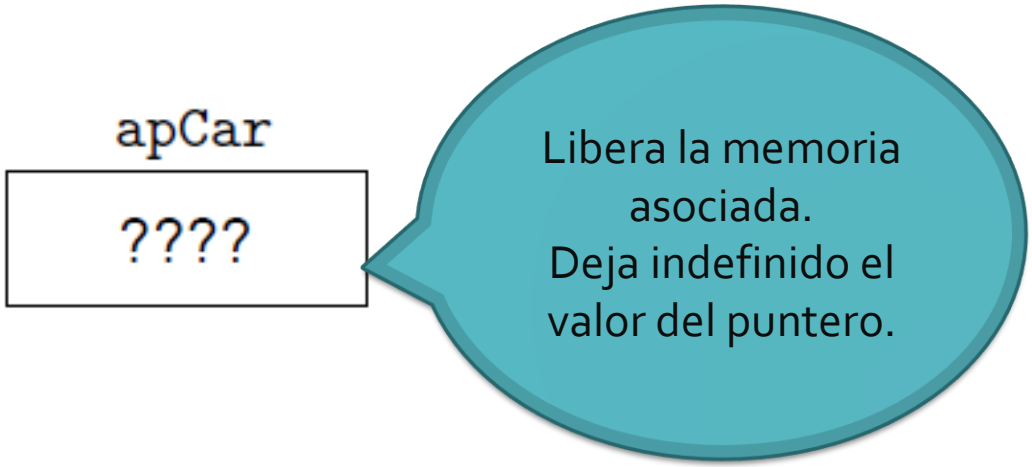
- La llamada a malloc( ) asigna espacio para un int y almacena la dirección de la asignación en *pEnt*.
- *pEnt* apunta ahora a la posición en el almacén libre/montículo donde se establece la memoria.

FUNCIÓN FREE()

- Permite liberar el espacio de memoria y dejarlo disponible para otros usos, una vez que se ha terminado de utilizar un bloque de memoria previamente asignado por malloc().

- El bloque de memoria suprimido se devuelve al espacio de almacenamiento libre, de modo que habrá más memoria disponible para asignar otros bloques de memoria.

Gráficamente



FUNCIÓN FREE - SINTAXIS

- Para un puntero:

  tipo *puntero;

- El formato de la llamada es:

  free (puntero) ;

- La variable puntero puede apuntar a una dirección de memoria de cualquier tipo.

ASIGNACIÓN DE MEMORIA - CONSIDERACIONES

- La función **malloc()** devuelve el tipo void*, por lo que será necesario hacer una conversión al tipo del puntero.
- Permite asignar memoria para cualquier tipo de dato especificado (int, float, struct, array…)
- La función malloc() tiene como argumento el número de bytes a reservar.
- El operador **sizeof** permite calcular el tamaño de un tipo de objeto para el que está asignando memoria.
- La función malloc() retorna **NULL** si no ha podido reservar la memoria requerida.

ASIGNACIÓN DE MEMORIA - CONSIDERACIONES

- La asignación dinámica de memoria permite utilizar tanta memoria como se necesite.
  - Se puede asignar espacio a una variable en el almacenamiento libre cuando se necesite y se libera la memoria cuando se desee.

- En C utilizaremos las función malloc() y free() para asignar y liberar memoria respectivamente.

- Cuando se termina de utilizar un bloque de memoria, se puede liberar con la función **free()**.

  - La memoria libre se devuelve al almacenamiento libre, de modo que quedará más memoria disponible.

RESUMIENDO

- Un **puntero** es una variable que contiene la dirección de una posición en memoria.
- Para declarar un puntero se sitúa un asterisco ***** entre el tipo de dato y el nombre de la variable → int *p;
- Para obtener el valor almacenado en la dirección utilizada por el puntero, se utiliza el operador de indirección **(*)**.
- El valor de *p* es una dirección de memoria y el valor de **p* es el dato entero almacenado en esa dirección de memoria.
- Para obtener la dirección de una variable existente, se utiliza el operador de dirección **(&)**.

RESUMIENDO

- Se debe declarar un puntero antes de su uso.
- Para inicializar un puntero que no apunta a nada, se utiliza la constante **NULL**.
- Una variable dinámica sólo se creará cuando sea necesario
  - lo que ocasiona la correspondiente ocupación de memoria y,
  - se destruirá una vez haya cumplido con su objetivo, con la consiguiente liberación de la misma.

BIBLIOGRAFÍA

- Mark Allen Weiss - Estructuras de Datos y Algoritmos - Florida International University - Año: 1995 - Editorial: Addison-Wesley Iberoamericana .
- Joyanes Aguilar, Luis - Programación en Pascal - 4ª Edición - Año: 2006 - Editorial: McGraw-Hill/Interamericana de España, S.A.U.
- Cristóbal Pareja Flores, Manuel Ojeda Aciego, Ángel Andeyro Quesada, Carlos Rossi Jiménez - Algoritmos y Programación en Pascal.
- Joyanes Aguilar, Luis - Fundamentos de la Programación. Algoritmos, Estructuras de Datos y Objetos - 3ª Edición - Editorial: McGraw-Hill
- Luis Joyanes Aguilar, Ignacio Zahonero Martinez - Programación en C. Metodología, algoritmos y estructura de datos - Editorial: McGraw-Hill