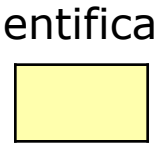
Estructuras fundamentales

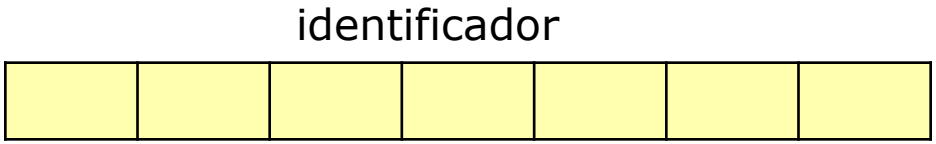
- Los datos a procesar por una computadora pueden clasificarse en: simples y estructurados. Los tipos de datos simples o primitivos significan que no están compuestos de otras estructuras de datos. Los tipos de datos compuestos están construidos basados en tipos de datos primitivos.

Estructuras fundamentales

- La principal característica de los datos simples es que ocupan sólo una localidad de memoria, por lo tanto una variable simple hace referencia a un único valor a la vez.

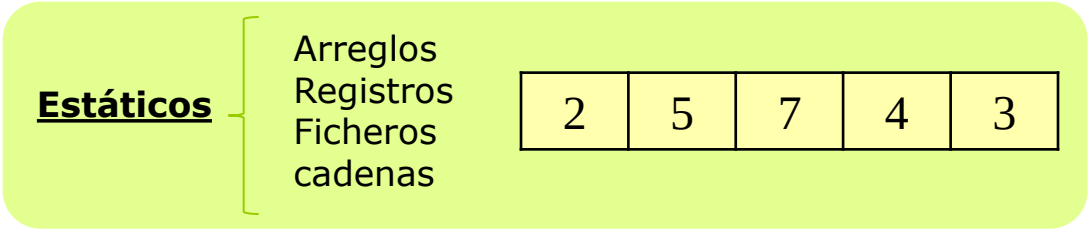


- Los datos estructurados se caracterizan por el hecho de que con un nombre se hace referencia a un grupo de localidades de memoria. Es decir, un dato estructurado tiene varios componentes. Cada uno de los componentes puede ser a su vez un dato simple o estructurado.

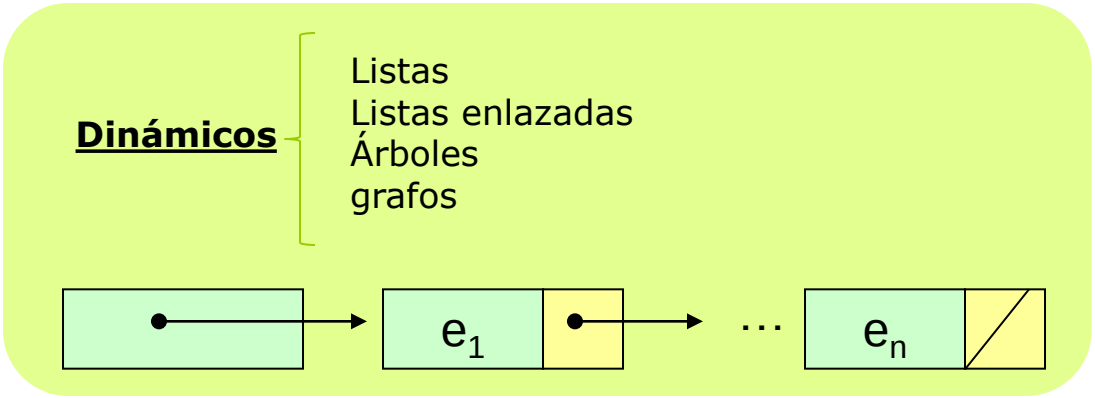


Estructuras de datos

- Los tipos de datos simples pueden ser organizados en estructuras de datos:
  - **Estáticas**.- El tamaño ocupado de memoria se define antes de que el programa se ejecute y no puede modificarse dicho tamaño durante la ejecución del programa.



  - **Dinámicas.-** Este tipo de datos pueden adquirir memoria a medida que se necesitan, y liberarlas cuando ya no se requieran.



Estructuras de datos

- Los arreglos y registros se denominan estructuras estáticas debido a que durante la compilación se les asigna espacio de memoria, y éste permanece inalterable a lo largo de la ejecución del programa. Este tipo de estructuras corresponden a las listas simples.

- Existen también otro tipo de listas que introducen el concepto de Estructuras de datos dinámicas, cuya principal ventaja consiste en que pueden adquirir memoria a medida que se necesita, y liberarla cuando ya no se requiera. El dinamismo de estas estructuras soluciona el problema de decidir cuál es la cantidad óptima de memoria que debe reservarse para un problema dado.

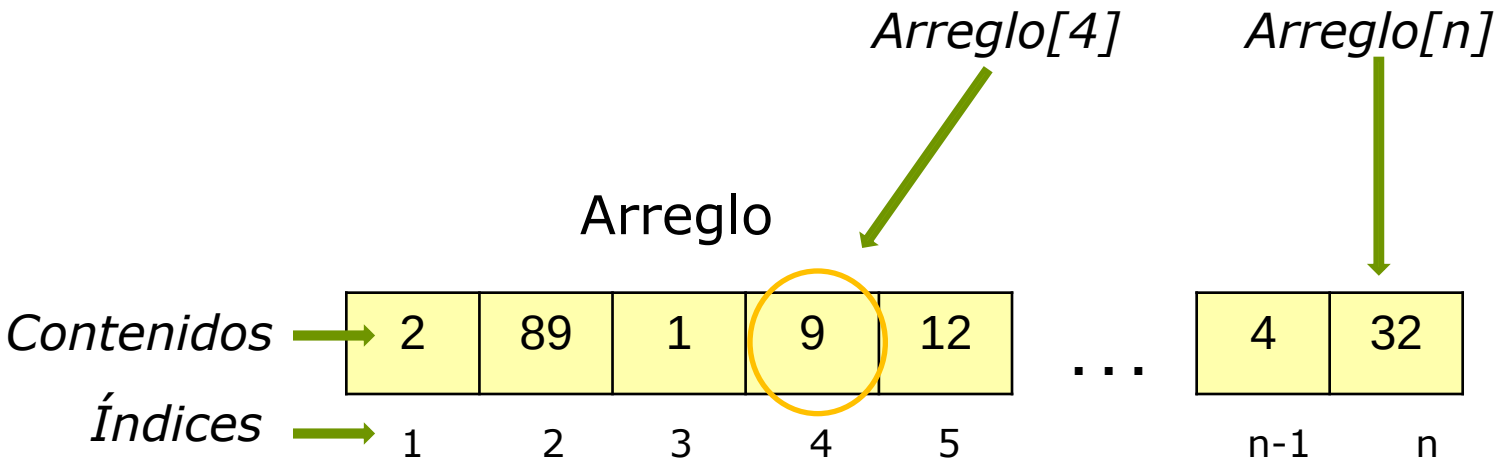
Arreglos

- Con frecuencia se presentan problemas cuya solución no resulta fácil de implementar (a veces imposible) si se utilizan datos simples. Los arreglos son tipos de datos estructurados que ayudan a resolver dichos problemas.

- Un arreglo se define como una colección finita (número fijo de elementos), homogénea (todos los elementos son del mismo tipo) y ordenada (se puede determinar cuál es el primer elemento, el segundo, … y el enésimo elemento) de elementos.

Arreglos

- Para hacer referencia a un elemento de un arreglo se utiliza: 1) el nombre del arreglo, y 2) el índice del elemento.



Operaciones con arreglos

- Las operaciones que se realizan con arreglos son:
  - Lectura / Escritura
  - Asignación
  - Actualización
    - Inserción
    - Eliminación
    - Modificación
  - Ordenación
  - Búsqueda

Operaciones con arreglos

- Inserción en arreglos desordenados:
  - Para insertar un elemento Y en un arreglo V desordenado debe verificarse que exista espacio. Si se cumple esta condición, entonces se asignará a la posición n+1 el nuevo elemento.

```
insertarDesordenado(V,N,Y)
{El algoritmo inserta un elemento en un arreglo desordenado. V es un
arreglo de 100 elementos. N es el número actual de elementos. Y es
el valor a insertar}

Si N<100 entonces
   Hacer N=N+1 y V[N] = Y
Sino
   Escribir "El elemento Y no puede insertarse"
Finsi
```

Operaciones con arreglos

- **Eliminación en arreglos desordenados:**
  - Para eliminar un elemento X de un arreglo V desordenado debe verificarse que el arreglo no esté varío y que X se encuentre en el arreglo. Si se cumplen las condiciones entonces se procederá a recorrer todos los elementos que están a su derecha una posición a la izquierda, decrementando finalmente el número de componentes del arreglo.

```
eliminaDesordenado(V,N,X)
{El algoritmo elimina un elemento en un arreglo desordenado. V es un arreglo de
100 elementos. N es el número actual de elementos. X es el valor a eliminar}
{I y K son variables de tipo entero. BAND es una variable de tipo Booleano}

Si N>=1 entonces
   Hacer I=1 y BAND=FALSO
   Mientras(I<=N) y (BAND=FALSO)
      Si V[I]=X entonces
          Hacer BAND=VERDADERO y N=N-1
          Repetir con K desde I hasta N
             Hacer V[K]=V[K+1]
          FinRepetir
      sino
          Hacer I=I+1
      Finsi
   FinMientras
   Si BAND=FALSO entonces
      Escribir "El elemento X no está en el arreglo"
   Finsi
Sino
   Escribir "El arreglo está vacío"
Finsi
```

Operaciones con arreglos

- **Modificación en arreglos desordenados:**
  - Para modificar un elemento X de un arreglo V desordenado debe verificarse que el arreglo no esté vacío y que X se encuentre en el arreglo. Si se cumplen estas condiciones entonces se procederá a su actualización.

```
modificarDesordenado(V,N,X,Y)
{El algoritmo modifica un elemento en un arreglo desordenado. V es un arreglo de
100 elementos. N es el número actual de elementos. X es el elemento a modificar
por el elemento Y}
{I es una variable de tipo entero. BAND es una variable de tipo booleano}

Si N>=1 entonces
   Hacer I=I+1 y BAND=FALSO
   Mientras (I<=N) y (BAND=FALSO)
      Si V[I]=X entonces
          Hacer V[I]=Y y BAND=VERDADERO
      sino
          Hacer I=I+1
      Finsi
   FinMientras
   Si BAND=FALSO entonces
      Escribir "El elemento X no está en el arreglo"
   sino
      Escribir "El arreglo está vacío"
   Finsi
Finsi
```

Operaciones con arreglos

- **Inserción en arreglos ordenados:**
  - Para insertar un elemento X en un arreglo V ordenado debe verificarse que exista espacio. Luego tendrá que encontrarse la posición en la que debería estar el nuevo valor para no alterar el orden del arreglo. Una vez detectada la posición, se procederá a recorrer todos los elementos desde la misma hasta la N-ésima posición, un lugar a la derecha. Finalmente se asignará el valor de X en la posición encontrada. Antes de presentar el algoritmo de inserción, se definirá una función auxiliar que se utilizará tanto en el proceso de inserción como en el de eliminación.

```
busca(V,N,X,POS)
{El algoritmo busca un elemento X en el arreglo ordenado V de N
elementos. POS es la posición de X en V, o la posición en la que X
debería estar}
{I es una variable de tipo entero}

Hacer I=1
Mientras (I<=N) y (V[I]<X)
   Hacer I=I+1
Fin Mientras
Si (I>N) o (V[I]>X) entonces
   Hacer POS=-I
sino
   Hacer POS=I
Finsi
```

Operaciones con arreglos

```
insertarOrdenado(V,N,Y)
{El algoritmo inserta un elemento Y en el arreglo ordenado V de N
elementos. POS e I son variables enteras}

Si N<100 entonces
   Llamar al algoritmo busca(V,N,Y,POS)
   Si POS>0 entonces {El elemento ya pertenece al arreglo}
      Escribir "El elemento ya existe"
   sino
      Hacer N=N+1 y POS =POS*(-1)
      Repetir con I desde N hasta POS+1
         Hacer V[I]=V[I-1]
      FinRepetir
      Hacer V[POS]=Y
   Finsi
Sino
    Escribir "No hay espacio en el arreglo"
Finsi
```

Operaciones con arreglos

- **Eliminación en arreglos ordenados:**
  - Para eliminar un elemento X de un arreglo ordenado V debe verificarse que el arreglo no esté vacío. Si se cumple esta condición, entonces tendrá que buscarse la posición del elemento a eliminar. Si el resultado de la función es un valor positivo, quiere decir que el elemento se encuentra en el arreglo y por lo tanto puede ser eliminado; en otro caso no se puede ejecutar la eliminación.

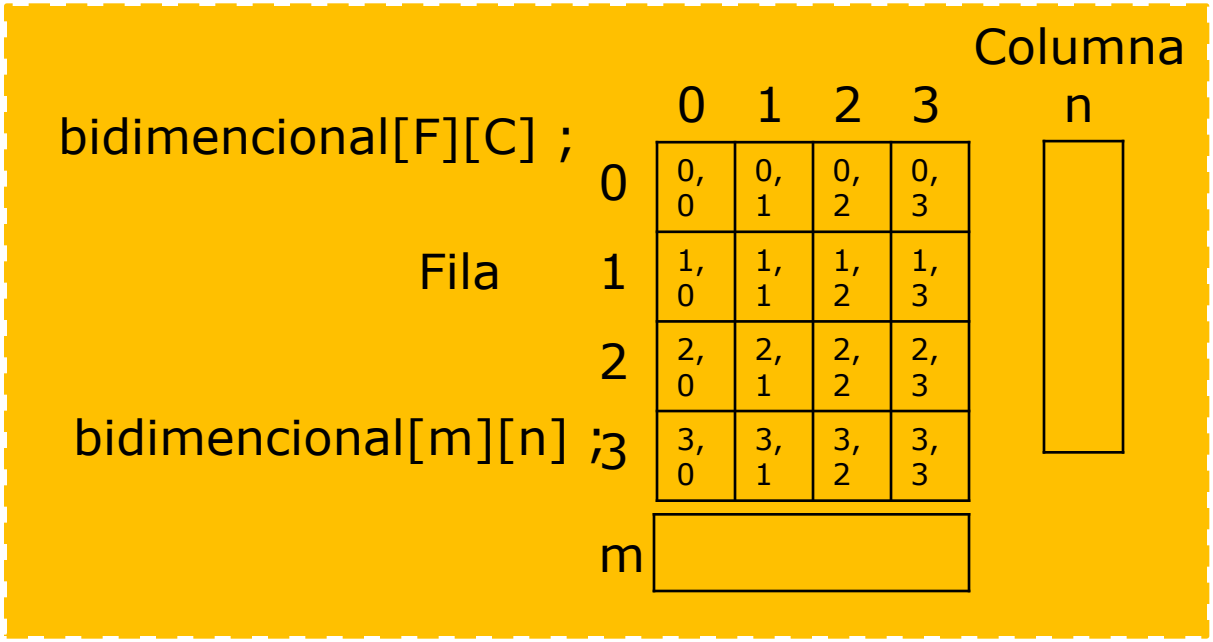
```
eliminarOrdenado(V,N,X)
{El algoritmo elimina un elemento X en el arreglo ordenado V de N
elementos. POS e I son variables enteras}

Si N>0 entonces
   Llamar al algoritmo busca(V,N,X,POS)
   Si POS<0 entonces {No se puede eliminar si no existe}
      Escribir "El elemento no existe"
   sino
      Hacer N=N-1
      Repetir con I desde POS hasta N
         Hacer V[I]=V[I+1]
      FinRepetir
   Finsi
Sino
    Escribir "El arreglo está vacío"
Finsi
```

¿Cómo sería la modificación en un arreglo ordenado?

Arreglos multidimensionales

- Los arreglos multidimensionales son aquellos que tienen más de una dimensión y, en consecuencia, más de un índice. Los arreglos más usuales son los de dos dimensiones, conocidos también como tablas o matrices. Sin embargo, es posible crear arreglos de tantas dimensiones como requieran sus aplicaciones, esto es, tres, cuatro o más dimensiones.



Arreglos multidimensionales

```
#include <stdio.h>
int main() {
    float matriz[4][4];
    int fila, col;
    for (fila = 0; fila < 4; fila++){
        for (col = 0; col < 4; col++){
            printf ("\n Introduzca un valor en la Posicion
    [%d][%d]:",fila,col) ;
            scanf ("%f",&matriz[fila][col]);
        }
    }
    for (fila = 0; fila < 4; fila++){
        for (col = 0; col < 1; col++){
            printf ("\n %.1f | %.1f | %.1f | %.1f\n",matriz[fila][col],
                                    matriz[fila][col+1],
    matriz[fila][col+2], matriz[fila][col+3]) ;
        }
    }
     return 0;
}
```

Ejemplo

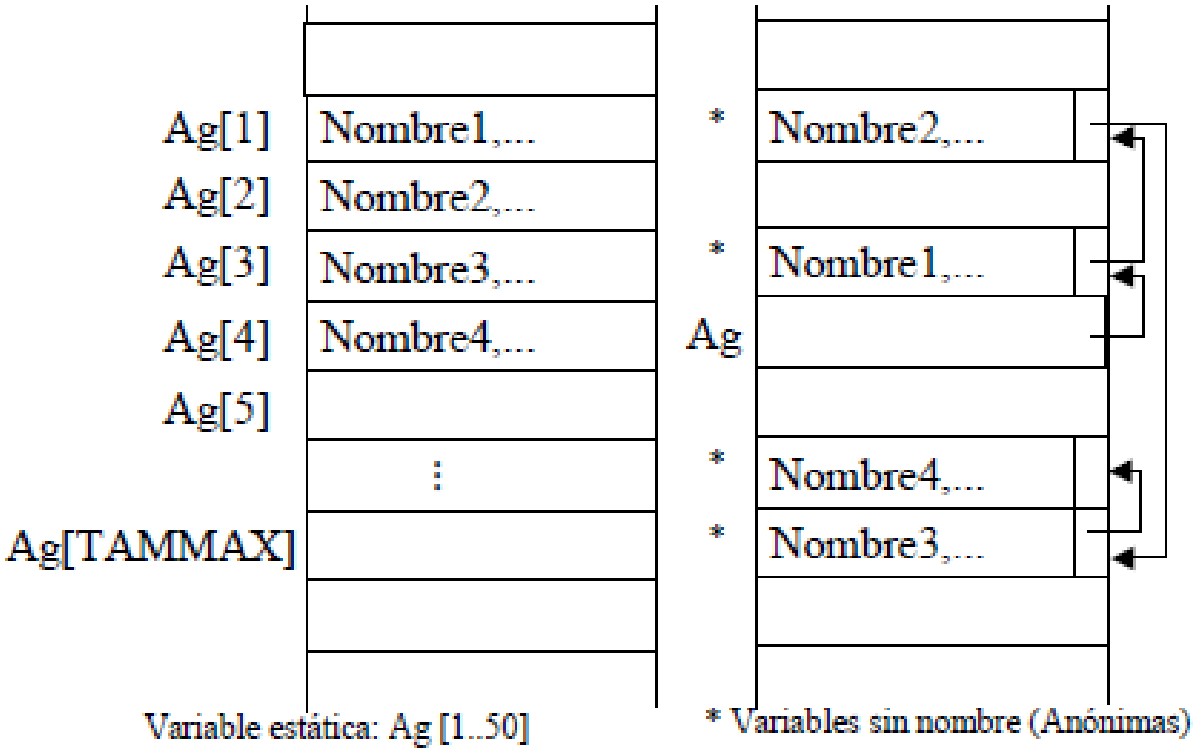
Determina si la matriz es simétrica

```
int simetrica(int a[][N],int n){
    int i,j;
    int es_simetrica;
    for (es_simetrica=1, i=0; i<n-1 && es_simetrica; i++)
          for (j=i+1; j<n && es_simetrica; j++)
                  if(a[i][j] != a[j][i])
                    es_simetrica=0;
   return es_simetrica;
}
```

Apuntadores

PUNTEROS

- ¿Qué sucede si a priori no conocemos la cantidad de espacio de almacenamiento que vamos a necesitar?

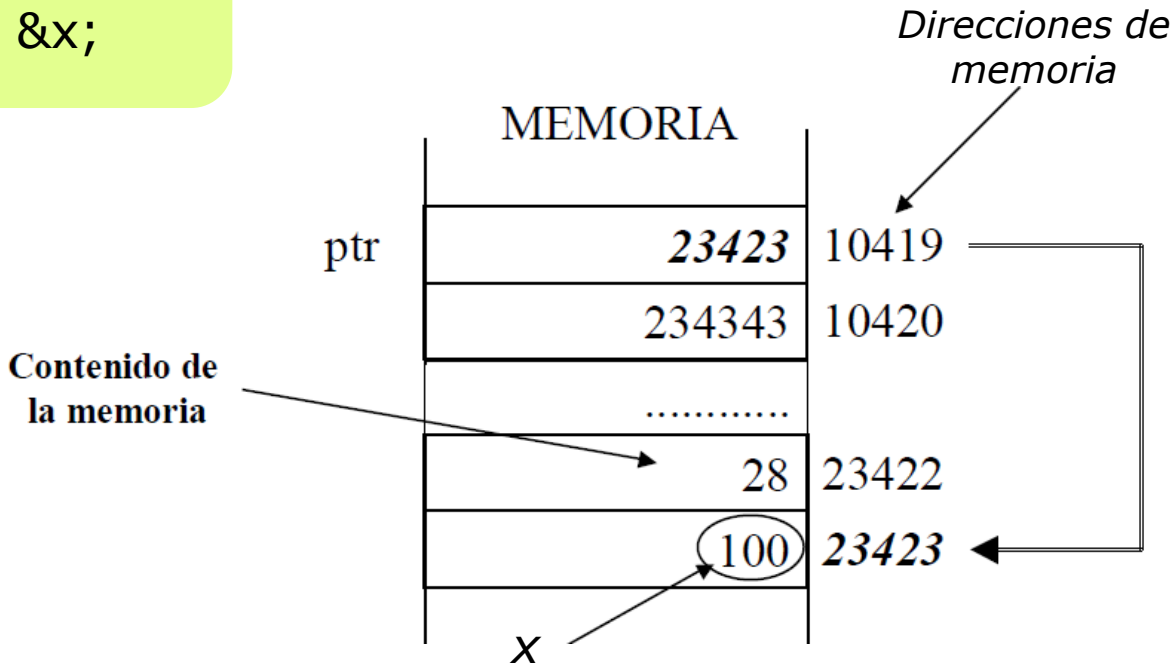


Apuntadores

PUNTEROS

- Un puntero es una variable que almacena una dirección de memoria.

```
int x=100;
int *ptr;
ptr = &x;
```



Apuntadores

PUNTEROS

Operaciones con punteros

Operación	Código	Significado
Desreferenciación	`*ptr`	Permite acceder a la variable anónima asociada al puntero *ptr*.
Comparación	`ptr1==ptr2`	Esta comparación será verdadera cuando *ptr1* y *ptr2* apunten a la misma dirección de memoria

Asignación	`ptr1=ptr2`
	
	`ptr=NULO`
		ptr1 pasa a apuntar a la misma variable anónima que *ptr2*.
		NIL indica que no apunta a ninguna dir. de memoria

Estructuras

ESTRUCTURAS

- Una estructura es una colección de uno o más tipos de elementos denominados **miembros**, cada uno de los cuales puede ser un tipo de dato diferente.

- Sintaxis de declaración:

```
struct nombre_estructura
{
  <tipo_dato_miembro> <nombre_miembro>
  <tipo_dato_miembro> <nombre_miembro>
  <tipo_dato_miembro> <nombre_miembro>
  <tipo_dato_miembro> <nombre_miembro>
};
```

Estructuras

EJEMPLO

Problema

Datos de un libro:

- Título
- Nombre
- Editorial
- Edición
- ISBN
- Precio

Solución

Nombre Miembro	Tipo Miembro
Título	Array de 50 caracteres
Nombre	Array de 40 caracteres
Editorial	Array de 50 caracteres
Edición	Entero
ISBN	Entero

Código

```
struct libro{
      char titulo[50];
      char nombre[40];
      char editorial[50];
      int edición;
      int ISBN;
};
```

Estructuras

DEFINICIÓN DE VARIABLES ESTRUCTURA

- Las variables de estructuras se pueden definir de dos formas:
  - Listándolas inmediatamente después de la llave de cierre de la declaración de la estructura.

```
struct libro{
    char titulo[50];
    char nombre[40];
    char editorial[50];
} libro1, libro2, libro3;
```

  - Listando el tipo de la estructura creado seguido por las variables correspondiente en cualquier lugar del programa antes de utilizarlas.

```
struct libro libro1, libro2, libro3;
```

Estructuras

ACCESO A LOS MIEMBROS DE UNA ESTRUCTURA

- Se puede acceder a los miembros de una estructura de una estructura mediante dos formas:
  - Utilizando el operador punto (.) Estático
  - Utilizando el operador puntero -> Dinámico

Estructuras

EJEMPLO DE ACCEDO CON EL OPERADOR “.”

```
#include <stdio.h>

int main(){
        struct complejo{
           float pr;
           float pi;
        };

        struct complejo z;

        printf("arte Real: ");
        scanf("%f", &z.pr);
        printf("Parte Imaginaria: ");
        scanf("%f", &z.pi);
        printf("Complejo: (%.1f + %.1fi)",z.pr, z.pi);
        system("pause");
        return 0;
}
```

Estructuras

EJEMPLO DE ACCEDO CON EL OPERADOR "->"

```
#include <stdio.h>

int main(){
	struct complejo{
		float pr;
		float pi;
	};

	struct complejo z, *ptr;

	printf("Parte Real: ");
	scanf("%f", &z.pr);
	printf("Parte Imaginaria: ");
	scanf("%f", &z.pi);
	ptr = &z;
	printf("Complejo: (%.1f + %.1fi)",ptr->pr, ptr->pi);
	system("pause");
	return 0;
}
```

Estructuras

ESTRUCTURAS ANIDADAS

```
struct empleado{
        char
        nombre_emp[30];
        char direccion[25];
        char ciudad[20];
        char provincia[20];
        long int cod_postal;
        double salario;
};
```

```
struct clientes {
        char nombre_cliente[30];
        char direccion[25];
        char ciudad[20];
        char provincia[20];
        long int cod_postal;
        double saldo;
};
```

Existe información repetida entre Empleado y Clientes

Estructuras

ESTRUCTURAS ANIDADAS

Contiene elementos comunes

```
struct info_dir {
    char direccion[25];
    char ciudad[20];
    char provincia[20];
    long int cod_postal;
};
```

Se usa la estructura info_dir

```
struct empleado {
    char nombre_emp[30];
    struct info_dir direccion_emp;
    double salario;
};
```

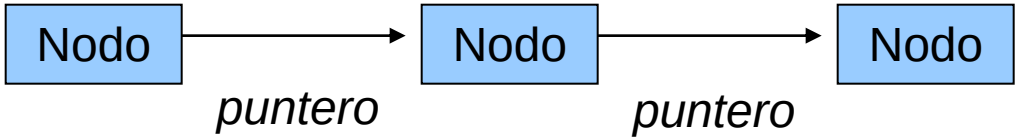
Se usa la estructura info_dir

```
struct clientes {
    char nombre_cliente[30];
    struct info_dir direccion_cliente;
    double saldo;
};
```

Listas simplemente ligadas

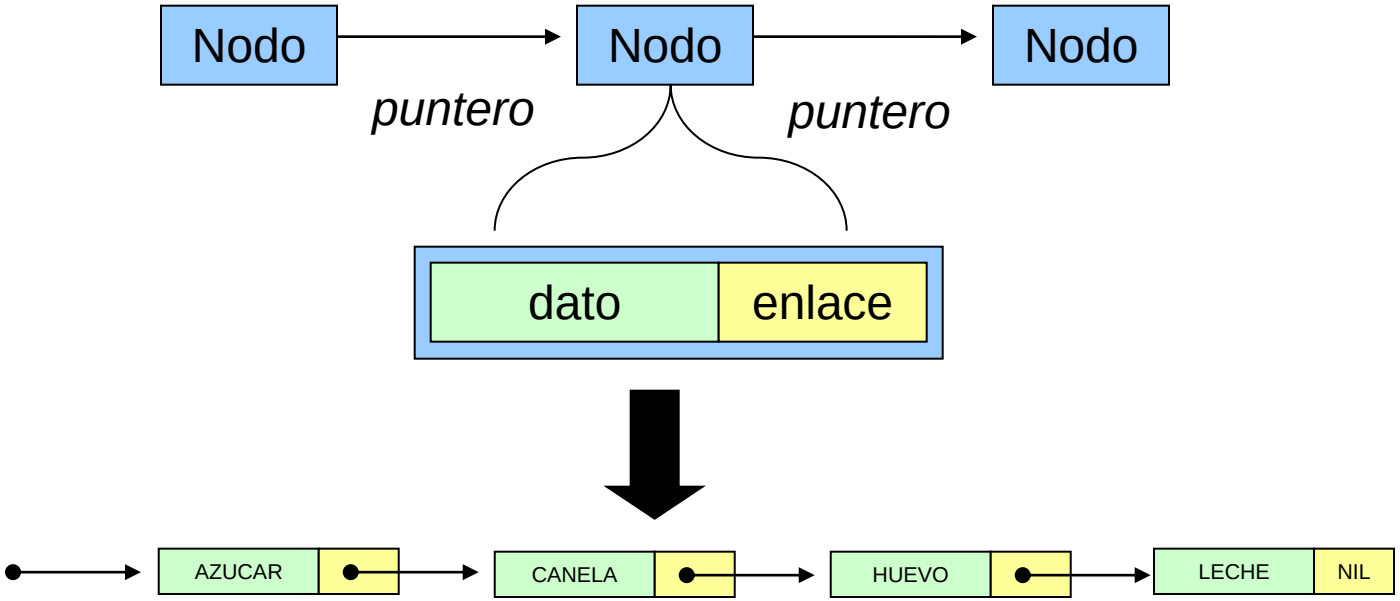
- Una lista ligada es un grupo de datos organizados secuencialmente, pero a diferencia de los arreglos, la organización no esta dada implícitamente por su posición sino por la **relación** entre elemento.

- Una lista simplemente ligada es una colección o secuencia de elementos (llamados generalmente nodos) dispuestos uno detrás de otro, en la que cada elemento se conecta al siguiente elemento por un enlace o puntero.



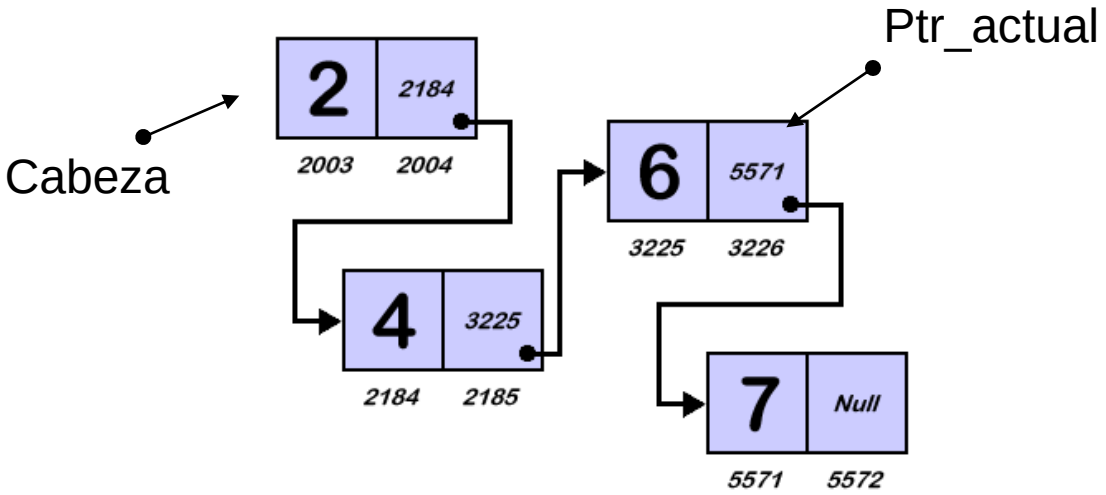
Listas simplemente ligadas

- Un *nodo* consta de un campo dato y una referencia que apunta al siguiente elemento de la lista.



Listas simplemente ligadas

- En una lista simplemente enlazada cada nodo contiene un único enlace que conecta a ese nodo al nodo siguiente o nodo sucesor.



- Entre las ventajas de las listas ligadas se encuentran las siguientes:
  - Permiten que sus tamaños cambien durante la ejecución del programa (optimizan el uso de memoria).
  - Proveen una mayor flexibilidad en el manejo de los datos.
  - La lista es eficiente en recorridos hacia delante.

Operaciones con listas simplemente ligadas

- Las operaciones que pueden llevarse a cabo en una lista son:
  - Recorrido de la lista.
  - Inserción de un elemento.
  - Borrado de un elemento.
  - Búsqueda de un elemento.

Operaciones con listas simplemente ligadas

- Crear el nodo inicial.

```
creaNodoInicial(DATO)
{Este algoritmo crea el primer nodo de una lista. DATO es la
información que se guardará en el nuevo nodo}
{creaNodoInicial devuelve un puntero al primer elemento de la
lista}

Crear NUEVO {Crea el primer nodo de la lista}
Hacer NUEVO^.INFORMACION=DATO y NUEVO^.ENLACE=NIL
Retornar NUEVO
```

Operaciones con listas simplemente ligadas

- Algoritmo para la inserción de elementos:

```
insertarNodo(DATO, NODO **P)
{Este algoritmo inserta un nodo en una lista. DATO es la
información que se guardará en el nuevo nodo y **P es una
variable puntero a un puntero de una estructura NODO}
{ACTUAL y ANTERIOR son variables puntero a estructuras NODO}

Hacer ACTUAL =*P y ANTERIOR=NIL
RepetirMientras ((ACTUAL<>NIL) y (ACTUAL^.INFORMACION < DATO))
    Hacer ANTERIOR=ACTUAL y ACTUAL=ACTUAL^.ENLACE
FinMientras
Crear NUEVO {Crea el nuevo nodo de la lista}
Si (ACTUAL=*P) entonces
   Hacer NUEVO^.INFORMACION=DATO
   Hacer NUEVO^.ENLACE=*P y *P=NUEVO
Sino
   Hacer NUEVO^.INFORMACION=DATO y NUEVO^.ENLACE=ANTERIOR^.ENLACE
   Hacer ANTERIOR^.ENLACE=NUEVO
Finsi
```

Operaciones con listas simplemente ligadas

- Algoritmo para la eliminación de elementos:

```
eliminarNodo(DATO, NODO **P)
{Este algoritmo elimina un nodo en una lista. DATO es la información que se
será evaluada para determinar el nodo que será eliminado y **P es una
variable puntero a un puntero de una estructura NODO}
{ENCONTRAO es una variable booleana. ACTUAL y ANTERIOR son variables
puntero a estructuras NODO}

Hacer ENCONTRADO=FALSO, ACTUAL =*P y ANTERIOR=NIL
RepetirMientras ((ACTUAL<>NIL) y (encontrado=FALSO))
   Si (ACTUAL^.INFORMACION=DATO) entonces
      Hacer ENCONTRADO=VERDADERO
   Sino
      Hacer ANTERIOR=ACTUAL Y ACTUAL=ACTUAL^.ENLACE
   Finsi
FinMientras
Si (ACTUAL<>NIL) entonces
   Si (ACTUAL=*P) entonces
      Hacer *P=ACTUAL^.ENLACE
   Sino
      ANTERIOR^.ENLACE=ACTUAL ^.ENLACE
      Escribir "Nodo eliminado"
   Finsi
Sino
   Escribir "Nodo no encontrado"
Finsi
Liberar ACTUAL
```

Ejercicio

- Las listas pueden utilizarse para almacenar los coeficientes diferentes de cero de un polinomio, junto al exponente. Realiza un programa que permita capturar el coeficiente y exponente de cada término de un polinomio P(x), y proporcione el valor de P(x) para una x dada.