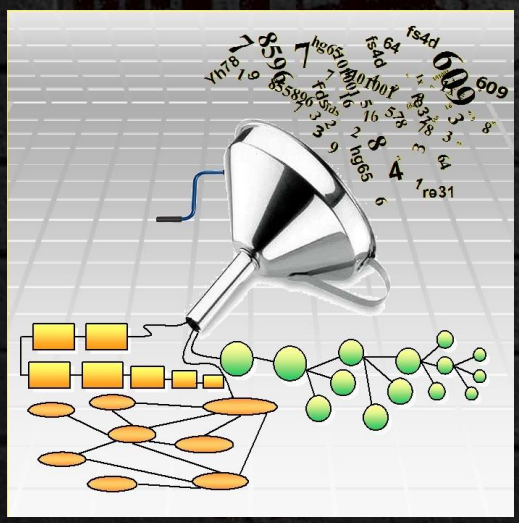
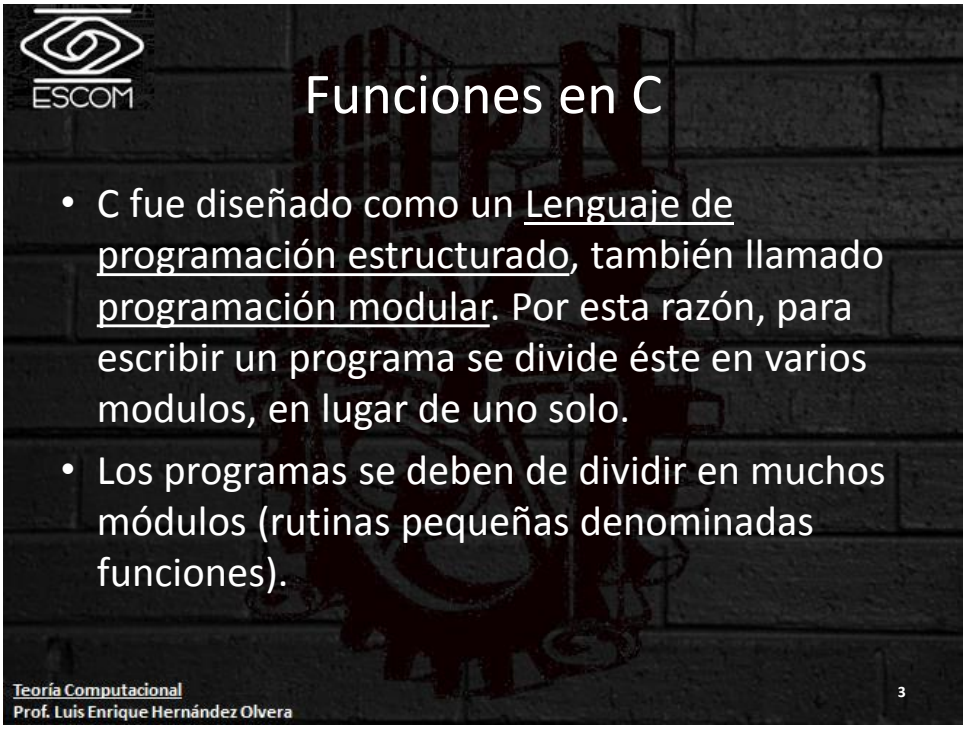
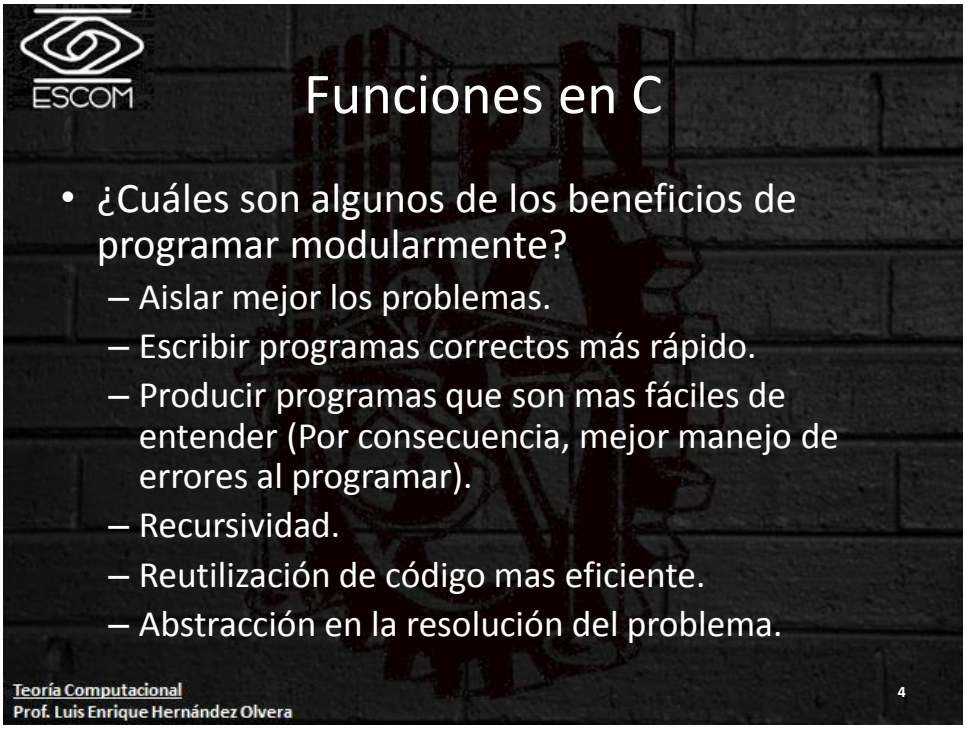
Repaso de C



Contenido

- Funciones
- Arreglos
- Punteros
- Cadenas
- Asignación de Memoria Dinámica
- Estructuras
- Listas


ESCOM
Funciones en C
• C fue diseñado como un Lenguaje de programación estructurado, también llamado programación modular. Por esta razón, para escribir un programa se divide éste en varios modulos, en lugar de uno solo.
• Los programas se deben de dividir en muchos módulos (rutinas pequeñas denominadas funciones).
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
3


ESCOM
Funciones en C
• ¿Cuáles son algunos de los beneficios de programar modularmente?
– Aislar mejor los problemas.
– Escribir programas correctos más rápido.
– Producir programas que son mas fáciles de entender (Por consecuencia, mejor manejo de errores al programar).
– Recursividad.
– Reutilización de código mas eficiente.
– Abstracción en la resolución del problema.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
4

Funciones (Ejemplo)

- Se está escribiendo un programa que obtenga una lista de caracteres del teclado, los ordene alfabéticamente y los visualice a continuación en la pantalla, esta claro que se puede escribir todo el código que haga estas acciones en un solo programa (main()).

```
Int main()
{
        /*Código c para obtener una lista de caracteres*/
        /*Código c para alfabetizar los caractes*/
        /*Código c para visualizar la lista por orden alfabético*/
        return 0;
}
```


Funciones (Ejemplo)

- El mejor medio para escribir el programa es usando funciones independientes para cada tarea que haga el programa:

```
Int main()
{
        obtenerCaracteres();
        alfabetizar();
        imprimirLetras();
        return 0;
}
```

```
void obtenerCaracteres(){
/*Código c para obtener una lista de caracteres*/
 }
void alfabetizar() {
/*Código c para alfabetizar los caractes*/
}
void imprimirLetras() {
/*Código c para visualizar la lista por orden alfabético*/
}
```

Funciones en C

- Las funciones en c no se pueden anidar. Esto significa queque una función no se puede declarar dentro de otra función.
- En C todas las funciones son externas o globales, es decir, pueden ser llamadas desde cualquier punto del programa.

Estructura de una función en C

```
Tipo-de-retorno   nombreFuncion  (  listaDeParametros)
{
        /*Cuerpo de la función */
        return   expresión;
}
```

- Donde:
  - **Tipo-de-retorno** es el valor devuelto por la función o la palabra reservada **void** si la función no devuelve ningún valor
  - **nombreFuncion** es el identificador o nombre de la función.
  - **ListaDeParametros** es la lista de declaraciones de los parámetros de la función separados por comas.
  - **Expresión** es el valor que devuelve la función.

Aspectos mas sobresalientes de las Funciones en C

- **Tipo de resultado**. Es el tipo de dato que devuelve la función en C y aparece antes del nombre de la función.
- **Lista de parámetros**. Es una lista de parámetros tipificados (con tipos) que utilizan el formato siguiente:
  - Tipo1 parametro1, tipo2 parametro2, ….
- **Cuerpo de la función**. Se encierra entre llaves de apertura ({) y cierre (}).

Aspectos mas sobresalientes de las Funciones en C

- **No se pueden declarar funciones anidadas.**
- **Declaración local**. Las constantes, tipos de datos y variables declaradas dentro de la función son locales a la misma y no perduran fuera de ella.
- **Valor devuelto por la función.** Mediante la palabra reservada **return** se devuelve el valor de la función.

Funciones en C

- Cuando se llama a una función, el control pasa a la misma para su ejecución; y cuando finaliza (normalmente cuando se encuentra una sentencia **return**), el control es devuelto de nuevo al módulo que llamó, para continuar con la ejecución del mismo a partir de la sentencia que efectuó la llamada.

```
Int main()
{
        funcion1();
        …
        funcion1();
        …
}
```

```
funcion1()
{
        …
        funcion2();
        …
}
```

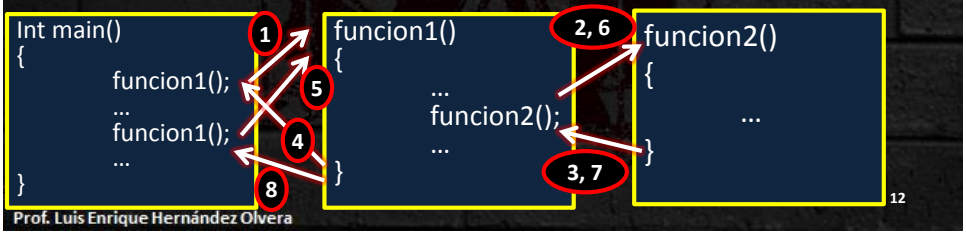
```
funcion2()
{
        …
}
```

Prof. Luis Enrique Hernández Olvera

11

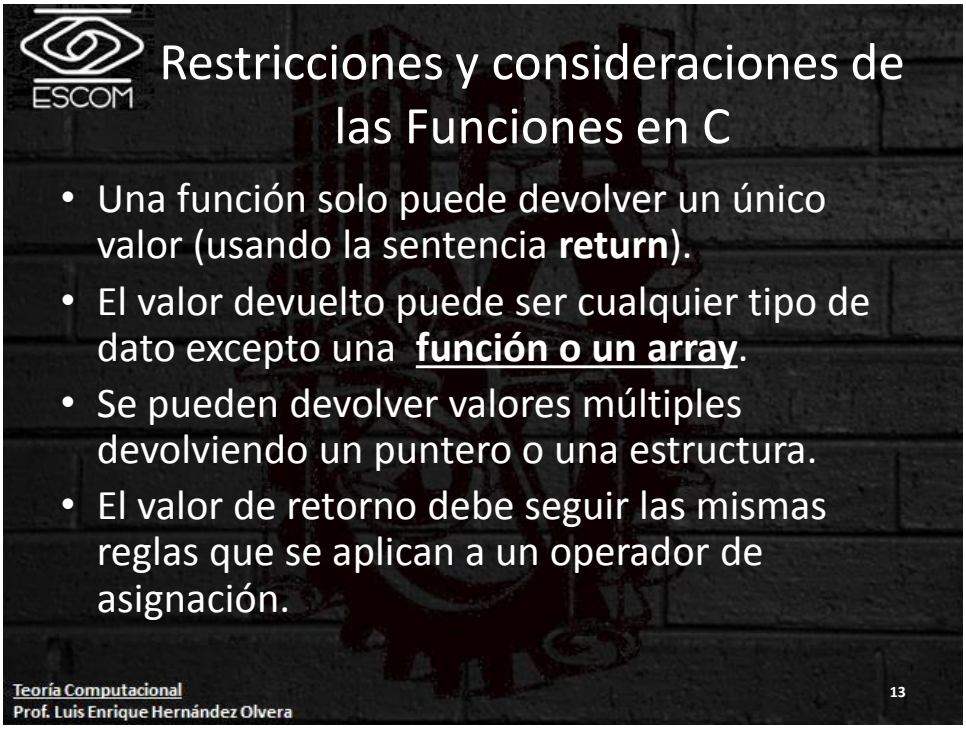
Funciones en C

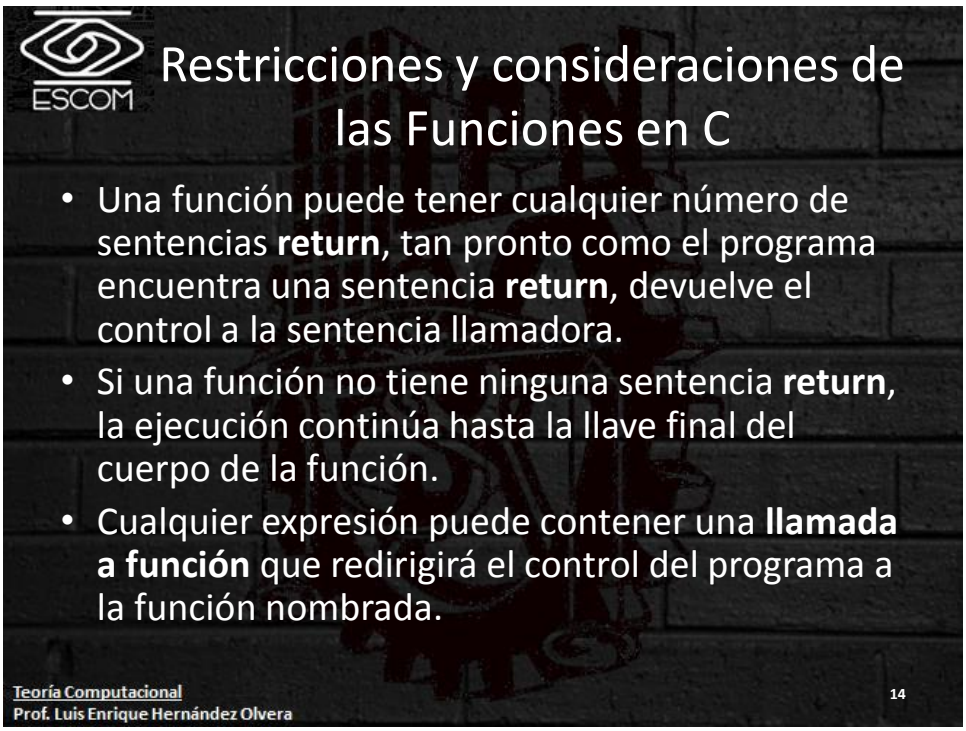
- Cuando se llama a una función, el control pasa a la misma para su ejecución; y cuando finaliza (normalmente cuando se encuentra una sentencia **return**), el control es devuelto de nuevo al módulo que llamó, para continuar con la ejecución del mismo a partir de la sentencia que efectuó la llamada.



Prof. Luis Enrique Hernández Olvera

12


ESCOM
Restricciones y consideraciones de las Funciones en C
• Una función solo puede devolver un único valor (usando la sentencia **return**).
• El valor devuelto puede ser cualquier tipo de dato excepto una **función o un array**.
• Se pueden devolver valores múltiples devolviendo un puntero o una estructura.
• El valor de retorno debe seguir las mismas reglas que se aplican a un operador de asignación.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
13


ESCOM
Restricciones y consideraciones de las Funciones en C
• Una función puede tener cualquier número de sentencias **return**, tan pronto como el programa encuentra una sentencia **return**, devuelve el control a la sentencia llamadora.
• Si una función no tiene ninguna sentencia **return**, la ejecución continúa hasta la llave final del cuerpo de la función.
• Cualquier expresión puede contener una **llamada a función** que redirigirá el control del programa a la función nombrada.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
14

Prototipo de las Funciones

- La declaración de una función se denomina **prototipo**. Los prototipos de una función contienen la cabecera de la función.
- Los prototipos de las funciones llamadas en un programa se incluyen en la cabecera del programa para que así sean reconocidas en todo el programa.

Prototipo de las Funciones

- Se recomienda que se declare una función si se llama a la función antes de que se defina.
- Los prototipos se sitúan normalmente al principio de un programa, antes de la definicion de la función main().

```
#include <stdio.h>
/*----- Declaración de prototipos ----*/
Int main( )
{
        return  0;
}
```

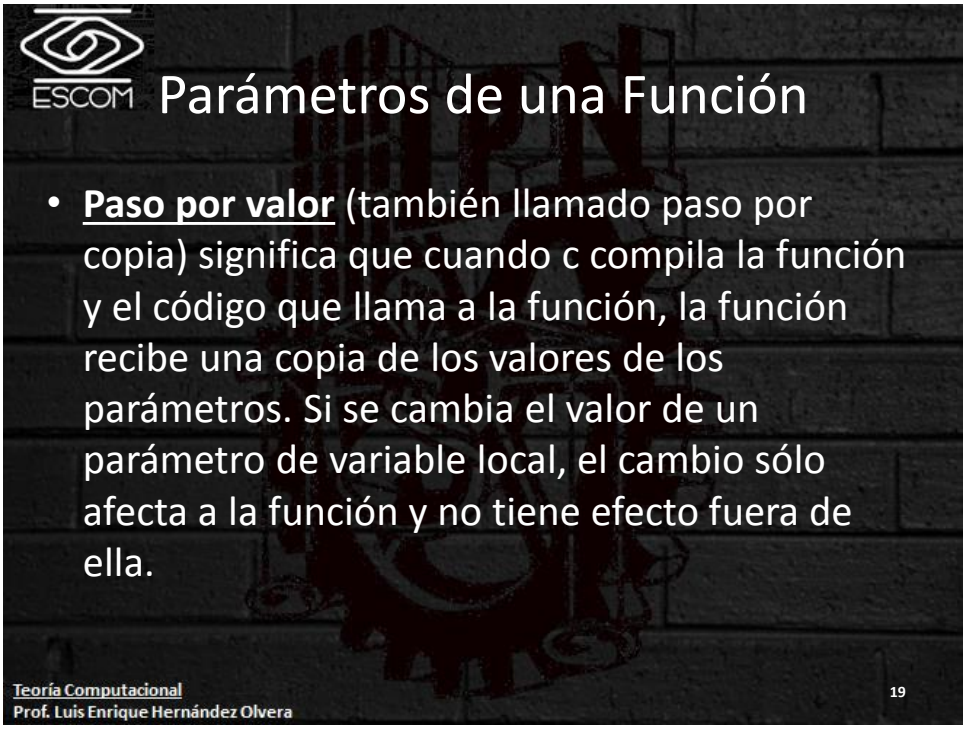
Sintaxis de un prototipo

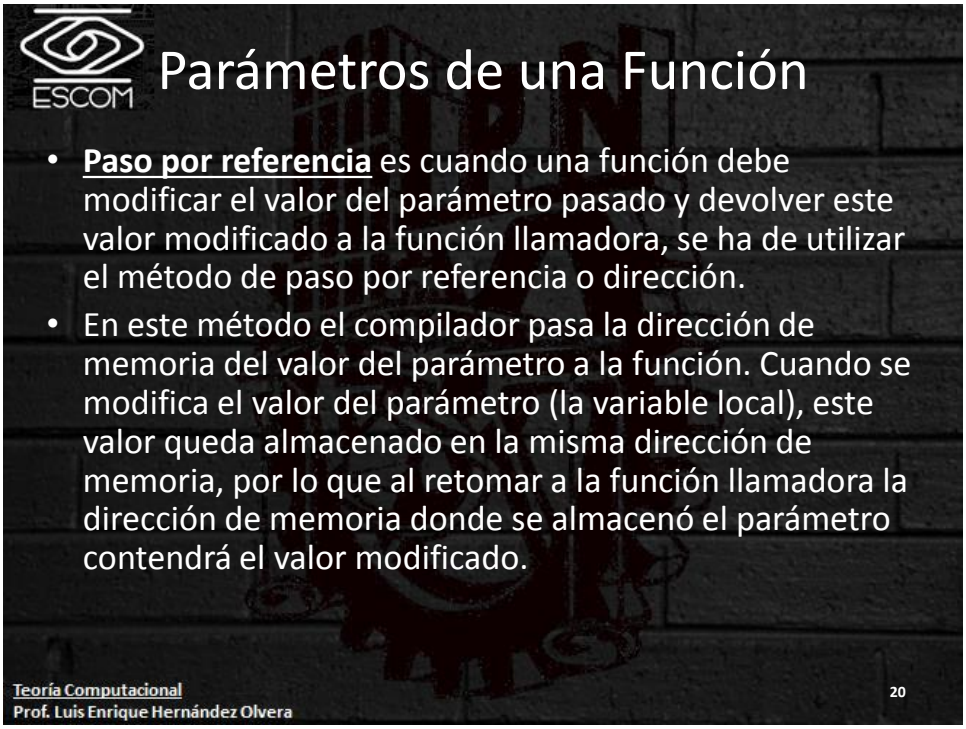
```
Tipo-de-retorno   nombreFuncion  (  listaDeDeclaracionParametros);
```

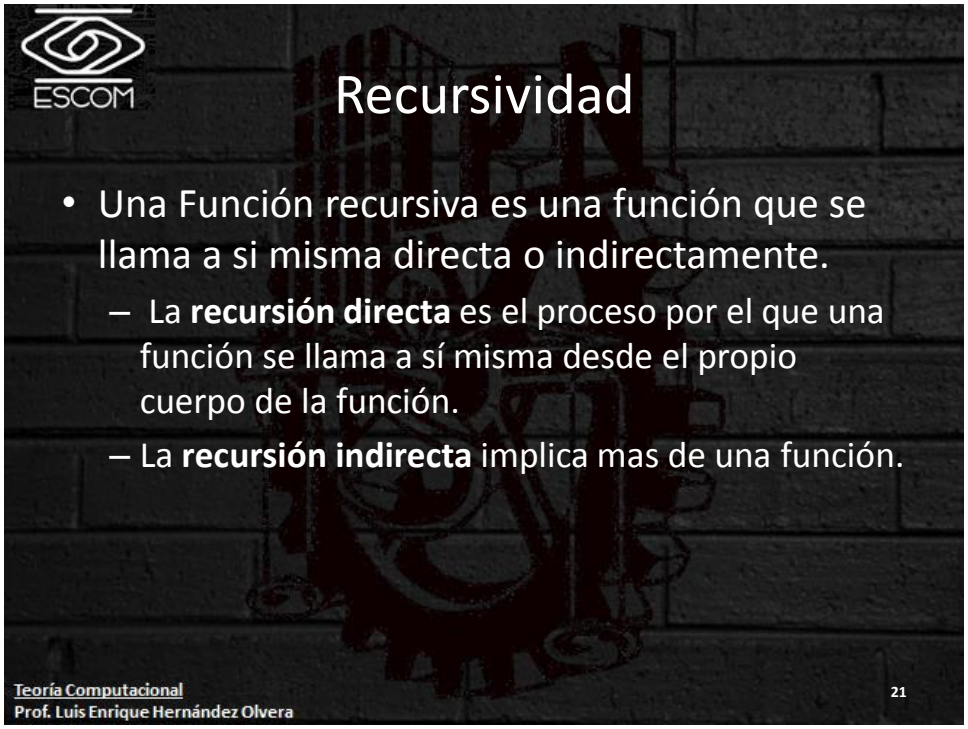
- Donde:
  - **Tipo-de-retorno** es el valor devuelto por la función o la palabra reservada **void** si la función no devuelve ningún valor
  - **nombreFuncion** es el identificador o nombre de la función.
  - **ListaDeDeclaracionParametros** es la lista de declaraciones de los parámetros de la función separados por comas (Los nombres de los parámetros son opcionales, pero es buena práctica incluirlos para indicar lo que representan).

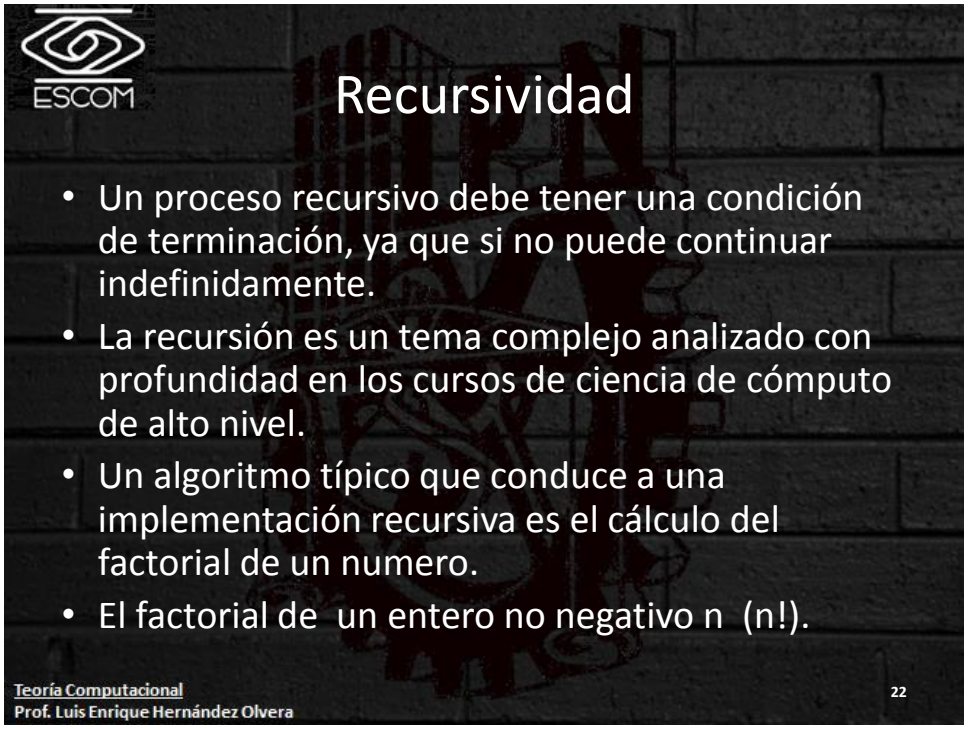
¿Por qué usar prototipos?

- El compilador utiliza los prototipos para validar que el numero y los tipos de datos de los argumentos reales de la llamada a la función son los mismos que el número y tipo de argumentos formales en la función llamada.
- Si se detecta una inconsistencia, se visualiza un mensaje de error. Sin prototipos, un error puede ocurrir si un argumento con un tipo de dato incorrecto se pasa a la función.


ESCOM
Parámetros de una Función
• Paso por valor (también llamado paso por copia) significa que cuando c compila la función y el código que llama a la función, la función recibe una copia de los valores de los parámetros. Si se cambia el valor de un parámetro de variable local, el cambio sólo afecta a la función y no tiene efecto fuera de ella.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
19


ESCOM
Parámetros de una Función
• Paso por referencia es cuando una función debe modificar el valor del parámetro pasado y devolver este valor modificado a la función llamadora, se ha de utilizar el método de paso por referencia o dirección.
• En este método el compilador pasa la dirección de memoria del valor del parámetro a la función. Cuando se modifica el valor del parámetro (la variable local), este valor queda almacenado en la misma dirección de memoria, por lo que al retomar a la función llamadora la dirección de memoria donde se almacenó el parámetro contendrá el valor modificado.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
20

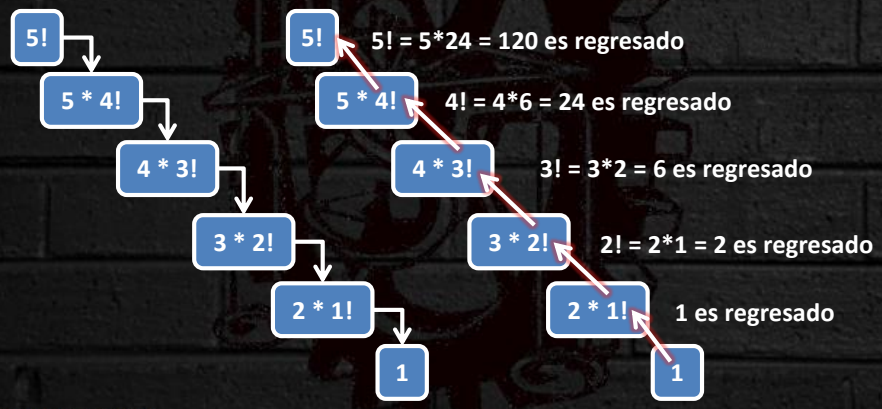

ESCOM
Recursividad
• Una Función recursiva es una función que se llama a si misma directa o indirectamente.
– La **recursión directa** es el proceso por el que una función se llama a sí misma desde el propio cuerpo de la función.
– La **recursión indirecta** implica mas de una función.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
21


ESCOM
Recursividad
• Un proceso recursivo debe tener una condición de terminación, ya que si no puede continuar indefinidamente.
• La recursión es un tema complejo analizado con profundidad en los cursos de ciencia de cómputo de alto nivel.
• Un algoritmo típico que conduce a una implementación recursiva es el cálculo del factorial de un numero.
• El factorial de un entero no negativo n (n!).
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
22

Recursividad

- El factorial de un entero no negativo n (n!).

$$n! = n * (n-1) * (n-2) * \ldots * 3 * 2 * 1$$



Arreglos (Definición)

- Un arreglo es un grupo de posiciones en memoria relacionadas entre si, por el hecho de que todas tienen el mismo nombre y son del mismo tipo.

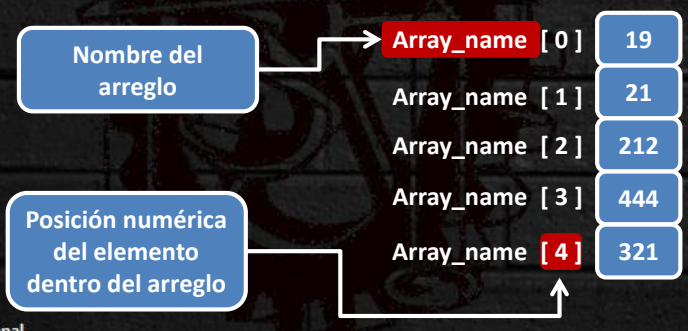
Array_name [0]	19
Array_name [1]	21
Array_name [2]	212
Array_name [3]	444
Array_name [4]	321

Arreglos

- En los arreglos se distinguen dos partes fundamentales:
  - Los Componentes
  - Los Subíndices
- Los Componentes, hacen referencia a los elementos que se almacenan en cada una de las celdas o casillas.
- El subíndice, especifica la forma de acceder a cada uno de estos elementos. Un subíndice debe ser un entero o una expresión entera.

Arreglos

- Para hacer referencia a un componente de un arreglo debemos utilizar tanto el nombre del arreglo como el subíndice del elemento.



ESCOM

Arreglos

- El primer elemento de un arreglo es el elemento cero, entonces, el primer elemento de un arreglo array_name se conoce como array_name [0], por lo que con un arreglo de longitud "n", el ultimo elemento se conoce como array_name [n - 1].

ESCOM

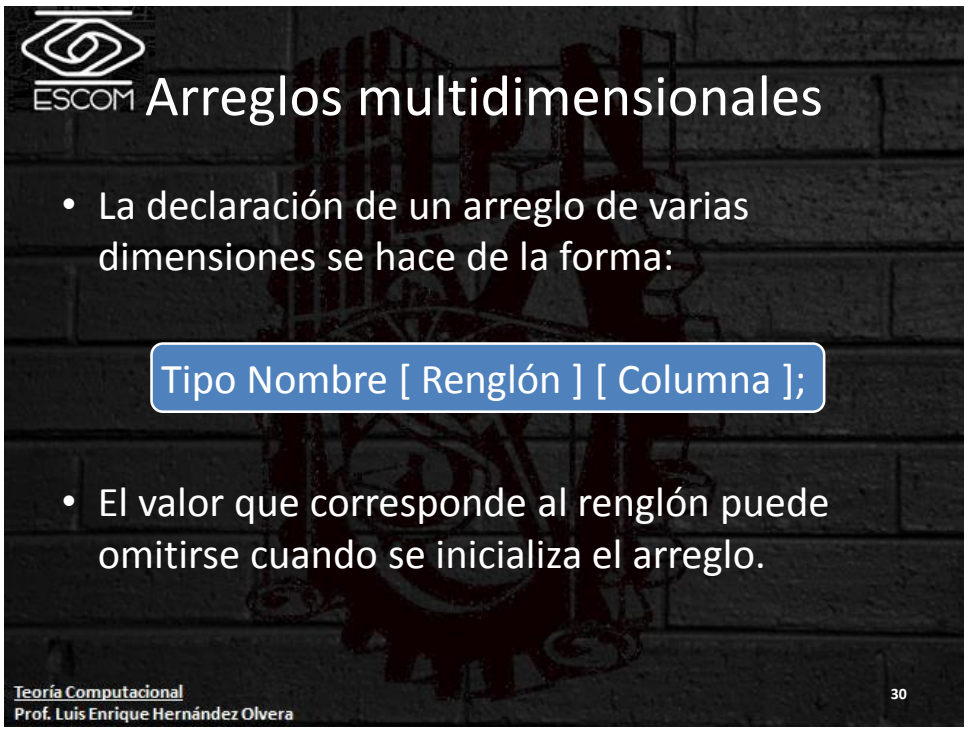
Declaración de arreglos

- La declaración de un array de una dimensión es de la siguiente forma:

```
Tipo   Nombre [ Tamaño ];
```

- Donde:
  - Tipo: Indica el tipo de los elementos del arreglo.
  - Nombre: Es un identificador que nombra al arreglo.
  - Tamaño: Es una constante que especifica el número de elementos del arreglo.


ESCOM
Arreglos multidimensionales
• En C los arreglos pueden tener múltiples subíndices. Una utilización común de los arreglos con múltiples subíndices es en la representación de tablas de valores o matrices.
• El Estándar ANSI indica que un sistema ANSI C debe soportar por lo menos 12 subíndices de arreglo.
• Instituto Nacional Estadounidense de Estándares (ANSI)
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
29


ESCOM
Arreglos multidimensionales
• La declaración de un arreglo de varias dimensiones se hace de la forma:
Tipo Nombre [Renglón] [Columna];
• El valor que corresponde al renglón puede omitirse cuando se inicializa el arreglo.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
30

Arreglos multidimensionales

- Para un arreglo array [4][3] su representación grafica seria la siguiente:

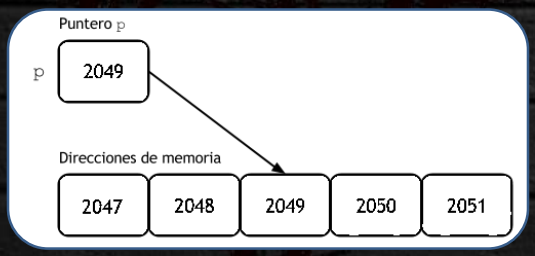
	Columna 0	Columna 1	Columna 2
Renglón 0	array [0] [0]	array [0] [1]	array [0] [2]
Renglón 1	array [1] [0]	array [1] [1]	array [1] [2]
Renglón 2	array [2] [0]	array [2] [1]	array [2] [2]
Renglón 3	array [3] [0]	array [3] [1]	array [3] [2]

Punteros (Definición)

- Un puntero es una variable que contiene **la dirección de memoria,** de un dato o de otra variable que contiene al dato. El puntero apunta al espacio físico donde está el dato o la variable. Un puntero puede apuntar a un objeto de cualquier tipo, incluyendo estructuras, funciones, etc.

Punteros (uso)

- Los punteros se pueden utilizar para crear y manipular estructuras de datos, para asignar memoria dinámicamente y para proveer el paso de argumentos por referencia en las llamadas a funciones.



Punteros (Creación)

- Un puntero se declara precediendo el identificador que referencia al puntero, por el **operador de indirección (*)**, el cual significa "puntero a". Un puntero siempre apunta a un objeto de un tipo particular.

```
Tipo  *var_puntero;
```

- Donde:
  - Tipo: Especifica el tipo del objeto apuntador.
  - var_puntero: nombre de la variable puntero.

Punteros (Creación)

- Ejemplo:

```
int  *p_int;
char  *p_char;
double  *p_d;
```

Recuerda:
Un puntero no inicializado tiene un valor desconocido.

- Donde:
  - p_int es un puntero a un entero
  - p_char es un puntero a una cadena de caracteres.
  - p_d es un puntero a real.

Punteros (Operadores)

- Para el uso de punteros, distinguimos dos oeradores:
  - Operador de dirección : **&**
  - Operador de indirección: ***
- El operador unitario **&**, devuelce como resultado la dirección de su operando.
- El operador unitario ***, toma su operando como una dirección y nos da como resultado su contenido.

Punteros (Operadores)

- Ejemplo. ¿Qué imprime este código?

```
1  #include <stdio.h>
2  main(){
3          /*Declaramos las variables enteras "a", "b" y
4          los punteros "p" y "q" */
5          int a = 10, b ,*p, *q;
6          q = &a;
7          b = *q;
8
9          *p = 20;
10
11
12         printf("En la direccion %.4X esta el dato %d\n", q, b);
13         printf("En la dirección %.4X esta el dato %d\n", p, *p);
14 }
```


Punteros (Operadores)

- Ejemplo (No funcional).

```
1  #include <stdio.h>
2  main(){
3          /*Declaramos las variables enteras "a", "b" y
4          los punteros "p" y "q" */
5          int a = 10, b ,*p, *q;
6          q = &a; // asigna la dirección de "a" a la variable "q"
7          b = *q; // Asigna a "b" el contenido de la dirección "q"
8
9          *p = 20; /* error: Asignación a un puntero no válido...
10                  * ¿A donde apunta "p"? */
11
12         printf("En la direccion %.4X esta el dato %d\n", q, b);
13         printf("En la dirección %.4X esta el dato %d\n", p, *p);
14 }
```

Depende del Sistema Operativo y la versión del compilador.

Punteros (Operadores)

- Ejemplo Funcional.

```c
#include <stdio.h>
main(){
        /*Declaramos las variables enteras "a", "b" y
        los punteros "p" y "q" */
        int a = 10, b ,*p, *q;
        q = &a; // asigna la dirección de "a" a la variable "q"
        b = *q; // Asigna a "b" el contenido de la dirección "q"

        p=(int *)malloc(sizeof(int));/* Se le asigna un espacio
                                        * en memoria a "p"  */

        *p = 20; // El puntero "p" ya tiene un espacio asignado

        printf("En la direccion %.4X esta el dato %d\n", q, b);
        printf("En la direccion %.4X esta el dato %d\n", p, *p);
        // ya funciona
}
```

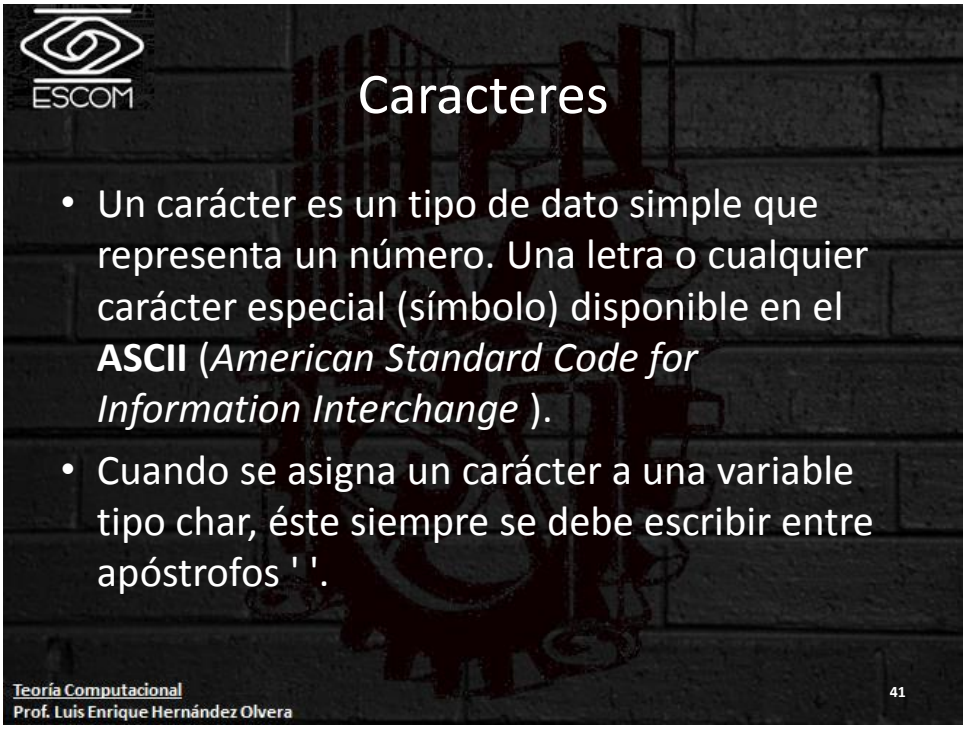

Punteros (Operadores)

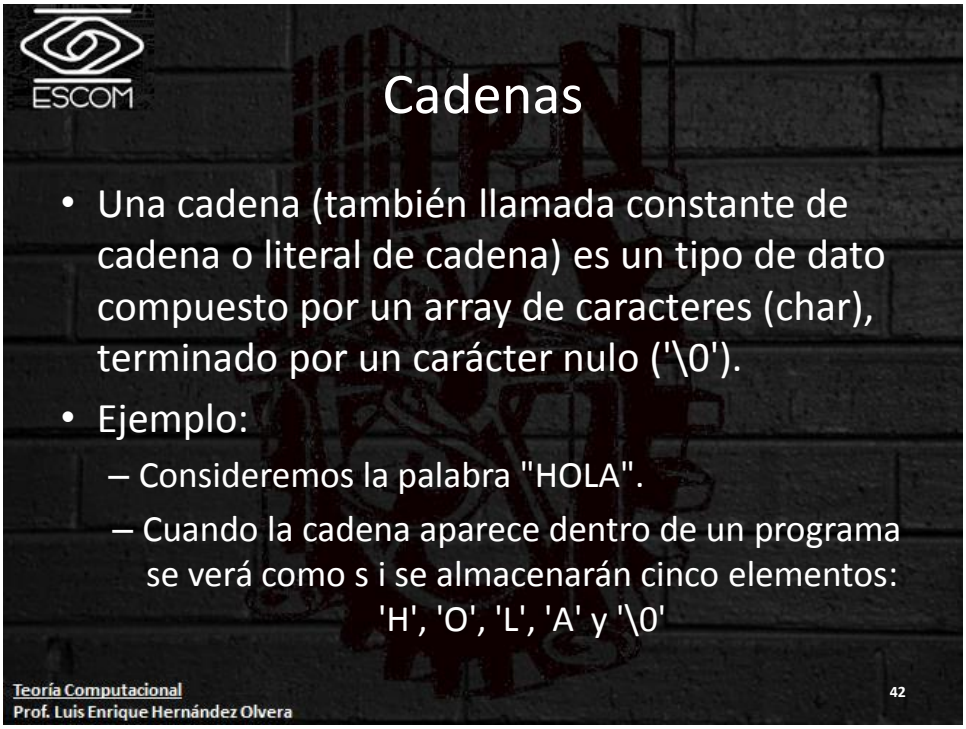
- Resultado del programa:

```
C:\PruebasC>gcc 1_6_221_corregido.c

C:\PruebasC>a.exe
En la direccion 28FF14 esta el dato 10
En la direccion 540D00 esta el dato 20

C:\PruebasC>a.exe
En la direccion 28FF14 esta el dato 10
En la direccion 350D00 esta el dato 20
```


ESCOM
Caracteres
• Un carácter es un tipo de dato simple que representa un número. Una letra o cualquier carácter especial (símbolo) disponible en el **ASCII** (*American Standard Code for Information Interchange*).
• Cuando se asigna un carácter a una variable tipo char, éste siempre se debe escribir entre apóstrofos ' '.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
41


ESCOM
Cadenas
• Una cadena (también llamada constante de cadena o literal de cadena) es un tipo de dato compuesto por un array de caracteres (char), terminado por un carácter nulo ('\0').
• Ejemplo:
– Consideremos la palabra "HOLA".
– Cuando la cadena aparece dentro de un programa se verá como s i se almacenarán cinco elementos: 'H', 'O', 'L', 'A' y '\0'
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
42

Declaración de variables de cadena

- Las cadenas se declaran como un array de tipo **char**. El operador postfijo [] contiene el tamaño maximo del objeto.

```
char  cad[21];
```

  - Declara una cadena **cad** que puede contener 20 caracteres.

```
char  nombre[ ] = "Luis Enrique";
```

  - Declara una cadena **nombre** y la inicializa con 13 elementos.

Cadenas

- Se considera que la cadena "HOLA" es un array de cinco elementos de tipo char.
- El valor real de esta cadena es la dirección de s primer carácter y su tipo es un puntero a char.
- Aplicando el operador * a un puntero a char se obtiene el carácter que forma su contenido; es posible también utilizar aritmética de direcciones con cadenas

Cadenas

- Ejemplo:

```
char Array[5] = "HOLA";
```

```
char *PArray = Array;
```

Array [0]	H
Array [1]	O
Array [2]	L
Array [3]	A
Array [4]	\0

* PArray	H
*(Parray + 1)	O
*(Parray + 2)	L
*(Parray + 3)	A
*(Parray + 4)	\0

Lectura y escritura de cadenas

- Declarar la cadena:

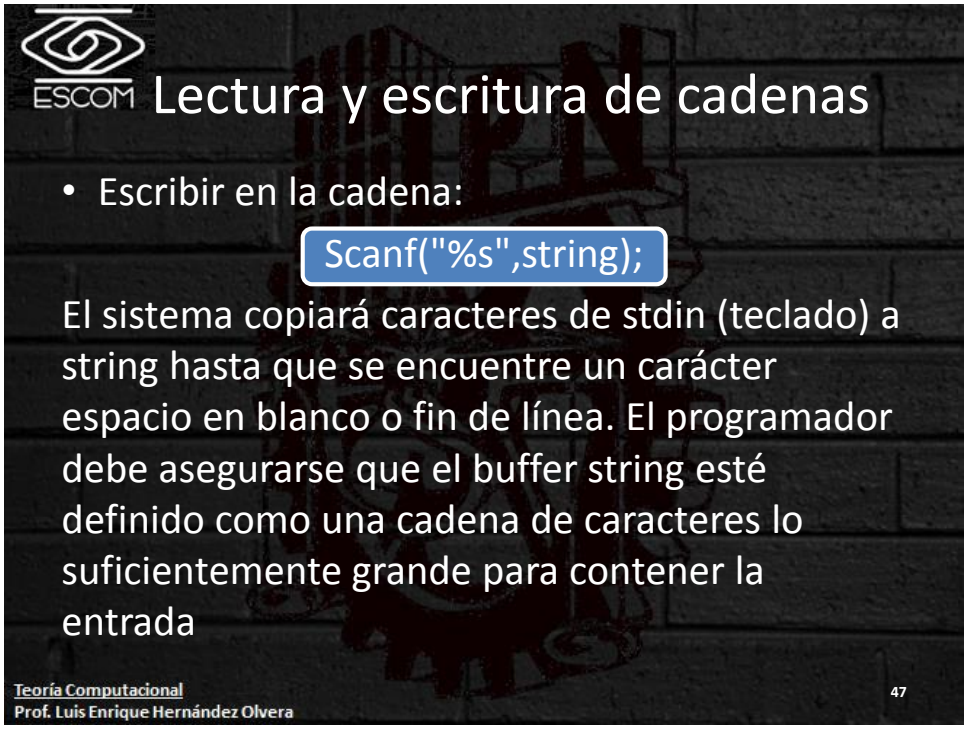
```
char  string[] = "HOLA";
```

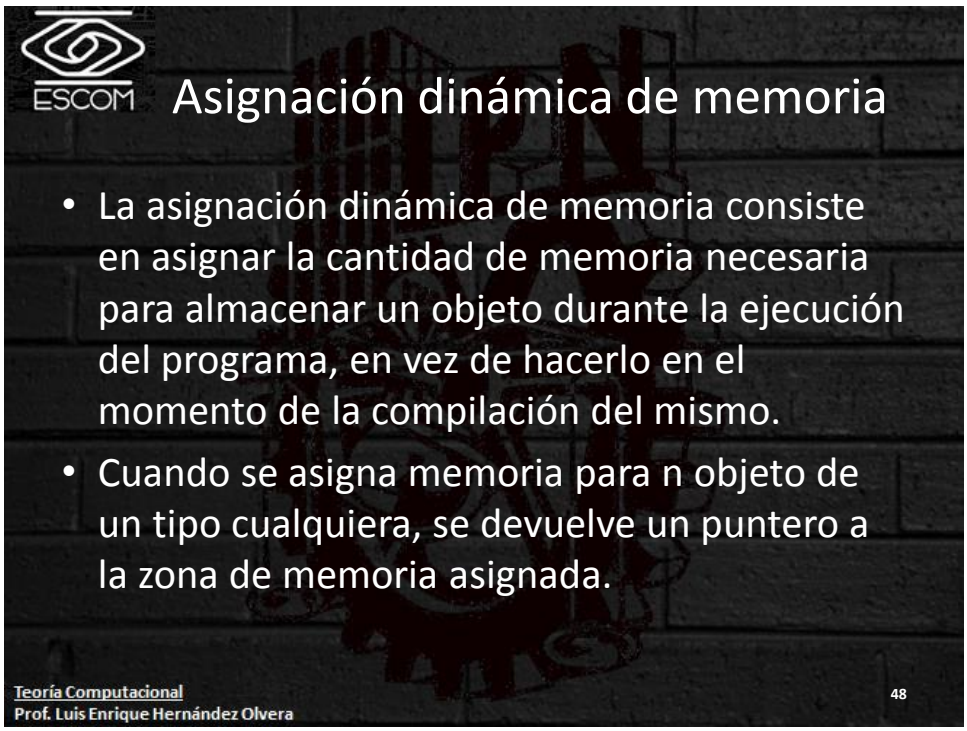
String tiene cinco caracteres ('H','O','L','A','\0')

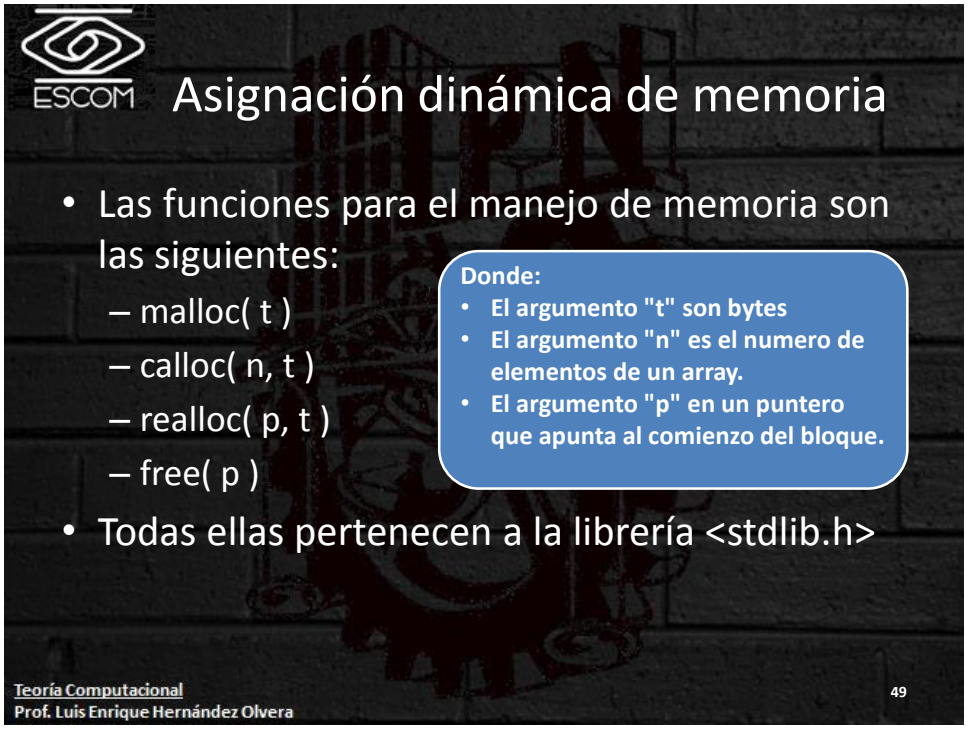
- Imprimir cadena:

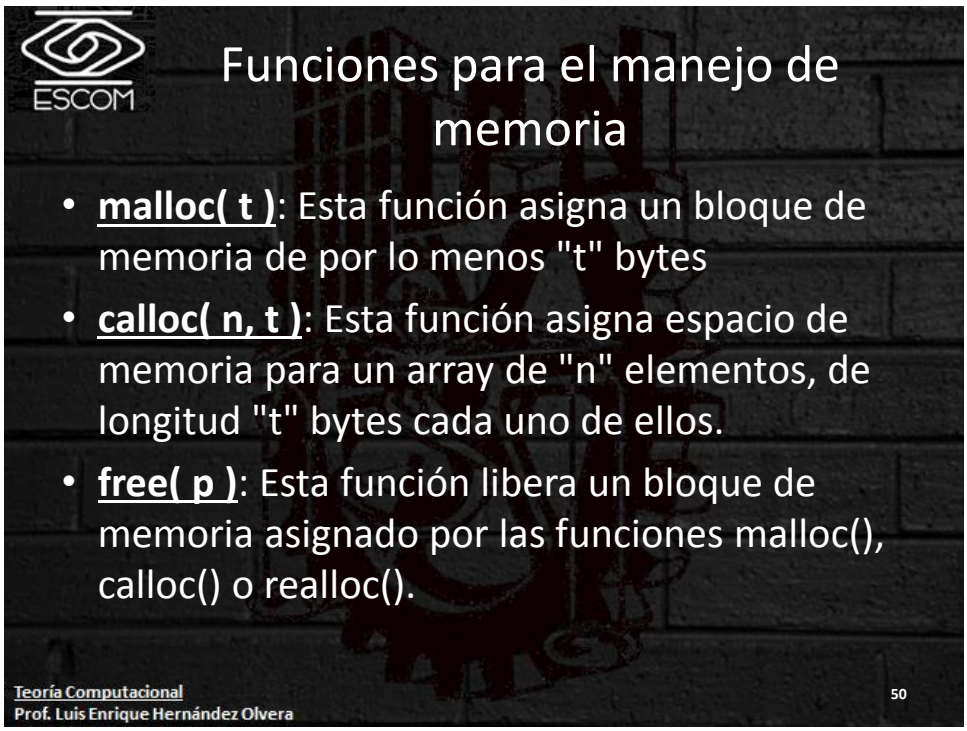
```
printf("%s",string);
```

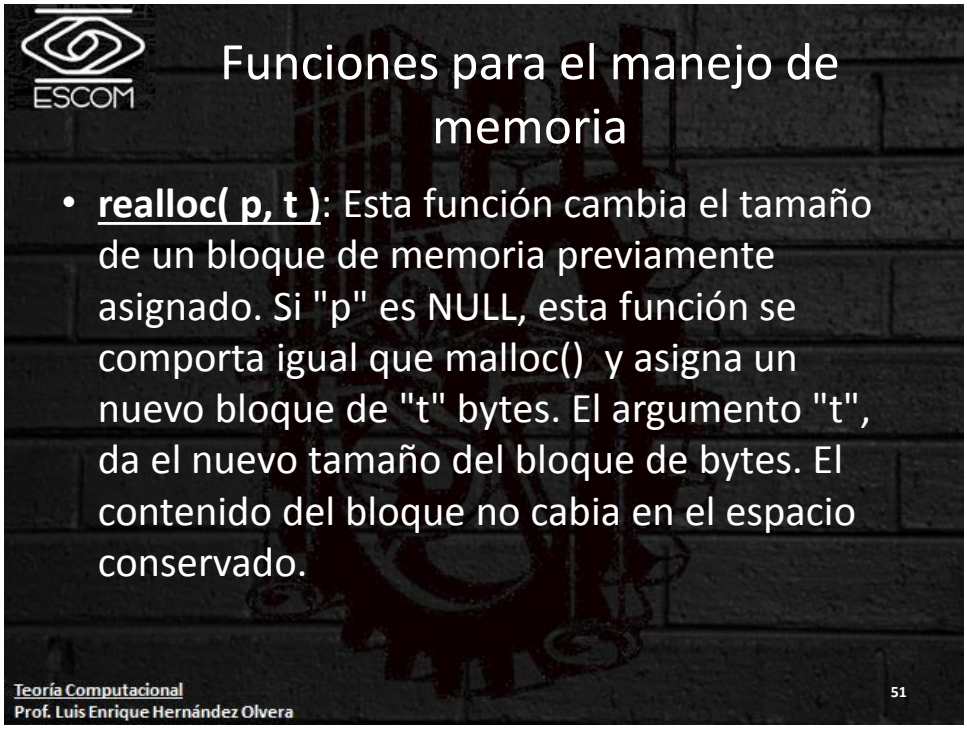
El sistema copiará caracteres de string a stdout (pantalla) hasta que el carácter NULL, '\0', se encuentre.


ESCOM
Lectura y escritura de cadenas
• Escribir en la cadena:
Scanf("%s",string);
El sistema copiará caracteres de stdin (teclado) a string hasta que se encuentre un carácter espacio en blanco o fin de línea. El programador debe asegurarse que el buffer string esté definido como una cadena de caracteres lo suficientemente grande para contener la entrada
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
47


ESCOM
Asignación dinámica de memoria
• La asignación dinámica de memoria consiste en asignar la cantidad de memoria necesaria para almacenar un objeto durante la ejecución del programa, en vez de hacerlo en el momento de la compilación del mismo.
• Cuando se asigna memoria para n objeto de un tipo cualquiera, se devuelve un puntero a la zona de memoria asignada.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
48


ESCOM
Asignación dinámica de memoria
• Las funciones para el manejo de memoria son las siguientes:
– malloc(t)
– calloc(n, t)
– realloc(p, t)
– free(p)
Donde:
• El argumento "t" son bytes
• El argumento "n" es el numero de elementos de un array.
• El argumento "p" en un puntero que apunta al comienzo del bloque.
• Todas ellas pertenecen a la librería <stdlib.h>
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
49


ESCOM
Funciones para el manejo de memoria
• malloc(t): Esta función asigna un bloque de memoria de por lo menos "t" bytes
• calloc(n, t): Esta función asigna espacio de memoria para un array de "n" elementos, de longitud "t" bytes cada uno de ellos.
• free(p): Esta función libera un bloque de memoria asignado por las funciones malloc(), calloc() o realloc().
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
50


ESCOM
Funciones para el manejo de memoria
• realloc(p, t): Esta función cambia el tamaño de un bloque de memoria previamente asignado. Si "p" es NULL, esta función se comporta igual que malloc() y asigna un nuevo bloque de "t" bytes. El argumento "t", da el nuevo tamaño del bloque de bytes. El contenido del bloque no cabia en el espacio conservado.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
51


ESCOM
Malloc (t)
• La función malloc() devuelve un puntero que referencia el espacio asignado, a un objeto de un tipo no especificado. Si hay suficiente espacio de memoria o si " t " es 0, la función retorna un puntero nulo (NULL).
void *malloc(size__t t);
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
52

ESCOM

Malloc (t)

- El espacio puede ser asignado a cualquier tipo de objeto. Para realizar la conversión al tipo deseado, utilizar la notación **cast** sobre el valor devuelto.
- Ejemplo:

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

cast

ESCOM

Calloc (n, t)

- La función calloc() devuelve un puntero al espacio asignado. Este espacio puede ser asignado a un array de cualquier tipo. Para ello, se debe utilizar la notación **cast** sobre el valor devuelto.

```
void *calloc(size__t  n, size__t  t);
```

ESCOM

Calloc (n, t)

- Si hay suficiente espacio de memoria, o si "n" o "t" son 0, la función retorna un puntero nulo (NULL). El espacio de memoria requerido debe ser inferior a 64K.
- Ejemplo:

```
int *lista
lista = (int *) calloc (100, sizeof( int ));
```

cast

ESCOM

Realloc (p, t)

- Esta función cambia el tamaño de un bloque de memoria previamente asignado. El argumento "p" es un puntero que apunta al comienzo del bloque. Si "p" es NULL, esta función se comporta igual que malloc() y asigna un nuevo bloque de "t" bytes. Si "p" no es nulo, entonces tiene que ser un puntero devuelto por las funciones malloc(), calloc(), la misma función realloc() o liberado por la función free() .

ESCOM

Realloc (p, t)

- Realloc devuelve un puntero al espacio asignado. El bloque puede ser movido al modificar el tamaño, esto quiere decir que p puede cambiar.
- Ejemplo:

```
int *lista
lista = (int *) realloc (lista, sizeof( int )*n);
```

cast

ESCOM

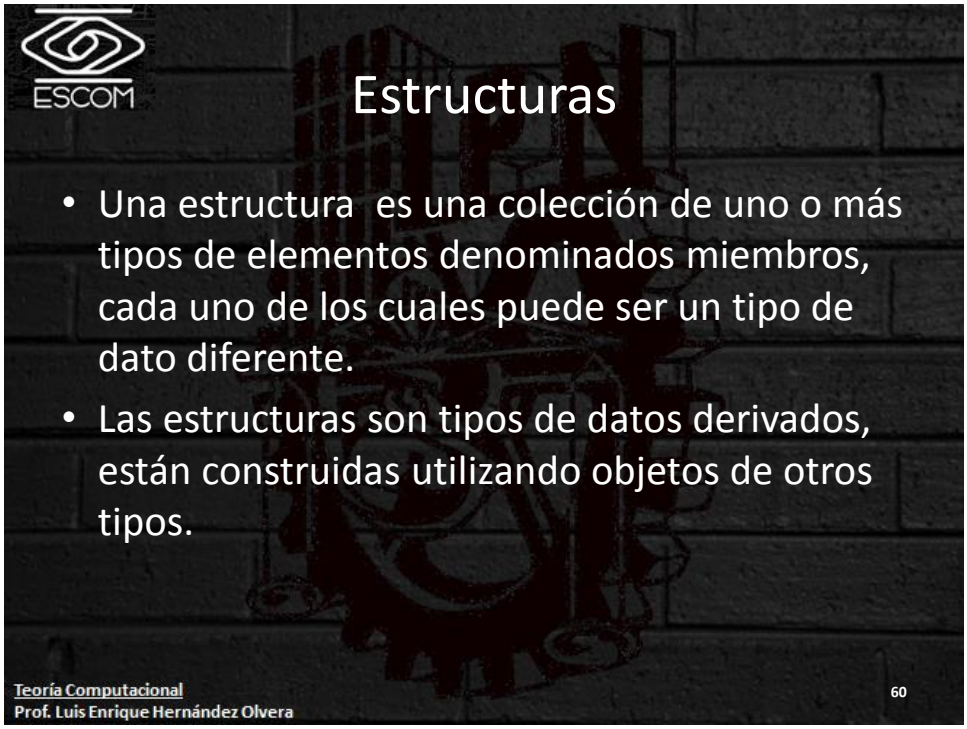
free(p)

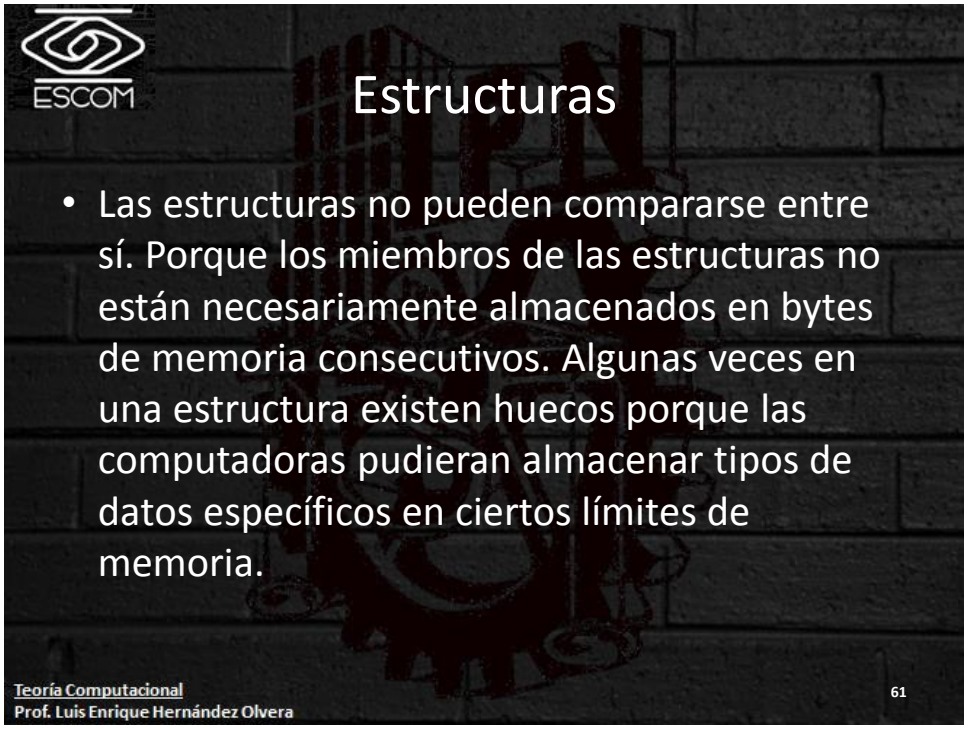
- Esta función libera un bloque de memoria asignado por las funciones malloc(), calloc() o realloc(). Un puntero nulo es ignorado.

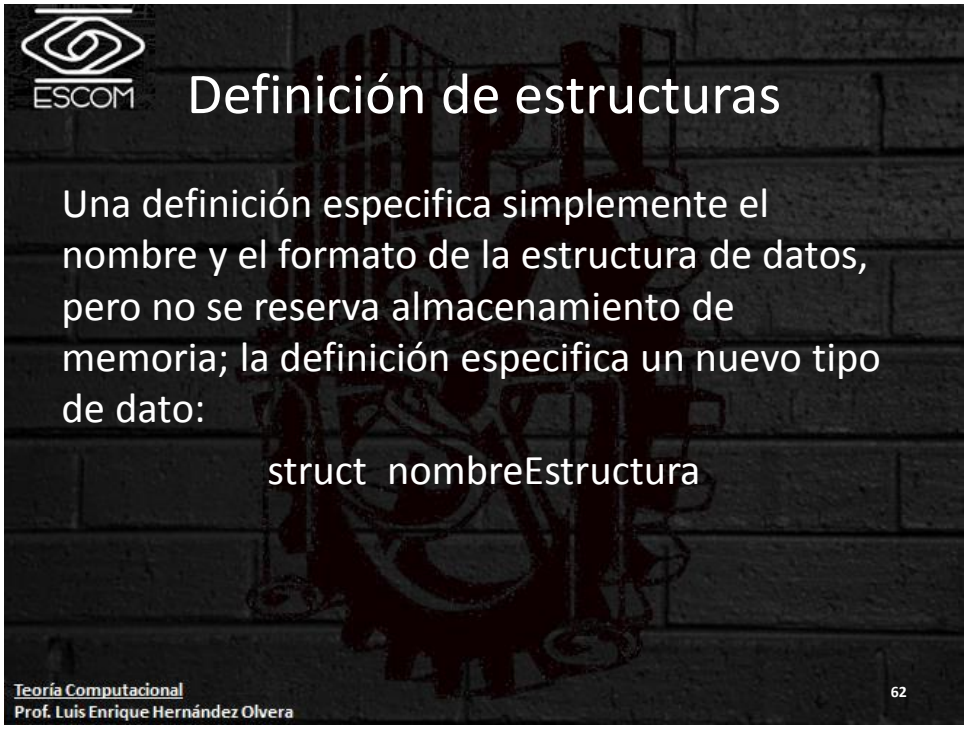
```
Void free (void *p);
```

- Esta función no retorna valor.


ESCOM
free (p)
• Ejemplo:
char *p;
p = (char *) malloc (21);
…
free(p);
• Free no iguala el puntero a NULL.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
59


ESCOM
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.
• Las estructuras son tipos de datos derivados, están construidas utilizando objetos de otros tipos.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
60


ESCOM
Estructuras
• Las estructuras no pueden compararse entre sí. Porque los miembros de las estructuras no están necesariamente almacenados en bytes de memoria consecutivos. Algunas veces en una estructura existen huecos porque las computadoras pudieran almacenar tipos de datos específicos en ciertos límites de memoria.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
61


ESCOM
Definición de estructuras
Una definición especifica simplemente el nombre y el formato de la estructura de datos, pero no se reserva almacenamiento de memoria; la definición especifica un nuevo tipo de dato:
struct nombreEstructura
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
62

Definición de estructuras

- Una estructura es un tipo de dato definido por el usuario, que se debe declarar antes de que se pueda utilizar. El formato es:

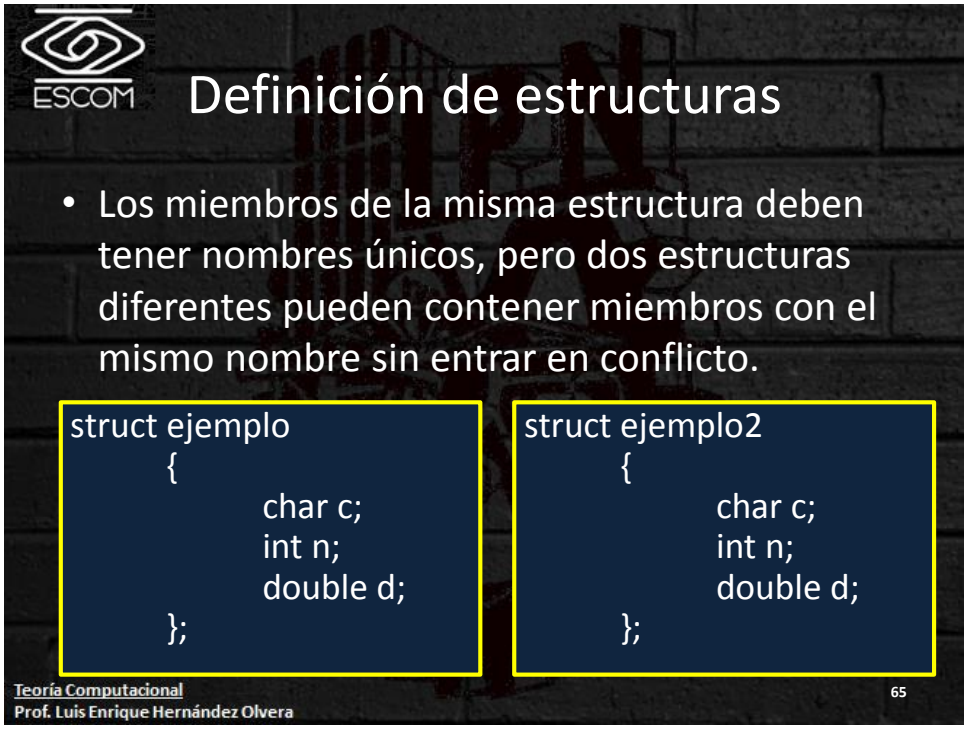
```
struct   nombreEstructura
        {
                tipoDato   nombreDato
                tipoDato   nombreDato
                …
        };
```

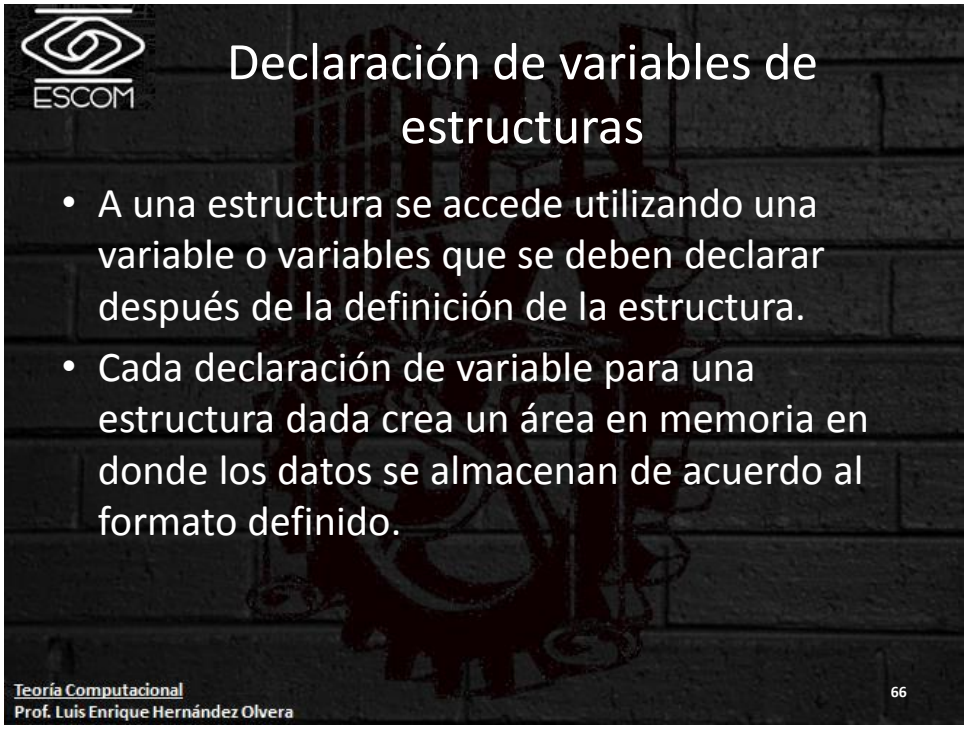

Definición de estructuras

Ejemplo:

- Las variables declaradas dentro de las llaves de la definición de estructura son los miembros de la estructura

```
struct ejemplo
        {
                char c;
                int n;
                double d;
        };
```


ESCOM
Definición de estructuras
• Los miembros de la misma estructura deben tener nombres únicos, pero dos estructuras diferentes pueden contener miembros con el mismo nombre sin entrar en conflicto.
struct ejemplo
{
char c;
int n;
double d;
};
struct ejemplo2
{
char c;
int n;
double d;
};
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
65


ESCOM
Declaración de variables de estructuras
• A una estructura se accede utilizando una variable o variables que se deben declarar después de la definición de la estructura.
• Cada declaración de variable para una estructura dada crea un área en memoria en donde los datos se almacenan de acuerdo al formato definido.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
66

Declaración de variables de estructuras

- Las variables de estructuras se pueden declarar de dos formas:

1) Listándolas inmediatamente después de la llave de cierre de la declaración de la estructura.

```
struct ejemplo
        {
                char c;
                int n;
                double d;
        } var1, var2[10], *var3;
```


Declaración de variables de estructuras

2) Listando el tipo de la estructura creado seguida por las variables correspondientes en cualquier lugar del programa antes de utilizarlas.

```
struct ejemplo2
        {
                char c;
                int n;
                double d;
        };

int main()
{
        struct ejemplo2  var1, var2[10], *var3;
        …
}
```

Inicializar una estructura

- Las estructuras pueden ser inicializadas mediante listas de inicialización como con los arreglos.

```
struct ejemplo
        {
                char *cadena;
                int n;
        };

int main()
{
        struct ejemplo var1 = {"Hola", 5};

}
```


Cómo acceder a los miembros de estructuras

- Para tener acceso a miembros de estructuras se utilizan dos operadores:
  - El operador de miembro de estructura " . " también conocido como operador punto.
  - El operador de apuntador de estructura " -> " también conocido como operador flecha.

Operador Punto

- El operador de miembro de estructura tiene acceso a un miembro de estructura mediante el nombre de la variable de estructura.
- Si quisiéramos imprimir los valores de la variable declarada en el ejemplo anterior:

```
printf("%s", var1.cadena);
printf ("%d", var1.n);
```


Operador de apuntador de estructura

- Consiste en un guion " - " seguido de in signo mayor que " > " sin espacios intermedios. Tiene acceso a un miembro de la estructura vía un apuntador a la estructura.

```
struct ejemplo var1={"Hola", 4}, *var2;
var2= &var1;
printf("%s\n",var2 -> cadena);
printf("%d\n",var2 -> n);
```

ESCOM

Uso de Typedef

- La palabra reservada **typedef** proporciona un mecanismo para la creación de sinónimos (o alias) para tipos de datos anteriormente definidos.
- Los nombres de los tipos de estructuras se definen a menudo utilizando typedef, a fin de crear nombres de tipo mas breves.

```
typedef int *p pint;
typedef struct ejemplo ejem;
```


ESCOM

Uso de Typedef

- Los programadores en C utilizan a menudo typedef para definir un tipo de estructura de tal forma que facilite su entendimiento.

```
typedef struct personaID
        {
                char *nombre;
                int edad;
                char *rfc;
        } ID;
```

```
struct personaID
        {
                char *nombre;
                int edad;
                char *rfc;
        } ;
typedef struct personaID ID;
```

El tipo abstracto de dato (TAD) Lista

Recordando

- Las operaciones más importantes que se realizan en las estructuras de datos son las de búsqueda, inserción y eliminación. Se utilizan también para comparar la eficiencia de las estructuras de datos y de esta forma observar cuál es la estructura que mejor se adapta al tipo de problema que se quiera resolver.

ESCOM

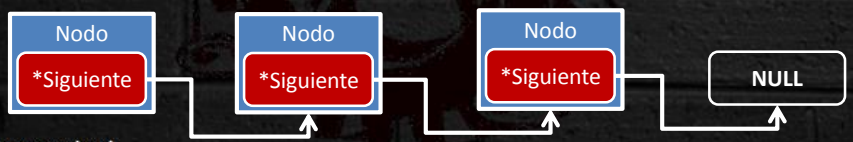
Definición de una Lista

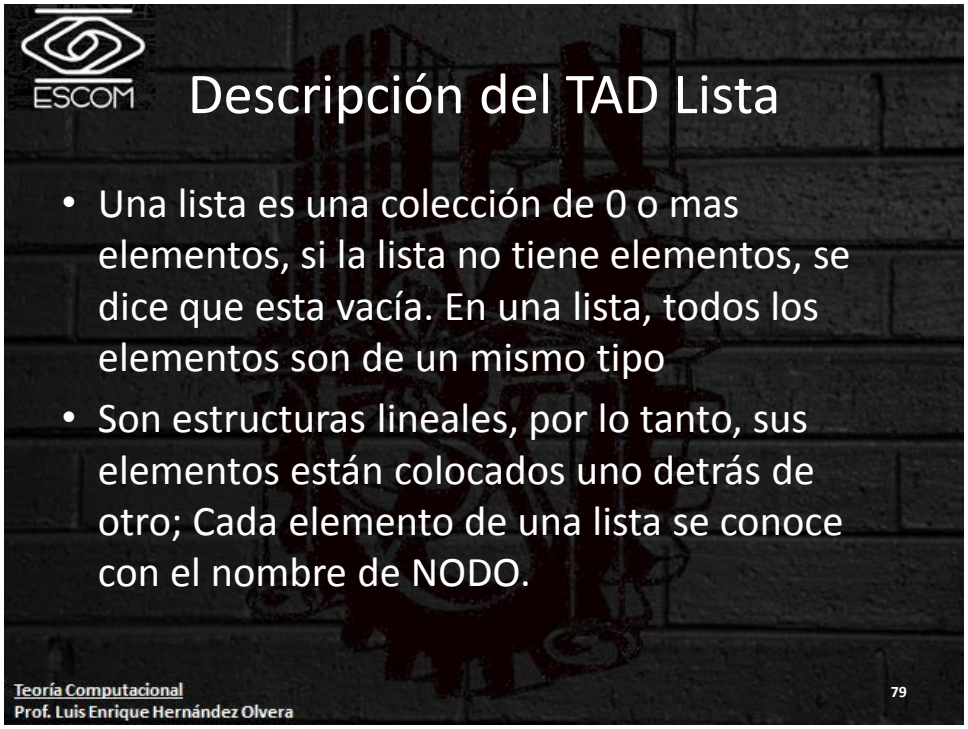
- Consiste en una secuencia de nodos, en los que se guardan elementos y una o dos referencias, enlaces o punteros al nodo anterior o posterior. El principal beneficio de las listas enlazadas respecto a los vectores convencionales o arreglos es que el orden de los elementos enlazados puede ser diferente al orden de almacenamiento en la memoria o el disco, permitiendo que el orden de recorrido de la lista sea diferente al de almacenamiento.

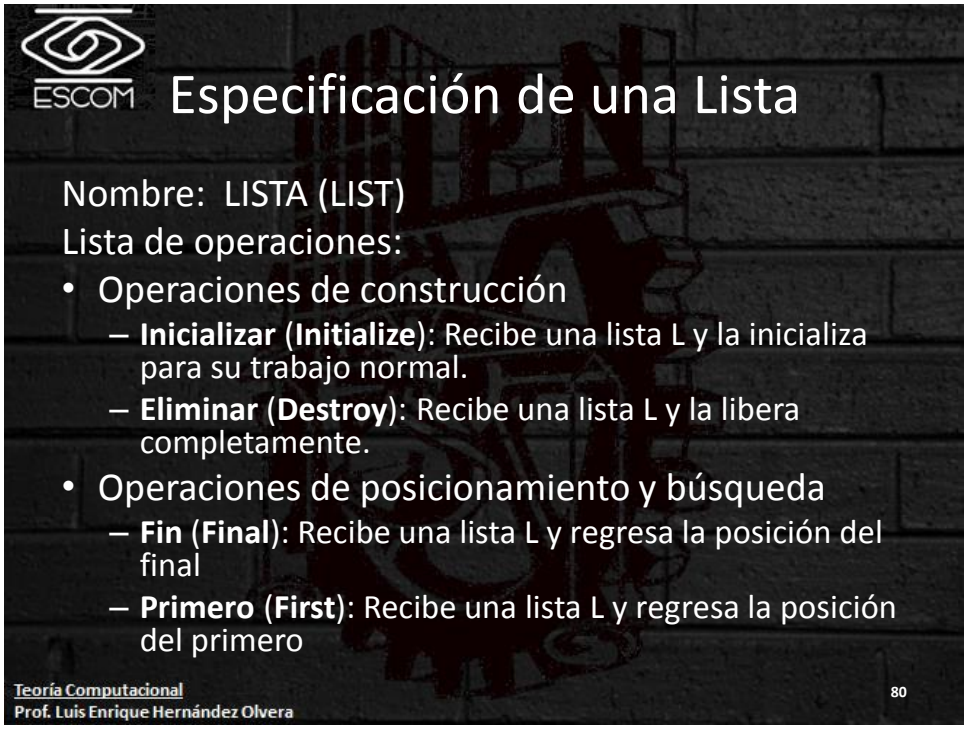
ESCOM

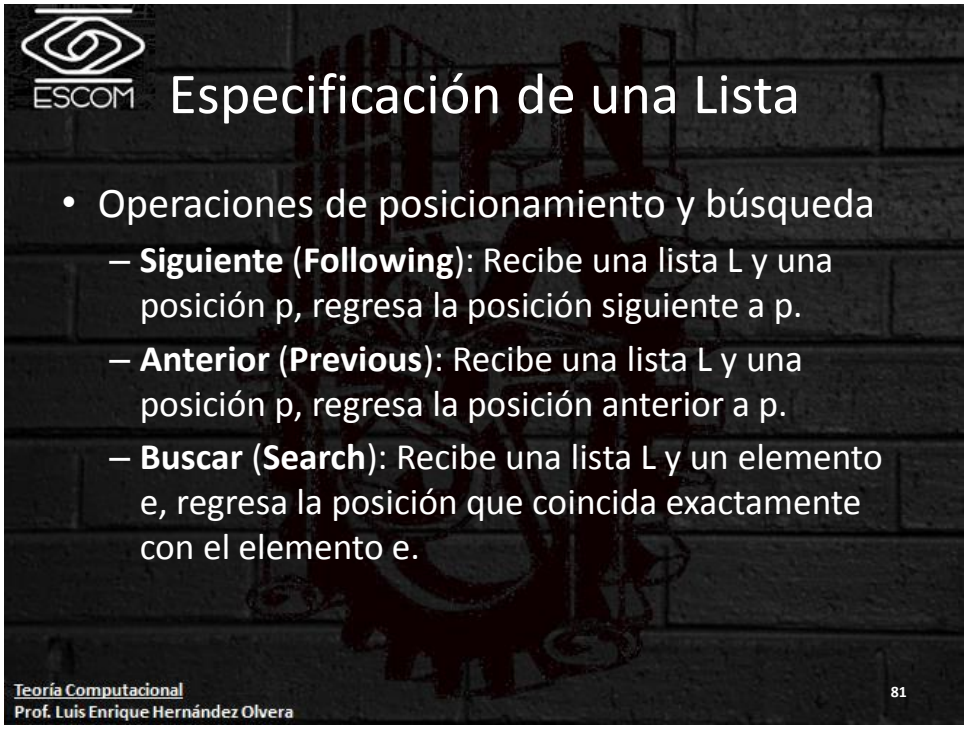
Listas

- Las ***listas*** *son la generalización de los dos TAD's (Pila y Cola): mientras que* en una pila y en una cola las operaciones sólo afectan a un extremo de la secuencia, en una lista se puede **insertar un elemento en cualquier posición**, y **borrar** y **consultar cualquier elemento**.



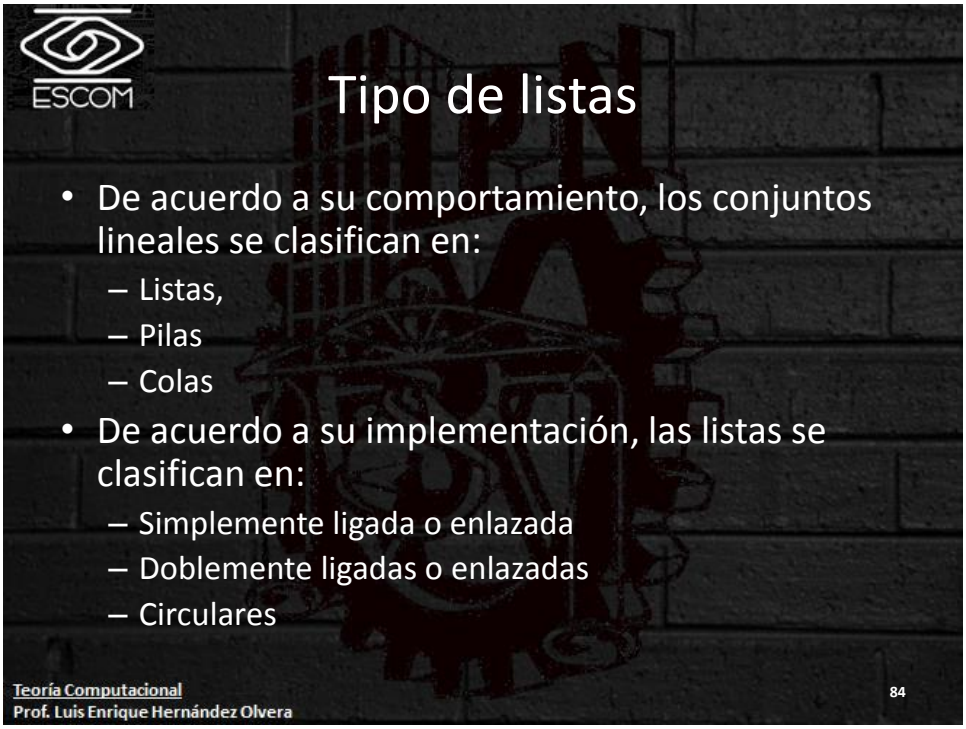

ESCOM
Descripción del TAD Lista
• Una lista es una colección de 0 o mas elementos, si la lista no tiene elementos, se dice que esta vacía. En una lista, todos los elementos son de un mismo tipo
• Son estructuras lineales, por lo tanto, sus elementos están colocados uno detrás de otro; Cada elemento de una lista se conoce con el nombre de NODO.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
79


ESCOM
Especificación de una Lista
Nombre: LISTA (LIST)
Lista de operaciones:
• Operaciones de construcción
– Inicializar (Initialize): Recibe una lista L y la inicializa para su trabajo normal.
– Eliminar (Destroy): Recibe una lista L y la libera completamente.
• Operaciones de posicionamiento y búsqueda
– Fin (Final): Recibe una lista L y regresa la posición del final
– Primero (First): Recibe una lista L y regresa la posición del primero
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
80


ESCOM
Especificación de una Lista
• Operaciones de posicionamiento y búsqueda
– Siguiente (Following): Recibe una lista L y una posición p, regresa la posición siguiente a p.
– Anterior (Previous): Recibe una lista L y una posición p, regresa la posición anterior a p.
– Buscar (Search): Recibe una lista L y un elemento e, regresa la posición que coincida exactamente con el elemento e.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
81

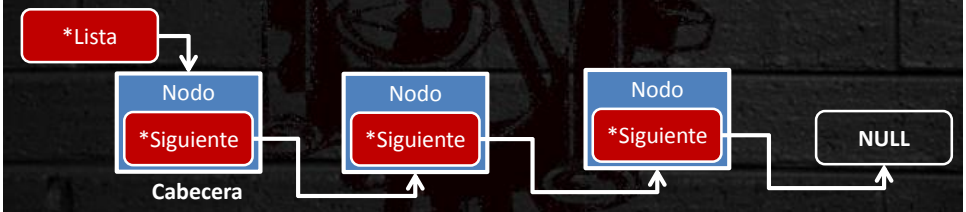

ESCOM
Especificación de una Lista
• Operación de consulta
– Posición (Position): Recibe una lista L y una posición p, devuelve el elemento en dicha posición.
– Validar posición (Validate Position): Recibe una lista L y una posición p, devuelve verdadero en caso de que en la lista L, p sea una posición valida (no nula).
– Tamaño (Size): Recibe una lista L y devuelve el tamaño de la lista
– Vacía (Empty): Recibe una lista L y devuelve verdadero en caso de que la lista l este vacía.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
82


ESCOM
Especificación de una Lista
• Operaciones de modificación
– **Inserta** (**Insert**): Recibe una lista L, un elemento e, una posición p y un valor booleano b e inserta al elemento e en la lista L enfrente de p si b es verdadero o atrás de p en caso de que b sea falso.
– **Agregar** (**Add**): Recibe una lista L y un elemento e, se inserta al elemento e al final de la lista L.
– **Remover** (**Remove**): Recibe una lista L y una posición p, y elimina al elemento en la posición p de la lista.
– **Sustituir** (**Replace**): Recibe una lista L, un elemento e, una posición p y sustituye al elemento ubicado en p por e.
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
83


ESCOM
Tipo de listas
• De acuerdo a su comportamiento, los conjuntos lineales se clasifican en:
– Listas,
– Pilas
– Colas
• De acuerdo a su implementación, las listas se clasifican en:
– Simplemente ligada o enlazada
– Doblemente ligadas o enlazadas
– Circulares
Teoría Computacional
Prof. Luis Enrique Hernández Olvera
84

Listas simplemente ligadas

- Se define como un conjunto de nodos, uno detrás de otro del cual siempre se puede conocer al nodo **inicial** y al **final.**
- De cada nodo de la lista, se conoce:
  - Un contenido, que es la información que almacena dentro puede ser de cualquier tipo de dato
  - Un sucesor único, excepto el ultimo nodo de la lista.



Listas simplemente ligadas vs Arreglos

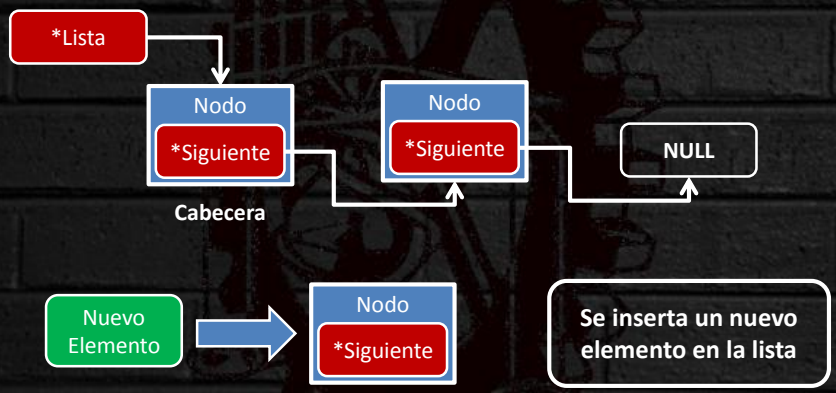
- Los arreglos tienen una longitud fija
- Para expandirlos es necesario crear un nuevo arreglo (de longitud mayor), y copiar el contenido del viejo arreglo al nuevo.
- Una mejor solución es usar apuntadores entre los elementos del arreglo.
- Cada elemento apunta al elemento siguiente.
- Esto es una lista simple (lista ligada).

Listas simplemente ligadas

- El modelo conceptual es como una "cadena" nuevos eslabones pueden insertarse al principio o al final, e incluso en cualquier otra posición con algo mas de recursos. Las extracciones pueden hacerse simplemente rompiendo la cadena.
- Ventajas:
  - **Se usa únicamente el espacio necesario.**
  - No es necesario conocer de antemano que tan grande será la estructura.
  - Es dinámica.

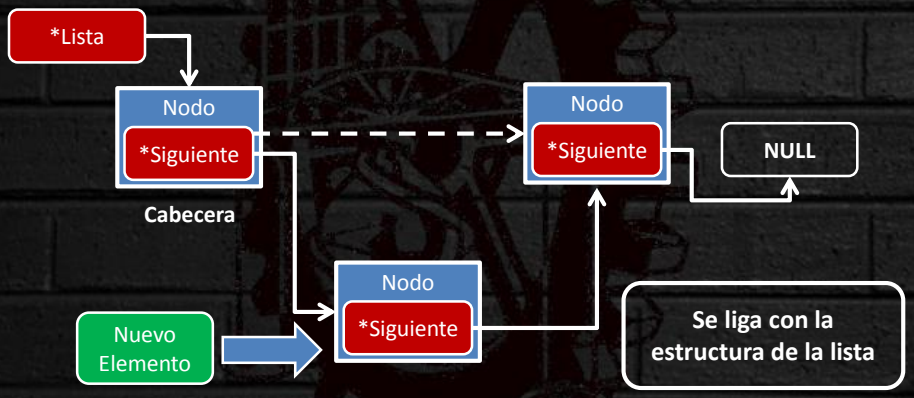
Operaciones básicas de una lista simplemente ligada

- Insertar un nuevo elemento a la lista:



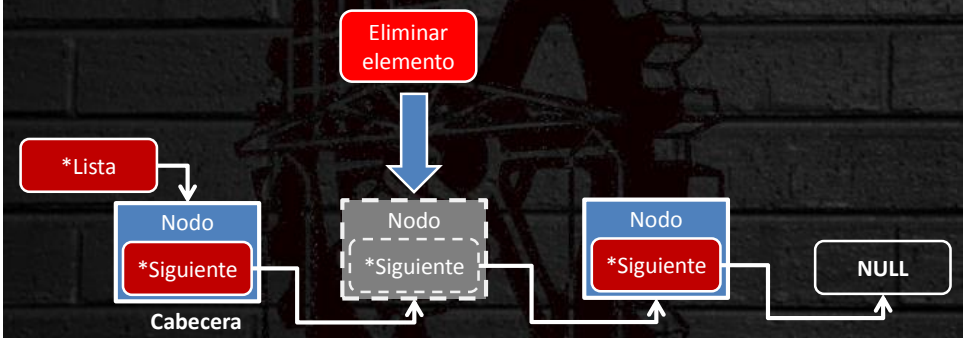
Operaciones básicas de una lista simplemente ligada

- Insertar un nuevo elemento a la lista:



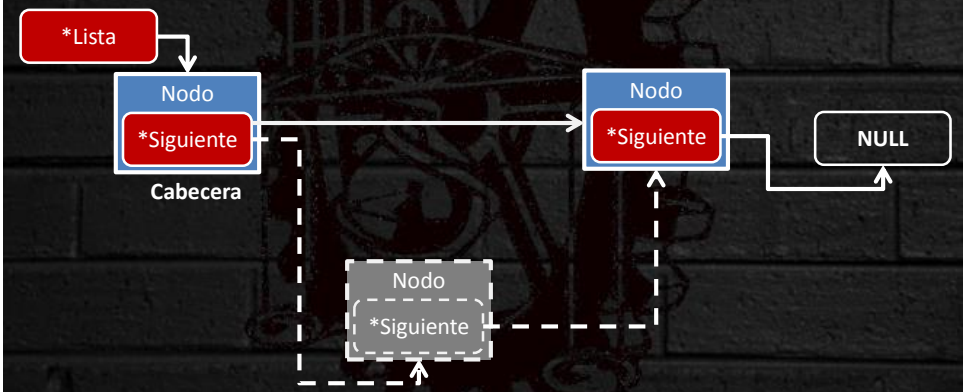
Operaciones básicas de una lista simplemente ligada

- Eliminar elemento de la lista:



Operaciones básicas de una lista simplemente ligada

- Eliminar elemento de la lista:



Recordando

- Los elementos de la lista **NO** necesariamente están en localidades de memoria adyacentes.
- Para tener acceso a la lista únicamente es necesario conocer la localidad de memoria donde comienza (Cabecera).

ESCOM

Preguntas?