Tema 3

Bibliotecas en C

Breve historia del lenguaje c

Escrito por **Dennis Ritchie**

Algunas características que nos interesa

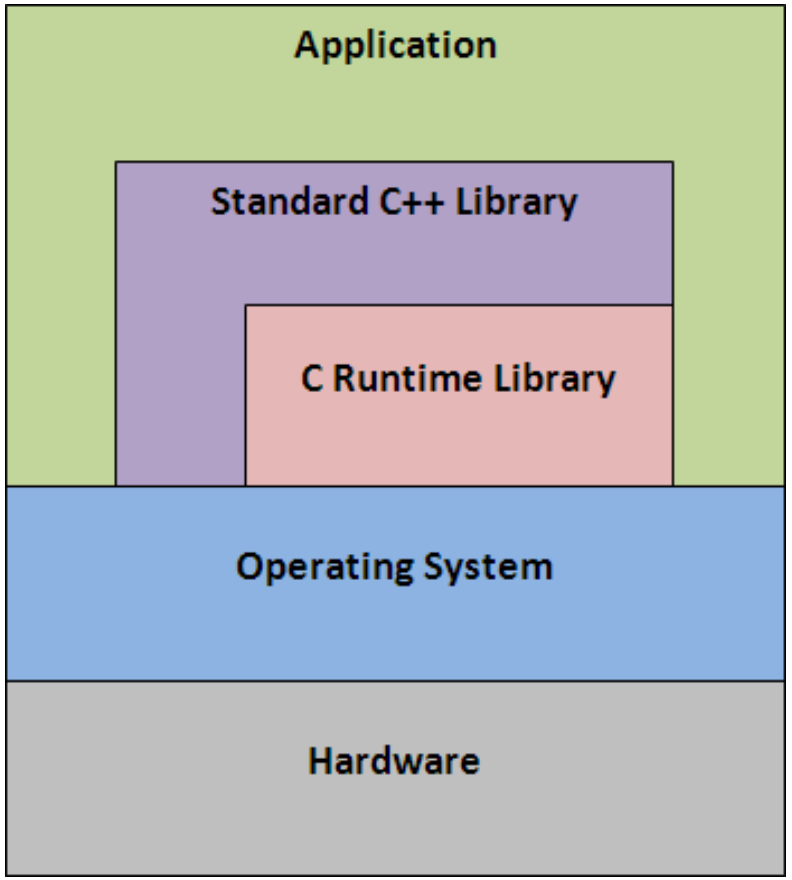
- Es un lenguaje muy flexible que permite programar con múltiples paradigmas.
- Acceso a memoria de bajo nivel mediante el uso de **punteros**.
- Un conjunto reducido de palabras clave.
- Código portable por medio de las **biblioteca estándar de C**

Bibliotecas en lenguaje c

Una **biblioteca** es un conjunto de elementos (funciones, clases, tipos predefinidos, constantes, variables globales, macros, etc) que es posible utilizar en un programa para facilitar la implementación de ese programa.

Bibliotecas en lenguaje c

La **biblioteca estándar de C** (también conocida como **stdio. h**) es una recopilación de archivos de cabecera y bibliotecas con rutinas, estandarizadas por un comité de la Organización Internacional para la Estandarización (ISO), que implementan operaciones comunes, tales como las de entrada y salida o el manejo de cadenas.



https://es.wikipedia.org/wiki/Biblioteca_est%C3%A1ndar_de_C

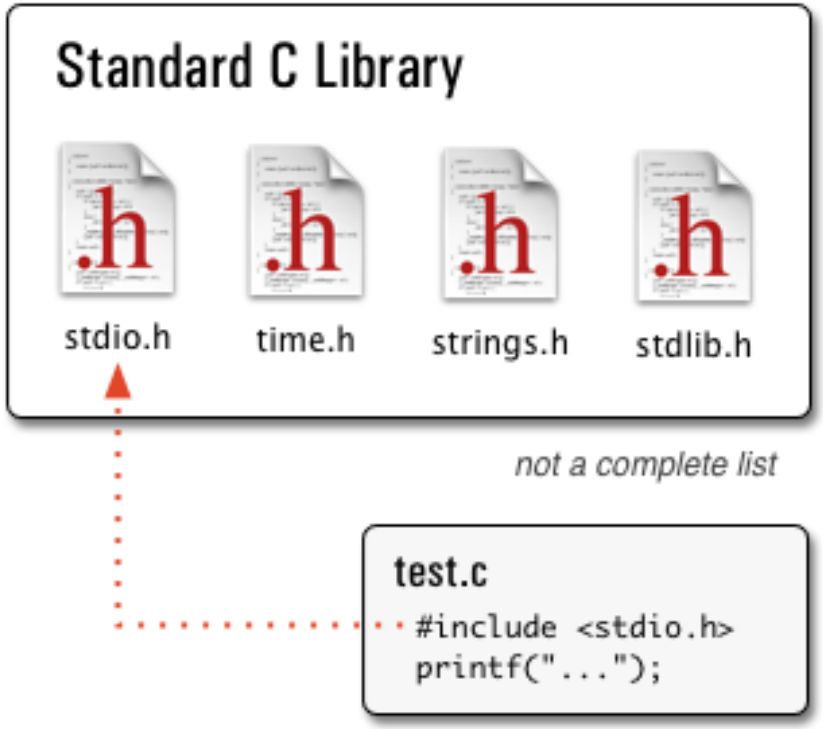
Bibliotecas populares en c

Algunas de las cabeceras más importantes de la librería estándar de C

<string.h>	Para manipulación de cadenas de caracteres.
<stdio.h>	Proporciona el núcleo de las capacidades de entrada/salida del lenguaje C.
<stdlib.h>	Para realizar ciertas operaciones como conversión de tipos, generación de números pseudo-aleatorios, gestión de memoria dinámica, control de procesos entre otras
<math.h>	Contiene las funciones matemáticas comunes.
<time.h>	Para tratamiento y conversión entre formatos de fecha y hora.

Bibliotecas populares en c

Incluyendo Cabeceras de la Librería Estándar de C en nuestro software



stdio.h

El Header del archive stdio.h da soporte para Standard Input Output. Tiene información relacionada a las funciones Input/output. Algunas de estas funciones ubicadas en stdio.h son:

Función	Descripción
printf()	It is used to print the strings, integer, character etc on the output screen.
scanf()	It reads the character, string, integer etc from the keyboard.
getc()	It reads the character from the file.
putc()	It writes the character to the file.
fopen()	It opens the file and all file handling functions are defined in stdio.h header file.
fclose()	It closes the opened file.
remove()	It deletes the file.
fflush()	It flushes the file.

Especificador de formato.

- %d o %i: Especifican un entero con signo.
- %u : Especifican un entero sin signo.
- %p : Dirección de un puntero u dato.
- %lld : Especifica un entero largo (long long). [*]
- %llu : Entero largo sin signo (unsigned long long) [*]
- %s : Especifica que el parámetro es un puntero a un arreglo de caracteres.
- %c : Un carácter.
- %x : Especifica un valor hexadecimal.
- %% : Muestra un literal de porcentaje.
- %f : Imprime un float o double.

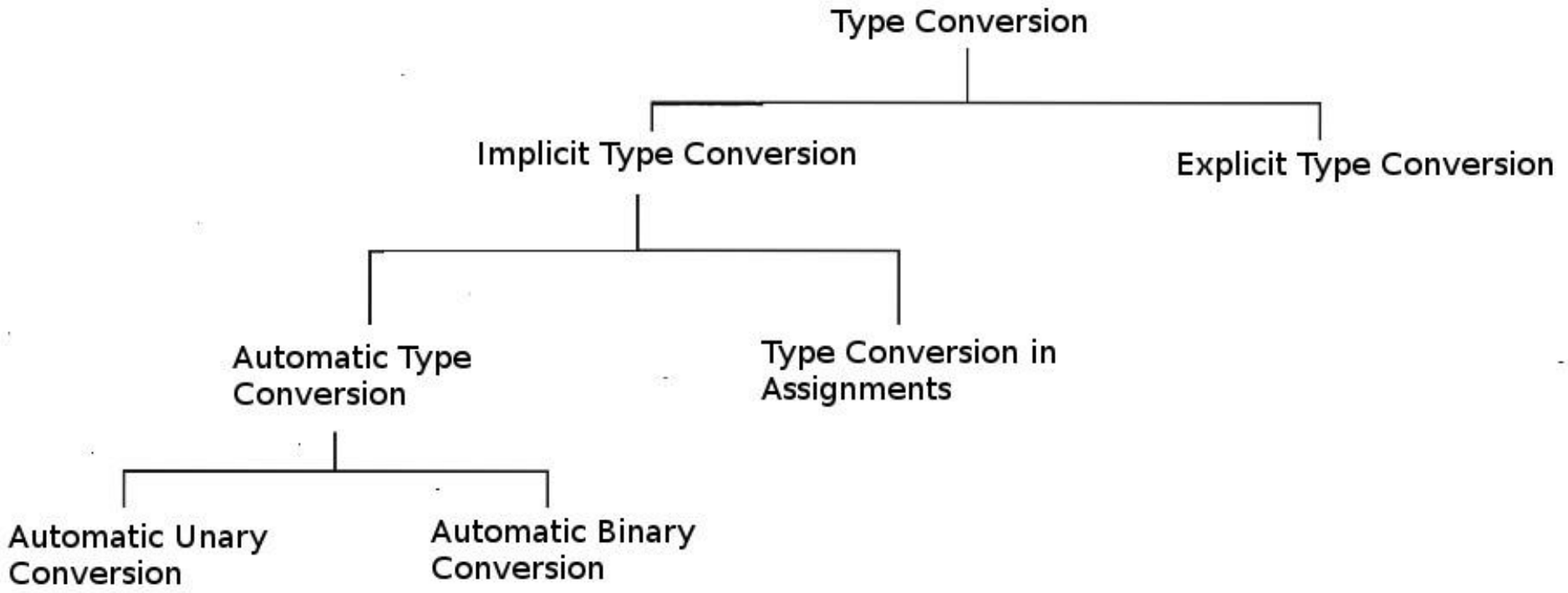
¿Cuál es la diferencia entre %i y %d?

https://www.geeksforgeeks.org/difference-d-format-specifier-c-language/

Tema 4
CAST’s

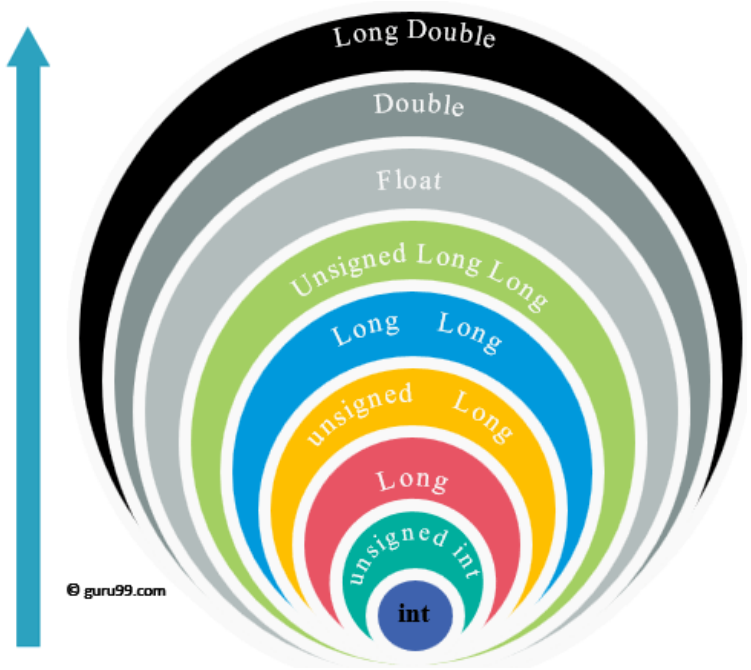
Cast’s

Conversión de un tipo de datos en otro tipo de datos.



Cast’s – Conversiones Implicitas de tipos (Implicit Type Conversions)

- La conversión Implícita de tipos también se denomina conversión de **tipo estándar**.
- No requerimos **ninguna palabra clave** o declaraciones especiales en el tipo implícito de conversión.
- La conversión de tipo implícita siempre ocurre con los tipos de **datos compatibles**.

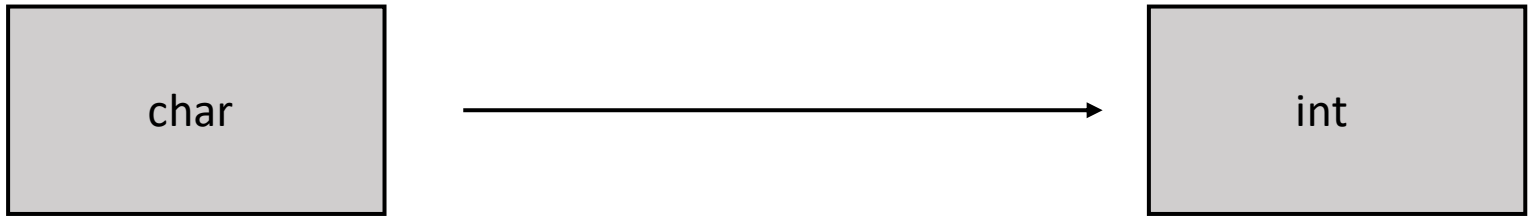


Fuente Imagen: https://www.guru99.com/c-type-casting.html

Cast’s – Conversiones Implicitas de tipos (Implicit Type Conversions)

Conversión en Unaria / binaria

- Las conversiones de tipo implícito las **realizan el compilador**.
- Los compiladores de C realizan estas conversiones de acuerdo con las **reglas predefinidas** del lenguaje C.

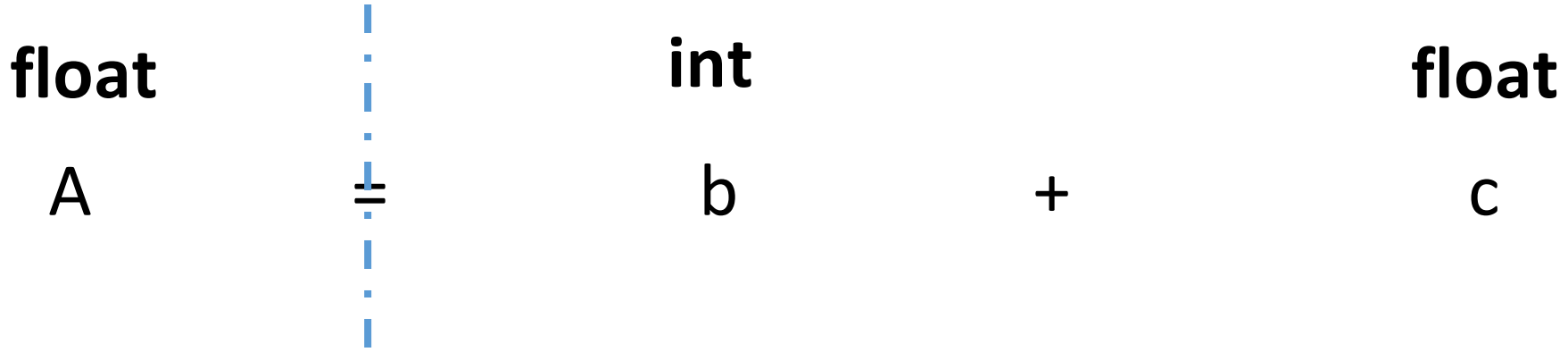


ejemplo: 1 y 2

Cast’s – Conversiones Implicitas de tipos (Implicit Type Conversions)

Conversión en asignación

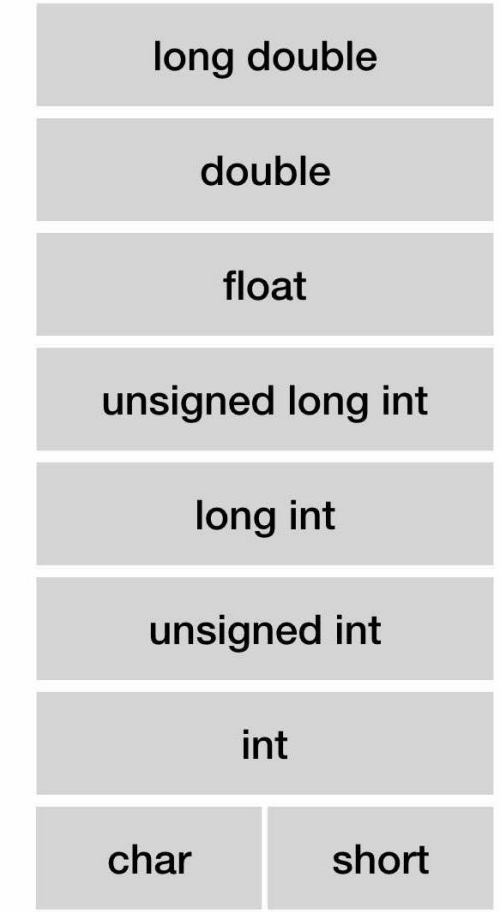
- Considere una expresión de asignación, donde los tipos de dos operandos son diferentes.
- El tipo de datos del operando en el lado derecho se convertirá en el tipo de datos del operando en el lado izquierdo. Durante esta conversión de tipo, promoción o degradación de operando en el lado derecho.



Ver ejemplo: 3 y 4

Cast’s – Conversiones Implicitas de tipos (Implicit Type Conversions)

Conversión en asignación



Promoción

Cuando la conversión se da de un tipo de rango inferior a rango superior se llama Promoción.

A = b + c → Si A es un tipo > c entonces c convierte al tipo de A y opera

Degradación

Cuando la conversión se da de un tipo de rango superior a rango inferior se llama Degradación.

A = b + c → Si A es un tipo < c entonces c convierte al tipo de A y opera

Ver ejemplo: 3 y 4

Cast’s – Conversiones Implicitas de tipos (Implicit Type Conversions)

Consecuencias de las conversiones

1. Algunos bits de orden superior pueden descartarse cuando long se convierte en int, o int se convierte en short int o char.

2. La parte fraccional puede truncarse durante la conversión de tipo flotante a tipo int.

3. Cuando el tipo doble se convierte en tipo flotante, los dígitos se redondean.

4. Cuando un tipo con signo se cambia a tipo sin signo, el signo se puede quitar.

5. Cuando un int se convierte en flotante, o flotante al doble, no habrá aumento en la precisión.

La programación 'C' proporciona otra forma de conversión de tipos que es la **conversión explícita** de tipos.

Cast’s – Conversiones Explicitas de tipos (Explicit type casting)

En la conversión de tipo implícito, el tipo de datos se convierte automáticamente. Hay algunos escenarios en los que tendremos que forzar la conversión de tipos. Por ejemplo en una división de dos operandos de tipo int que dan como resultado un float.

Sintaxis: **(tipo) expresión**

Ejemplo:

```
int a = 1;
int b = 2;
float b = a / b; // b debería valer 0,5 pero….
```

Cast’s – Conversiones Explicitas de tipos (Explicit type casting)

Pasando en limpio, algunos casos:

(int) 100.23223
Aquí la constante 100,23223 se convierte al tipo entero.
Por lo tanto, la parte fraccional se pierde y el resultado final será 100.

(float) 100/3
Aquí la constante 100 se convierte al tipo float, luego se divide por 3. El valor obtenido es 33.33.

(float) (100/3)
Aquí, al principio, 100 se divide por 3 y luego el resultado se convierte en tipo float. El resultado será 33.00.

(doble) (x + y -z)
Aquí, en la expresión anterior, el resultado de la expresión x + y-z se convierte en doble.

(doble) x + y-z
Aquí, en la expresión anterior, x se convierte primero en doble y luego se usa en la expresión.

Cast’s – Problemas de tipos

https://youtube.com/shorts/xxGr3T8tDwE?feature=share

8bits

128	64	32	16	8	4	2	1	Decimal
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1	255

Un bug en el juego Civilization 1:

- El “nivel de agresividad” de gandy, en el juego era 1
- Cuando Gandhi elegía la democracia restaba -2 al

nivel de agresividad. Dado que usaron un byte para representar el nivel de agresividad y si restamos -2 a 1 el resultado (-1) no se puede representar por lo tanto pasa al máximo superior y se vuelve a “255” (el máximo nivel de agresión del juego).
Conclusión: Gandhi el personaje más pacífico del juego, te declaraba la guerra nuclear.

Tema 5
Arreglos

Arreglos

- Un **arreglo** es un conjunto de datos.
- Mas precisamente: es una **estructura de datos homogéneos** que se encuentran ubicados en **forma consecutiva en la memoria RAM**.

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
1	3	5	7	9	11	13	15	17	19

Arreglos

Declaración **<Tipo>** nombre [dimensión];

Ejemplo **int** Buff[10];

int Buff [] = { 0, 3, 5, 7, 9, 11, 13, 15, 17, 19};

Buff [3] → 7

Arreglos

```
int Buff []   = { 0, 3, 5, 7, 9, 11, 13, 15, 17, 19};
```

Organización en memoria de la arreglo

posición →	[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
valor →	0	3	5	7	9	11	13	15	17	19
memoria →	1000	1004	1008	1012	1016	1020	1024	1028	1032	1036

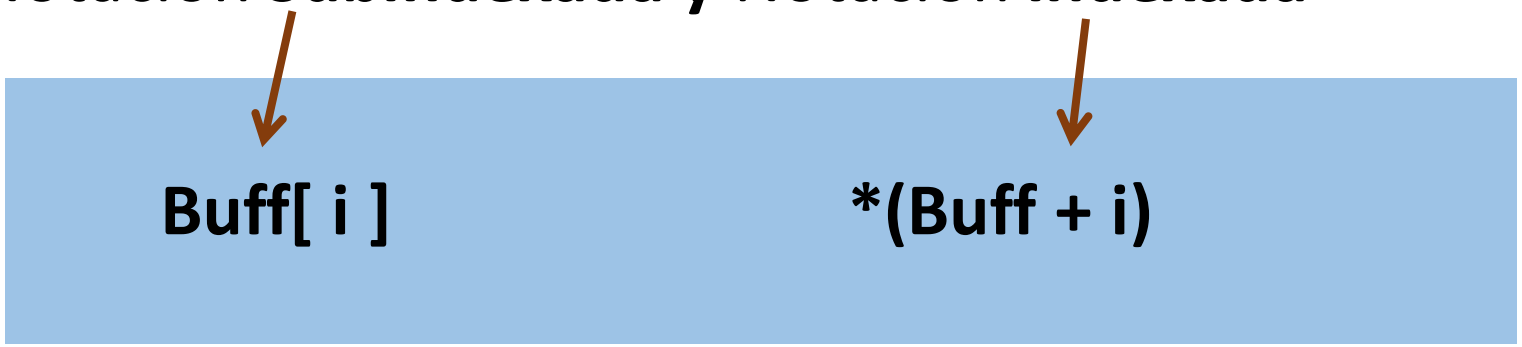
¿Arreglos y punteros?

Arreglos y punteros

- Para “C” los **arreglos** son un caso particular de ***puntero***.
- Los maneja implícitamente como tales.
- El nombre de un **arreglo** es equivalente a la **dirección** del primer elemento **&arreglo[0]**
- Todas las operaciones que utilizan vectores e índices pueden realizarse mediante punteros.
- Cuando declaramos: `int Buff[DIM];`
- **Buff** contienen la dirección de **Buff[0]**, entonces es un **puntero.**

ARREGLOS Y PUNTEROS

Notación **Subindexada y** Notación **Indexada**



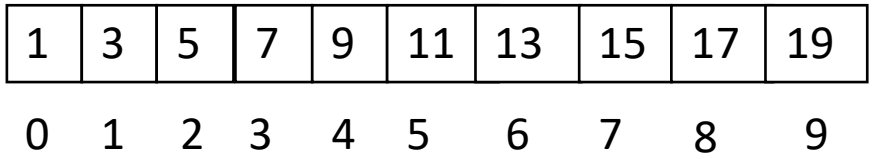
- En la notación **subindexada**, tratamos a **Buff** como **arreglo**.
- En la notación **indexada**, consideramos como **puntero** a **Buff**.

Buf++ => Buff = Buff + 1*sizeof (tipo del puntero)
Buff + n => Buff + n*sizeof (tipo del puntero)

ARREGLOS Y PUNTEROS

int Buff[10];

Buff[0]



- No es necesario utilizar el **operador de dirección** para apuntar al primer elemento de un **arreglo**.

</> Código

```
int* p;  //puntero a entero
int Buff[10], x;
// dirección del 1er. elemento
p = &Buff[0];          //equivale a    p = Buff;
x = Buff[0] ;          //equivale a    x = *p;
Buff[1]                //equivale a    *(Buff + 1)
&Buff[i]               //equivale a    Buff + i
```

EJEMPLOS

```
int v[10], *p;
```

p = &v[0]	*Apunta a la posición inicial del vector*
p + 0	*Apunta a la posición inicial del vector*
p + 1	*Apunta a la 2ª posición del vector*
p + i	*Apunta a la posición **i+1** del vector*
p = &v**[9]**;	*Apunta a la última posición del vector*
p - 1	*Apunta a la novena posición del vector*
p - i	*Se refiere a la posición **9 – i** del vector*

ARITMÉTICA DE PUNTEROS

- Los siguiente casos son válidos:

```
int v[10],
Int *pENT;
pENT = &v[0]
```

Notación	Significado
pENT++	Incrementa en 1 la dirección.
pENT—	Decrementa en 1 la dirección.
++(*pENT)	Incrementa lo referenciado y luego utiliza dicho valor.
++(*pENT++)	Incr. lo referenciado, utiliza y vuelve a incrementar la dirección.
*pENT++	Utiliza lo referenciado e incrementa la dirección.
*(pENT++)	Usa lo referenciado e incrementa dirección.
(*pENT)++	Utiliza lo referenciado e incrementa lo referenciado.
*(pENT+i)	Utiliza lo referenciado y luego incrementa en “i” la dirección

Aritmética de Punteros. **Diferencia importante**

pENT++ ó pENT+ = i

- Modifica el contenido del puntero.

pENT[i] ó *(pENT+ i)

- No modifica el contenido
- Continúa apuntando a la dirección original.

Clasificación de punteros

Tipificados

- Aritmética de punteros (++ / -- / + = k / etc.)
- Referencia datos específicos (int * / char * / etc.)
- Permite operaciones aritméticas (operandos)

No-Tipificados (void *)

- Son usados en las **reservas dinámicas**.
- **No** admiten aritmética de punteros.

ARREGLOS Y PUNTEROS

int Buff[5];

Buff[0]

1	3	5	7	9

int

0 1 2 3 4

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

void main()
{
        int Buff [5] = {5,15,30,10,35};
        int *p;
        p= Buff;
  ....
}
```

ACTIVIDAD

- Acceda a los elementos mediante notación subindexada
- Acceda a los elementos del array mediante el puntero p con aritmética de punteros
- Acceda a los elementos del array mediante el arreglo con aritmética de punteros
- obtener las direcciones de memoria y mostrarlas por pantalla mediante arismética de punteros
- ¿Que tamaño tiene el arreglo Buff y el puntero? Muestre por pantalla.

Cadena de caracteres

- Los elementos del tipo caracter (tipo **char** en C) se pueden agrupar para formar secuencias que se denominan **cadenas de caracteres**.

- En un programa en C, las **cadenas** se delimitan por dobles comillas: **“CH?*$A7!”** y “**soy cadena**”.

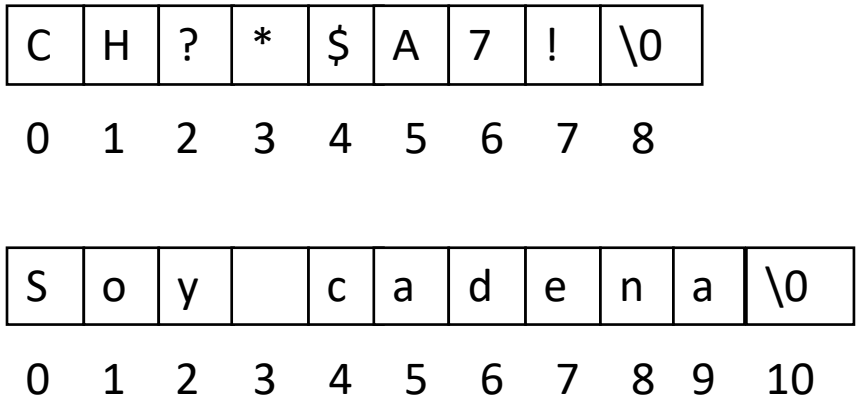
- En la memoria RAM una cadena se guarda en un **arreglo de tipo caracter**, de tal manera que cada símbolo de la cadena ocupa una **casilla** del arreglo.

- Se utiliza una casilla adicional del arreglo para guardar un **carácter especial** que se llama **terminador** de cadena: **´\0´** .

- Este carácter especial indica que la **cadena** termina.

Cadena de caracteres

- Las dos cadenas del ejemplo anterior se representan en la memoria de la computadora:



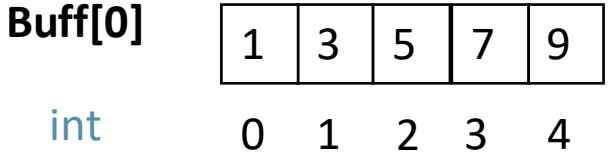
IMPORTANTE

La **longitud** de una cadena se define como el número de símbolos que la componen, **sin contar el terminador de cadena**.

El **terminador de cadena** no forma parte de la cadena, pero sí ocupa una casilla de memoria en el arreglo.

ARREGLOS Y PUNTEROS

int Buff[5];




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

void main()
{
        char Buff [] =“cadena”;
        char Buff2 [7] ;
        ….
}
```

ACTIVIDAD

- Copie desde una cadena hacia otra.
- Muestre por pantalla
- Indique por pantalla el tamaño de las cadena
- ¿Que tamaño tiene buff y buff2?

TIP: Investigue la función **strcpy** c Funcionamiento y aplicación

Arreglos Multidimensionales

Declaración **TipoDeVariable** nombreDelArray [dimensión1] [dimensión2] [...] [dimensiónN]

Ejemplo **int** MiMatriz [3] [4];

```
Int MiMatriz  [3][4] =
{
      { 1 ,  2 ,  3 ,  4},
      { 5 ,  6 ,  7 ,  8},
      { 9 , 10 , 11 , 12}
};
```

Arreglos Multidimensionales

```
Int MiMatriz  [3][4] =
{
        { 1 ,  2 ,  3 ,  4},
        { 5 ,  6 ,  7 ,  8},
        { 9 , 10 , 11 , 12}
};
```

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 10 & 11 & 12 \end{pmatrix}$$

Organización en memoria de la matriz

valor	1	2	3	4	5	6	7	8	9	10	11	12
Posición en memoria	1000	1004	1008	1012	1016	1020	1024	1028	1032	1036	1040	1044

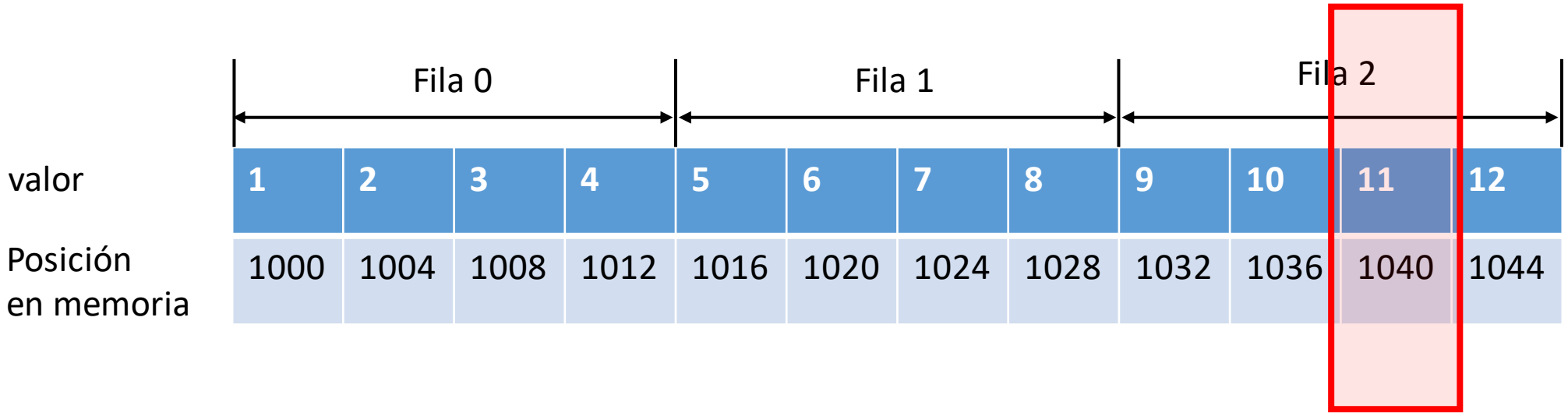
Arreglos Multidimensionales

MiMatriz[i][j] = *(MiMatriz + (i * numerodeColumnas + j))

```
Int MiMatriz  [3][4] =
{
        { 1 ,   2 ,   3 ,   4},
        { 5 ,   6 ,   7 ,   8},
        { 9 , 10 , 11 , 12}
};
```

```
// value at num[2][3] donde,  i = 2 y j = 3

num[2][3] =  *(ptr + (i * no_of_cols  + j))
          =  *(ptr + (2 * 4 + 3))
          = *(ptr + 11)
```



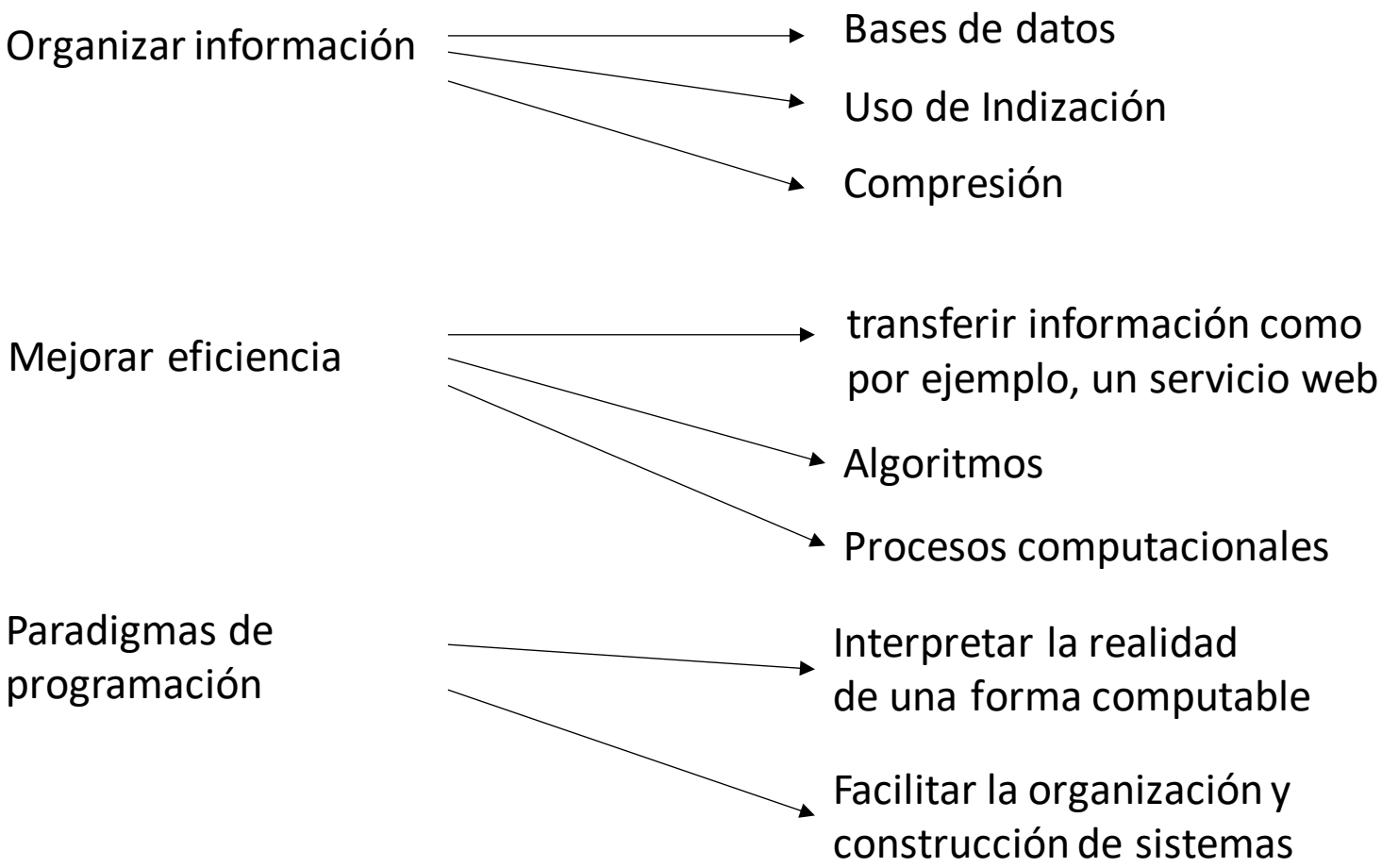
Tema 6
Estructuras

Estructuras

definiciones

Según Wikipedia

[…] Una **estructura de datos** es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.[…]



Estructuras

definiciones

Struct

- Un **Struct (o estructuras)** son colecciones de variables relacionadas bajo un nombre.
- Los **Struct** pueden contener variables de muchos tipos diferentes de datos - a diferencia de los arreglos que contienen únicamente elementos de un mismo tipo de datos.
- Se utilizan para crear tipos de datos definidos por el usuario (UDT).
- Son útiles si se desea que una única variable contenga varios datos relacionados.
- Sus miembros pueden ser también otras estructuras.

Estructuras

declaraciones

Struct

- Se las declara en la zona general del programa a partir de la palabra reservada **Struct** entre **{ }**
- Una **estructura** debe tener al menos un **elemento**.
- Declarar cada **elemento** y especificar el **tipo de datos** del mismo**.**

```
struct TRectang
{
        int Alto;
        int Ancho;
        char Letra;
 };
```

Sus miembros

Tipos definidos por el usuario

typedef es una palabra reservada en el lenguaje de programación C y C++.
Su función es asignar un nombre alternativo a tipos existentes, a menudo cuando su declaración normal es extensa, potencialmente confusa o probablemente variable de una implementación a otra.

Sintaxis

```
typedef Tipo Identificador;
```

Ejemplo

```
typedef int ValorEntero;

ValorEntero NotasDeAlumnos = 10;
```

Tipos definidos por el usuario

Forma de uso

Palabra reservada

Tipo

```
Typedef   struct Rectang
{
        int Alto;
        int Ancho;
        char Letra;
} TRectang;
```

Alias para el tipo

DECLARACIÓN DE
UNA ESTRUCTURA
(Tipo definido por el usuario)
Con Typedef para poner un alias.

Tipos definidos por el usuario

Forma de uso

```
struct Rectang
{
        int Alto;
        int Ancho;
        char Letra;
 };
```

DECLARACIÓN DE
UNA ESTRUCTURA
(Tipo definido por el usuario)

```
Typedef  struct Rectang  Trectang;
```

Uso de TypeDef para
Poner un alias a la estructura

Palabra reservada	Tipo	Alias para el tipo

Bibliografía

https://es.stackoverflow.com/questions/59590/cual-es-el-uso-del-operador-en-printf-de-variables-en-lenguaje-c

https://es.wikipedia.org/wiki/Estructura_de_datos

https://emertxe.com/blog/2018/02/type-promotion-in-c.html

Referencia bibliográfica

https://es.stackoverflow.com/questions/23239/que-diferencia-existe-entre-api-biblioteca-y-framework

•

•**API** viene del inglés "Application programming interface" que significa "Interfaz para programación de aplicaciones". Es la parte de una biblioteca a la que accede un programa que usa la biblioteca; haciendo así el uso de la biblioteca independiente de los detalles de implementación. Una API puede ser implementadas por distintas bibliotecas.

Otra definición: API (Application Programming Interface), is a tool that lets software developers make integrations between apps.

•Un **framework** es un conjunto integrado de herramientas que facilitan un desarrollo software. Puede incluir APIs y bibliotecas. Pero también puede incluir otros elementos como herramientas de depuración, diseño gráfico, prototipado, edición, etc.