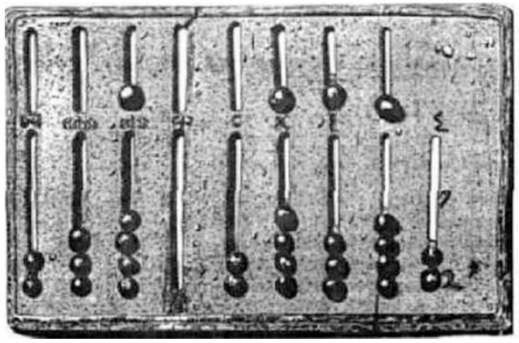
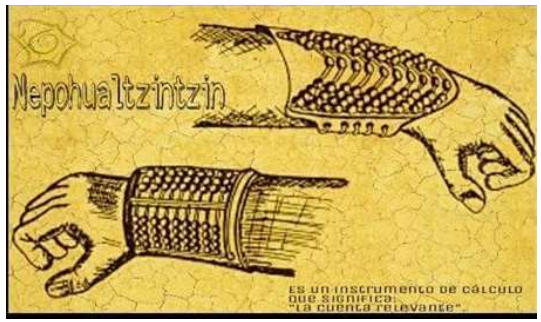
TEMA 1.
Estructura de datos

1.0 Antecedentes y Generalidades

- La historia de la computación tiene sus orígenes en el inicio mismo de la civilización.
- La búsqueda de nuevos métodos para realizar cálculos ha evolucionado desde la antigüedad.





1.0 Antecedentes y Generalidades

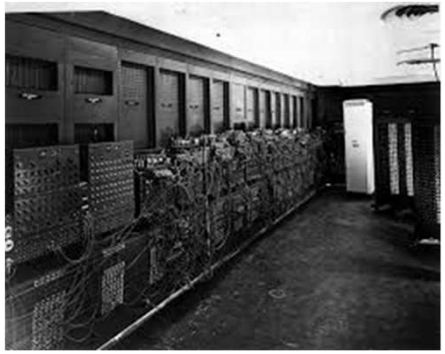
- Una computadora digital es un dispositivo electrónico, utilizado para procesar información y obtener resultados, capaz de ejecutar cálculos a velocidades considerablemente más rápidas de lo que pueden hacerlo los seres humanos.
- Componentes físicos:
  - CPU
  - DISPOSITIVOS E/S
  - MEMORIA

1.0 Antecedentes y Generalidades

- La computación como disciplina nace a principios de 1940 con base en la teoría de algoritmos, la lógica matemática y la aparición del concepto de **programa**.
- Un programa es un conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora

1.0 Antecedentes y Generalidades

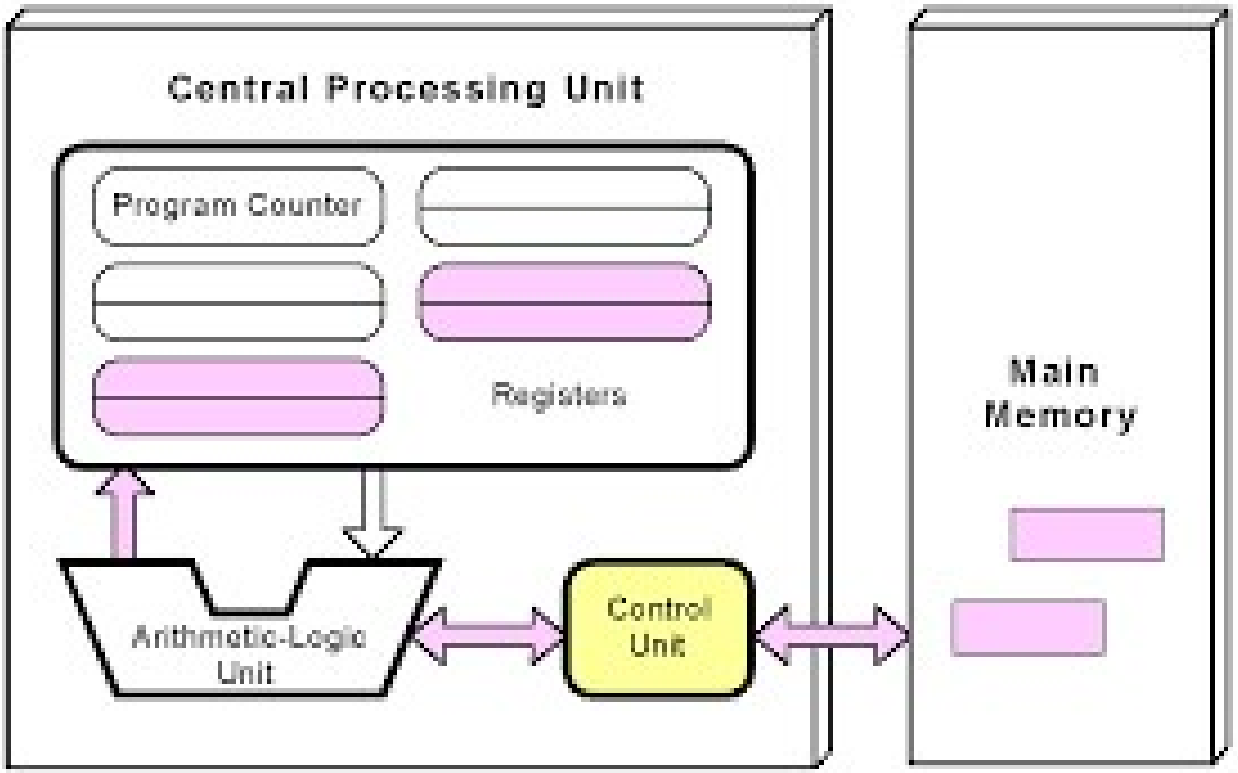
- John von Neumann propuso un esquema con una computadora con una memoria para almacenar datos y programas.
- Este modelo se conoce como arquitectura von Neumann y es el modelo que se sigue utilizando en las computadoras actuales.



https://histinf.blogs.upv.es/2011/12/05/proyecto-eniac/

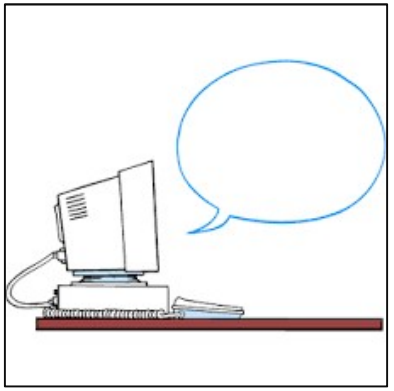
https://es.wikipedia.org/wiki/John_von_Neumann

1.0 Antecedentes y Generalidades



1.0 Antecedentes y Generalidades

- Un **lenguaje de programación** es un lenguaje artificial diseñado para expresar instrucciones que pueden ser llevadas a cabo por máquinas como las computadoras.
  - Lenguajes de alto nivel
  - Lenguajes de bajo nivel



1.0 Antecedentes y Generalidades

Clasificaciones de Lenguajes de Programación

- Manera de Ejecutarse
  - ✓**Compilados**. Código fuente > código objeto > Programa ejecutable
  - ✓**Interpretados**. Un programa ejecuta las instrucciones de manera directa
- Manera de Abordar la tarea
  - ✓**Imperativos**. Indican como hay que hacer la tarea
  - ✓**Declarativos**. Indican qué tarea hay que hacer sin embargo no indica cómo realizarla

1.0 Generalidades

Clasificaciones de Lenguajes de Programación

- De acuerdo al paradigma de Programación
  - ✓ **Procedural.** Divide el problema en partes más pequeñas.
  - ✓ **Orientado a Objetos**. Sistema de clases y objetos siguiendo el mundo real.
  - ✓ **Funcional**. Evaluación de funciones.
  - ✓ **Lógica.** Tareas expresadas empleando lógica formal matemática.

1.1 Representación de datos

- La información que se procesa en una computadora tiene diferentes tipos de datos
- Para el manejo adecuado de datos en una computadora digital, se implementó una representación uniforme de los datos.
- Esta representación se conoce como Patrón de Bits

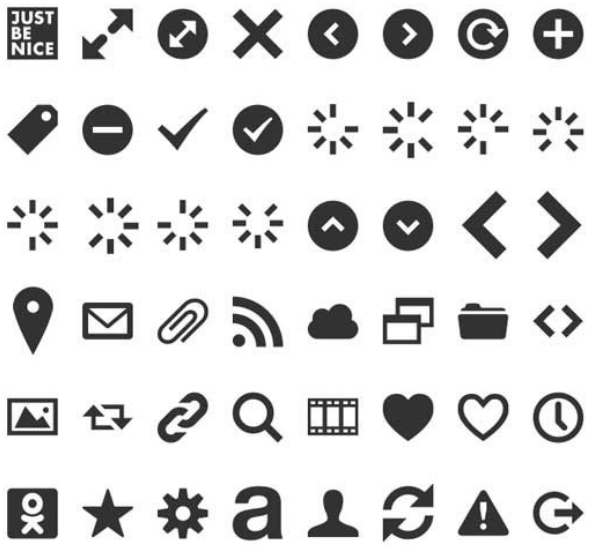
1.1 Representación de datos

- BIT: Es la unidad mínima de almacenamiento en las computadoras y representa dos estados.

- BYTE: Es un patrón de 8 bits mediante el cual se mide el tamaño de una memoria o de otros dispositivos de almacenamiento

1.1 Representación de datos

- ¿Cuántos bits se necesitan en un patrón para representar un símbolo?



1.1 Representación de datos

Numero de Símbolos	Longitud del Patrón de Bits
2	1
4	2
8	3
16	4
…	…
128	7
256	8
…	…
65 536	16

1.1.1 Tipos de datos

- Existen diversas cadenas de bits que se establecen como secuencias de patrones para representar símbolos de texto, numéricos, etc.
- El proceso de representar los símbolos se conoce como codificación.

Texto

- Los diferentes estándares para representar texto han evolucionado conforme a la necesidad de representar una mayor cantidad de símbolos.

ASCII

- Código Norteamericano de Estándares para Intercambio de Información (1967)
- (American Standard Code for Information Interchange).
- Este código utiliza siete bits para cada símbolo.
- **ASCII extendido**: Para hacer que el tamaño de cada patrón sea de 1 byte (8 bits), a los patrones de bits ASCII se les aumenta un cero mas a la izquierda.
- Cada patrón cabe fácilmente en un byte de memoria

ASCII

USASCII code chart

b_7 b_6 b_5 → / Bits ↓				0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
b_4	b_3	b_2	b_1	Row ↓ Column →	0	1	2	3	4	5	6	7
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

ASCII EXTENDIDO

128	Ç	144	É	160	á	176	▒	193	┴	209	╤	225	ß	241	±
129	ü	145	æ	161	í	177	▒	194	┬	210	╥	226	Γ	242	≥
130	é	146	Æ	162	ó	178	▓	195	├	211	╙	227	π	243	≤
131	â	147	ô	163	ú	179	│	196	─	212	╘	228	Σ	244	⌠
132	ä	148	ö	164	ñ	180	┤	197	┼	213	╒	229	σ	245	⌡
133	à	149	ò	165	Ñ	181	╡	198	╞	214	╓	230	µ	246	÷
134	å	150	û	166	ª	182	╢	199	╟	215	╫	231	τ	247	≈
135	ç	151	ù	167	º	183	╖	200	╚	216	╪	232	Φ	248	°
136	ê	152	_	168	¿	184	╕	201	╔	217	┘	233	Θ	249	·
137	ë	153	Ö	169	_	185	╣	202	╩	218	┌	234	Ω	250	·
138	è	154	Ü	170	¬	186	║	203	╦	219	▉	235	δ	251	√
139	ï	156	£	171	½	187	╗	204	╠	220	▄	236	∞	252	_
140	î	157	¥	172	¼	188	╝	205	═	221	▌	237	ϕ	253	²
141	ì	158	_	173	¡	189	╜	206	╬	222	▐	238	ε	254	■
142	Ä	159	ƒ	174	«	190	╛	207	╧	223	▀	239	∩	255	
143	Å	192	└	175	»	191	┐	208	╨	224	α	240	≡		

EBCDIC

- Extended Binary Coded Decimal Interchange Code (EBCDIC)
- Representa caracteres alfanuméricos, controles y signos de puntuación.
- Cada carácter está compuesto por 8 bits, define un total de 256 caracteres.
- Es un código estándar usado por computadoras mainframe IBM.

EBCDIC

Bits 3210	7654 0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111
0000	NUL	DLE	DS	IRS	SP	&	-	ø	Ø	°	µ	¢	ä	å	E	0
0001	SOH	DC1	SOS	ITB	RSP	'	/	\	a	j	ü	£	A	J	÷	1
0010	STX	DC2	FS	SYN	á	é	A	E	b	k	s	¥	B	K	S	2
0011	ETX	TM	WUS	IR	{	è	#	E	c	l	t	·	C	L	T	3
0100	PF	RES	BYP	PN	à	ê	A	E	d	m	u	©	D	M	U	4
0101	HT	NL	LF	RS	á	ì	A	I	e	n	v	[	E	N	V	5
0110	LC	BS	ETB	UC	ã	î	A	I	f	o	w	¶	F	O	W	6
0111	DEL	IL	ESC	EOT	}	I	$	I	g	p	x	¼	G	P	X	7
1000	GE	CAN	SA	SBS	ç	ì	Ç	I	h	q	y	½	H	Q	Y	8
1001	SPS	EM	SFE	IT	ñ	ß	N	è	i	r	z	¾	I	R	Z	9
1010	SMM	CC	SM	RFF	§	¤	ö	:	«	ª	¡	-	SHY	¹	²	³
1011	VT	CU1	CU2	CU3	.	Á	,	A	»	º	¿	]	ò	û	O	U
1100	FF	IFS	MFA	DC4	<	*	%	O	ð	æ	Ð	—	¦	~	@	U
1101	CR	IGS	ENQ	NAK	(	)	_	'	ý	¸	Ý	]	ò	ù	O	U
1110	SO	IRS	ACK		+	;	>	=	Þ	Æ	þ	`	ó	ú	O	U
1111	SI	IUS	BEL	SUB	!	^	?	"	±	]	®	×	õ	ÿ	O	EO

UNICODE

- Este estándar es mantenido por el Unicode Technical Committee (UTC) mantiene estrecha relación con ISO/IEC, alcanzando el acuerdo de sincronizar sus estándares que contienen los mismos caracteres y puntos de código.
- En la actualidad soporta tres formatos para representar millones de caracteres.
  - UTF-8
  - UTF-16
  - UTF-32

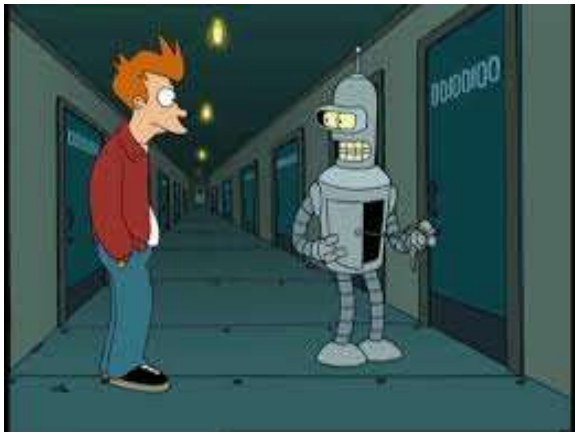
  https://unicode-table.com/es/

Números Enteros

- Los enteros de la computadora se representan mediante agrupaciones de dígitos binarios, se tienen las siguientes formas de representación:
  - SIN SIGNO
  - CON SIGNO
    - SIGNO Y MAGNITUD
    - COMPLEMENTO A 1
    - COMPLEMENTO A 2

Enteros sin signo

- También conocido como **binario convencional o binario puro**, permite representar enteros positivos y el cero.
- El intervalo de números que puede representar depende del número de bits disponibles



Enteros con signo y magnitud

- Se utiliza de la misma manera que la representación convencional, la diferencia radica en que en este tipo de representación se destina el primer bit para indicar el signo del número
  - 0, número positivo
  - 1, número negativo
- Debido a que se utiliza un bit para identificar el signo, se reduce la cantidad de valores que se pueden representar

Números reales

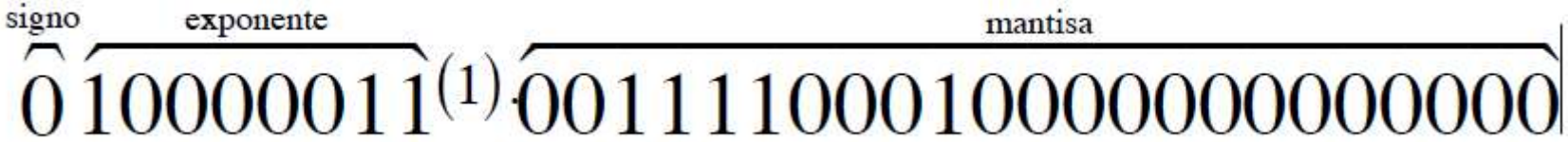
- Debido a que las computadoras tienen un número finito de bits. No pueden almacenar los números reales en forma exacta, de forma similar a lo que ocurre con los números irracionales (pi, e, etc.) por lo tanto se utilizan aproximaciones.
  - Punto fijo
  - Punto flotante

Punto Fijo

- Consiste en destinar una cantidad de dígitos para la parte entera y el resto para la parte fraccionaria.
- La cantidad de dígitos destinados a la parte fraccionaria indica la posición del punto dentro del número.
- La limitante es que dependiendo de los dígitos que se asignen a la parte fraccionaria disminuye el número máximo para la parte entera

Notación de punto flotante

- Esta notación se encuentra definida en la convención IEEE 754.
- Cada número de precisión simple ocupa 32 bits
- El número se construye de la siguiente forma
  - 1 bit para el signo
  - 8 bits para el número correspondiente al exponente
  - 23 bits para la parte fraccionaria de la mantisa.



Notación de punto flotante

- Para encontrar la representación decimal de un número binario en punto flotante de precisión simple se debe realizar lo siguiente:
  1. Verificar el signo del número. (a)
  2. Convertir los 8 dígitos del exponente a decimal
  3. A ese número restar la constante E=127. El número obtenido será el exponente para el número 2 (b)
  4. Convertir los dígitos de la mantisa de la misma forma que se hace en notación de punto fijo.
  5. Al número obtenido anteponer un 1 antes del punto para obtener un número de la forma 1.xxxx (c)
  6. Multiplicar los 3 números obtenidos (a x b x c)

Notación de punto flotante

- De esta manera un número está representado por 3 cantidades:

$$num. = (-1)^{signo} \times 2^{exponente-E} \times 1.mantisa$$

Donde:

signo = primer dígito

exponente = siguientes 8 dígitos

E= 127

Notación de punto flotante

- Para encontrar la representación binaria en punto flotante de precisión simple a partir de un número decimal se realiza el siguiente procedimiento:
  1. Verificar el signo del número.
  2. Convertir la parte entera del número decimal de la forma convencional.
  3. Recorrer el punto decimal **n** posiciones hasta obtener una notación de la forma 1.xxxx
  4. A la constante E = 127, sumarle **n**
  5. Convertir el número obtenido a binario.

Notación de punto flotante

6. Concatenar los bits obtenidos en el paso 1, el paso 5, los bits restantes del paso 3.
7. El número correspondiente a la parte real se debe multiplicar por 2 de manera sucesiva, los bits que se deben considerar son la parte entera del resultado de dicha multiplicación.
8. El proceso termina cuando ya se han llenado los 32 bits o cuando al realizar la multiplicación se obtiene un cero

Notación de punto flotante

- En doble precisión se utilizan 64 bits de la siguiente forma
  - 1 bit para el signo
  - 11 bits para el exponente
  - 52 bits para la parte fraccionaria
  - La constante E = 1023

1.1 Tipos primitivos

- Los tipos primitivos son los tipos de datos básicos que maneja cada lenguaje de programación para representar datos y almacenar valores.
- El manejo de los datos es una *capa de abstracción* que proporciona el lenguaje y que hace transparente para el usuario el manejo de este tipo de datos en los programas.

1.1 Tipos primitivos

Tipo	Representación	Tamaño (bits)
char	Carácter	8
int	Entero con signo	32
short	Entero con signo	16
long	Entero con signo	32
float	Punto flotante simple	32
double	Punto flotante doble	64
unsigned	Entero positivo	32

1.1 Tipos primitivos (java)

Tipo	Representación	Tamaño (bits)
bool	True or false	1
char	Carácter	16
byte	Entero con signo	8
short	Entero con signo	16
int	Entero con signo	32
long	Entero con signo	64
float	Punto flotante	32
double	Punto flotante	64

1.1.2 Arreglos

Definición

- Un arreglo es una colección o conjunto de variables del mismo tipo que se asocian a un nombre común
- Los arreglos constan de posiciones de memoria contiguas.
- Pueden tener una o varias dimensiones.
- Son entidades estáticas debido a que se conservan del mismo tamaño a lo largo de la ejecución del programa

1.1.2 Arreglos

Uso de Arreglos

- El manejo de los elementos de un arreglo se realiza con índices.
- Los índices indican la posición del elemento dentro del arreglo.
- En el índice es posible utilizar variables, constantes o expresiones aritméticas **.**

1.1.2 Arreglos

Declaración:

```
tipoDeDato identificador[numeroElementos];
```

```
int c[20];
int arreglo[5]
char letra[10];

int[20] c;   //ERROR
```

1.1.2 Arreglos

Acceso a elementos en Arreglos

```
int arreglo[20];
arreglo[20] = 8;    // ERROR
int a = 2;
arreglo[a] = 2;
double b = 5.0;
arreglo[b]= 1;      // ERROR
arreglo[a*2]=7;
```

1.1.2 Arreglos

Acceso a elementos en Arreglos

El índice de un arreglo incluso se puede referenciar con el índice de otro arreglo

```
int arreglo1[];     // ERROR
arreglo1[0]=1;
arreglo1[1]=2;
arreglo1[2]=3;
int arreglo2[3];
arreglo2[arreglo1[1]]= 1;
```

1.1.2 Arreglos

Acceso a elementos en Arreglos

```
printf("El valor es: %d",a[2])
int suma = a[0] + a[3];
printf("resultado = %d",suma)
a[1] = 25;
int x = 5;
a[x] = 30;
```

1.1.2 Arreglos

Acceso a elementos en Arreglos

```
char letras[10];

letras[0]='a';

letras[1]='}';

letras[2]=64;
```

1.1.2 Arreglos

Inicialización:

```
int c[]={11,2,8};

int c[3]={11,2,8};

int c[8]={10,20,30};

char cadena[] = “Hola”;

char cadena[] = {'H','o','l','a’,\0};

char cadena[5] = “hola”

char cadena[15] =“adiós”
```

1.1.2 Arreglos

Declaración de arreglos (Java)

- byte[ ] edad = new byte[4];
- short[ ] edad = new short[4];
- int[ ] edad = new int[4];
- float[ ] estatura = new float[3];
- boolean[ ] estado = new boolean[5];
- char[ ] sexo = new char[2];
- String[ ] nombre = new String[2]

1.1.2 Arreglos

Arreglos Unidimensionales

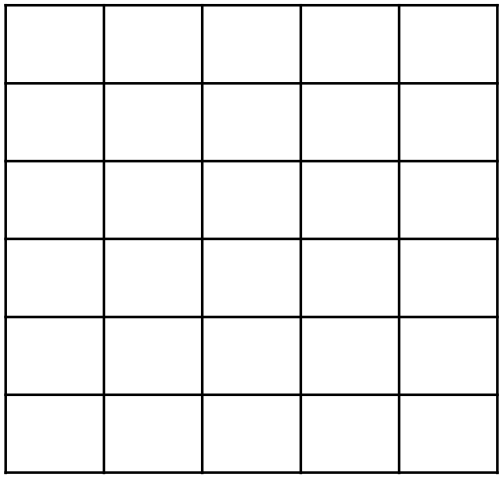
- Son los arreglos más simples y constan de un solo índice, también suelen llamarse vectores.
- Como los elementos se almacenan de forma adyacente, su representación en la memoria es simple

Dirección	Elemento
...	...
z	$elem\,(0)$
$z+1$	$elem\,(1)$
...	...
$z+n-1$	$elem\,(n-1)$
...	...

1.1.2 Arreglos

Arreglos Multidimensionales

- Estos arreglos constan de más de un índice, cuando son de dos dimensiones también se llaman matrices.



1.1.2 Arreglos Multidimensionales

Declaración:

```
int c[3][3];
int arr[10,3];      //error
int matriz[2][3]
int arreglo[5][3][1]
char letra[10][20][20][5];
```

1.1.2 Arreglos

- Dado un arreglo bidimensional, en la memoria se almacena ordenando primero los renglones y posteriormente las columnas

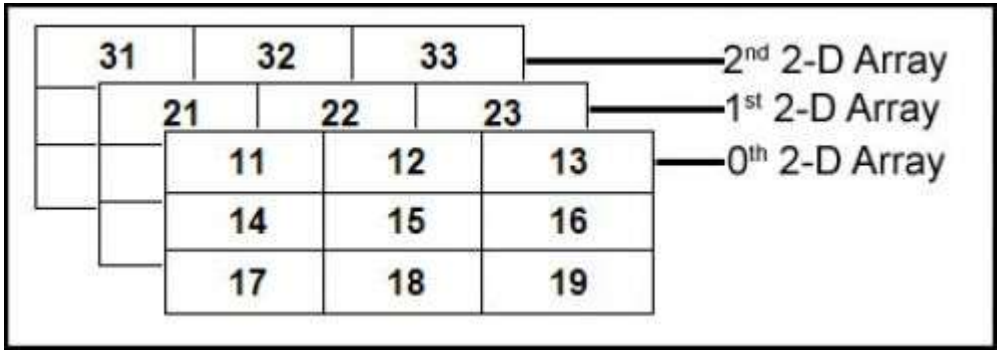
```
int a[3][2]
```

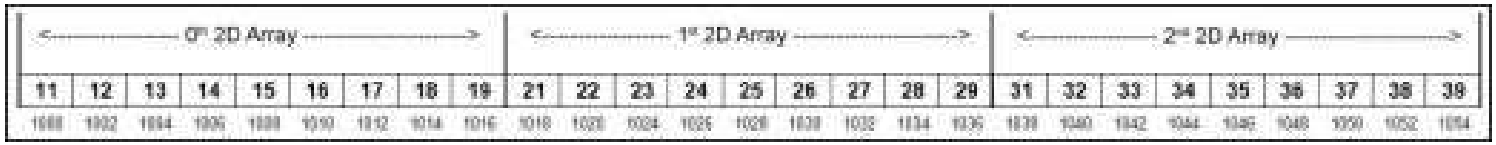
50	60
70	80
90	100

En memoria:

…
50
60
70
80
90
100
…

1.1.2 Arreglos





1.1.2 Arreglos Multidimensionales

Inicialización arreglos multidimensionales:

Existen 2 alternativas para inicializar arreglos multidimensionales

```
int c[3][3]={1,2,3,4,5,6,7,8,9};
int otro[3][3]= {{1,2,3},{4,5,6},{7,8,9}};

int unomas[2][4]={10,20,30,40,50,60,70,80};
int nuevo[2][4]={{10,20,30,40},{50,60,70,80}}
```

1.1.2 Arreglos

```
int a[5][5];

a[3][2]= 80;
a[2][3]= 80;
a[1][2]= 80;
a[1][4]= ?
a[4][4]= 200;
a[3][1]= 33
```

✘
✘
✓
100

10	20	30	40	50
60	70	80	90	100
110	120	130	140	150

1.1.2 Arreglos

Manejo práctico de arreglos

```
void main(){
     int i,arreglo[100];
     for (i=0;i<=99;i++){
          arreglo[i]=i;
     }
     return 0;
}
```

1.1.2 Arreglos

Manejo práctico de arreglos

```
void main(){
    int i,j,num=2
    int numeros[3][3];
    for (i=0;i<3;i++)
    {
        for (j=0;j<3;j++)
        {
            numeros[i][j]=num;
            num=num*2;
        }
    }
```

1.1.2 Arreglos

Lo esencial…

- Todos los elementos son del mismo tipo
- Son estáticos.
- Se almacenan en localidades de memoria consecutivas.
- El primer índice de un arreglo siempre es 0

Paso por valor / Paso por referencia

```
principal() {
    int a,b
    a = 5
    b = 10
    función (a,b)
    print (“a=%d y b=%d”,a,b)
}
void función(int a, int b){
    a=100
    b=200
}
```

Ejemplo

```
var i, j : integer;
procedure P(k, m : integer);
begin
    k := k - m;
    m := k + m;
    k := m - k;
end;
i := 2;
j := 3;
P(i, j);
```

Ejemplo

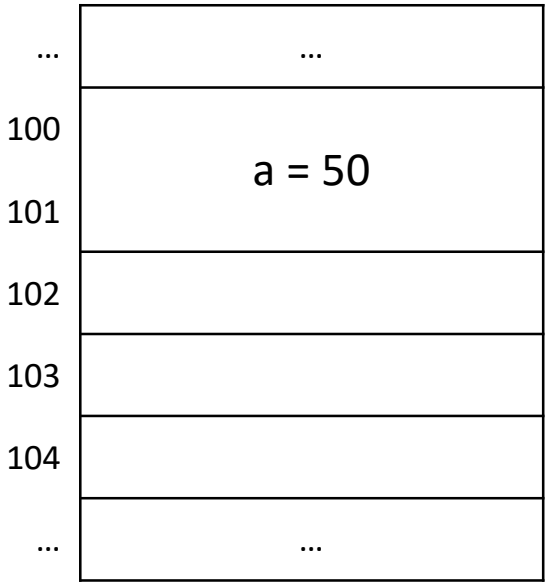
```
program main ()
begin
    integer a, b, c, i
    a = 6
    b = 7
    c = 8
    i = fun(a, b, c)
    print i, a, b, c
end

integer fun (integer x, integer y, integer z)
begin
    if (x > 6) then
        y = 25
    z = x + y
    return y + z
end
```

1.2.1 Apuntadores

- Un apuntador es un tipo especial de variable cuyo valor es una dirección de memoria.
- Esa dirección de memoria se refiere a otra variable , que a su vez, contiene un valor específico.
- Es decir, un apuntador hace referencia indirecta a un valor.
- Al proceso de hacer referencia a un valor a través de un apuntador se le conoce comúnmente como indirección

El operador dirección (&)

- Cuando se antepone este operador al identificador de una variable , se hace referencia a la dirección de memoria donde se almacena la variable.



int a = 50;

&a = ?

El operador indirección (*)

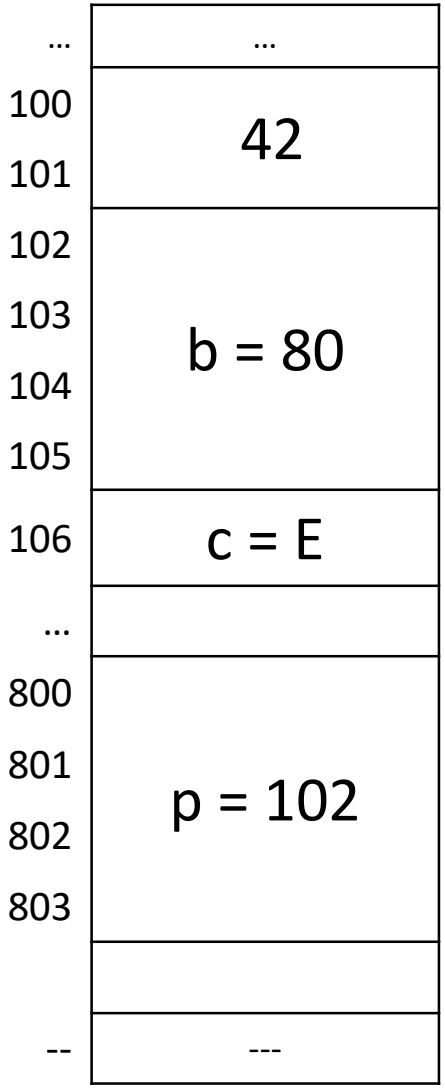
- Cuando se utiliza una variable de tipo apuntador se debe anteponer el operador * al identificador, de esta forma es posible acceder al espacio de memoria que este direcciona.

Declaración

```
[tipo dato] *[identificador];
```

En Memoria…

```
int a = 42;
long b = 80;
char c = "E";
long *p;
p = &b;
```



Ejemplo

```
#include <stdio.h>
int main(){
    int a = 50;
    int *p;
    p = &a;
    *p = 12;
    printf(" a = %d” , a);
  return 0;
}
```

Ejemplo

```
#include <stdio.h>
main() {
    int a=5, b=20, c=10, d;
    int *p1, *p2;
    p1=&c;
    p2=&b;
    d = *p1 + *p2;
    printf("suma %d\n", d);
    return 0;
}
```

Ejemplo

```
#include <stdio.h>
main() {
    int a=5, b=20, c=10, d;
    int *p1, *p2;
    p1=&c;
    p2=&b;
    *p1=100;
    *p2= 50;
    printf("suma %d\n", b+c);
    return 0;
}
```

Ejemplo

```
void main() {
    int x,*px;
    px = &x;
    *px = 0;
    printf("%d \n",x);
    *px = *px + 5;
    printf("%d \n",x);
    *px = *px * 2;
    printf("%d \n",x);
    *px = *px + 4;
    printf("%d \n",x);
    *px *= x * 2;
    printf("%d \n",x);
```

Apuntadores

- Los 3 elementos necesarios para uso de apuntadores
  - Declaración de la variable (tipo apuntador)
  - Asociación con una dirección de memoria
  - Asignación de valores