

# TEMA 3:

Fundamentos para la construcción de código a partir del algoritmo



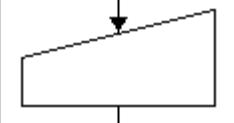
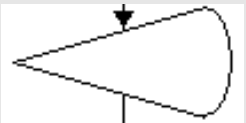


# TEMA 3:

## 3.2 Variables, tipos, expresiones y asignación

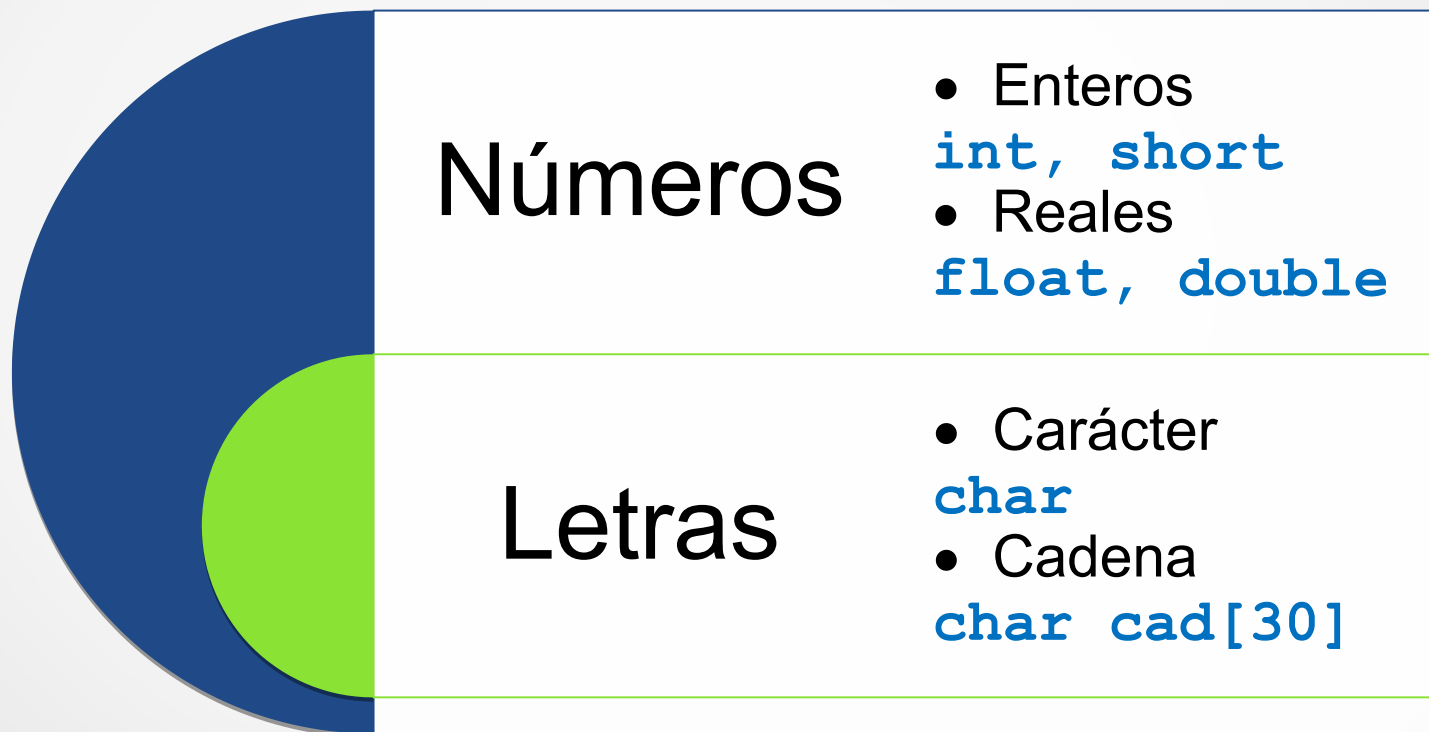
# Objetivo

- El alumno construirá programas utilizando el lenguaje de programación C a través de un análisis y modelado algorítmico previo.

# Pseudocódigo – DFD - C

ACCIÓN	DFD	LENGUAJE C
INICIO / ALGORITMO		<pre>int main(void) int main(int argc, char* argv[])</pre>
ASIGNAR / DECLARAR / DEFINIR		<pre>int x=0; float y=5.1;</pre>
LEER Y GUARDAR EN / LEER Y ALMACENAR EN		<pre>scanf("%d", &amp;x); scanf("%f", &amp;y); scanf("%d,%f", &amp;x, &amp;y);</pre>
IMPRIMIR / ESCRIBIR / MOSTRAR MENSAJE		<pre>printf("Hola Mundo"); printf("La suma es: %d", x);</pre>
CALCULAR / REALIZAR OPERACIÓN		<pre>z= x + y; suma= suma + conde;</pre>
FIN / FIN ALGORITMO		<pre>return 0;</pre>

# Tipos de datos



# Rango de tipos de datos

Tipo	Bits	Rango / Uso
<code>unsigned char</code>	8	$0 \leq X \leq 255$ Números pequeños y juego caracteres del PC.
<code>char (signed)</code>	8	$-128 \leq X \leq 127$ Números muy pequeños y juego de caracteres ASCII
<code>short (signed)</code>	16	$-32,768 \leq X \leq 32,767$ Números muy pequeños, control de bucles pequeños
<code>unsigned short</code>	16	$0 \leq X \leq 65,535$ Números muy pequeños, control de bucles pequeños
<code>unsigned int</code>	32	$0 \leq X \leq 4,294,967,295$ Números grandes
<code>int (signed)</code>	32	$-2,147,483,648 \leq X \leq 2,147,483,647$ Números pequeños, control de bucles <b>NOTA:</b> <i>Depende de la arquitectura puede tener el rango de un tipo short</i>

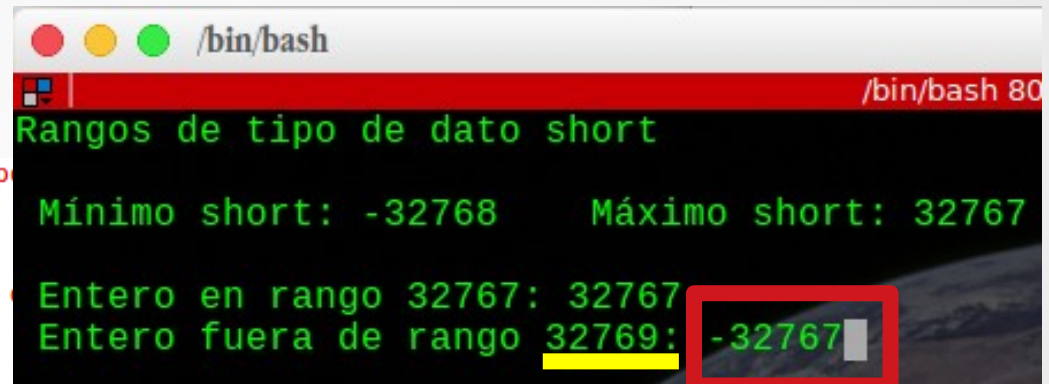
# Rango de tipos de datos

Tipo	Bits	Rango / Uso
<code>unsigned long</code>	32	$0 \leq X \leq 4,294,967,295$ Distancias astronómicas
<code>enum</code>	32	$-2,147,483,648 \leq X \leq 2,147,483,647$ Conjuntos de valores ordenados
<code>long (int)</code>	32	$-2,147,483,648 \leq X \leq 2,147,483,647$ Números grandes. <b>NOTA:</b> <i>Depende de la arquitectura puede tener un rango de -9223372036854775808 a 9223372036854775807</i>
<code>float</code>	32	$1.18e-38 \leq  X  \leq 3.40e38$ Precisión científica ( 7-dígitos)
<code>double</code>	64	$2.23e-308 \leq  X  \leq 1.79e308$ Precisión científica (15-dígitos)
<code>long double</code>	80	$3.37e-4932 \leq  X  \leq 1.18e4932$ Precisión científica (18-dígitos)

# Ejemplo de desborde de tipo short

- El tipo short puede almacenar un número dentro del rango:  $-32,768 \leq X \leq 32,767$ , si el número ingresado está fuera el tipo se desborda y el dato se pierde.

```
1  /*programa para mostrar los límites de tipo short
2  * hecho por huicho */
3
4  #include <stdio.h>
5  #include <limits.h> //para las constantes
6
7  int main(int argc, char* argv[])
8  {
9      short shortentero= 32767;
10     short shortenterodesbor= 32769;
11
12     printf("Rangos de tipo de dato short \n");
13
14     printf("\n M%cnimo short: %d", 161, SHRT_MIN);
15     printf("\t M%cximo short: %d", 160, SHRT_MAX);
16     printf("\n\n Entero en rango 32767: %d", shortentero);
17     printf("\n Entero fuera de rango 32769: %d", shortenterodesbor);
18
19     getchar();
20     return 0;
21 }
22
```



```
/bin/bash
Rangos de tipo de dato short
Mínimo short: -32768    Máximo short: 32767
Entero en rango 32767: 32767
Entero fuera de rango 32769: -32767
```

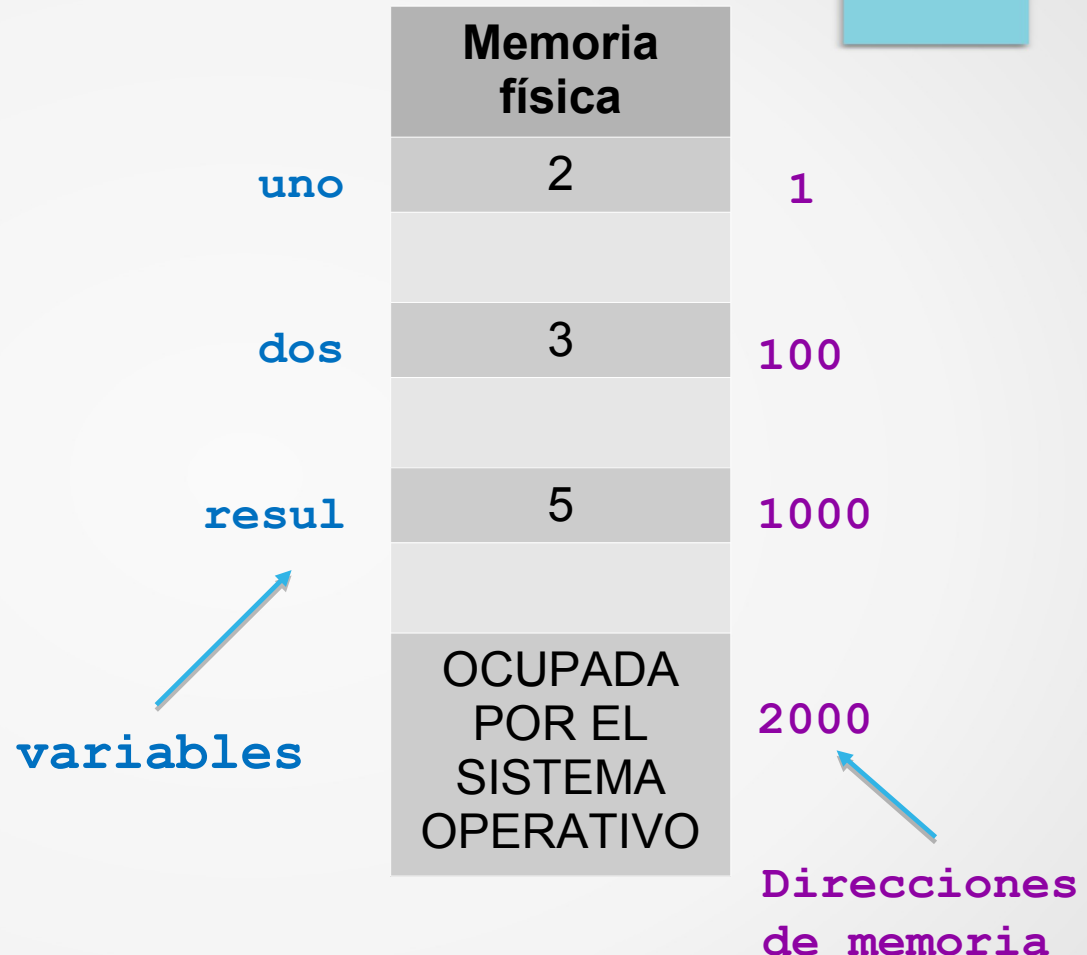


# Variables o identificadores

- Su nombre identifica una posición de memoria donde se guardará el dato.
- TODA variable debe ser declarada antes de poder usarse.
- Son GLOBALES si se declaran entre la biblioteca y una función, serán visibles en todo el programa.
- Son LOCALES si se declaran dentro de una función y son visibles sólo dentro de la función.

# Diagrama de memoria

- Cada vez que una variable se declara se reserva un espacio de memoria al cual se tiene acceso por el nombre de la variable o el operador "&"



# Nombres para variables o identificadores

- Iniciar con una letra de alfabeto inglés o con el símbolo (\_).
- No debe contener caracteres especiales como \$, #, @
- Después de la primera letra puede contener más, o números o (\_).
- No tener espacios como "mi variable".
- Distingue mayúsculas y minúsculas.
- No usar palabras reservadas del lenguaje como int, do, while, etc.
- Emplear notación de camello con la cual se unen palabras sin guiones colocando en mayúscula la inicial de la siguiente palabra:

`numUno, numDos, valorInicio, valorFinal`

# Declarar e inicializar variables

- Se recomienda inicializar toda variable de tipo numérico en cero antes de utilizarla, sino podría traer contenido basura y ser operado sin darse cuenta en el programa.

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int entera;
    float flotante;
    char letra;
    double doble;
```

```
/bin/bash
Este programa muestra la basura al no inicializar

Tu variable entera es -294451200
Tu variable flotante es 0.00
Tu variable caracter es:
Tu variable double es: 0.000000
```

```
#include <stdio.h>

int main(int argc, char *argv[])
{
    int entera=0;
    float flotante=0;
    char letra='\0';
    double doble=0;
```

```
/bin/bash
Este programa muestra variables inicializadas

Tu variable entera es 0
Tu variable flotante es 0.00
Tu variable caracter es:
Tu variable double es: 0.000000
```

# Constantes

- Valores que una vez compilado el programa no pueden cambiarse.
- Son visibles en cualquier parte del programa.
- Colocarse entre las bibliotecas y una función.
- Empleando `#define` busca toda coincidencia dentro del código y la sustituye
- Declaración con define:

```
#define pi 3.14159
```

# Alías de tipos de datos

Tipos de datos	Especificadores de conversión printf	Especificadores de conversión scanf
long double	%Lf	%Lf
double	%f	%lf
float	%f	%f
unsigned long int	%lu	%lu
long int	%ld	%ld
unsigned int	%u	%u
int	%d	%d
short	%hd	%hd
char	%c	%c

# Componentes de la función scanf

Cadena de control  
de formato

Indica a scanf la posición  
en memoria donde se  
almacenará la variable

`scanf ("%d", &variable);`

El *especificador de conversión* **%d** indica que los datos deben ser de tipo entero

El operador de dirección en C **&** (el ampersand) va seguido de la variable.

# Argumentos de la función main

- Son pasados desde línea de comandos a continuación del nombre del ejecutable del programa.
- La variable entera `argc` guarda el número de argumentos pasados al programa con valor mínimo de 1 por contar el nombre del ejecutable del programa.
- La variable `argv` corresponde a un arreglo de apuntadores a caracteres que almacena los argumentos pasados donde el elemento 0 es el nombre del ejecutable.



# Argumentos de la función main

- Aunque se ingresen números como argumentos en la línea de comandos o terminal éstos serán considerados cadenas y deberán convertirse antes a números si se desean operar.

`argv[0]`

`argv[1]`

`argv[2]`

## Memoria física

argumentos

5

8

OCUPADA POR EL  
SISTEMA  
OPERATIVO

```
huicho@NX-01: ~/Documentos
huicho@NX-01: ~/Documentos 80x24
huicho@NX-01:~/Documentos$ gcc argumentos.c -o argumentos
huicho@NX-01:~/Documentos$ ./argumentos 5 8
Este programa muestra los argumentos pasados al ejecutar

El argumento [0] es el nombre: ./argumentos
El argumento [1] es el primer dato: 5
El argumento [2] es el primer dato: 8
```

# Acentos y caracteres especiales en la consola de **Windows**

- Hay dos maneras de mostrar letras con acentos y otros caracteres especiales en C/C++ sobre Windows.
- Tenemos por ejemplo la palabra **árbol** que podemos mostrarla de las dos siguientes formas:

`printf("%crbol",160);` → valor ASCII decimal

`printf("%xA0rbol");` → valor ASCII hexadecimal

# Código ASCII incluyendo extendido

## El código ASCII

sigla en inglés de American Standard Code for Information Interchange  
(Código Estadounidense Estándar para el Intercambio de Información)

[www.elcodigoascii.com.ar](http://www.elcodigoascii.com.ar)

### Caracteres de control ASCII

DEC	HEX	Simbolo ASCII
00	00h	NULL (carácter nulo)
01	01h	SOH (inicio encabezado)
02	02h	STX (inicio texto)
03	03h	ETX (fin de texto)
04	04h	EOT (fin transmisión)
05	05h	ENQ (enquiry)
06	06h	ACK (acknowledgement)
07	07h	BEL (timbre)
08	08h	BS (retroceso)
09	09h	HT (tab horizontal)
10	0Ah	LF (salto de línea)
11	0Bh	VT (tab vertical)
12	0Ch	FF (form feed)
13	0Dh	CR (retorno de carro)
14	0Eh	SO (shift Out)
15	0Fh	SI (shift In)
16	10h	DLE (data link escape)
17	11h	DC1 (device control 1)
18	12h	DC2 (device control 2)
19	13h	DC3 (device control 3)
20	14h	DC4 (device control 4)
21	15h	NAK (negative acknowledge)
22	16h	SYN (synchronous idle)
23	17h	ETB (end of trans. block)
24	18h	CAN (cancel)
25	19h	EM (end of medium)
26	1Ah	SUB (substitute)
27	1Bh	ESC (escape)
28	1Ch	FS (file separator)
29	1Dh	GS (group separator)
30	1Eh	RS (record separator)
31	1Fh	US (unit separator)
127	20h	DEL (delete)

### Caracteres ASCII imprimibles

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
32	20h	espacio	64	40h	@	96	60h	`
33	21h	!	65	41h	A	97	61h	a
34	22h	"	66	42h	B	98	62h	b
35	23h	#	67	43h	C	99	63h	c
36	24h	\$	68	44h	D	100	64h	d
37	25h	%	69	45h	E	101	65h	e
38	26h	&	70	46h	F	102	66h	f
39	27h	'	71	47h	G	103	67h	g
40	28h	(	72	48h	H	104	68h	h
41	29h	)	73	49h	I	105	69h	i
42	2Ah	*	74	4Ah	J	106	6Ah	j
43	2Bh	+	75	4Bh	K	107	6Bh	k
44	2Ch	,	76	4Ch	L	108	6Ch	l
45	2Dh	-	77	4Dh	M	109	6Dh	m
46	2Eh	.	78	4Eh	N	110	6Eh	n
47	2Fh	/	79	4Fh	O	111	6Fh	o
48	30h	0	80	50h	P	112	70h	p
49	31h	1	81	51h	Q	113	71h	q
50	32h	2	82	52h	R	114	72h	r
51	33h	3	83	53h	S	115	73h	s
52	34h	4	84	54h	T	116	74h	t
53	35h	5	85	55h	U	117	75h	u
54	36h	6	86	56h	V	118	76h	v
55	37h	7	87	57h	W	119	77h	w
56	38h	8	88	58h	X	120	78h	x
57	39h	9	89	59h	Y	121	79h	y
58	3Ah	:	90	5Ah	Z	122	7Ah	z
59	3Bh	;	91	5Bh	[	123	7Bh	{
60	3Ch	<	92	5Ch	\	124	7Ch	
61	3Dh	=	93	5Dh	]	125	7Dh	}
62	3Eh	>	94	5Eh	^	126	7Eh	~
63	3Fh	?	95	5Fh	_			

[elCodigoASCII.com.ar](http://elCodigoASCII.com.ar)

### ASCII extendido

DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo	DEC	HEX	Simbolo
128	80h	Ç	160	A0h	á	192	C0h	Ł	224	E0h	Ó
129	81h	ü	161	A1h	í	193	C1h	ł	225	E1h	ô
130	82h	é	162	A2h	ó	194	C2h	Ł	226	E2h	Ô
131	83h	â	163	A3h	ú	195	C3h	ł	227	E3h	Õ
132	84h	à	164	A4h	ñ	196	C4h	Ł	228	E4h	ö
133	85h	ä	165	A5h	Ñ	197	C5h	ł	229	E5h	Ö
134	86h	å	166	A6h	ª	198	C6h	Ł	230	E6h	µ
135	87h	ç	167	A7h	º	199	C7h	ł	231	E7h	þ
136	88h	ê	168	A8h	¿	200	C8h	Ł	232	E8h	ß
137	89h	ë	169	A9h	®	201	C9h	ł	233	E9h	Û
138	8Ah	è	170	AAh	¬	202	CAh	Ł	234	EAh	Ü
139	8Bh	ï	171	ABh	½	203	CBh	ł	235	EBh	Ý
140	8Ch	ì	172	ACH	¼	204	CCh	Ł	236	ECh	ý
141	8Dh	í	173	ADh	¾	205	CDh	ł	237	EDh	ÿ
142	8Eh	Ä	174	Aeh	«	206	CEh	Ł	238	Eeh	ÿ
143	8Fh	Å	175	Afh	»	207	CFh	ł	239	Efh	·
144	90h	É	176	B0h	⋮	208	D0h	Ł	240	F0h	±
145	91h	æ	177	B1h	⋮	209	D1h	ł	241	F1h	±
146	92h	Æ	178	B2h	⋮	210	D2h	Ł	242	F2h	±
147	93h	ø	179	B3h	⋮	211	D3h	ł	243	F3h	¾
148	94h	ò	180	B4h	⋮	212	D4h	Ł	244	F4h	¾
149	95h	ó	181	B5h	⋮	213	D5h	ł	245	F5h	¾
150	96h	û	182	B6h	⋮	214	D6h	Ł	246	F6h	¾
151	97h	ù	183	B7h	⋮	215	D7h	ł	247	F7h	¾
152	98h	ÿ	184	B8h	©	216	D8h	Ł	248	F8h	¾
153	99h	Ö	185	B9h	⋮	217	D9h	ł	249	F9h	¾
154	9Ah	Ü	186	BAh	⋮	218	DAh	Ł	250	FAh	¾
155	9Bh	ø	187	BBh	⋮	219	DBh	ł	251	FBh	¾
156	9Ch	£	188	BCh	⋮	220	DCh	Ł	252	FCh	¾
157	9Dh	Ø	189	BDh	⋮	221	DDh	ł	253	FDh	¾
158	9Eh	x	190	BEh	⋮	222	DEh	Ł	254	FEh	¾
159	9Fh	f	191	BFh	⋮	223	DFh	ł	255	FFh	¾

# ASCII de acentos y caracteres especiales

```
1  /* programa con acentos y ñ
2  * hecho por huicho */
3
4  #include <stdio.h>
5
6  int main(int argc, char *argv[])
7  {
8      printf("Minúsculas con acento y ñ\n", 163, 164);
9      printf("\n\t ASCII 160 -> %c", 160);
10     printf("\n\t ASCII 130 -> %c", 130);
11     printf("\n\t ASCII 161 -> %c", 161);
12     printf("\n\t ASCII 162 -> %c", 162);
13     printf("\n\t ASCII 163 -> %c", 163);
14     printf("\n\t ASCII 164 -> %c", 164);
15
16     printf("\n\nMayúsculas con acento y ñ\n", 163, 165);
17     printf("\n\t ASCII 181 -> %c", 181);
18     printf("\n\t ASCII 144 -> %c", 144);
19     printf("\n\t ASCII 214 -> %c", 214);
20     printf("\n\t ASCII 224 -> %c", 224);
21     printf("\n\t ASCII 233 -> %c", 233);
22     printf("\n\t ASCII 165 -> %c", 165);
23
24     getchar();
25     return 0;
26 }
```



```
/bin/bash
Minúsculas con acento y ñ

    ASCII 160 -> á
    ASCII 130 -> é
    ASCII 161 -> í
    ASCII 162 -> ó
    ASCII 163 -> ú
    ASCII 164 -> ñ

Mayúsculas con acento y Ñ

    ASCII 181 -> Á
    ASCII 144 -> É
    ASCII 214 -> Í
    ASCII 224 -> Ó
    ASCII 233 -> Ú
    ASCII 165 -> Ñ
```

# Acentos y caracteres especiales en la consola de Windows

- La función `setlocale(LC_ALL, "")` asigna el idioma predeterminado del ambiente donde correrá. Para español:

`setlocale(LC_ALL, "Spanish");`

`setlocale(LC_ALL, "Spanish_Mexico");`

```
1  /*Programa para hacer compatibles ñ y acentos sin ascii
2   * cambia la localidad total del programa o porciones
3   * como el formato de fecha, moneda, la separación decimal
4   * hecho por Huicho*/
5
6  #include <stdio.h> //para printf y getchar
7  #include <locale.h> //para setlocale
8
9  int main(int argc, char* argv[])
10 {
11     //setlocale(LC_ALL, ""); //asigna localidad default del sistema
12     //setlocale(LC_ALL, "Spanish"); //asigna localidad en español
13     setlocale(LC_ALL, "Spanish_Mexico"); //asigna en español de México
14
15     printf("\n\n Soy niño, me llamo Jorge Luis López y tengo -- años");
16     getchar();
17     return 0;
18 }
```



# Suma de 2 números inicializados

INICIO

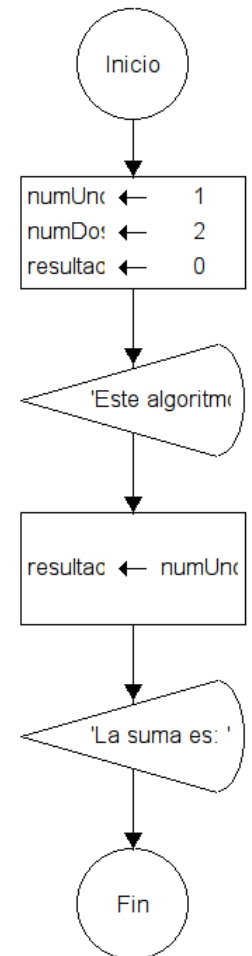
```
1. DEFINIR Entero: numUno=1
2. DEFINIR Entero: numDos=2
3. DEFINIR Entero: resultado=0

4. IMPRIMIR 'Este algoritmo suma 2
números inicializados en 1 y 2'
5. CALCULAR resultado=numUno+numDos
6. IMPRIMIR 'La suma es: ',resultado
```

FIN

- Partimos de un análisis y diseño previo en forma de pseudocódigo o diagrama de flujo realizado en cualquier herramienta

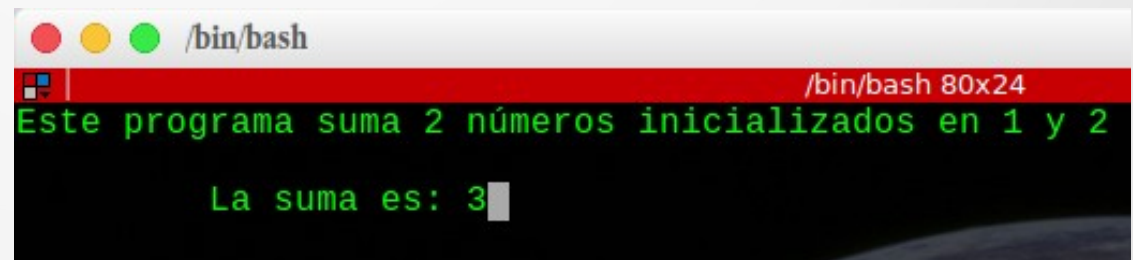
```
1 Algoritmo sumaInicializados
2   DEFINIR numUno,numDos,resultado COMO ENTERO
3   numUno<-1
4   numDos<-2
5   resultado<-0
6   Escribir 'Este algoritmo suma 2 números enteros inicializados en 1 y 2'
7   resultado<-numUno+numDos
8   Escribir 'La suma es: ',resultado
9 FinAlgoritmo
```



# Código en C y ejecución

- Buscar el equivalente en lenguaje C
- Agregar elementos propios del lenguaje como **biblioteca** y **pausa** en ejecución
- Agregar elementos de presentación como **acentos**, **saltos de línea** y **tabulador**

```
1  /*Programa que suma 2 números inicializados
2   * hecho por Huicho*/
3
4   #include <stdio.h> //para printf y getchar
5
6   int main(int argc, char* argv[])
7   {
8       short numUno= 1;
9       short numDos= 2;
10      short resultado= 0;
11
12      printf("Este programa suma 2 números inicializados en 1 y 2", 163);
13      resultado= numUno+numDos;
14      printf("\n\n\t La suma es: %hd", resultado);
15      getchar();
16      return 0;
17  }
```



```
/bin/bash
Este programa suma 2 números inicializados en 1 y 2

La suma es: 3
```

# Suma de 2 números

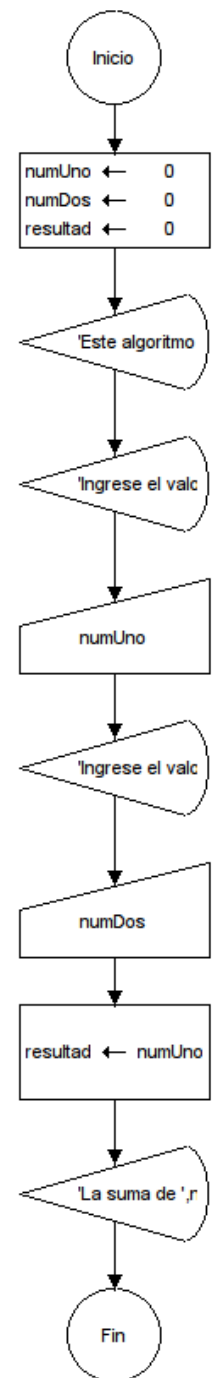
INICIO

```
1. DEFINIR Entero: numUno=0
2. DEFINIR Entero: numDos=0
3. DEFINIR Entero: resultado=0

4. IMPRIMIR 'Este algoritmo suma 2 números enteros dados por el usuario'
5. IMPRIMIR 'Ingrese el valor del primer número a sumar: '
6. LEER y ALMACENAR EN numUno
7. IMPRIMIR 'Ingrese el valor del segundo número a sumar: '
8. LEER y ALMACENAR EN numDos
9. CALCULAR resultado=numUno+numDos
10. IMPRIMIR 'La suma de ',uno,' + ',dos,' es: ',resultado
```

FIN

```
1  Algoritmo sumaUsuario2
2  DEFINIR numUno,numDos,resultado COMO ENTERO
3  numUno<-0
4  numDos<-0
5  resultado<-0
6  Escribir 'Este algoritmo suma 2 números enteros dados por el usuario'
7  Escribir 'Ingrese el valor del primer número a sumar: '
8  Leer numUno
9  Escribir 'Ingrese el valor del segundo número a sumar: '
10 Leer numDos
11 resultado<-numUno+numDos
12 Escribir 'La suma de ',numUno,' + ',numDos,' es: ',resultado
13 FinAlgoritmo
```



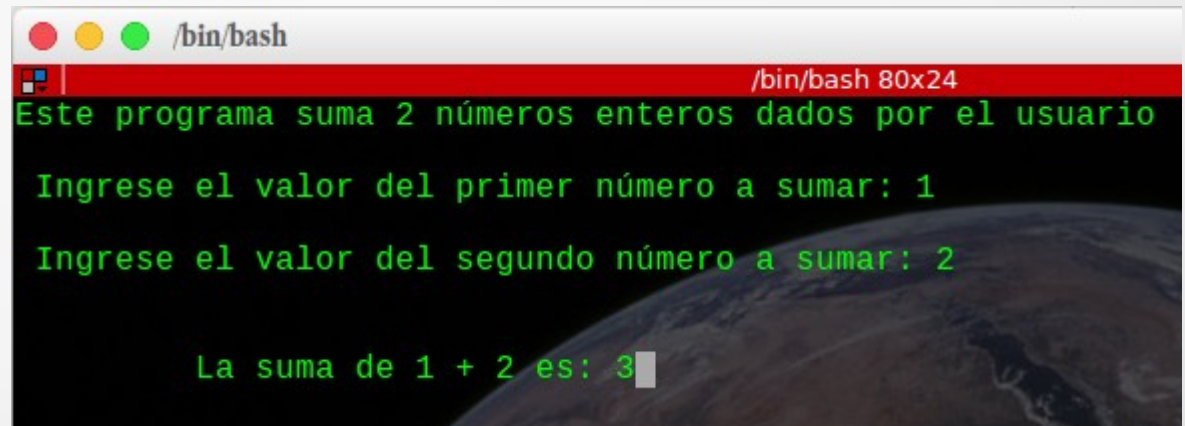
Hecho por Huicho :)



# Código en C y ejecución

- `ú` → 163
- `%hd` es el alias para leer e imprimir un entero corto
- `2 getchar()`, uno para atrapar el enter del scanf y otro para la pausa
- **No concatenar**, colocar los alias en el lugar donde mostrará el contenido de la o las variables

```
1  /*Programa que suma 2 números
2  * hecho por Huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5
6  int main(int argc, char* argv[])
7  {
8      short numUno= 0, numDos= 0, resultado= 0;
9
10     printf("Este programa suma 2 números enteros dados por el usuario \n", 163);
11     printf("\n Ingrese el valor del primer número a sumar: ", 163);
12     scanf("%hd", &numUno);
13     printf("\n Ingrese el valor del segundo número a sumar: ", 163);
14     scanf("%hd", &numDos);
15     resultado= numUno+numDos;
16     printf("\n\n\t La suma de %hd + %hd es: %hd", numUno, numDos, resultado);
17     getchar(); //atrapa el enter del scanf
18     getchar(); //mantiene en espera la ejecución
19     return 0;
20 }
```



```
/bin/bash
/bin/bash 80x24
Este programa suma 2 números enteros dados por el usuario

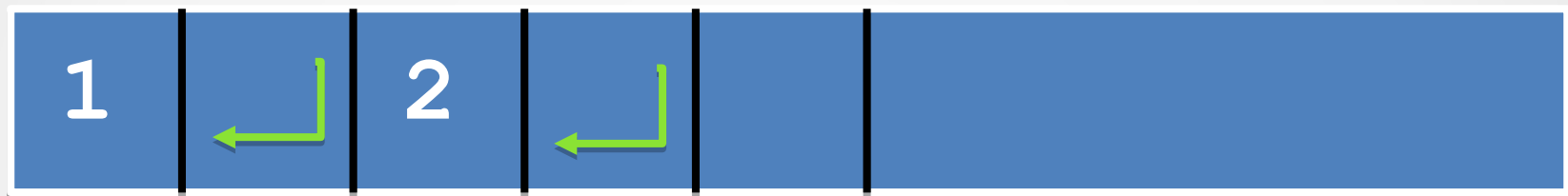
Ingrese el valor del primer número a sumar: 1

Ingrese el valor del segundo número a sumar: 2

La suma de 1 + 2 es: 3
```

# Usar más de un getch

- Cada tecla presionada en la ejecución del programa se guarda en un buffer temporal del teclado incluyendo la tecla "Enter".



*Al leer un entero el **scanf** solo toma el 2*

*El **"enter"** pulsado después del 2 queda libre y es tomado por el **getchar***

*Se coloca otro **getchar** en espera para capturar cualquier tecla y terminar la ejecución*

```
1 /*Programa que suma 2 números
2  * hecho por Huicho*/
3
4 #include <stdio.h> //para printf, scanf y getchar
5 #include <locale.h> //para setlocale
6
7 int main(int argc, char* argv[])
8 {
9     setlocale(LC_ALL, "Spanish_Mexico"); //asigna en español de México
10    short numUno= 0, numDos= 0, resultado= 0;
11
12    printf("Este programa suma 2 números enteros dados por el usuario \n");
13    printf("\n Ingrese el valor del primer número a sumar: ");
14    scanf("%hd", &numUno);
15    printf("\n Ingrese el valor del segundo número a sumar: ");
16    scanf("%hd", &numDos);
17    resultado= numUno+numDos;
18    printf("\n\n\t La suma de %hd + %hd es: %hd", numUno, numDos, resultado);
19    getchar(); //atrapa el enter del scanf
20    getchar(); //mantiene en espera la ejecución
21    return 0;
22 }
```

```
huicho@NX-01:~/Documentos$ gcc sumaUsuario3.c -o sumaUsuario3
```

```
huicho@NX-01:~/Documentos$ ./sumaUsuario3
```

```
Este programa suma 2 números enteros dados por el usuario
```

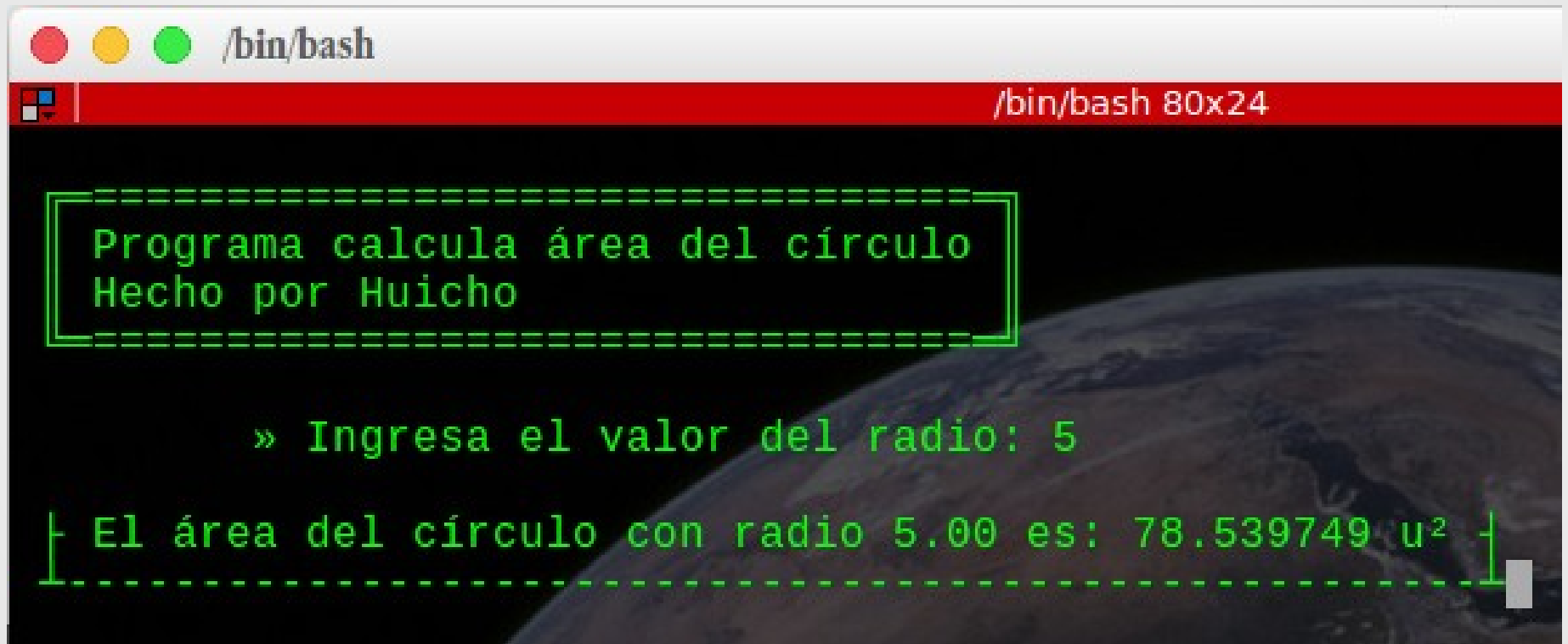
```
    Ingrese el valor del primer número a sumar: 1
```

```
    Ingrese el valor del segundo número a sumar: 2
```

```
        La suma de 1 + 2 es: 3
```

Hecho por Huicho :)

# Ejemplo con más presentación



```
/bin/bash
/bin/bash 80x24

=====
Programa calcula área del círculo
Hecho por Huicho
=====

» Ingresa el valor del radio: 5

| El área del círculo con radio 5.00 es: 78.539749 u² |
-----
```

# Ejemplo con más presentación

```
1  /*programa que calcula el área del círculo
2  * hecho por huicho*/
3
4  #include <stdio.h> //para printf, scanf y getchar
5  #include <math.h> //para pow
6
7  #define pi 3.14159 //declaro constante pi
8
9  int main(int argc, char* argv[]) //función para que sea ejecutable
10 { //abre función principal
11     float area=0; //declaro variable para área
12     float radio=0; //declaro variable para radio
13     //imprimo msj al usuario con márgenes y acentos con ascii y %c:
14     // á -> 160, í -> 161,
15     //extremos: 201, 187, 200, 188, horizontal: 205, vertical: 186
16     printf("\n %c%c===== %c%c", 201, 205, 205, 187);
17     printf("\n %c Programa calcula %c el área del círculo %c", 186, 160, 161, 186);
18     printf("\n %c Hecho por Huicho %c", 186, 186);
19     printf("\n %c===== %c", 200, 205, 205, 188);
20     printf("\n\n\t %c Ingresa el valor del radio: ", 175); //pido radio
21     scanf("%f", &radio); //leo radio
22     area= pi*pow(radio,2); //calcula área usando función potencia
23     //imprimo resultado de area, limito a 2 decimales con %.2f y unidades cuadradas
24     printf("\n %c El %c el área del círculo con radio %.2f es: %f u%c %c", 195, 160, 161, radio, area, 253, 180);
25     printf("\n %c----- %c", 193, 193);
26     getchar(); //atrapa el enter del scanf
27     getchar(); //mantiene estática la pantalla
28     return 0; //termina la ejecución devolviendo cero
29 }
```


```
┌(201) =(205) =(205) ┐(187)
│(186)  Progra      │(186)
│(186)  Hecho       │(186)
└(200) =(205) =(205) ┘(188)
```



# Cast

- Es la conversión explícita de un tipo de dato a otro
- Colocar (tipo de dato) delante de la variable a convertir

```
1  /*Programa que convierte entero a flotante temporalmente
2  * hecho por huicho*/
3
4  #include <stdio.h>
5
6  int main(int argc, char* argv[])
7  {
8      int uno=5, dos=2; //variables enteras a dividir
9      int div=0; //variable entera para guardar división
10     float division=0; //variable flotante para guardar división
11
12     printf("Divisi%cn de %d / %d = \n", 162, uno, dos);
13
14     div= uno/dos; //división de enteros guardada en entero
15     printf("\n\t En variable entera: %d", div);
16
17     division= uno/dos; //división de enteros guardada en flotante
18     printf("\n\t En variable flotante: %f", division);
19
20     division= (float)uno/dos; //división de enteros pasados a reales
21     printf("\n\n\t En variable flotante CON CAST: %f", division);
22
23     getchar();
24     return 0;
25 }
```



```
/bin/bash
/bin/bash 80x24
División de 5 / 2 =
En variable entera: 2
En variable flotante: 2.000000
En variable flotante CON CAST: 2.500000
```

# Operadores y funciones en C

OPERACIÓN	C	EJEMPLO Y NOTAS
SUMA	+	<code>a+b</code>
RESTA	-	<code>a-b</code>
MULTIPLICACIÓN	*	<code>a*b</code> <i>//No usar: 2a ni 2(a)</i>
DIVISIÓN	/	<code>a/b</code>
MÓDULO	<code>%</code> <code>fmod(a,b)</code>	<code>a%b</code> <i>//para enteros</i> <code>fmod(a,b)</code> <i>//para reales</i> <i>//pedir a math.h</i>
RAÍZ CUADRADA	<code>sqrt(x)</code>	<code>sqrt(x)</code> <i>//pedir a math.h</i>
POTENCIA	<code>pow(a,b)</code>	<code>pow(x,2)</code> <i>//eleva x al cuadrado</i> <i>//pedir a math.h</i>
FUNCIÓN PISO	<code>floor(x)</code>	<code>floor(x)</code> <i>//si x= 6.74 da 6</i> <i>//pedir a math.h</i>

**IMPORTANTE:** Al compilar desde terminal debe incluirse el modificador `-lm` si se pidieron funciones de `math.h` ]\$ `gcc algo.c -o algo -lm`

Hecho por Huicho :)

# Operadores y funciones en C

OPERACIÓN	C	EJEMPLO Y NOTAS
VALOR ABSOLUTO	<code>fabs (x)</code>	<code>fabs (x)</code> <i>//pedir a math.h</i>
EXPONENCIAL	<code>exp (x)</code>	<code>exp (x)</code> <i>//pedir a math.h</i>
LOGARITMO NATURAL	<code>log (x)</code>	<code>log (x)</code> <i>//pedir a math.h</i>
PSEUDO-ALEATORIO	<code>rand( )</code>	<code>rand()</code> <i>% x //el tope es x-1</i> <i>//pedir a stdlib.h</i>
FUNCIÓN TECHO	<code>ceil (x)</code>	<code>ceil(x)</code> <i>//si x= 6.74 da 7</i> <i>//pedir a math.h</i>
OPERADOR LÓGICO Y	<code>&amp;&amp;</code>	<code>x &amp;&amp; y</code>
OPERADOR LÓGICO O	<code>  </code>	<code>x    y</code>
OPERADOR LÓGICO NEGACIÓN	<code>!</code>	<code>!x</code>

**IMPORTANTE:** Al compilar desde terminal debe incluirse el modificador `-lm` si se pidieron funciones de `math.h` ]\$ `gcc algo.c -o algo -lm`