Lenguaje de programación

Introducción al lenguaje. Estructura secuencial

El lenguaje de programación C es un lenguaje de alto nivel que se puede caracterizar por los siguientes aspectos:

- Es de propósito general, esto significa que puede ser usado tanto para el desarrollo de sistemas operativos como para programas científicos, programas de aplicación o programas de educación y juegos.
- Posee una alta transportabilidad ya que los programas escritos en C pueden ser llevados de un tipo de ordenador a otro y funcionarán. Los cambios que hay que realizar son mínimos.
- Los compiladores generan ejecutables más pequeños porque por norma general, los programas escritos en lenguaje C poseen menos líneas de código que otros lenguajes de programación de alto nivel.
- El lenguaje de programación C es un lenguaje imperativo o procedimental. Esto significa que indica secuencias de acciones con el fin de llegar a un objetivo. Generalmente los lenguajes imperativos o procedimentales contienen una parte declarativa. El lenguaje C da órdenes a la máquina.
- Es un lenguaje estructurado ya que los programas escritos con él se pueden organizar en módulos.

- **Palabras reservadas en lenguaje C**

Las palabras reservadas tienen un significado especial para el compilador y representan una instrucción propia de su lenguaje, no se pueden utilizar para identificar variables y se escriben siempre en minúsculas.

Tabla 1. Palabras reservadas en lenguaje C

Asm	Auto	Bool	break	Case	catch
Char	Class	Const	const_cast	continue	default
Delete	Do	Double	dynamic_cast	Else	enum
Explicit	Extern	False	float	For	friend
Goto	If	Inline	int	Long	mutable
namespace	New	Operator	private	protected	Public
Register	reinterpret_cast	Return	short	Signed	Sizeof
Static	static_cast	Struct	switch	template	This
Throw	True	Try	typedef	Typeid	Typename
Unión	Unsigned	Using	while	Void	Volatile

- **Librerias en lenguaje C**

Las librerías o archivos de cabecera en lenguaje C, son los que contienen o almacenan

funciones que realizan operaciones y cálculos de uso frecuente y son parte de cada compilador. El programador debe invocar todos aquellos archivos o bibliotecas que necesite. A continuación algunas librerías más comunes en el lenguaje C.

- **#include<stdio.h>:** contiene los prototipos de las funciones, macros, y tipos para manipular datos de entrada y salida

Archivos que utiliza la librería <stdio.h>

clearerr	fclose	feof	ferror	fflush	fgetc	fgetpos
fgets	fopen	formato	fprintf	fputc	fputs	fread
freopen	fscanf	fseek	fsetpos	ftell	fwrite	getc
getchar	gets	perror	printf	putc	putchar	puts
remove	rename	rewind	scanf	setbuf	setybuf	sprintf
sscanf	tmpfile	tmpnam	ungetc	vfprintf	vprintf	vsprintf

- **#include<stdlib.h>:** contiene tipos, macros y funciones para la conversión numérica, generación de números aleatorios, búsquedas y ordenación, gestión de memoria y tareas similares.

Archivos que utiliza la librería < stdlib.h >

abort	abs	atexit	atof	atoi	atol	bsearch	abort
calloc	div	exit	free	getenv	labs	ldiv	calloc
malloc	mblen	mbstowcs	mbtowc	qsort	rand	Realloc	malloc
srand	strtod	strtol	strtoul	system	wctomb		srand

- **#include<string.h>:** contiene los prototipos de las funciones y macros de clasificación de caracteres.

Archivos que utiliza la librería < string.h >

memchr	memcmp	memcpy	memmove	memset	strcat	strchr	memchr
strcmp	strcoll	strcpy	strcspn	strerror	strlen	strmcat	strcmp
strmcmp	strmcpy	strpbrk	strrchr	strspn	strstr	strtok	strmcmp
strxfrm							strxfrm

- **#include<ctype.h>:** contiene varias funciones para comprobación de tipos y transformación de caracteres.

Archivos que utiliza la librería < ctype.h >

tolower	toupper					

Toupper ⇒ instrucción que convierte a mayúscula por ejemplo [s -> S]

- **#include<math.h>:** contiene los prototipos de las funciones y otras definiciones para el uso y manipulación de funciones matemáticas.

Archivos que utiliza la librería < math.h >

acos	asin	atan	atan2	ceil	cos	cosh
exp	fabs	floor	fmod	frexp	ldexp	log
log10	modf	pow	sin	sinh	sqrt	tan
tanh						

- **#include<time.h>:** contiene los prototipos de las funciones, macros, y tipos para manipular la hora y la fecha del sistema.

Archivos que utiliza la librería < time.h >

asctime	clock	ctime	difftime	Gmtime	localtime	mktime
strftime	time					

- **#include<conio.h>:** contiene los prototipos de las funciones, macros, y constantes para preparar y manipular la consola en modo texto en el entorno de MS-DOS®.

Archivos que utiliza la librería < conio.h >

cgets	clreol	clrscr	cprintf	cputs	cscanf	delline
getch e	getpass	gettext	gettextinfo	gotoxy	highvideo	inport
insline	Getch	lowvideo	movetext	normvideo	outport	putch
puttext	setcursortype	textattr	textbackground	textcolor	textmode	ungetch

Ejemplo utilizando la librería **<stdio.h>**.

```
/* cuadrado de un numero real */
# include <stdio.h>
# include <conio.h>
main()
{
```

```
    float x, cuadrado;
    printf("Teclee un numero real: ");
    scanf("%f",&x);
    cuadrado=x*x;
    printf("su cuadrado es %f : ",cuadrado);
    getch();
}
```

- **Cadenas de formato:** las cadenas de formatos como printf, permite dar formato específico a la salida y a cada símbolo cuando le antecede un %.
  Un **dato** es la expresión general que describe los objetos con los cuales opera una computadora. Los tipos de datos más comunes son: char, int, float, long, short, doublé, void etc.

Cadenas de formatos

%d,% i	entero decimal con signo, (eje: 38)
%o	entero octal sin signo, (eje: o43)
%u	entero decimal sin signo, (eje: 47)
%x	entero hexadecimal sin signo (en minúsculas) , (eje: 0x32)
%X	entero hexadecimal sin signo (en mayúsculas) , (eje: 0X32)
%f	Coma flotante en la forma [-]dddd.dddd , (eje: -3.2, 56.896)
%e	Coma flotante en la forma [-]d.dddd e[+/-]ddd, (eje: 2.3e4, 4.879e-4)
%g	Coma flotante según el valor
%E	Como e pero en mayúsculas, (eje: 2.3E4, 4.879E-4)
%G	Como g pero en mayúsculas
%c	un carácter, (eje: a, A)
%s	cadena de caracteres terminada en '\0' o null, (eje: ‘juan’, ‘JUAN’)
%5s	primeros cinco caracteres o delimitador
%8.2f	tamaño total de 8 dígitos, con dos decimales

- **Secuencias de escape:** las secuencias de escape son constantes especiales que se diferencian por llevar adelante una barra invertida (\) y cumplen una función específica, Las más usadas son:

Secuencias de escape

\a	Alerta
\b	Espacio atrás
\f	Salto de página
\n	Salto de línea

\r	Retorno de carro
\t	Tabulación horizontal
\v	Tabulación vertical
\\	Barra invertida
\'	Comilla simple
\"	Comillas dobles

- **Operadores relacionales:** los operadores relacionales, permiten comparar 2 valores arrojando un resultado basado en si la comparación es verdadera o falsa. Si la comparación es falsa el resultado es 0, si es verdadera el resultado es 1.

- **Operadores lógicos: l**os operadores lógicos, dan como resultado cierto o falso, siguiendo las reglas de la lógica formal. Estos operadores se combinan con operadores relacionales o comparación.

Operadores relacionales y lógicos

Operadores relacionales	Ejemplo	Operadores lógicos	Ejemplo
<	1° menor 2°	&&	And ⇒ (3>1)&&(2>7) salida no = 0
>	1° mayor 2°	¦¦	Or ⇒ (1<3) ¦¦ (2>0) salida si = 1
<=	1° menor o igual 2°	!	Not ⇒ !(2<1) Salida si = 1
>=	1° mayor o igual 2°		
==	1° igual al 2°		
!=	1° diferente al 2°		

- **Operadores unarios:** los operadores unarios admiten un único argumento y cuentan con operador de cambio de signo (-) y también con los de incremento (++) y decremento (- -) y son una forma fácil de sumar o restar 1 a una variable.

Operadores unarios

variable ++	Postincremento
++variable	Preincremento
variable - -	Postdecremento
- -variable	Predecremento

Ejemplos:

- ++a; ⇒ a=a+1;
- - -b; ⇒ b=b-1;

Los formatos postfijos se conforman de modo diferente según la expresión en que se aplica, ejemplos:

- b=++a; ⇒ a=a+1; b=a;
- b=a++; ⇒ b=a; a=a+1;
- int i,j,k=5;

  k++; ⇒ k ahora vale 6, es igual que ++k;

  --k; ⇒ k vale ahora 4, es igual que k--;

  i=4*k++; ⇒ k vale ahora 5 e i=20

  j=4*++k; ⇒ k vale ahora 6 e j=24

- **Operadores aritméticos:** los operadores aritméticos se utilizan para hacer cálculos aritméticos. Ejemplos:

  División entera

  19%6=1

  19/6=3

  15/2=7

  15%2=1

  División real

  10/3=3.33333

Operadores aritméticos

Nombre	**Función**
Multiplicación	*
División	/
Módulo o resto de una división entera	%
Suma	+
Resta	-

Ejemplo : cuántos pares e impares hay, utilizando el modulo división entera? /* cuantos pares e impares hay?*/

```
#include <stdio.h>
#include<stdlib.h>
main()
{
int a,b,c,impar,par,ed,ef,eg;
printf("Digite tres números enteros\n");
scanf("%d %d %d",&a,&b,&c);
impar=((a%2)+(b%2)+(c%2));
par=3-impar;
ed=(a%2);
ef=(b%2);
eg=(c%2);
printf("\nHay %d numero(s) impar(es)\n\n",impar);
printf("\nHay %d numero(s) par(es)\n\n",par);
printf("\nporque en el modulo de una division entera 0 es par y 1 es impar
"); printf("\nentonces hay %d %d %d \n\n ",ed,ef,eg);
}
```

Código ascii: muestra el código ascii imprimibles y no imprimibles.

Código ascii

Caracteres no imprimibles				Caracteres imprimibles					
Nombre	Dec	Car.		Dec	Car.	Dec	Car.	Dec	Car.
Nulo	0	NUL		32	Espacio	64	@	96	`
Inicio de cabecera	1	SOH		33	!	65	A	97	a
Inicio de texto	2	STX		34	"	66	B	98	b
Fin de texto	3	ETX		35	#	67	C	99	c
Fin de transmisión	4	EOT		36	$	68	D	100	d
enquiry	5	ENQ		37	%	69	E	101	e
acknowledge	6	ACK		38	&	70	F	102	f
Campanilla (beep)	7	BEL		39	'	71	G	103	g
backspace	8	BS		40	(	72	H	104	h
Tabulador horizontal	9	HT		41	)	73	I	105	i
Salto de linea	10	LF		42	*	74	J	106	j
Tabulador vertical	11	VT		43	+	75	K	107	k
Salto de página	12	FF		44	,	76	L	108	l
Retorno de carro	13	CR		45	-	77	M	109	m
Shift fuera	14	SO		46	.	78	N	110	n
Shift dentro	15	SI		47	/	79	O	111	o
Escape linea de datos	16	DLE		48	0	80	P	112	p
Control dispositivo 1	17	DC1		49	1	81	Q	113	q
Control dispositivo 2	18	DC2		50	2	82	R	114	r
Control dispositivo 3	19	DC3		51	3	83	S	115	s
Control dispositivo 4	20	DC4		52	4	84	T	116	t
neg acknowledge	21	NAK		53	5	85	U	117	u
Sincronismo	22	SYN		54	6	86	V	118	v
Fin bloque transmitido	23	ETB		55	7	87	W	119	w
Cancelar	24	CAN		56	8	88	X	120	x
Fin medio	25	EM		57	9	89	Y	121	y
Sustituto	26	SUB		58	:	90	Z	122	z
Escape	27	ESC		59	;	91	[	123	{
Separador archivos	28	FS		60	<	92	\	124	|
Separador grupos	29	GS		61	=	93	]	125	}
Separador registros	30	RS		62	>	94	^	126	~
Separador unidades	31	US		63	?	95	_	127	DEL

- **Creación de un programa en C**

Para crear un programa, se debe definir primero un esquema general del problema, el siguiente paso, es desarrollar el programa en C o C++, es decir crear un código fuente.

Compilación: después que se ha terminado de codificar el programa en el editor, el siguiente paso es la compilación, o sea la traducción del código fuente a código objeto (lenguaje de máquina entendible por el computador). Esta compilación genera un archivo con extensión .cpp si es C++.

Si el compilador es BorlandC, genera 4 archivos con extensión .cpp, .obj, .bak y .exe
Si el compilador es C++, genera 2 archivos con extensión .cpp y .exe

- **Estructura secuencial**

Una estructura secuencial es aquella que nos permite entrar datos, hacer un cálculo y luego mostrar la salida o los resultados del programa. Siguiendo la tradición, la mejor forma de aprender a programar en cualquier lenguaje es editar, compilar, corregir y ejecutar pequeños programas descriptivos. Analicemos por lo tanto los siguientes ejemplos que ilustran la forma como se edita un programa en C o en Dev-C++:

Ejemplo 1: #include, main(), printf()

Comenzaremos por un ejemplo sencillo: un programa que muestra en pantalla una frase.

```
/* Ejemplo 1. Programa DOCENA.C */
#include <stdio.h>
main ()
{
int docena;
docena = 12;
printf ("Una docena son %d unidades\n", docena);
}
```

Este programa hace aparecer en pantalla la frase
"**Una docena son 12 unidades**".

Veamos el significado de cada una de las líneas del programa.

/* Ejemplo 1. Programa DOCENA.C */

Es un comentario. El compilador de Turbo C ignora todo lo que está entre los símbolos de comienzo (**/***) y fin (***/**) de un comentario. Los comentarios delimitados por estos símbolos pueden ocupar varias líneas.

#include <stdio.h>

Le dice a Turbo C o Dev-C++ que en el proceso de compilación incluya un archivo denominado **stdio.h**. Este fichero se suministra como parte del compilador de Turbo C y contiene la información necesaria para el correcto funcionamiento de la E/S de datos.

La sentencia **#include** no es una instrucción C. El símbolo # la identifica como una directiva, es decir, una orden para el preprocesador de C, responsable de realizar ciertas tareas previas a la compilación.

Los archivo ***.h** se denominan **archivos de cabecera**. Todos los programas C requieren la inclusión de uno o varios archivos de este tipo, por lo que normalmente es necesario utilizar varias líneas **#include**.

main ()

Es el nombre de una función. Un programa C se compone de una o más funciones, pero al menos una de ellas debe llamarse **main()**, pues los programas C empiezan a ejecutarse por esta función.

Los paréntesis identifican a **main()** como una función. Generalmente, dentro de ellos se incluye información que se envía a la función. En este caso no hay traspaso de información por lo que no hay nada escrito en su interior; aun así son obligatorios. El **cuerpo de una función** (conjunto de sentencias que la componen) va enmarcado entre llaves **{** y **}**. Ese es el significado de las llaves que aparecen en el ejemplo.

int docena;

Es una sentencia declarativa. Indica que se va a utilizar una variable llamada **docena** que es de tipo entero. La palabra **int** es una palabra clave de C que identifica uno de los tipos básicos de datos que tiene C. En C es obligatorio declarar todas las variables antes de ser utilizadas. El "**;**" identifica la línea como una sentencia C.

docena = 12;

Es una sentencia de asignación. Almacena el valor **12** a la variable **docena**. Obsérvese que acaba con punto y coma. Como en la mayoría de los lenguajes, el operador de asignación en C es el signo igual "**=**".

printf ("Una docena son %d unidades\n", docena);

Esta sentencia es importante por dos razones: en primer lugar, es un ejemplo de llamada a una función. Además ilustra el uso de una función estándar de salida: la función **printf()**.

La sentencia consta de dos partes:

- El nombre de la función: **printf()**.
- Los argumentos. En este caso hay dos separados por una coma:
  - **"Una docena son %d unidades\n"**
  - **docena**

Como toda sentencia C acaba con punto y coma.

La función **printf()** funciona de la siguiente forma: el primer argumento es una **cadena de formato**. Esta cadena será lo que, básicamente, se mostrará en pantalla. En la cadena de formato pueden aparecer **códigos de formato** y **caracteres de escape**.

Un **código de formato** comienza por el símbolo **%** e indica la posición dentro de la cadena en donde se imprimirá el segundo argumento, en este caso, la variable **docena**.

Más adelante estudiaremos todos los códigos de formato de Turbo C o Dev-C++. En este ejemplo, **%d** indica que en su lugar se visualizará un número entero decimal. Un **carácter de escape** comienza por el símbolo ****. Son caracteres que tienen una interpretación especial. La secuencia **\n** es el carácter **nueva línea** y equivale a la secuencia LF+CR (salto de línea + retorno de cursor).
La función **printf()** pertenece a la biblioteca estándar de C. Las definiciones necesarias para que funcione correctamente se encuentran en el archivo **stdio.h**, de ahí que sea necesaria la sentencia **#include <stdio.h>**.

Ejemplo 2: scanf()

El siguiente programa realiza la conversión de pies a metros usando la

equivalencia: 1 pie = 0.3084 metros

El programa solicita por teclado el número de pies y visualiza en pantalla los metros correspondientes.

```
/* Ejemplo 2. Programa PIES.C */
#include <stdio.h>
main ()
{
int pies;
float metros;
printf ("\n¿Pies?: ");
scanf ("%d", &pies);
metros = pies * 0.3084;
printf ("\n%d pies equivalen a %f metros\n", pies, metros);
}
```

Este programa hace aparecer en pantalla:

¿Pies?: 24
24 pies equivalen a 7.315200 metros

Estudiaremos ahora las novedades que aparecen en este programa.

float metros;

Es una sentencia declarativa que indica que se va a utilizar una variable llamada **metros**, que es del tipo **float**. Este tipo de dato se utiliza para declarar variables numéricas que pueden tener decimales.

printf ("\n¿Pies?: ");

Es la función **printf()** comentada antes. En esta ocasión sólo tiene un argumento: la cadena de control sin códigos de formato. Esta sentencia simplemente sitúa el cursor al principio de la siguiente línea (**\n**) y visualiza la cadena tal como aparece en el argumento.

scanf ("%d", &pies);

scanf() es una función de la biblioteca estándar de C (como **printf()**), que permite leer datos del teclado y almacenarlos en una variable. En el ejemplo, el primer argumento, **%d**, le dice a **scanf()** que tome del teclado un número entero. El segundo argumento, **&pies**, indica en qué variable se almacenará el dato leído. El símbolo **&** antes del nombre de la variable es necesario para que **scanf()** funcione correctamente.

metros = pies * 0.3084;

Se almacena en la variable **metros** el resultado de multiplicar la variable **pies** por **0.3084**. El símbolo ***** es el operador que usa C para la multiplicación.

printf ("\n%d pies equivalen a %f metros\n", pies, metros);

Aquí **printf()** tiene 3 argumentos. El primero es la cadena de control, con dos códigos de formato: **%d** y **%f**. Esto implica que **printf()** necesita dos argumentos adicionales. Estos argumentos encajan en orden, de izquierda a derecha, con los códigos de formato. Se usa **%d** para la variable **pies** y **%f** para la variable **metros**.

printf ("\n**%d** pies equivalen a **%f** metros\n", **pies**, **metros**);

El código **%f** se usa para representar variables del tipo **float**.

Ejemplo 3: Funciones que devuelven valores, return( );.

En el siguiente ejemplo, veremos un ejemplo que utiliza una función que devuelve un valor. El siguiente programa lee dos números enteros del teclado y muestra su producto en pantalla. Para el cálculo se usa una función que recibe los dos números y devuelve el producto de ambos.

```
/* Ejemplo 3 - Programa MULT.C */
 /*programa multiplica mediante una funcion*/
 #include <stdio.h>
 #include <conio.h>
 float multiplica(int a,int b); /*declaro la función */
 int main()
 {
  int a,b,producto;
 printf("\nTeclee dos numeros enteros: ");
 scanf("%d %d",&a,&b);
 producto=multiplica(a,b);
 printf("\nEl resultado es %d" ,producto);
 getch();
 }

/*declaración de la función multiplica()*/
float multiplica(int a,int b) /*aquí la función no lleva ;*/
  {
    int c;
    c=a*b;
    return(c);
    }
```

Este programa hace aparecer en pantalla:

```
Teclee dos números enteros: 23
 4
 El resultado es 92
```

Las novedades que se presentan en este programa se comentan a

continuación. **scanf ("%d %d", &a, &b);**

La cadena de control de **scanf()** contiene dos códigos de formato. Al igual que ocurre en **printf()**, se precisan dos argumentos más, uno por cada código de formato. Los dos números se teclean separados por espacios en blanco, tabuladores o por la tecla Intro.

return (c);

La palabra clave **return** se usa dentro de las funciones para salir de ellas devolviendo un valor. El valor devuelto mediante **return** es el que asume la función. Eso permite tener sentencias como

producto = multiplica (int a, int b);

es decir, sentencias en las que la función está a la derecha del operador de asignación. Para nuestros propósitos actuales podemos decir (aunque esto no sea exacto) que después de la sentencia **return** la función **multiplica()** actúa como si fuese una variable que almacena el valor devuelto.

Los paréntesis son opcionales, se incluyen únicamente para clarificar la expresión que acompaña a **return**. No deben confundirse con los paréntesis de las funciones.

Ejemplo 4: Calcular el área de un círculo con printf y scanf

```
/* Programa para calcular el área de un círculo */
# include <stdio.h>
# include <conio.h>
main()
{
    float radio, area;
    printf("Radio = ? ");
    scanf("%f",&radio);
    area=3.1416*radio*radio;
    printf("Area = %f",area);
    getch();
}
```

Este programa hace aparecer en pantalla:
Radio = ? 4
Area = 50.265598

Ejemplo 5: Cadenas de un vector con printf

```
/*cadenas en un vector*/
#include <stdio.h>
#include <conio.h>
main()
{
   char nombre[15], apellido1[15], apellido2[15];

   printf("Introduce tu nombre: ");
   scanf("%s",nombre);
   printf("Introduce tu primer apellido: ");
   scanf("%s",apellido1);
   printf("Introduce tu segundo apellido: ");
   scanf("%s",apellido2);

/*para obtener la salida puedo decir*/
 printf("Usted es %s %s %s\n",nombre,apellido1,apellido2);

 /* o escribir la salida utilizando la directiva puts (colocar)*/

   puts(nombre);
   puts(apellido1);
   puts(apellido2);
```

/*Al final cierro con esta directiva que regresa un carácter tan pronto se oprime una tecla sin

esperar que el usuario oprima *Enter* ↵*/, y se coloca antes de cerrar el programa para poder visualizarlo.

```
    getch ();
}
```

1.2 Estructuras de decision if then else y switch case

Las Estructuras de control como de las de decisión, repetitivas y estructuras de selección múltiple permiten transferir el control a otras estructuras sin seguir la ejecución secuencial. Las estructuras selectivas se utilizan para tomar decisiones lógicas; el cual se suelen denominar también estructuras de decisión o alternativas simple doble o múltiple.

La estructura switch case, es una estructura de decisión múltiple, donde el compilador prueba o busca el valor contenido en una variable contra una lista de constantes de tipo entero o char. Si dicha variable no existe ejecuta la instrucción por default.

Una cadena está definida como un arreglo de caracteres ascii. Las cadenas de caracteres o strings son un tipo particular de vectores de tipo char que tienen una marca de fin o caracter *NULL* o *\0* que indica que la cadena ha terminado. También podemos escribir texto como comentarios dentro de comillas dobles como por ejemplo:

- **Manejo de cadenas**

En los siguientes ejemplos 1, 2 y 3 se muestra como se hace un menú, cadenas de caracteres y como se escribe una cadena en un vector.

Ejemplo 1: menú

```
#include <stdio.h>
#include <conio.h>
main()
{
    char nombre[20];
    int i;
    printf("Elije el numero del menu:\n\n");
    printf("1- Cargar fichero\n");
    printf("2- Guardar en un fichero\n");
    printf("3- Mostrar datos\n");
    printf("0- Salir\n\n");
    printf("opcion:");
    printf("\nHas elegido:%c",getchar());
    return(0);
}
```

Ejemplo 2: cadenas de caracteres

```
/*cadena de caracteres*/
#include <stdio.h>
#include <conio.h>

int main()
{
```

```
   char cadena[6]; /* Define un vector o cadena de 6 caracteres */


   cadena[0]='L';
   cadena[1]='e';
   cadena[2]='t';
   cadena[3]='r';
   cadena[4]='a';
   cadena[5]='s';
   cadena[6]=0; /* Carácter nulo, significa el fin del texto */

   printf("La cadena es: %s\n", cadena);
   printf("La tercera letra de la cadena es: %c\n", cadena[2]);
   printf("Las últimas tres letras de la cadena son: %s\n",
   &cadena[3]); getch ();
}
```

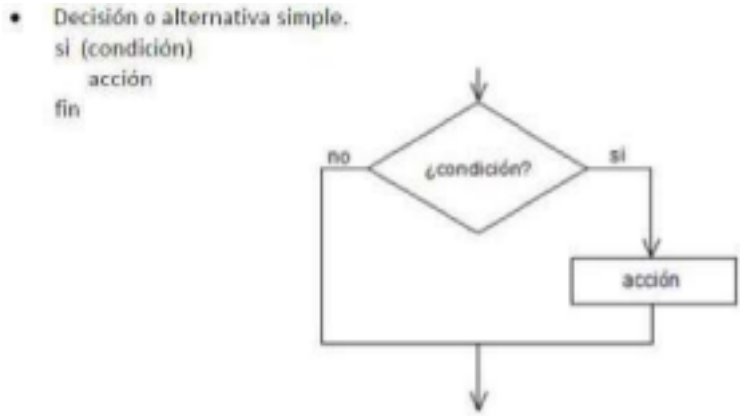
Ejemplo 3: cadenas en un vector

```
/*cadenas en un vector*/
#include <stdio.h>
#include <conio.h>

main()
{
   char nombre[15], apellido1[15], apellido2[15];

   printf("Introduce tu nombre: ");
   scanf("%s",nombre);
   printf("Introduce tu primer apellido: ");
   scanf("%s",apellido1);
   printf("Introduce tu segundo apellido: ");
   scanf("%s",apellido2);
   printf("Usted es %s %s

   %s\n",nombre,apellido1,apellido2); printf("o escribirlo

   por filas con puts: ");


   puts(nombre);
   puts(apellido1);
   puts(apellido2);
   getch ();
}
```

- **Estructuras de decisión o selección**

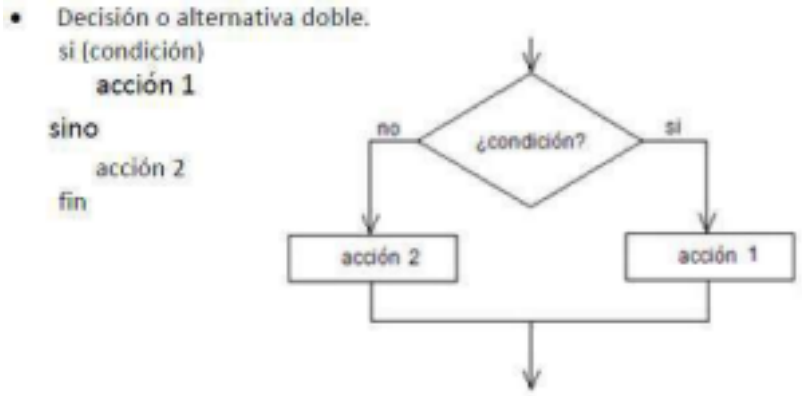
Las sentencias de **decisión** o también llamadas **estructuras** de control que realizan una pregunta la cual retorna verdadero o falso (evalúa una condición) y selecciona la siguiente instrucción a ejecutar dependiendo la respuesta o resultado.

Figura 2: estructura de decisión o alternativa simple




```
#include <stdio.h>
#include <conio.h>
int main()
{
int num;
 printf( "Introduce un número " );
scanf( "%i", &num );
if (num==10)
{
 printf( "El número es correcto\n" );
}
getch();
}
```

Figura 3: estructura de decisión o alternativa doble



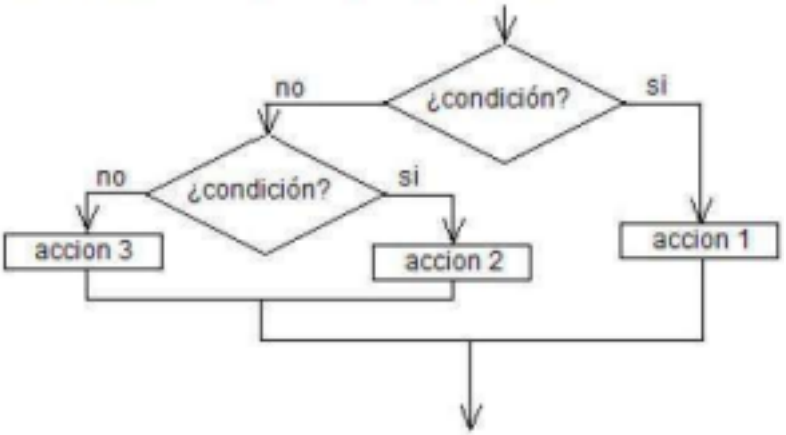

```
#include <stdio.h>
#include <conio.h>
int main()
{
int a;
 printf("Introduce un número ");
scanf( "%i", &a );
if ( a==8 )
{
 printf ("El número introducido era un ocho.\n");
}
else
{
```

```
printf("Pero si no has escrito un ocho!!!\n");
}
getch();
}
```

Figura 4: estructura de decisión o alternativa múltiple

- Decisión o alternativa múltiple (es una combinación de ambas).

```
si (condición)
    acción 1
si (condición)
   acción 2
sino
   acción 3
fin
```

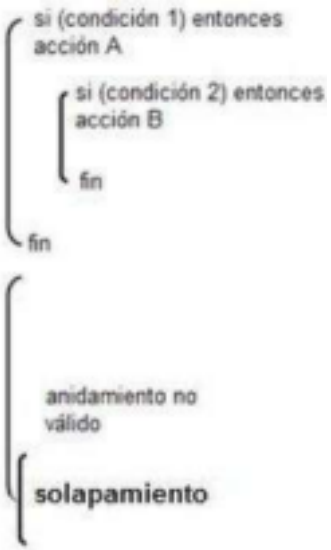



```
#include <stdio.h>
#include <conio.h>
int main()
{
int a;
printf( "Introduce un número " );
scanf( "%i", &a );
if ( a<10 )
{
printf ( "El número introducido era menor de 10.\n" );
}
else if ( a>10 && a<100 )
{
printf ( "El número está entre 10 y 100\n" );
}
else if ( a>100 )
{
printf( "El número es mayor que 100\n" );
}
printf( "Fin del programa\n" );
getch();
}
```

Anidamientos de estructuras de decisión.

Se anidan colocando una estructura en el interior de otra estructura, no debe haber solapamiento. También se cumple para las demás estructuras repetitivas. Ver figura 5.

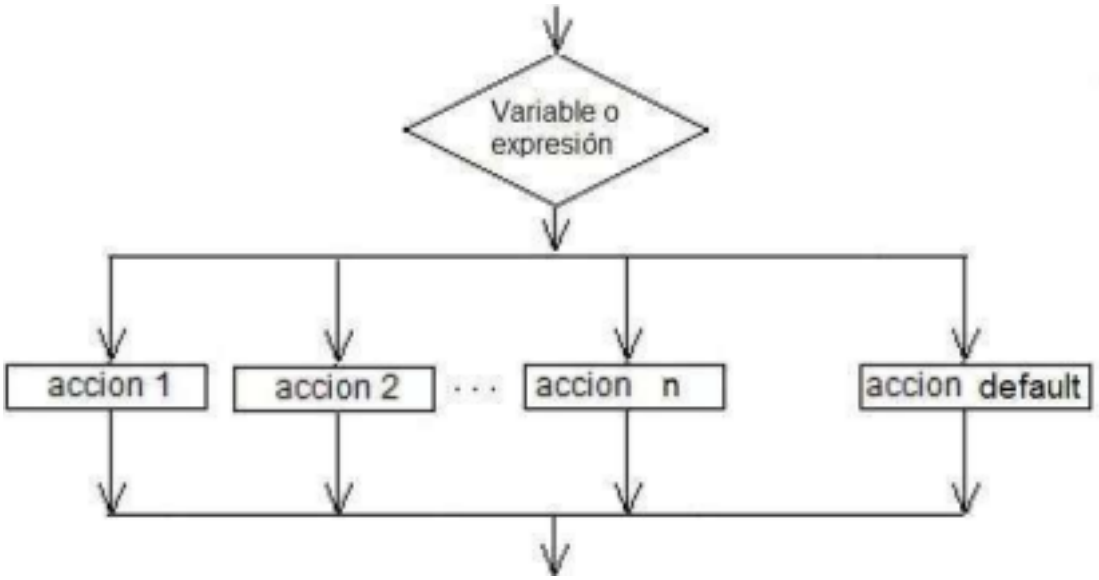
Figura 5. Anidamientos



- **Estructura switch-case (según-caso)**

La estructura según caso, es una estructura de decisión múltiple. Se evalúa una expresión y en función del valor resultante se realiza una determinada tarea o conjunto de tareas. En caso de existir un valor no comprendido entre 1 y n, se realiza una acción excluyente o por defecto (default). En la figura 6, se muestra el diagrama de flujo de esta estructura.

Figura 6. Estructura switch case (según caso)



Código fuente:

```
switch (expresion)
{
    case 1:
       instruccion;
       break;
    case 2:
       instruccion;
       break;
    case 3:
       instruccion;
       break;
    case n:
       instruccion;
```

```
        break;
    default :
}
```

Cuando se ejecuta la instrucción switch, la expresión se evalúa y se transfiere el control directamente al grupo de instrucciones cuya etiqueta case tenga el mismo valor de expresión. Si ninguno de los valores de las etiquetas case coincide con el valor de expresión, entonces no se seleccionará ninguno de los grupos de la instrucción switch o se realizara la instrucción por defecto (default). El valor por defecto (default) es una forma conveniente de generar un mensaje de error en caso de tener entradas inválidas.

break: es una sentencia de salto que se utiliza cuando se requiere salir de un ciclo incondicionalmente antes de que se termine la iteración actual.

Probar las estructuras según caso y entenderlas

Ejemplo 7: introduce un número

```
#include <stdio.h>
#include <conio.h>
int main()
{
int num;
printf( "\nIntroduce un numero entero \n\n " );
printf( "\n1-Verifique si es el número " );
printf( "\n2-Verifique si es el número " );
printf( "\n3-Verifique si es el número " );
printf( "\n4-por defecto \n\n " );
scanf( "%i", &num );
switch( num )
{
case 1:
printf( "\nEs un 1\n" );
break;
case 2:
printf( "\nEs un 2\n" );
break;
case 3:
printf( "\nEs un 3\n" );
break;
default:
printf( "\nNo es ni 1, ni 2, ni 3\n" );
}
getch();
}
```

Ejemplo 8: 4 operaciones básicas (+, -, /,*)

```
#include <stdio.h>
#include <conio.h>
int main()
{
float R1, R2;
char character;
printf( "Introduce dos valores :\n" );
scanf( "%f%f", &R1,&R2 );
printf( "Seleccione una operacion:\n" );
```

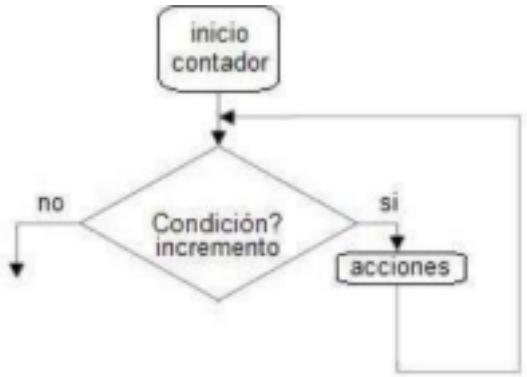
```
printf( "s: para sumar \n r: para restar \n" );
printf( "p: para multiplicar \n d: para dividir \n" );
character=getch();
switch( character )
{
case 's':
printf( "El resultado es %f", R1+R2);
break;
case 'r':
printf( "El resultado es %f", R1-R2);
break;
case 'p':
printf( "El resultado es %f", R1*R2);
break;
case 'd':
printf( "El resultado es %f", R1/R2);
break;
default:
printf( "Opcion no valida\n" );
}
getch();
}
```

1.3 Estructuras repetitivas for, while y do while

- Estructura repetitiva for: la estructura for, es una estructura repetitiva, donde el número de iteraciones o repeticiones se conoce con anterioridad y por ello no se necesita ninguna condición de salida para detener las instrucciones. Ver figura 7.
  Código fuente:

```
for(variable = valor inicial; condición; incremento)
  {
  acciones
  }
```

Figura 7. Estructura repetitiva for



Ejemplo 9: lista de n números enteros positivos

```
/*Lista de n números enteros positivos*/
#include<stdio.h>
#include<conio.h>
main()
{
    int dig;
```

```
    for(dig=0;dig<=9;++dig)
    printf("Numero: %d \n",dig);
    getch();
}
```

Salida:

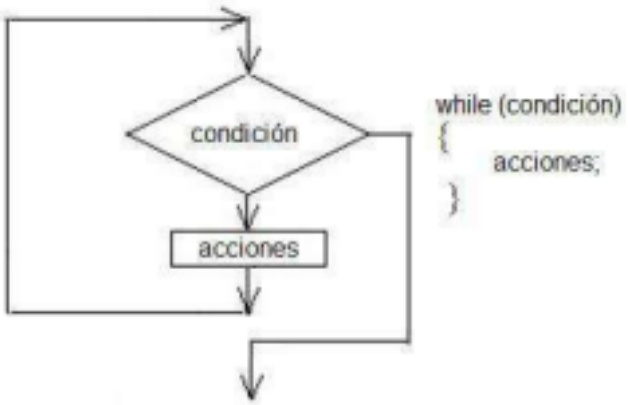
Numero: 0
Numero: 1
Numero: 2
Numero: 3
Numero: 4
Numero: 5
Numero: 6
Numero: 7
Numero: 8
Numero: 9

- Estructura repetitiva mientras (while): la estructura repetitiva while, en ella la condición o expresión sea verdadera, realizar la(s) acciones es la más utilizada. Ver figura 8.

Código fuente:

```
While (condición)
    {
    acciones
    }
```

Figura 8. Estructura while



Ejemplo 10: lista de números enteros de 1 a 9

```
/* lista de números enteros de 1 a 9*/
#include<stdio.h>
#include<conio.h>
main()
{
    int digito=0;
    while(digito<=9)
    printf("%d \n",digito++);
    getch();
}
```

SALIDA:

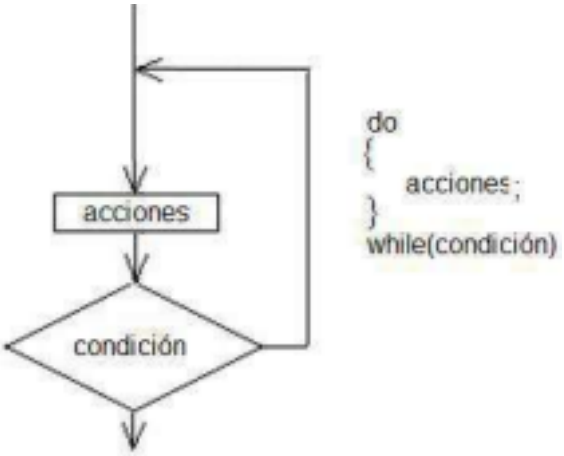
1
2
3
4
5
6
7
8
9

- Estructura repetitiva hacer mientras (do while): la estructura repetitiva do while, en ella se ejecuta al menos una vez la(s) acciones, antes de que se evalúe la expresión o condición. Ver figura 9.

Código fuente:

do {
acciones
}
While (condición)

Figura 9. Estructura do while



Ejemplo 11: suma de los primeros 6 números enteros positivos

```
/* suma de los primeros 6 números enteros positivos */
#include<stdio.h>
#include<conio.h>
#define N 5
main()
{
    int i=0,sum=0;
    do{
       sum=sum+i;
       i++;
      }while(i<=N);
     printf("la suma de los primeros %d números incluido el cero es %d
    ",i,sum); getch();
}
```

SALIDA: la suma de los primeros 6 numeros incluido el cero es 15

Ejemplo 12: **utilizando la sentencia (do while), hacer suma de pares entre 1 y n, donde n entra por teclado.**

```
/* suma de pares entre 1 y n, donde n entra por teclado */

#include <stdio.h>
#include <conio.h>
main()
{
    int i, suma_par=0, num1;
    i=0;
    printf("\n digite un numero entero: ");
    scanf ("%i",&num1);
    do
    {
    if(i%2==0) /* este operador % me devuelve el residuo de la div entre dos enteros*/
    suma_par +=i;
    i++;
    }
    while (i<=num1);
    printf ("\n La suma de numeros pares es %d\n", suma_par);
    getch();
    }
```

SALIDA: digite un numero entero: 3
La suma de números pares es 2

2.1 Conversión de un algoritmo a función

Conversión de un algoritmo a función: en el lenguaje C se conoce como funciones, aquellos trozos de códigos utilizados para dividir un programa en subprogramas, con el objetivo de que cada bloque o función realice una tarea específica. Es decir; a medida que los programas crecen en tamaño y dificultad, el programador debe dividirlos en secciones más pequeñas o módulos, denominadas funciones. De ahí el nombre de programación estructurada o modular. Esta estructura permite tanto al programador como al usuario seguir los pasos en la ejecución de un programa para revisar y corregir errores.

PAPEL DE LAS FUNCIONES.

1. **Modularidad:** un programa por extenso que sea, es más fácil y agradable manipularlo cuando se ha dividido en varios módulos o funciones.

2. **Facilidad de seguimiento:** las funciones permiten ahorrarle tiempo al programador cuando se trata de buscar errores o cuando se quiere depurar el programa. 3. **Reutilización del código:** sí una función requiere utilizarse más de una vez en el mismo programa, no se necesita volver a escribir su código (definición de la función). Basta hacer el llamado a partir de su nombre que la identifica.

Para construir una función de debe tener en cuenta:

- Qué hace exactamente la función (tarea específica).
- Qué datos necesita la función (parámetros formales).
- Qué tipo de dato o valor devuelve (o que resultados debe producir)

Sí se encuentra que en una función hay involucrada más de una tarea, debe subdividir en dos o más funciones según el caso. En las funciones es importante declarar las variables y estas pueden ser locales o globales.

Variables globales: las variables globales se crean durante la ejecución del programa, y son globales porque son llamadas, leídas, modificadas, etc. Desde cualquier función y se definen antes del programa principal (main).

Variables locales: estas, solo pueden ser utilizadas únicamente en la función que hayan sido declaradas.
La sintaxis de una función es la siguiente:

tipo_de_datos nombre_de_la_función (tipo o nombre de argumentos)
{
acciones
}

Donde:
tipos_de_datos: es el tipo de dato que devolverá esa función, que puede ser real, entero, o tipo void (es decir que no devolverá ningún valor).
nombre_de_la_función: es el identificador o el nombre que se le da a una función. **tipo o nombre de argumentos:** son los parámetros que recibe la función y pueden existir funciones que no reciban argumentos. Los argumentos de una función no son más que variables locales que reciben un valor.

Entre abrir y cerrar {}, se encuentran las **acciones** o lo que queremos que haga la función; al final antes de cerrar con llave debemos colocar una instrucción return(), el cual devolverá un valor específico; si la función es de tipo void no devuelve ningún valor específico, debido a que esta instrucción reconoce todo tipo de dato.

TIPOS DE FUNCIONES.

Una función es un bloque de código dentro de un programa que se encarga de una tarea específica. El programa puede tener muchas funciones y no puede faltar la función principal main(). Entonces un programa, está constituido por más de una función, las llamadas a la misma pueden realizarse desde cualquier parte del programa y no solo una vez, se pueden llamar muchas veces y empiezan a ejecutarse las acciones que están escritas en código.
Los tipos de funciones son de acuerdo al tipo de dato que manejan como:

- Funciones que no devuelven ningún valor (void)
- Funciones que devuelven un valor entero (return())
- Funciones que devuelven un valor real (return())
- Funciones combinadas (return())

• Funciones en las que usamos un menú

DECLARACIÓN DE UNA FUNCIÓN.

Una función en C, se declara antes de ser utilizada por el programa principal (main()) y se declaran después de invocar las librerías. El prototipo de una función es su cabecera terminada en punto y coma (;), como se muestra en el siguiente ejemplo:

```
/* suma de los n primeros numeros enteros */
#include<conio.h>
#include <stdio.h>
int suma(int n); // Declaro la función antes de main( ) y debe llevar (;)

main() //programa principal
{
  int sum,n;
  printf("entre el valor de n: " );
  scanf("%d",&n);
  sum=suma(n); // aquí llamo la función suma
  printf("la suma de los n primeros numeros enteros es: %d" ,sum);
  //salida getch();
}
int suma(int n) // definición de la función sin el (;) y va después del programa
  principal {
  int i,sum=0;
  for(i=1; i<=n; i++)
     sum+=i;
     return(sum);
  }
```

SALIDA:
entre el valor de n: 4
la suma de los n primeros numeros enteros es: 10

Los dos siguientes ejemplos: 13) programa para calcular el área de un triángulo rectángulo y 14) programa que multiplica dos números enteros, son codificados en forma secuencial entrada, proceso y salida. Seguido se proceden a transformarlos en subprogramas llamados funciones en las que devuelven un valor entero, para luego integrarlos en un "switch case", que es un programa que integra estos dos subprogramas en un programa principal.

Ejemplos 13: programa área de un triángulo rectángulo como secuencial y como función.

```
/*programa area de un triangulo rectángulo como secuencial*/
#include<stdio.h>
#include<stdlib.h>
main()
{
 int b,h,area;
```

```
 printf("Teclee la base y la altura:\n");
 scanf("%d%d",&b,&h);
 area=(b*h)/2;
 printf("El area del triangulo rectangulo es %d\n\n", area);
 }

/*programa area de un triangulo rectángulo como funcion*/
#include<stdio.h>
#include<stdlib.h>
int areatriangulo(int b, int h);
int main()
{
 int b,h,area;
 printf("Teclee la base y la altura :\n");
 scanf("%d %d",&b,&h);
 area=areatriangulo(b, h);
 printf("El area del triangulo rectangulo es %d \n\n", area);
}
int areatriangulo(int b, int h)
 {
   int a;
   a=(b*h)/2;
   return(a);
 }
```

Ejemplos 14: programa multiplica dos números enteros como secuencial y como función. /* programa multiplica dos números enteros como secuencial*/

```
 #include <stdio.h>
 #include <conio.h>
  int main()
  {
  int a, c, producto;
    printf ("Teclee dos numeros enteros:\n ");
    scanf ("%d %d", &a, &c);
    producto=a*c;
    printf ("\nEl resultado es %d", producto);
    getch();
     }

/* programa multiplica dos numeros enteros como funcion*/
#include <stdio.h>
#include <conio.h>
 int multiplica(int a, int c);

  int main()
  {
   int a, c, producto;
    printf ("Teclee dos numeros enteros:\n ");
```

```
 scanf ("%d %d", &a, &c);
 producto = multiplica (a, c);
 printf ("\nEl resultado es %d", producto);
 getch();
}

 int multiplica(int a, int c)
 {
 int pro;
 pro=a*c;
 return(pro);
 }
```

Con los dos ejemplos anteriores 13 y 14, convertidos en subprogramas o funciones, se agrupan en un programa principal llamado menú funciones con la estructura “switch case”, el cual muestra un menú de opciones para escoger.

Ejemplos 15: programa menú funciones con switch case.

```
/*menu funciones con switch case*/
#include <stdio.h>
#include <conio.h>
#include<stdlib.h>
int areatriangulo(int b, int h);
int multiplica(int a, int c);

int main()
{
                int opc;
                printf( "\n\tPROGRAMA MENU FUNCIONES\n\n" );
                printf( "\n\t1-area de un triangulo rectangulo" );
                printf( "\n\t2-multiplicacon de dos numeros" );
                printf( "\n\n\tTeclea tu opcion " );
                scanf( "%d",&opc);
                switch(opc)
                        {
                        case 1:
                        system("COLOR F1");
                         int b,h,area;
                        printf("Teclee la base y la altura :\n");
                        scanf("%d %d",&b,&h);
                        area=areatriangulo(b, h);
                        printf("El area del triangulo rectangulo es %d \n\n", area);
                        return 0;
                        break;

                        case 2:
                        int a, c, producto;
                        printf ("Teclee dos numeros enteros:\n ");
```

```
                    scanf ("%d %d", &a, &c);
                    producto = multiplica (a, c);
                    printf ("\nEl resultado es %d", producto);
                    getch();
                    break;
                    default:
                    printf( "\n\tElija una opcion del menu\n" );
                    }
                    getch();
}

int areatriangulo(int b, int h)
  {
  int a;
  a=(b*h)/2;
  return(a);
  }
int multiplica(int a, int c)
   {
   int pro;
   pro=a*c;
   return(pro);
   }
```

ANIDAMIENTOS DE BUCLES

Los anidamientos de bucles son útiles cuando queremos diseñar un programa donde nos indica por software si deseamos terminar el programa o continuar ejecutándolo. Los siguientes ejemplos 16 y 17, muestran como anidar las sentencias **for** e **if then else** en un bucle **do while** con el objetivo de que el programa se pueda cerrar por software.

Ejemplos 16: cuadrado y cubo de n números enteros.

```
#include <stdio.h>
int main()
{
char seguir;
int i, num;

do
{
  printf( "\n Introduzca un numero entero: " );
  scanf( "%d", &num );
  printf( "\n El cuadrado y cubo de los %d primeros numeros enteros: \n\n ",

   num); /* Inicio del anidamiento */

   for ( i = 1 ; i <= num ; i++ )
    {
```

```
      printf( "\n numero: %d cuadrado: %d cubo: %d ", i, i*i, i*i*i );
    }
   /* Fin del anidamiento */

 printf( "\n\n Desea volver a repetir (s/n)?: " );
 fflush( stdin );
 scanf( "%c", &seguir );

} while ( seguir != 'n' );
return 0;
}
```

Ejemplos 17: Cálculo de raíces reales de la ecuación cuadrática

```
#include <stdio.h>
#include <conio.h>
#include <math.h>

main()
  {
        float a, b, c, raiz, R1, R2;
        char resp;

        do {
              printf ("\nCalculo de raices reales de una ecuacion
              cuadratica"); printf ("\nde la forma ax^2+bx+c=0 :");
              printf ("\n\nIntroduzca las constantes a, b y c : \n");
              scanf ("%f%f%f", &a,&b,&c);

              raiz=(b*b)-4*a*c; //calculo del discriminante

                 /* Inicio del anidamiento */
              if (raiz>=0)
                   {
                        R1=(-b+sqrt(raiz))/(2*a);
                        R2=(-b-sqrt(raiz))/(2*a);
                        printf ("\nLas raices reales son %.2f y %.2f ", R1, R2);
                    }
              else
                        printf ("\nLas constantes a,b y c calculan raices complejas");
                       printf ("\n\nDesea cambiar las constantes? (s/n) \n\n ");
                       resp = getche();
                   /* Fin del anidamiento */

            }
            while (resp=='s' || resp=='S');
           getch();
 }
```

2.2 Programa menú funciones con anidamientos

Con el conocimiento adquirido en el ítem anterior sobre anidamientos, podemos restructurar el programa de menú funciones en un programa mucho más vistoso, con avisos y mensajes ilustrativos, utilizando anidamientos para salir del programa por software, con cambio de color de fuente y de texto y utilizando el tipo de puntero universal (void) para declarar las funciones y que no tienen valor de retorno.

‘’ Cuando se utiliza como un tipo de valor devuelto de función, la palabra clave **void** especifica que la función no devuelve ningún valor. Cuando se utiliza para la lista de parámetros de una función, void especifica que la función no toma ningún parámetro. Cuando se utiliza en la declaración de un puntero, void especifica que el puntero es "universal" ‘’.

Este programa queda como plantilla para que cada estudiante construya su trabajo final de menú funciones.

/*programa area de un triangulo rectángulo como funcion*/

```
#include<stdio.h>
#include<stdlib.h>
void areatriangulo(int b, int h);
 main()
{
 int b,h;
 printf("Teclee la base y la altura :\n");
 scanf("%d %d",&b,&h);
 areatriangulo(b, h);
}
void areatriangulo(int b, int h)
 {
   int area;
   area=(b*h)/2;
    printf("El area del triangulo rectangulo es %d \n\n", area);
 }
```

Salida:

```
Teclee la base y la altura:
4
5
El área del triangulo rectangulo es 10
Presione una tecla para continuar. . .
```

/*programa multiplica dos números enteros como función*/

```
#include<stdio.h>
#include<stdlib.h>
void multiplica(int a, int b);
 main()
{
 int a,b;
 printf("Teclee dos números enteros: \n");
 scanf("%d %d",&a,&b);
```

```
 multiplica(a, b);
}
void multiplica(int a, int b)
 {
   int c;
   c=a*b;
    printf("El resultado es %d \n\n", c);
 }
```

Salida:

```
Teclee dos números enteros:
4
5
El resultado es 20
Presione una tecla para continuar. . .
```

CONSTRUCCIÓN DE LA PLANTILLA PARA EL PROGRAMA MENU FUNCIONES CON SWITCH CASE

```
/*menu funciones con switch case*/
#include <stdio.h>
#include <conio.h>
#include<stdlib.h>
#include <ctype.h>

#define MENSAJE printf("\n\n<<<<PRESIONE UNA TECLA PARA VOLVER A MENU");getch();
#define SI printf("\n\n<<<<SI...ENTONCES PULSA [S]");
#define NO printf("\n\n<<<<NO...ENTONCES PULSA [N] PARA VOLVER AL MENU");

void areatriangulo(int b, int h);
void multiplica(int a, int b);
char menu()
{                                                    printf("\n\n\t\tMENU DE OPCIONES\n\n");
                                                     printf("\n*****************************
                                                     *************************\n");
                                                     printf("\n\t1-area de un triangulo
                                                     rectangulo");
                                                     printf("\n\t2-multiplica dos números
                                                     enteros");
                                                     printf("\n\n\t3-SALIR");
                                                     printf("\n\n\tESCOJA UNA OPCION");
                                                     printf("\n*****************************
}                                                    *************************\n");
main() {
printf("\n\n\n\t\tBIENVENIDO/A\n\n");
                                          /*declaracion de variables globales*/
        char c,resp;
```

```
  do
    { /*primer ciclo do while para salir o continuar*/ do
        { /*segundo ciclo do while, inicio del anidamiento*/ system("CLS()"); /*limpia
          pantalla*/
          menu();
          c=getche();
          getch();
          switch(c)
           {
             case '1': system("CLS");
             system("COLOR F1");
             int b,h;
              printf("Teclee la base y la altura :\n");
              scanf("%d %d",&b,&h);
              areatriangulo(b, h);
              MENSAJE;
              break;
             case '2': system("CLS");
             system("COLOR F1");
             int a, c;
              printf ("Teclee dos numeros enteros:\n");
             scanf ("%d %d", &a, &b);
             multiplica (a, b);
              MENSAJE;
              break;
             case '3': system("CLS"); /*limpia pantalla*/
             default:printf("\n\n\n\n\tERROR<<<<<DEBES ELEGIR UNA OPCION VALIDA ");getch();
              break;
           }
        } while(c!='3'); /*fin del anidamiento*/
          printf("\n\nQUIERES SALIR DEL PROGRAMA?");
          SI;
          NO;
          resp=toupper(getch()); /*INSTRUCCION QUE CONVIERTE A MAYUSCULA [s->S]*/
                                                  /*toupper pertenece a la libreria ctype*/
            } while(resp!='S'); /*fin del primer ciclo do while para salir o continuar*/
   return 0;
} /*FIN DEL PROGRAMA PRINCIPAL*/

void areatriangulo(int b, int h)
  {
  int a;
  a=(b*h)/2;
  printf("El area del triangulo rectangulo es %d \n\n", a);
  }
void multiplica(int a, int b)
   {
   int pro;
```

```
pro=a*b;
printf ("\nEl resultado es %d \n\n",
pro); }
```