FUNDAMENTOS DE INFORMÁTICA

CAPITULO II

Algoritmos de Programación

PROF. DANIEL SLAVUTSKY

PROF. EDGARDO BARCIA

PROF. JORGE ZÁRATE

AÑO 2014

www.utnfravirtual.org.ar
daniel@slavutsky.com.ar

ALGORITMOS DE PROGRAMACIÓN

INTRODUCCIÓN

La idea general de este capítulo es introducir a los alumnos a la resolución de Algoritmos por medio del uso de Pseudocódigo y el Lenguaje C de Programación.

Cada alumno deberá estar capacitado a resolver problemas desde el punto vista de la lógica informática, que le permitirá entender de una manera más explícita la forma de comunicarse con una computadora mediante la creación de programas sencillos que no son ni más ni menos el inicio en esta etapa del conocimiento que le permitirá entender el modo de funcionamiento de un sistema de computación.

Por este motivo el desarrollo de algoritmos es un tema fundamental en el diseño de programas por lo cual el alumno debe tener buenas bases que le sirvan para poder desarrollar de manera fácil y rápida sus programas.

CONCEPTOS GENERALES

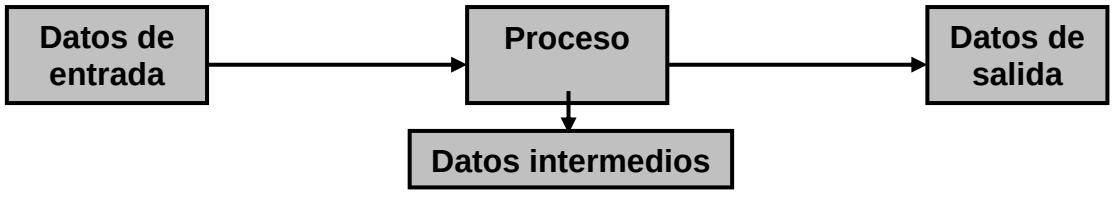
La computadora no solamente es una maquina que puede realizar procesos para darnos resultados, sin que tengamos la noción exacta de las operaciones que realiza para llegar a esos resultados.

Con la computadora además de lo anterior también podemos diseñar soluciones a la medida, de problemas específicos que se nos presenten. Más aun, si estos involucran operaciones matemáticas complejas y/o repetitivas, o requieren del manejo de un volumen muy grande de datos.

El diseño de soluciones a la medida de nuestros problemas, requiere como en otras disciplinas una metodología que nos enseñe de manera gradual, la forma de llegar a estas soluciones. A las soluciones creadas por computadora se les conoce como programas y no son más que una serie de operaciones que realiza la computadora para llegar a un resultado, con un grupo de datos específicos.

Lo anterior nos lleva al razonamiento de que un programa nos sirve para solucionar un problema específico. Para poder realizar programas, además de conocer la metodología mencionada, también debemos de conocer, de manera específica las funciones que pueden realizar las computadoras y las formas en que se pueden manejar los elementos que hay en la misma.

Proceso de información en la computadora:



DEFINICIONES IMPORTANTES

Programa: Es el conjunto de instrucciones escritas de algún lenguaje de programación y que ejecutadas secuencialmente, resuelven un problema especifico.

Lenguaje: Es una serie de símbolos que sirven para transmitir uno o más mensajes (ideas) entre dos entidades diferentes. A la transmisión de mensajes se le conoce comúnmente como **comunicación**.

Comunicación: Es un proceso complejo que requiere una serie de reglas simples, pero indispensables para poderse llevar a cabo. Las dos principales son las siguientes:

- Los mensajes deben correr en un sentido a la vez.
- Debe forzosamente existir 4 elementos: Emisor, Receptor, Medio de Comunicación y Mensaje.

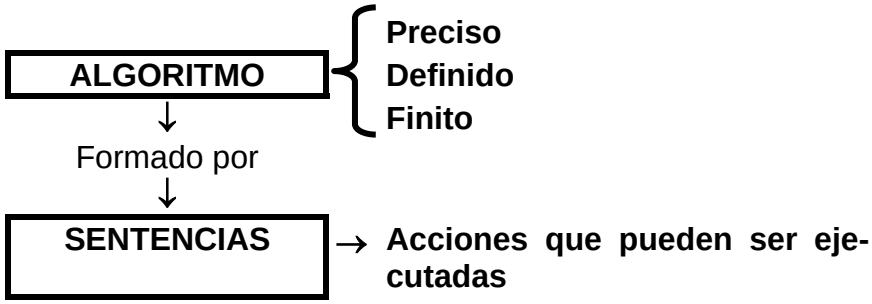
Algoritmo: La palabra algoritmo se deriva de la traducción al latín de la palabra árabe alkhowarizmi, nombre de un matemático y astrónomo árabe que escribió un tratado sobre manipulación de números y ecuaciones en el siglo IX.

Un algoritmo es una serie de pasos organizados que describe el proceso que se debe seguir, para dar solución a un problema específico.

Es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos.

- **Preciso:** No se presta a interpretaciones ambiguas.
- **Definido:** Si se siguen 2 o más veces los pasos, se obtiene el mismo resultado.
- **Finito:** Tiene comienzo y fin; tiene un número determinado de pasos.

Los algoritmos se pueden expresar en forma de diagramas de flujo, por fórmulas matemáticas y en **Pseudocódigo**. Esta última herramienta es la más usada en lenguajes estructurados como el **Lenguaje C**.



Tipos de algoritmos:

- **Cualitativos:** Son aquellos en los que se describen los pasos utilizando palabras.
- **Cuantitativos:** Son aquellos en los que se utilizan cálculos numéricos para definir los pasos del proceso.

Diseño de un algoritmo:

- Debe tener un punto particular de inicio.
- Debe ser definido, no debe permitir dobles interpretaciones.
- Debe ser general, es decir, soportar la mayoría de las variantes que se puedan presentar en la definición del problema.
- Debe ser finito en tamaño y tiempo de ejecución.
- Existen varias herramientas para hacerlo, entre ellas el Pseudocódigo.

Pseudocódigo:

Es una mezcla de un lenguaje de programación y el idioma de un país, se utiliza en la programación estructurada para realizar el diseño de un algoritmo. En general, al Pseudocódigo se lo puede definir como un lenguaje de especificaciones de algoritmos.

Es la representación narrativa de los pasos que debe seguir en un algoritmo para dar solución a un problema determinado. El Pseudocódigo utiliza palabras que indican el proceso a realizar.

Ventajas de utilizar un Pseudocódigo:

- Permite representar en forma fácil operaciones repetitivas complejas
- Es muy fácil pasar de Pseudocódigo a un programa en algún lenguaje de programación.
- Si se siguen las reglas se puede observar claramente los niveles que tiene cada operación.

TIPOS DE DATOS

Todos los datos tienen un tipo asociado con ellos. Un dato puede ser un simple carácter, tal como 'b', un valor entero tal como 35. El tipo de dato determina la naturaleza del conjunto de valores que puede tomar una variable.

En la tabla se muestra estos tipos:

TIPOS	TAMAÑO EN BITS	RANGO
char	**8**	0 a 255
int	**16**	-32768 a 32767
float	**32**	3.4e-38 a 3.4e+38
double	**64**	1.7e-308 a 1.7e-308
void	**0**	Sin valor

Los valores del tipo **char** se usan normalmente para guardar valores definidos en el juego de caracteres ASCII, así como cualquier cantidad de 8 bits. Los valores de tipo **int** se usan para guardar cantidades enteras y los valores del tipo **float** o **double,** se usan para guardar números reales (Los números reales tienen un componente entero y uno fraccionario).

ARREGLOS (ARRAYS)

Los arreglos son tipos de datos que reservan un espacio en la memoria RAM para almacenar temporalmente los datos ingresados. La cantidad de espacio que se reserve dependerá del tipo de dato y de la cantidad de elementos que lo compongan.

Los arreglos pueden almacenar tanto caracteres como cadena de caracteres y valores numéricos de distinto tipo (enteros y reales).

Como se declaran los Arreglos:

VECTORES:

char nombre[15]; /*Creamos un vector de caracteres para almacenar una cadena de caracteres de 15 elementos. Reservando un espacio en memoria RAM de 15 bytes. (Cada carácter ocupa en memoria 1 byte).*/

float valores[10]; /*Creamos un vector numérico del tipo real para almacenar 10 valores. Reserva un espacio en memoria RAM de 40 bytes. (Cada número entero ocupa en memoria 4 bytes).*/

int valores[10]; /*Creamos un vector numérico del tipo entero para almacenar 10 valores. Reserva un espacio en memoria RAM de 20 bytes. (Cada número entero ocupa en memoria 2 bytes).*/

MATRICES:

char nombres[5][20]; /*Creamos una matriz para almacenar a 5 cadenas de caracteres de hasta 20 elementos cada una. Reserva un espacio en memoria RAM de 100 bytes. (Cada letra ocupa en memoria 1 byte).*/

float matriz[3][4]; /* Creamos una matriz de 3 filas por 4 columnas para almacenar valores numéricos reales. Reserva un espacio en memoria RAM de 48 bytes. (Cada número real ocupa en memoria 4 bytes).

int matriz[3][4]; /* Creamos una matriz de 3 filas por 4 columnas para almacenar valores numéricos enteros. Reserva un espacio en memoria RAM de 24 bytes. (Cada número entero ocupa en memoria 2 bytes).

VARIABLES

Son objetos de un programa cuyo valor puede cambiar durante la ejecución del mismo. Datos que pueden sufrir modificaciones a lo largo de un programa. El cambio se produce mediante sentencias ejecutables como por ejemplo la asignación o el ingreso de datos. **Es** en realidad **una porción** (o posición) **de memoria con nombre** que permite almacenar temporalmente un dato durante la ejecución de un proceso. Para poder reconocer una variable en la memoria de la computadora, es necesario darle un nombre con el cual podamos identificarla dentro de un algoritmo (a esto se lo denomina Declaración de variables). Al declararlas se reserva una posición de memoria para la misma, donde se almacenará el valor que toma cada variable en cada momento del programa. El nombre de la posición es el NOMBRE DE LA VARIABLE y el valor almacenado es el VALOR DE LA VARIABLE. Las variables pueden ser de todos los tipos de datos conocidos: entero, reales, carácter y cadena de caracteres.

DECLARACIÓN DE VARIABLES

Todas las variables han de ser declaradas antes de poder ser usadas. La forma general de declaración es la que se muestra a continuación: **tipo** (hace referencia al tipo de datos) **lista de variables** (son los nombres de las variables.

Aquí, tipo debe ser un tipo de datos válido de C y la lista de variables puede consistir en uno o más nombres de identificadores separados por comas. A continuación se muestran algunas declaraciones en Pseudocódigo y en C:

En Pseudocódigo	En C
entero cantidad;	int cantidad;
real valor;	float valor;
carácter letra;	char letra;
cadena palabra[cantidad];	char palabra[cantidad];

Recuerde que en C el nombre de una variable no tiene nada que ver con su tipo. Existen dos tipos básicos donde se pueden declarar variables: dentro de las funciones y fuera de todas las funciones. Estas variables son, respectivamente, las variables locales y las variables globales.

I. Variables locales.

Las variables que se declaran dentro de una función se denominan variables locales. Las variables locales pueden ser denominadas sólo por sentencias que estén dentro del bloque en el que han sido declaradas. O dicho de otra forma, las variables locales no son conocidas fuera de su propio bloque de código. Recuerde que un bloque de código comienza con una llave abierta y termina con una llave cerrada.

Una de las cosas más importantes que hay que comprender sobre las variables locales es que sólo existen mientras se está ejecutando el bloque de código en el que son declaradas, o sea, la variables local se crea al entrar en el bloque y se destruye al salir de él.

II. Variables globales.

A diferencia de las variables locales, las variables globales se conocen a lo largo de todo el programa y se pueden usar en cualquier parte del código. Además, mantienen todo su valor durante toda la ejecución del programa. Las variables globales se crean al declararlas en cualquier expresión independientemente de la función en la que se encuentre la expresión.

INICIALIZACIÓN DE VARIABLES

En Pseudocódigo	En C
entero cantidad ← 10;	int cantidad = 10;
real valor ← 8.2;	float valor = 8.2;
carácter letra ← ‘a’ ;	char letra = ’a’;
entero nro1 ← 3, nro2 ← 80;	int nro1 = 3, nro2 = 80;

CONSTANTES

Dato invariable a lo largo del programa. Es un valor que no puede cambiar durante la ejecución del programa; recibe un valor en el momento de la compilación del programa y este valor no puede ser modificado.

El lenguaje C soporta todo tipo de dato como constante, además de las de los tipos de datos predefinidos. Se trata de una cadena. Una constante de cadena siempre irá encerrada entre dobles comillas, como en "LA FACULTAD”. Una constante de un solo carácter va entre comillas simples, como 'a'.

En Pseudocódigo	En C
MAXIMO 100	#define MAXIMO 100
NOMBRE “LA FACULTAD”	#define NOMBRE “LA FACULTAD”
SIMBOLO ‘a’	#define SIMBOLO ‘a’
VALOR 7.50	#define VALOR 7.50

Para la declaración de una constante se utiliza la sentencia define

El carácter numeral “#” sirve para que le indiquemos al C que vamos a utilizar una sentencia del preprocesador. Este carácter se utiliza tanto para la declaración de constantes como para incluir las librerías internas o archivos de encabezado que vamos a utilizar para la ejecución de un programa.

En Pseudocódigo	En C
No existe	#include <librería.h>

Reglas para formar el identificador (nombre) de una variable o constante:

Para la declaración de una librería se utiliza la sentencia include. Durante este curso usaremos exclusivamente las librerías que están indicadas en el glosario. La extensión “.h” indica que son archivos de encabezados del Lenguaje C (Librerías internas).

- Debe comenzar con una letra (mayúscula o minúscula), nunca con un número o espacio, y no deben contener espacios en blanco.
- Letras, números y el carácter guión (-) están permitidos después del primer carácter.
- La longitud de identificadores. El nombre de un identificador debe ser descriptivo de aquello que representa, y considerando además la practicidad de su invocación durante el desarrollo del algoritmo. Si es muy extenso estaremos más expuestos a errores al nombrarlo.
- Normalmente los nombres de las variables se colocan en letras minúsculas y las constantes con letras mayúsculas.

OPERADORES Y OPERANDOS

- **Operadores:** Son elementos que relacionan de forma diferente, los valores de una o más variables y/o constantes. Es decir, los operadores nos permiten manipular valores.
- **Operandos:** Son los datos a ser procesados.
- **Tipos de Operadores**
  - Asignación
  - Aritméticos
  - Relacionales
  - Lógicos

OPERADOR DE ASIGNACIÓN

La asignación consiste, en el paso de valores o resultados a una zona de la memoria. Dicha zona será reconocida con el nombre de la variable que recibe el valor.

variable ← expresión
(Una expresión puede ser una variable, una constante, una expresión o fórmula a evaluar)

La asignación se puede clasificar de la siguiente forma:

- **Simples:** Consiste en pasar un valor constante a una variable:

En Pseudocódigo	Ejemplo	En C	Ejemplo
←	a ← 15	=	a = 15

- **Contador:** Es una variable que se incrementa, cuando se ejecuta, en una unidad o en una cantidad constante:

En Pseudocódigo	En C	Acción
contador ← contador + 1	**contador = contador + 1**	Contador
multiplo ← multiplo + 3	**multiplo = multiplo + 3**	Múltiplo

- **Acumulador:** Es una variable que se incrementa en una cantidad variable

En Pseudocódigo	En C	Acción
suma ← suma + numero	**suma = suma + numero**	Acumulador

(Donde "numero" es una variable que puede recibir distintos valores)

Considerar:

- Una variable del lado derecho debe tener valor antes de que la sentencia se ejecute. Si "numero" no tiene valor antes de: suma ← suma + numero Se produce un Error LÓGICO. Se dice que "numero" no se ha **inicializado.**
- A la izquierda de una sentencia de asignación sólo puede haber variables. No puede haber operaciones.

Nota: la operación de asignación es una **operación destructiva** debido a que el valor almacenado en una variable se pierde o destruye y se sustituye por el nuevo valor de asignación.

Ejemplos: numero ← 16
numero ← -23

"La variable **numero** conservará el último valor asignado, en este caso **-23"**

- **Entrada:**

La entrada de datos consiste en recibir desde un dispositivo de entrada (por ejemplo un teclado) un valor. Esta operación se representa en Pseudocódigo como sigue:

Leer (a)
Leer (b)

Donde "**a**" y "**b**" son las variables que recibirán los valores

- **Salida:**

Consiste en mandar por un dispositivo de salida (por ejemplo: monitor o impresora) un resultado o mensaje. Este proceso se representa en pseudocódigo como sigue:

Mostrar ("El resultado es:", R)

Donde "**El resultado es:**" Es el mensaje que vamos a ver y **R** es una variable que contiene un valor.

OPERADORES ARITMÉTICOS

- Los operadores aritméticos permiten la realización de operaciones matemáticas con los valores (variables y constantes).
- Los operadores aritméticos pueden ser utilizados con tipos de datos enteros o reales. Si ambos son enteros, el resultado es entero; si alguno de ellos es real, el resultado es real.

En Pseudocódigo	Ejemplo	En C	Ejemplo	Acción
-	a ← 6 - 3	**-**	a = 6 - 3	Resta
+	a ← 6 + 3	**+**	a = 6 + 3	Suma
*****	a ← 6 * 3	*****	a = 6 * 3	Multiplicación (Producto)
/	a ← 6 / 3	**/**	a = 6 / 3	División
mod	a ← 6 mod 3	**%**	a = 6 % 3	Resto de la división
Fórmula	a ← a + 1	**++**	a = ++	Incremento
Fórmula	a ← a - 1	**--**	a = --	Decremento

OPERADORES RELACIONALES

- Se utilizan para establecer una relación entre dos valores.
- Compara estos valores entre sí y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).
- Los operadores relaciónales comparan valores del mismo tipo (numéricos o cadenas)
- Tienen el mismo nivel de prioridad en su evaluación.
- Los operadores relaciónales tiene menor prioridad que los aritméticos.

En Pseudocódico	En C	Acción
>	**>**	Mayor que
<	**<**	Menor que
>=	**>=**	Mayor o igual que
<=	**<=**	Menor o igual que
=	**==**	Igual que
<>	**!=**	Distinto o no igual

OPERADORES LÓGICOS

- Estos operadores se utilizan para establecer relaciones entre valores lógicos.
- Estos valores pueden ser resultado de una expresión relacional.
- En el término operador lógico la palabra lógico se refiere a las formas en que esas relaciones pueden conectarse entre sí siguiendo las reglas de la lógica formal.

En Pseudocódigo	En C	Acción
And / Y	**&&**	Producto Lógico
Or / O	**||**	Suma Lógica
Not / No	**!**	Negación

- La clave de los conceptos de operadores lógicos es la idea de verdadero y falso. En C, verdadero es cualquier valor distinto de 0, y falso es 0. Las expresiones que utilizan los operadores lógicos devuelven el valor 1 en caso de verdadero y 0 en caso de falso, siendo la tabla de verdad para los operadores lógicos:

TABLAS DE VERDAD

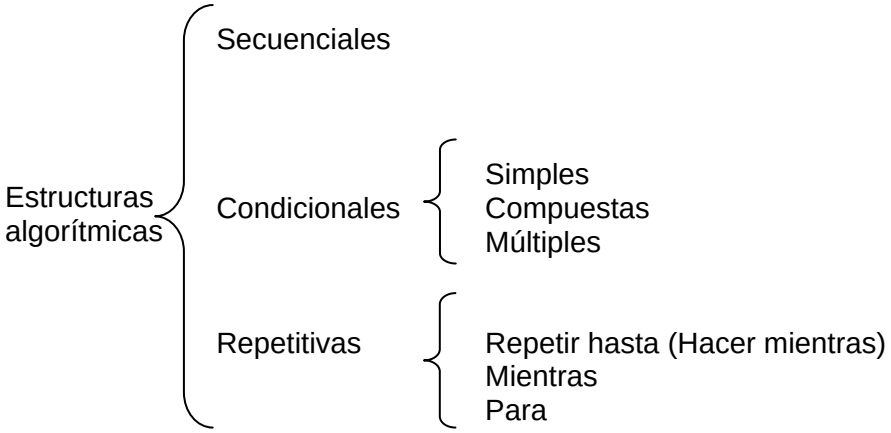
Operador AND = &&		
Operando1	Operando2	Resultado
1	1	**1**
1	0	**0**
0	1	**0**
0	0	**0**

Operador OR = ||		
Operando1	Operando2	Resultado
1	1	**1**
1	0	**1**
0	1	**1**
0	0	**0**

Operador NOT = !	
Operando	Resultado
1	**0**
0	**1**

ESTRUCTURAS ALGORITMICAS

Las estructuras de operación de programas son un grupo de formas de trabajo, que permiten, mediante la manipulación de variables, realizar ciertos procesos específicos que nos lleven a la solución de problemas. Estas estructuras se clasifican de acuerdo con su complejidad en:



ESTRUCTURAS SECUENCIALES

La estructura secuencial es aquella en la que una acción (instrucción) sigue a otra en secuencia. Las tareas se suceden de tal modo que la salida de una es la entrada de la siguiente y así sucesivamente hasta el fin del proceso. Una estructura secuencial se representa de la siguiente forma:

CARACTERÍSTICAS

En Pseudocódigo	En C
COMIENZO Accion1 Accion2 . . AccionN FIN	**main()** /*Siempre se utiliza la función main() para definir la estructura principal del programa*/ **{** // Inicio del programa. **Sentencia1;** **Sentencia2;** **.** **.** **SentenciaN;** **}** // Fin del programa.

La función main() como en cualquier otra función del Lenguaje C devuelven un valor, para evitar ese inconveniente usaremos el tipo de datos void que permite que en la función en donde los utilicemos no devuelva ningún valor de retorno, con esta función, la usaremos siempre con la siguiente sintaxis: void main(void)

EJEMPLOS

1. Se deben ingresar dos números enteros cualquiera para realizar las cuatro operaciones matemáticas básicas. Mostrar los resultados correspondientes.

En Pseudocódigo

```
Comienzo // del programa.
  /*Declaro las variables.*/
  entero nro1
  entero nro2
  entero suma
  entero resta
  entero producto
  entero division
  /*Ingreso los datos*/
  leer (nro1)
  leer (nro2)
  /*Realizo los cálculos*/
  suma = nro1 + nro2
  resta = nro1 – nro2
  producto = nro1 * nro2
  division = nro1 / nro2
  /*Muestro los resultados*/
  mostrar (suma)
  mostrar (resta)
  mostrar (producto)
  mostrar (division)
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
  /*Declaro las variables.*/
  int nro1;
  int nro2;
  int suma;
  int resta;
  int producto;
  int division;
  /*Ingreso los datos*/
  scanf (“%i”, &nro1);
  scanf (“%i”, &nro2);
```

```
   /*Realizo los cálculos*/
   suma = nro1 + nro2;
   resta = nro1 – nro2;
   producto = nro1 * nro2;
   division = nro1 / nro2;
/*Muestro los resultados*/
   printf (“%i”, suma);
   printf (“%i”, resta);
   printf (“%i”, producto);
   printf (“%i”, division);
}  // Fin del programa.
```

2. Una tienda ofrece un descuento del 15% sobre el total de la compra y un cliente desea saber cuanto deberá pagar finalmente.

En Pseudocódigo

```
Comienzo // del programa.
/*Declaro las variables todas juntas*/
   real compra, descuento, pago;
   /*Ingreso el dato*/
   leer (compra)
   /*Realizo el cálculo*/
   descuento ← compra * 0.15
   pago ← compra - descuento
   /*Muestro el resultado*/
   mostrar (pago)
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
   /*Declaro las variables todas juntas*/
   float compra, descuento, pago;
   /*Ingreso el dato*/
   scanf(“%f”, &compra);
   /*Realizo los cálculos*/
   descuento = compra * 0.15;
   pago = compra -  descuento;
   /*Muestro el resultado*/
   printf(“%f”, pago);
}  // Fin del programa.
```

El carácter amperson "&" permite ingresar un dato a una variable declarada y las sintaxis "%f" y "%i" son modificadores de formato que permiten reconocer a los tipos de datos con las funciones scanf() y printf(). La 1º función corresponde a la entrada de datos y la 2º a la salida de datos.

Ver "M o d i f i c a d o r e s d e F o r m a t o" de los Anexos

ESTRUCTURAS CONDICIONALES

Las estructuras condicionales comparan una variable contra otro(s) valor(es), para que, sobre la base del resultado de esta comparación, se siga un curso de acción dentro del programa. Cabe mencionar que la comparación se puede hacer contra otra variable o contra una constante, según se necesite. Existen dos tipos básicos, las simples y las múltiples.

CARACTERÍSTICAS

Simples:

Las estructuras condicionales simples se les conoce como "Tomas de decisión". Estas tomas de decisión tienen la siguiente forma:

En Pseudocódigo	En C
SI (condición) ENTONCES Acciones FIN-SI	**if (condición)** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque.

Análisis del Pseudocódigo:	
Si ……	Indica el comando de comparación
Condición……	Indica la condición a evaluar
Entonces……	Precede a las acciones a realizar cuando se cumple la condición
Acción(es)……	Son las acciones a realizar cuando se cumple o no la condición

Compuestas:

Las estructuras condicionales compuestas permiten elegir entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición. Se representa de la siguiente forma:

En Pseudocódigo	En C
SI (condición) ENTONCES Acciones SINO Acciones FIN-SI	**if (condición)** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque. **else** **{** // Inicio del bloque. **Sentencias;** **}** // Fin del bloque.

Análisis del Pseudocódigo:	
Si ……	Indica el comando de comparación
Condición……	Indica la condición a evaluar
entonces……	Precede a las acciones a realizar cuando se cumple la condición
Acción(es)……	Son las acciones a realizar cuando se cumple o no la condición
Sino……	Precede a las acciones a realizar cuando no se cumple la condición

Nota: Dependiendo del resultado de la comparación, que puede ser verdadera o falsa, se pueden realizar una o mas acciones.

Múltiples:

Las estructuras de comparación múltiples, son tomas de decisiones especializadas que permiten comparar una variable, contra distintos posibles resultados, ejecutando para cada caso una serie de instrucciones específicas. La forma común es la siguiente:

En Pseudocódigo:	En C
LEER (variable) CON-SELECCIÓN (variable) HACER CASO constante 1: Acciones ROMPER CASO constante 2: Acciones ROMPER CASO constante N: Acciones ROMPER OTROS CASOS: Acciones FIN-SELECCIÓN	**scanf(variable);** **switch (variable)** **{** // Inicio del bloque de la estructura. **case 1:** // El 1 es la constante. **sentencias;** **break;** /* Fuerza la salida de cada condición*/ **case 2:** **sentencias;** **break;** **case N:** **sentencias;** **break;** **default:** /*Entra en esta condición si las demás no son verdaderas*/ **sentencias;** **}** // Fin del bloque de la estructura.

Análisis de una estructura múltiple:

- La estructura de selección múltiple sólo compara por igualdad el valor de la variable con cada una de las constantes de cada caso. Al encontrar una coincidencia comienza a ejecutar las sentencias en forma secuencial hasta encontrar el fin de la estructura o una instrucción que rompa la misma.
- Puede tener hasta 256 casos.
- No puede haber 2 casos con el mismo valor en la constante.
- Sólo se pueden utilizar variables de tipo carácter o enteras.
- Si la variable que se está seleccionando es de tipo carácter, las constantes de tipo carácter se colocan entre comillas simples o apóstrofes, para el caso de variables de tipo enteras, las constantes numéricas se colocan directamente.
- Puede contener casos vacíos.

EJEMPLOS

1. **Simples:** Ingresar un número positivo (debe ser mayor a 0) y mostrar el mensaje si la condición es verdadera, "**El número Nº es positivo**".

En Pseudocódigo

```
Comienzo // del programa.
   /*Declaro la variable*/
   entero nro
   /*Ingreso el dato*/
   leer (nro)
   /*Utilizo la estructura condicional simple*/
   si (nro > 0) Entonces
      /*Muestro el mensaje*/
       mostrar ("El número nro es positivo")
   fin-si
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
   /*Declaro la variable*/
   int nro;
   /*Ingreso el dato*/
   scanf("%i", &nro);
```

```
/*Utilizo la estructura condicional simple*/
  if (nro > 0)
    /*Muestro el mensaje*/
    printf("El número %i es positivo", nro);
}  // Fin del programa.
```

2. **Compuestas:** Hacer un algoritmo que calcule el total a pagar por la compra de camisas. Si se compran tres camisas o más se aplica un descuento del 20% sobre el total de la compra y si son menos de tres camisas un descuento del 10%.

En Pseudocódigo

```
Comienzo // del programa.
  /*Declaro las variables*/
  entero total, cantidad, precio;
  /*Ingreso los datos*/
  leer (cantidad)
  leer (precio)
  /*Cálculo el total*/
  total ← cantidad * precio
  /*Utilizo la estructura condicional compuesta*/
  si (cantidad >= 3 ) Entonces
    /*Cálculo el total para la condición verdadera*/
    total ← total - total * 0.20
  sino
    /*Cálculo el total para la condición falsa*/
    total ← total - total * 0.10
  fin-si
  /*Muestro el resultado*/
  mostrar (total)
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
  /*Declaro la variable*/
  int total, cantidad, precio;
  /*Ingreso los datos*/
  scanf("%i", &cantidad);
  scanf("%i", &precio);
  /*Cálculo el total*/
  total = cantidad * precio
```

```
  /*Utilizo la estructura condicional compuesta*/
   if (cantidad >= 3 )
      /*Cálculo el total para la condición verdadera*/
      total = total - total * 0.20;
   else
      /*Cálculo el total para la condición falsa*/
      total = total - total * 0.10;
   /*Muestro el resultado*/
   printf("%i",  total);
}  // Fin del programa.
```

3. **Múltiples:** Ingresar 2 números y realizar un menú de opciones para que los sume, reste o los muestre.

En Pseudocódigo

```
Comienzo // del programa.
   /*Declaro las variables*/
   entero nro1, nro2, opcion;
   entero suma, resta;
   /*Ingreso los datos*/
   leer (nro1)
   leer (nro2)
   /*Realizo el menú*/
   mostrar ("Menú")
   mostrar ("1.Suma – 2.Resta")
   /*Ingreso la opción para el menú*/
   leer (opcion)
   /*Utilizo la estructura condicional múltiple/
   con-selección (opcion) Hacer
    /*Según la opción seleccionada se ejecutará uno de los casos*/
       caso 1:
       /*Acciones para la opción 1*/
          suma ← nro1 + nro2
          mostrar ("La suma es: ", suma)
          romper
       caso 2:
       /*Acciones para la opción 2*/
          resta ← nro1 – nro2
          mostrar ("La resta es: ", resta)
          romper
       otros casos:
       /*Acciones para la opción mayor a 2*/
          mostrar ("Los Nº ingresados son:")
          mostrar (nro1, nro2)
   fin-selección
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
   /*Declaro las variables*/
   int nro1, nro2, opcion;
   int suma, resta;
/*Ingreso los datos*/
   scanf("%i", &nro1);
   scanf("%i", &nro2);
   /*Realizo el menú*/
   printf("\t\tMenú\n");
   printf("1.Suma – 2.Resta\n");
   /*Ingreso la opción para el menú*/
   scanf("%i", &opcion);
   /*Utilizo la estructura condicional múltiplo*/
   switch (opcion)
   { // Inicio de la estructura múltiple.
        case 1 :
        /*Acciones para la opción 1*/
          suma = nro1 + nro2;
          printf("\nLa suma es:  %i", suma);
          break;
        case 2:
        /*Acciones para la opción 2*/
          resta = nro1 – nro2;
          printf("\nLa resta es:  %i", resta);
          break;
        default:
        /*Acciones para la opción mayor a 2*/
          printf("\nLos Nº ingresados son:\n");
          printf("%i\t-\t%i", nro1, nro2);
   } // Fin de la estructura múltiple.
} // Fin del programa.
```

En la estructura de condicional múltiple comenzamos a utilizar las secuencias de escape "\t" para la tabulación horizontal y "\n" para el salto de línea o ENTER.

Ver "S e c u e n c i a s d e E s c a p e" de los Anexos

ESTRUCTURAS REPETITIVAS

Se llaman problemas repetitivos o cíclicos a aquellos en cuya solución es necesario utilizar un mismo conjunto de acciones que se puedan ejecutar una cantidad específica de veces. Esta cantidad puede ser fija (previamente determinada por el programador) o puede ser variable (estar en función de algún dato dentro del programa).

Estos tipos de estructuras generan un número de iteraciones que no se conocen con exactitud, ya que esta dado en función de un dato dentro del programa.

Este tipo de estructura se utiliza normalmente para los arreglos (**arrays**) unidimensionales (**vectores**) y bidimensionales o multidimensionales (**matrices**).

Las estructuras repetitivas se clasifican en: **MIENTRAS-HACER, HACER-MIENTRAS y PARA-HACER**.

CARACTERÍSTICAS

1. **MIENTRAS-HACER:** Ejecuta las sentencias que contiene mientras la condición sea VERDADERA. Primero evalúa la condición, y de ser VERDADERA ejecuta las sentencias. Esto supone que la variable que forma parte de la condición tenga un valor inicial.

En Pseudocódigo	En C
MIENTRAS (condición) HACER Acciones FIN-MIENTRAS	**while (condición)** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura.

2. **HACER-MIENTRAS:** Ejecuta las sentencias que contiene mientras la condición sea VERDADERA. A diferencia de la estructura anterior, primero ejecuta las sentencias y luego evalúa la condición.

En Pseudocódigo	En C
HACER Acciones MIENTRAS (condición)	**Do** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura. **while (condición);**

3. **PARA-HACER:** Son aquellos en que el numero de iteraciones se conoce antes de ejecutarse el ciclo. Esta estructura cuenta con tres campos: variable, condición y el contador.

Estructura del PARA-HACER:

En Pseudocódigo	En C
PARA (variable; condición; contador) HACER Acciones FIN-PARA	**for (variable; condición; contador)** **{** // Inicio de la estructura. **Sentencias;** **}** // Fin de la estructura.

Análisis de la estructura:
variable: Esta variable cumple la función de asignarle un valor inicial al contador, el valor puede ser cualquiera. A está variable contador se la conoce en los arreglos como **índice**. **condición**: La condición le indica a la estructura PARA-HACER **desde** y **hasta** que número se va a repetir el ciclo. Desde, nos indica el límite inferir y hasta límite superior. **contador:** El contador permite incrementar o decrementar su valor inicial hasta que se cumpla la condición. Se puede incrementar o decrementar con los que sean necesarios para el programa (de 1 en 1, de 2 en 2, etc.)

Nota: En este ciclo la variable de control toma el valor inicial del ciclo y el ciclo se repite hasta que la variable de control llegue al límite superior. La cantidad de repeticiones que tenga depende del límite superior y del incremento de la variable.

EJEMPLOS

MIENTRAS-HACER: Leer 10 números, sumarlos y mostrar el resultado.

En Pseudocódigo

```
Comienzo // del programa.
/*Declaramos e inicializamos las variables para el contador y el acumulador*/
  entero suma ← 0  // Acumulador.
  entero contador ← 0 // Contador.
  entero numero
  borrar pantalla
  /*Utilizo la estructura repetitiva mientras-hacer que el contador sea menor a 10*/
  mientras (contador < 10)  hacer
    /*Ingreso el dato*/
    leer (numero)
```

```
    /*Incremento el contador*/
    contador ← contador + 1
    /*Uso el acumulador suma*/
    suma ← suma + numero
  fin-mientras
  /*Mostramos el contenido del acumulador*/
  mostrar (suma)
  /*Hago una pausa en espero que se pulse una tecla para continuar*/
  pausa
Fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
#include <conio.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
   /*Declaramos e inicializamos las variables para el contador y el acumulador*/
   int suma = 0;
   int contador = 0;
   int numero;
   /*Esta función borra la pantalla*/
   clrscr();
   /*Utilizo la estructura repetitiva mientras que el contador sea menor a 10*/
   while (contador<10)
   {  // Inicio la estructura.
      /*Ingreso el dato*/
      scanf("%i", &numero);
      /*Incremento el contador*/
      contador = contador + 1;
      /*Uso el acumulador suma*/
      suma = suma + numero;
   }  // Fin de la estructura.
   /*Mostramos el contenido de acumulador*/
   printf("\nLa suma es: %i", suma);
   /*Esta función espera que se pulse una tecla para continuar*/
   getch();
} // Fin del programa.
```

HACER-MIENTRAS: Ingresar **n** cantidad de números y averiguar cual es el número mayor y el menor. Mostrarlos.

En Pseudocódigo

```
comienzo // del programa.
/*Declaramos e inicializamos las variables para averiguar el valor
   máximo y mínimo*/
   entero mayor ← 0  /*Se debe inicializar con el valor mínimo
   posible para que cualquier número ingresado sea mayor a esta
   variable*/
   entero menor ← 9999 /*Se debe inicializar con el valor mayor
posible para que cualquier número ingresado sea menor a esta
   variable*/
   entero numero
   /*Declaro la variable para poder salir del programa*/
   carácter continuar ← 's' /*Las variables del tipo carácter se
   inicializan con el carácter entre apóstrofes*/
   borrar pantalla
   /*Utilizo la estructura repetitiva hacer-mientras hasta que
   decidamos lo contrario*/
   hacer
      /*Ingreso el dato*/
      leer (numero)
      /*Averiguo el valor máximo*/
      si (numero > mayor) entonces
      /*Le asigno el valor de numero a la  variable mayor*/
        mayor ← numero
      fin-si
      /*Averiguo el valor mínimo*/
      si (numero < menor) entonces
      /*Le asigno el valor de numero a la variable menor*/
        menor ← numero
      fin-si
       /*Decido si quiero ingresar más números*/
      mostrar ("Continuo S/N")
      leer (continuar)
  /*Sí pulsamos un carácter distinto de 's' continuamos dentro del
programa*/
   mientras (continuar ← 's')
   /*Mostramos los valores máximo y mínimo*/
   mostrar ("Máximo: ", mayor)
   mostrar ("Mínimo: ", menor)
   /*Hago una pausa en espero que se pulse una tecla para conti-
nuar*/
   pausa
fin // del programa.
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
/*La función main() con el tipo de dato void define la estructura principal del programa*/
void main(void)
{  // Comienzo del programa.
/*Declaramos e inicializamos las  variables para averiguar el valor máximo y mínimo*/
   int mayor = 0;  /*Se debe inicializar con el valor mínimo posible para que  cualquier número ingresado sea  mayor a esta variable*/
   int menor = 9999; /*Se debe inicializar con el valor mayor posible para que cualquier número ingresado sea menor a esta variable*/
   int numero;
/*Declaro la variable para poder salir  del programa*/
   char continuar = 's'; /*Las variables del tipo carácter se inicializan con el  carácter entre apóstrofes*/
   system("cls"); /*De esta forma ejecutamos al comando cls del DOS para borrar la pantalla*/
   /*Utilizo la estructura repetitiva hacer-mientras hasta que decidamos lo contrario*/
   do
   { // Inicio la estructura.
      /*Ingreso el dato*/
      scanf("%i", &numero);
      /*Averiguo el valor máximo*/
      if (numero > mayor)
      /*Le asigno el valor de numero a la variable mayor*/
         mayor = numero;
      /*Averiguo el valor mínimo*/
      if (numero < menor)
      /*Le asigno el valor de numero a la variable menor*/
         menor = numero;
      /*Decido si quiero ingresar más números*/
      printf("\nContinuo S/N:\t");
     /*Para ingresar un solo carácter conviene hacerlos de esta forma*/
      continuar = getche(); //Esta función muestra el carácter que
                                 pulsamos.
  /*Sí pulsamos un carácter distinto de 's' continuamos dentro del programa*/
   } // Fin de la estructura.
   while (continuar == 's');
```

```
  /*Mostramos los valores máximo y mínimo*/
  printf("\nMáximo:\t%i", mayor);
  printf("\nMínimo:\t%i", menor);
  /*Hago una pausa en espero que se pulse una tecla para continuar*/
  system("pause"); /*De esta forma ejecutamos al comando pause del DOS para detener el programa, aparecerá en pantalla el mensaje de pulsar una tecla para continuar sin necesidad de escribir dicho mensaje con la función printf()*/
}// Fin del programa.
```

PARA-HACER: Ingresar 10 números y mostrarlos en forma de lista. Sacar el promedio y mostrar el resultado.

En Pseudocódigo

```
CANTIDAD 10  /*Cantidad es una constante*/
comienzo
  /*declaro las variables*/
  entero vector[CANTIDAD] /*Entre corchetes se ponen la cantidad de elementos del arreglo*/
  entero contador /*El contador cumple la función de un índice para los arreglos*/
  real promedio
  entero suma ← 0
/*Está variable la utilizamos para ingresar el título que queremos*/
  carácter titulo[30]
  borrar pantalla
  para /*Inicializo el contador*/
        (contador ← 0; /*La condición es desde el límite inferior 0 hasta el límite superior 9*/
        contador < 10; /*Incremento el contador de 1 en 1*/
        contador ← contador + 1) /*Ingreso un número en el arreglo en la posición en que se encuentre el contador (índice)*/
     leer (vector[contador])
/*Realizamos el cálculo con los valores ingresados en el  arreglo*/
     suma←suma+vector[contador]
  fin-para
/*Ingresamos el siguiente título "Estos son los Nº ingresados"*/
  mostrar ("Ingresamos el Título:");
  leer ("%s", titulo);
  borrar pantalla
/*Los títulos se ponen siempre fuera de las estructuras repetitivas*/
  mostrar ("titulo")
```

```
   para (/*Inicializo el contador*/
            contador ← 0;
            /*La condición es desde el límite    inferior 0 hasta el límite
superior 9, en este caso en vez de poner el número de elementos
del arreglo ponemos directamente la constante*/
            contador < CANTIDAD;
            /*Incremento el contador de 1 en 1*/
            contador ← contador + 1)
/*Muestro el número que encuentra en la posición del contador
(índice) dentro del arreglo*/
      mostrar (vector[contador])
   fin-para
/*Realizo el cálculo del promedio usando el acumulador y el conta-
dor*/
   promedio ← suma / contador
/*Muestro el promedio*/
   mostrar ("Promedio: ", promedio)
/*Hago una pausa en espero que se    pulse una tecla para conti-
nuar*/
   pausa
Fin
```

En C

```
/*Indico las librerías que voy a utilizar*/
#include <stdio.h>
#include <conio.h>
#include <string.h>
#define CANTIDAD 10 /*Cantidad es una constante*/
void main(void)
{ //Comienzo del programa.
   /*declaro las variables*/
   int vector[CANTIDAD]; /*Entre corchetes se ponen la cantidad de
elementos del arreglo*/
   int contador; /*El contador cumple la función de un índice para
los arreglos*/
   float promedio;
   int suma = 0;
/*Está variable la utilizamos para ingresar el título que queremos*/
   char titulo[30];
   system("cls"); /*De esta forma ejecutamos al comando cls del
DOS para borrar la pantalla*/
   for (contador = 0;
/*Inicializo el contador*/
            contador < 10;
/*La condición es desde el límite inferior 0 hasta el límite superior 9*/
            contador = contador + 1)
/*Incremento el contador de 1 en 1*/
```

```
   { //Inicio de la estructura
/*Ingreso un número en el arreglo en la posición en que se encuentre el contador (índice)*/
      scanf("%i", &vector[contador]);
/*Realizamos el cálculo con los valores ingresados en el  arreglo*/
      suma = suma+vector[contador];
   } // Fin de la estructura
/*Ingresamos el siguiente título "Estos son los Nº ingresados"*/
   printf("\nIngresamos el Título:\t");
   scanf("%s", titulo);
   system("cls"); /*De esta forma ejecutamos al comando cls del DOS para borrar la pantalla*/
/*Los títulos se ponen siempre fuera de las estructuras repetitivas*/
   printf("\n%s\n", titulo)
  for /*Inicializo el contador*/
        (contador = 0;
/*La condición es desde el límite inferior 0 hasta el límite superior 9, en este caso en vez de poner el número de elementos del arreglo ponemos directamente la constante*/
        contador < CANTIDAD;
/*Incremento el contador de 1 en 1*/
        contador = contador + 1)
   { // Inicio de la estructura
/*Muestro el número que encuentra en la posición del contador (índice) dentro del arreglo*/
      printf("%i\n", vector[contador]);
    } // Fin de la estructura
/*Realizo el cálculo del promedio usando el acumulador y el contador*/
    promedio = suma / contador;
/*Muestro el promedio*/
    printf("Promedio:%f\t", promedio);
   system("pause"); /*De esta forma ejecutamos al comando pause del DOS para detener el programa, aparecerá en pantalla el mensaje de pulsar una tecla para continuar sin necesidad de escribir dicho mensaje con la función printf()*/
} // Fin del programa.
```

ESTILO DE PROGRAMACIÓN

Antes de escribir los algoritmos de los ejercicios en Pseudocódigo considerar:

Un programa legible y comprensible **es más fácil de:**
- **Entender**
- **Corregir**
- **Mantener**

Seguir las siguientes sugerencias:

1. **SANGRADO O INDENTACIÓN**. En cada estructura, y alineando las instrucciones (sentencias) dentro de cada una de ellas y dentro de todo el algoritmo (Comienzo, Fin)
2. **LÍNEAS EN BLANCO**. Dejarlas entre partes importantes o que estén lógicamente separadas. Recomendable luego de cada estructura (Repetitiva o Selectiva), luego de declaración de variables.
3. **COMENTARIOS**. Parte importante de la documentación de un programa que permite mayor comprensión del mismo usaremos el mismo formato que en el lenguaje C:
   - **Párrafo: /***Esto es un comentario (puede ocupar varias líneas.***/**
   - **Línea: //**Esto va en la misma línea.
4. **NOMBRES SIGNIFICATIVOS DE IDENTIFICADORES**. que representen aquello que estamos tratando. Si son palabras compuestas usar guión común. Todos deben comenzar con una letra.
5. **CADA SENTENCIA EN UNA LÍNEA DISTINTA**. Al colocar una nueva sentencia comenzar una nueva línea, incluso las palabras claves de las estructuras en líneas separadas.
6. **ESPACIOS ENTRE ELEMENTOS DE UNA SENTENCIA**. lo hace más legible. Por ejemplo: **suma ← suma + numero**

CREACIÓN Y DESARROLLO DEL LENGUAJE C[1]

La versión original de C fue desarrollada en los laboratorios de la Bell Telephone por Denis Ritchie, en los años 70. Fue implementada por Ritchie en una DEC PDP 11, con sistema operativo UNIX.

Ritchie creó el lenguaje basándose en otros dos existentes: BCPL, desarrollado por Martin Richards en 1967, y basado en el lenguaje CPL, que había sido creado en 1963 en la Universidad de Cambridge y el B, que había ideado Ken Thompson,también basado en el BCPL. Todos estos lenguajes - el BCPL, el B y, como dijimos, el C - fueron desarrollados en los laboratorios de la Bell Telephone. Se creaban para uso exclusivo de los laboratorios y no eran conocidos fuera de ellos. Estaban ligados, fundamentalmente, al sistema operativo UNIX, en cuyo entorno se trabajaba y que Denis Ritchie rescribe en C. Por todo ello se piensa en C como un lenguaje de programación de sistemas, aunque, al salir de los laboratorios Bell, comienza a ser utilizado para muchas otras aplicaciones.

Recién en 1978 el lenguaje C sale de los laboratorios Bell cuando Ritchie, en colaboración con Brian Kernighan, publica "*The C Programming Language" ,* editado por Prentice Hall.

A partir de allí comienza a extenderse el uso del lenguaje. Se escriben varios compiladores para permitir el uso del C en distintos equipos y se desarrollan infinidad de programas para aplicaciones comerciales y científicas.

A pesar de la alta portabilidad del lenguaje, las distintas implementaciones de C tenían algunas incompatibilidades.

Esto lleva al Instituto Nacional Americano de Estándares (ANSI) a formar, en 1983, una comisión para lograr una versión normalizada del lenguaje. Esta versión se completa con 1989: se la denomina ANSI C.

Pero desde 1980 Bjorne Stroustrup, también de los laboratorios Bell, trabajaba en un lenguaje que puede ser considerado como un súper conjunto del lenguaje C aunque posee también componentes del Algol 68 y del Simula 67. Este lenguaje es el C++., que Stroustrup describe en su libro "*The C++ Programming Language".*

En 1983 comienza a ser utilizado el lenguaje fuera del grupo de su creador. C++ mantiene la compatibilidad con C.

Lo más importante de C++ es su posibilidad de utilización en programación orientada a objetos.

CARACTERÍSTICAS DEL LENGUAJE C[2]

Vamos a encontrar a menudo que a C se lo califica como un lenguaje de nivel medio. Esto es debido a que C no sólo puede utilizarse como lenguaje de alto nivel sino que también permite trabajar a bajo nivel, a la par de un lenguaje ensamblador. Como lenguaje de alto nivel es un lenguaje estructurado, como el Algol, el Pascal y el ADA entre otros.

Recordemos que en programación estructurada una de las normas esenciales es la modularidad. El lenguaje C lleva a sus últimos extremos esta modularidad: todo programa es un conjunto de bloques totalmente independientes.

Estos bloques se denominan *funciones*. Una función es una subrutina que contiene una o más instrucciones y que realiza una o mas acciones. Existe una biblioteca estándar de C que constituye una importante recopilación, que el programador puede utilizar, además de las funciones que él mismo cree.

Cada función está individualizada por un *identificador*, es decir, un nombre, que elige libremente el usuario respetando ciertas normas que daremos en el próximo apartado y, eventualmente, una lista de argumentos entre paréntesis. Si no necesita parámetros se coloca a continuación del identificador, paréntesis vacíos.

La función que inicia el programa - una especie de programa principal - es la única a la que no podemos asignarle libremente un identificador; ella recibe el nombre de *main.*

Dentro de cada bloque las estructuras de control permiten crear sub bloques que pueden considerarse como una unidad. Estos sub bloques se encierran entre llaves: una llave que abre (**{**) al comienzo y una que cierra (**}**) al final del bloque.

C tiene un conjunto de instrucciones pequeño con respecto a otros lenguajes de alto nivel. Las instrucciones básicas están apoyadas y mejoradas por las numerosas funciones de la biblioteca estándar.

Los compiladores son de pequeña dimensión así como los programas objeto generados por ellos.

1-2 Introducción del libro escrito por el Ing. Gustavo Viard para la asignatura Informática I de la especialidad Ingeniería Electrónica de la Facultad Regional Avellaneda perteneciente a la UTN.

LIBRERIAS + FUNCIONES

LIBRERIAS	TRADUCCIÓN	FUNCIONES ASOCIADAS
conio.h	Consola de E/S	**clrscr();** // BORRA LA PANTALLA. **getch();** // ESPERA QUE SE PULSE UNA TECLA PARA CONTINUAR, "NO SE ESCRIBE EN PANTALLA EL CARÁCTER SELECCIONADO". **getche();** // ESPERA QUE SE PULSE UNA TECLA PARA CONTINUAR, "SE ESCRIBE EN PANTALLA EL CARÁCTER SELECCIONADO". **gotoxy();** // POSICIONA CURSOR.
ctype.h	Operaciones con caracteres	**tolower();** //PASA UNA LETRA EN MAYÚSCULAS A MINÚSCULAS. **toupper();** //PASA UNA LETRA EN MINÚSCULAS A MAYUSCULAS.
stdio.h	Estándar de E/S	**printf();** //MOSTRAR MENSAJES EN PANTALLA. **scanf();** //INGRESO DE DATOS DE DISTINTOS TIPOS, EN EL CASO QUE EL TIPO DE DATO SEA UNA CADENA DE CARACTERES "NO SE PUEDEN INGRESAR CADENAS DE CARACTERES COMPUESTAS".
stdlib.h	Librería Estándar de Propósito General	**exit();** //TERMINA LA EJECUCION DEL PROGRAMA. **system();** //EJECUTA UN COMANDO / PROCESO EXTERNO.
string.h	Cadenas de Caracteres	**gets();** //INGRESO DE CADENAS DE CARACTERES COMPUESTAS. **puts();** //MOSTRAR CADENAS DE CARACTERES. **strcat();** //CONCATENA CADENAS. **strlen();** //CUENTA LOS CARACTERES DE UNA CADENA. **strcpy();** //COPIA CADENAS. **strcmp();** //COMPARA CADENAS.

MODIFICADORES DE FORMATO

Hemos visto por encima cómo mostrar datos en pantalla y cómo aceptar la introducción de datos por parte del usuario, mediante "printf" y "scanf", respectivamente. Veamos ahora su manejo y algunas de sus posibilidades con más detalle, junto con otras órdenes alternativas: El formato de printf es printf(formato, lista de variables); Dentro del apartado de "formato" habíamos comentado que "%d" indicaba que se iba a escribir un número entero, y que "%s" indicaba una cadena de texto. Vamos a resumir en una tabla las demás posibilidades, que no habíamos tratado todavía:

MODIFICADOR	FORMATO POR TIPO DE DATO
%i o **%d**	Número entero con signo, en notación decimal.
%u	Número entero sin signo, en notación decimal.
%o	Número entero sin signo, en notación octal (base 8).
%x	Número entero sin signo, en hexadecimal (base 16), minúsculas.
%X	Número entero sin signo, en hexadecimal, mayúsculas.
%f	Número con decimales (coma flotante o punto flotante).
%e	Número real en notación científica.
%g	Usa el más corto entre %e y %f.
%c	Un único carácter.
%s	Cadena de caracteres.
%%	Signo de tanto por ciento: %
%p	Puntero (dirección de memoria)
%n	Se debe indicar la dirección de una variable entera (como en scanf), y en ella quedará guardado el número de caracteres impresos hasta ese momento.

EJEMPLO: Uso de los Modificadores de Formato.

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*Declaro variables globales*/
int   entero = 1234, enteroNeg = -1234, contador;
float real = 234.567;
char  letra = 'E', mensaje[20] = "Un texto";
main() //Esta es la función principal del programa.
{
  system("cls");
  printf("El número entero vale %d en notación decimal,\n", entero);
  printf("  y %o en notación octal,\n", entero);
  printf("  y %x en notación hexadecimal,\n", entero);
  printf("  y %X en notación hexadecimal en mayúsculas,\n", entero);
  printf("  y %ld si le hacemos que crea que es entero largo,\n", entero);
  printf("  y %10d si obligamos a una cierta anchura,\n", entero);
  printf("  y %-10d si ajustamos a la izquierda.\n", entero);
  printf("El entero negativo vale %d\n", enteroNeg);
  printf("  y podemos hacer que crea que es positivo: %u (incorrecto).\n",
         enteroNeg);
  printf("El número real vale %f en notación normal\n", real);
  printf("  y %3.2f si limitamos a dos decimales,\n", real);
  printf("  y %e en notación científica (exponencial).\n", real);
  printf("La letra es %c y el texto %s.\n", letra, mensaje);
  printf("  Podemos poner \"tanto por ciento\": 50%%.\n");
  printf("Finalmente, podemos escribir direcciones. de memoria: %p.\n",
          letra);
  printf("  y contar lo escrito hasta aquí %n", &contador);
  printf(", que ha sido: %d letras.\n", contador);
  system("pause");
  return 0; /* Esta sentencia reemplaza al void en una función, pero a
diferencia retorna el valor que le indiquemos que puede ser una variable o una
constante*/
}
```

SECUENCIAS DE ESCAPE	
Las secuencias de escape cumplen cuatro funciones:	1. Cambiar de lugar al cursor. 2. Colocar algún tipo de carácter específico (comillas, apóstrofe o contrabarra). 3. Indicar el fin de una cadena de caracteres. 4. Hacer una llamada.
SECUENCIA	**FUNCIÓN**
\n	Nueva línea (Enter)
\r	Retorno de carro (impresora).
\b	Retroceso (backspace)
\f	Salto de página.
\t	Tabulación horizontal
\"	Comillas dobles (")
\'	Apóstrofe o comillas simples (')
****	Barra invertida ó Contrabarra (\)
\a	Alerta (sonido).
\0	Carácter nulo ó Fin de cadena de caracteres.
\ddd dígitos)	Constante octal (máximo tres)
\xddd	Constante hexadecimal (ídem)

Ejemplo: Uso de las Secuencias de Escape

```
#include <stdio.h>
#include <conio.h>
main()
{
        clrscr();
        printf("\"Uso de Comillas\"");
        printf("\tTabulación Horizontal");
        printf("\nSalto de Línea\n");
        printf("\¿Entre signos de interrogación\?\n");
        printf("\'Apóstrofo\'\n");
        printf("\\Contrabarra\b\b\b Retroceso");
        printf("\nFin de una cadena de\0 caracteres");
        printf("\nS\a o\a n\a i\a d\a o\a s\a");
        getch();
        return(0); /* Esta sentencia reemplaza al void en una función, pero a diferencia retorna el valor que le indiquemos que puede ser una variable o una constante*/
}
```

COMENTARIOS A TENER EN CUENTA

1. Las constantes ó variables del tipo carácter se colocan entre apóstrofes (comillas simples) cuando es para un sólo carácter y las cadenas siempre entre comillas dobles.
2. Para salir de una estructura repetitiva antes de que finalice (por su condición) o alcance el número de iteraciones indicados (en el para) utilizamos la sentencia ROMPER.
3. Para salir de cada CASO de una estructura de selección múltiple se debe utilizar la sentencia ROMPER.
4. Los comentarios los hacemos como lo indica él lenguaje C de la siguiente manera: **/*Para un Párrafo*/** y **//Para una Línea**.
5. Por cada acción en Pseudocódigo debe existir una sentencia de un lenguaje de programación asociada.
6. Cuando dentro de una estructura hay más de una sentencia las mismas se deben poner entre llaves **{ Sentencias; }** que indican el comienzo o final de un bloque respectivamente.
7. Cada bloque de un programa en el lenguaje C comienza con una llave de apertura **"{"**, y debe terminar con una la llave de cierre **"}"**.
8. Después de cada sentencia en el lenguaje C se debe poner punto y coma "**;**", en el pseudocódigo no es necesario.
9. Todo contador cumple la función de un índice.
10. Un acumulador cumple la función de un sumador.
11. Tanto los acumuladores como los contadores deben ser inicializados para que comiencen a sumar o contar a partir de un número determinado.
12. Todos los mensajes de salida deben poner entre comillas dobles.
13. Las constantes en el lenguaje C se escriben para diferenciarlas de las variables siempre en mayúsculas.
14. Las variables y constantes en el lenguaje C no deben empezar con un número ni deben tener espacios entre caracteres.
15. Los nombres de las constantes y variables deben ser claros.
16. La mayoría de las sentencias del lenguaje C son funciones.
17. Toda función para poder ser utilizada debe tener como archivo de encabezamiento a la librería a la cual pertenece.
18. Al lenguaje C se lo cataloga con un lenguaje de bajo nivel, dado que sirve para programar al hardware.
19. En el lenguaje C se pueden utilizar por defecto hasta 16 colores, tanto para los fondos como para la fuente. Si se quieren usar más se deben incorporar como archivos de encabezados a librerías específicas.
20. Las librerías cumplen la función de brindarle información a los programas, tanto para su creación como para su ejecución. Una librería puede formar parte de un programa (**internas)**, o sea que se las incorpora dentro de él, como ocurre con los archivos de encabezado que se utilizan en el lenguaje C (***.h**), o pueden ser compartidas por varios programas que requieren de información adicional para poder continuar con su ejecución (**externas**), como ocurre por ejemplo con las librerías dinámicas de Windows (***.DLL**).
21. Con el lenguaje C fue creado el sistema operativo Unix.

EJERCICIO TIPO

Hacer un programa que mediante un menú de opciones realice las siguientes acciones:

a. Ingresar los siguientes datos de un stock de materiales compuesto de 100 artículos: Código (Según el índice), Descripción (30 caracteres), Cantidad (Pueden ser negativa) y Precio Unitario.
b. Calcular el monto de cada artículo y mostrar el capital actual del Stock.
c. Buscar los datos de un artículo determinado por su nombre (Descripción), y mostrar a todos los datos incluyendo el monto.
d. Hacer los listados de los artículos existentes y faltantes, por separado, mostrar a todos los datos.
x. Salir del programa.*/

FUNCIONES UTILIZADAS PARA LA RESOLUCIÓN DEL PROBLEMA

fflush(stdin):	**Para vaciar el buffer del teclado.**
scanf():	**Para el ingreso de todo tipos de datos.**
gets():	**Para el ingreso de las cadenas de caracteres.**
printf():	**Para mostrar los distintos tipos de mensajes o para la ejecución de las secuencias de escape.**
strcpy():	**Para asignarle un valor a una cadena de caracteres.**
strcmp():	**Para comparar el contenido de dos cadenas de caracteres.**
strlen():	**Para calcular el tamaño de una cadena de caracteres.**
getch():	**Para detener el programa esperando que se pulse cualquier tecla.**
system():	**Nos permite ejecutar programas externos al programa que estamos utilizando.**
exit():	**Forzamos la salida de un programa.**
break:	**Rompemos la ejecución de un caso de la estructura switch().**
toupper():	**Convertimos un carácter en mayúsculas.**

RESOLUCIÓN DEL PROGRAMA:

Declaramos las librerías internas del programa:

#include <conio.h> //Librería consola de E/S.
#include <stdio.h> //Librería estándar de E/S.
#include <string.h> //Librería para el manejo de las cadenas de caracteres.
#include <stdlib.h> //Librería estándar.
#include <ctype.h> //Librería para la conversión de caracteres.

Declaramos las constantes:

#define M 100 //Defino la constante.

Inicio de la estructura principal del programa:

```
main()
{    //Primero declaramos todas las variables.
     int i; //Índice.
     int control=0; //Variable para controlar que opcion del menú debemos
                      hacer.
     char opcion='Z'; //Para seleccionar la opcion del menú
     char auxop; //Para cargar en forma temporal la opcion del menú
                   seleccionada.
     int código[M]; //Para almacenar el numero de código de articulo.
     char auxdes[30]; //Para cargar el forma temporal la cadena de caracteres
                        a validar.
     char descripcion[M][30]; //Para almacenar el nombre del articulo.
     float pu[M]; //Para almacenar el precio unitario del articulo.
     int cantidad[M]; //Para almacenar la cantidad de artículos del stock.
     float auxprecio; //Para cargar en forma temporal el precio unitario.
     float monto[M]; //Para el almacenar el precio unitario.
     double capital=0; //Sirve como acumulador de los montos calculados.
```

Diseñamos el menú de opciones:

```
     while(opcion!='X')
     {    system("cls"); //El proceso ejecutado por esta función borra la
                           pantalla.
          //Menú de opciones.
          printf("\n\tMENU");
          printf("\nA. ALTAS");
          printf("\nB. CALCULOS");
          printf("\nC. BUSQUEDA");
          printf("\nD. LISTADO");
          printf("\nX. SALIR");
          printf("\nElegir Opción:\t");
          fflush(stdin); //Vaciamos el buffer del teclado.
          scanf("%c", &auxop); //Seleccionamos la opción.
          opcion=toupper(auxop); /*Pasamos la opción seleccionada a
                                   mayúsculas.*
```

Resolvemos cada opción del menú:

ALTAS: Ingreso de datos.

```
switch(opcion)
{               case 'A':
        if(control==0) /*Controlamos si esta hecho la opción A del
                        menú.*/
        {   control=1; /*A la variable control le asignamos el valor 1
                        para saber que vamos a realizar el ingreso de
                        los datos.*/
            for(i=0; i<M; i=i+1)
            {   codigo[i]=i+1; //Ingresamos el código del artículo.
                printf("\nArtículo Nro:\t%i", codigo[i]);
                printf("\nNombre:\t");
                fflush(stdin);
                gets(auxdes); //Se puede usar: scanf("%s", auxdes);
                while(strlen(auxdes)>30)
                {   printf("Ingresar una nueva descripcion:\t");
                    fflush(stdin);
                    gets(auxdes);
                }
                strcpy(descripcion[i], auxdes);
                printf("\nCantidad:\t");
                scanf("%i", &cantidad[i]);
                printf("\nPrecio unitario:\t");
                scanf("%f", &auxprecio);
                pu[i]=auxprecio;
            }
        }
        break;
```

CALCULOS: Realizamos en este caso todos los cálculos pedidos en el programa.

```
    case 'B':
        if(control==1) /*Controlamos si ya esta hecha la opción A del
                        menú.*/
        {   control=2; /*A la variable control le asignamos el valor 2
                        para saber que vamos a realizar los cálculos
                        del programa.*/
            for(i=0; i<M; i=i+1)
            {   monto[i]=pu[i]*cantidad[i]; //Cálculo el monto.
                capital=capital+monto[i]; //Cálculo el capital.
            }
            printf("\nEl capital existente es %8.2f", capital);
        }
```

```
 if(control==2)
     printf("El capital ya fue calculado!!!");
 else
     printf("Primero ingresamos los datos");
break;
```

BUSQUEDA: Por el nombre del artículo.

```
case 'C':
   if(control!=2)  /*Antes de hacer cada caso controlamos si los
                     casos A y B fueron realizados*/
   {   printf("Debemos hacer primero las opciones A y B del
       menú");
   }
   else
   {   printf("Buscar al siguiente artículo\n");
       printf("Nombre:\t");
       fflush(stdin);
       gets(auxdes); //se puede usar: scanf("%s", auxdes);
       while(strlen(auxdes)>30)
       {    printf("Ingresar una nueva descripción:\t");
            fflush(stdin);
            gets(auxdes);
       }
       for(i=0; i<M; i=i+1)
       {    if(strcmp(descripcion[i], auxdes)==0)
            {    printf("\nCodigo:\t%i", codigo[i]);
                 printf("\nCantidad:\t%i", cantidad[i]);
                 printf("\tPrecio Unitario:\t%8.2f", pu[i]);
                 printf("\tMonto:\t%8.2f", monto[i]);
                 break;
            }
       }
       if(strcmp(descripcion[i], auxdes)!=0)
       {    printf("\nArticulo inexistente!!!");
       }
   }
   break;
```

LISTADOS: Bajo distintas condiciones.

```
case 'D':
   if(control!=2) /*Antes de hacer cada caso controlamos si los
                    casos A y B fueron realizados*/
   {   printf("Debemos hacer primero las opciones A y B del
                menu\n");
   }
```

```
                else
                {   printf("\nListado Artículos existentes");
                    printf("\nCod  Descripción        Cant  Precio  Total \n");
                    for(i=0; i<M; i=i+1)
                    {    if(cantidad[i]>0)
                         {    printf("\n%3i", codigo[i]);
                              printf("%20s", descripcion[i]);
                              printf("%8i", cantidad[i]);
                              printf("%8.2f", pu[i]);
                              printf("%8.2f", monto[i]);
                         }
                    }
                    printf("\nListado Artículos faltantes");
                    printf("\nCod  Descripción        Cant  Precio  Total \n");
                    for(i=0; i<M; i=i+1)
                    {    if(cantidad[i]<=0)
                         {    printf("\n%3i", codigo[i]);
                              printf("%20s", descripcion[i]);
                              printf("%8i", cantidad[i]);
                              printf("%8.2f", pu[i]);
                              printf("%8.2f", monto[i]);
                         }
                    }
                }
                break;

            //SALIR: Opcion obligatoria para salir del programa.

            case 'X':
                printf("Este es el fin del Programa ...");
                getch();
                exit(0);
            /*Esta es la opcion de falso de la estructura SWITCH(), todas las
              demás para que se cumplan tienen que ser verdaderas*/
            default:
                printf("OPCION INCORRECTA !!!");
        } //Fin del switch(opcion).

        /*Con este procedimiento aparece el mensaje para todas las
          opciones, menos la "A" de pulsar una tecla para continuar.*/

        if(opcion!='A')
        {   printf("\n"); //Realiza un salto de linea.
            system("pause"); //El proceso ejecutado por esta función hace
                                una pausa dentro del programa.
        }
    } //Fin del while(opcion!='Z').
    return (0);
} //Fin del main().
```

EJERCITACIÓN SOBRE ALGORITMOS

1. Ingresar 3 números y mostrarlos.
2. Escribir el algoritmo necesario para calcular la suma de dos números. Mostrar el resultado.
3. Ingresar 2 números y mostrar al mayor de los dos.
4. Se leen tres números, A, B, y C. Se pide escribir el mayor de ellos.
5. Se deben ingresar 3 números enteros, sumarlos y controlar si el resultado es mayor a 100. Mostrar mensaje por si es mayor o si es menor*/
6. Pedir al operador que ingrese 5 números y mostrar por pantalla el mayor.
7. Pedir al operador que ingrese un número entero positivo mayor a 10. Determinar si el par o impar (sin usar división) y mostrar por pantalla el mensaje Número Par o Impar.
8. Pedir al operador que ingrese 20 números enteros mayores a 15, mostrar por pantalla la suma, el promedio, la cantidad de números pares y la cantidad de números impares ingresados.
9. Ingresar una serie de números enteros debiéndose averiguar la cantidad de enteros positivos y la cantidad de negativos. Realizar la sumatoria, si el resultado es positivo, mostrar la cantidad de números positivos y si es negativo, mostrar el resultado de la suma.
10. Ingresar 15 números, averiguar al mayor y al menor, mostrar a todos los ingresados en forma de lista y al mayor y al menor.
11. Un profesor de matemática de un establecimiento educativo registra de cada alumno Nº de legajo, nombre y promedio. Según el promedio desea saber cuantos alumnos aprobaron (promedio mayor o igual a 7), cuantos rinden en diciembre (promedio menor a 7 y mayor o igual a 4) y cuantos rinden examen en marzo (promedio menor a 4). Además desea conocer el Nº de legajo y nombre de los alumnos con mejor promedio.
12. Crear un programa con un menú de opciones. La opción 1 permitir cargar en los legajos, nombres de 40 alumnos de un curso y las notas de programación de cada uno de ellos. La opción 2 mostrar la mayor nota y a que alumnos le correspondió esta nota. La opción 3 mostrar el listado de los datos ingresados. La opción 4 permitir salir del programa.
13. Hacer un programa que mediante un menú de opciones nos permita realizar las siguientes acciones:
1. Ingresar los nombres y las notas de 50 alumnos, debiéndose controlar que la nota sea positiva y menor o igual a 10.
2. Averiguar la nota mayor y la menor.
3. Hacer dos listados, el primero con todos los alumnos que tengan la mayor nota, y el segundo con todos los alumnos que tengan la menor nota.
4. Salir del programa.
14. Hacer un programa que por medio de un menú de opciones nos permita realizar las siguientes acciones:
a. Ingresar el nombre y el precio unitario de 100 artículos. El precio unitario debe ser mayor que 0 (cero).
b. Calcular la ganancia del 25 % para cada artículo y mostrar los datos para de cada uno.
c. Indicar cuantos artículos tienen la menor ganancia.
d. Hacer un listado ordenado por el nombre en forma ascendente de todos los artículos que superen el promedio de la ganancia. Debe aparecer el orden de ingreso.
e. Salir del programa.

CUESTIONARIO GENERAL TEÓRICO

1. ¿A que se llama IDE?
2. ¿A que se llama lenguaje de programación?
3. ¿A que se llama pseudocódigo?
4. ¿Cuál es el lenguaje que se utiliza internamente el hardware y cuales son los símbolos que lo componen?
5. ¿Qué variantes del lenguaje C de programación podemos encontrar?
6. ¿De que elementos deben estar compuestos los programas para que funcionen?
7. ¿De que manera se pueden clasificar los lenguajes de programación y que tienen en cuenta los siguientes ítems de la clasificación: Por su nivel de abstracción, por su forma de ejecutarse y por su propósito.
8. ¿Para que se utilizan las funciones dentro de un programa escrito con el lenguaje C de programación?
9. ¿Para que se utilizan las llaves, los corchetes y los paréntesis dentro de un programa?
10. ¿Para que se utilizan las secuencias de escape, cuales existen y como se utilizan dentro de un programa?
11. ¿Para que se utilizan los modificadores de formato, cuales existen y como se utilizan dentro de un programa?
12. ¿Qué es un programa?
13. ¿Qué extensión tienen los archivos creados con el lenguaje C de programación?
14. ¿Qué extensiones de archivos nos indican que programa fue escrito con el lenguaje C de programación?
15. ¿Qué funciones cumple un compilador?
16. ¿Qué funciones cumple un editor de textos?
17. ¿Qué funciones cumple un linkeador?
18. ¿Qué son las constantes y como se las utilizan dentro de un programa?
19. ¿Qué son las librerías, de que tipo existen y cuales utilizamos en el lenguaje C? Definir a las funciones que están incluidas en cada una de ellas.
20. ¿Qué son las variables y de que tipo existen?
21. ¿Qué son los algoritmos y cuales son las características que deben cumplir?
22. ¿Qué son los arreglos y de que tipo existen?
23. ¿Qué son los operandos y que los operadores?
24. ¿Qué tipos de comentarios podemos incorporar dentro de un programa?
25. ¿Qué tipos de datos se pueden utilizar para realizar los programas y de que manera se declaran y se inicializan?
26. ¿Qué tipos de estructuras de programación existen? Indicarlas en pseudocódigo y en el lenguaje C.
27. ¿Quiénes fueron los creadores del lenguaje C de programación y que sistemas operativos fueron creados con la utilización de este lenguaje?
28. Indicar que tipos de operadores existen, de que símbolos están compuestos y funciones cumplen cada de ellos.
29. ¿A que se llama código fuente?
30. ¿Cuáles son los pasos que se deben cumplir para crear un programa?