



# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## FACULTAD DE INGENIERÍA

ESTRUCTURA DE DATOS Y ALGORITMOS I (1227)

*Profesor: M.I. Marco Antonio Martínez Quintana*

*Semestre 2021-2*



### **Actividad Asíncrona #1 Miércoles 24 de Febrero**

**Nombre del alumno:** Cadena Luna Iván Adrián

**Grupo:** 15

**Fecha:** (10/03/2021)

- Realizar un pequeño repaso de lo que aprendieron en la materia Fundamentos de Programación y lo van a plasmar en la redacción de un documento de al menos 5 cuartillas.

La introducción del curso se basó en fundamentos de **algoritmos**. Un **algoritmo** es un método para resolver problemas mediante una serie de pasos, precisos, definidos y finitos. Es una serie de operaciones detalladas que se pueden formular de muchas formas siempre cuidando no generar ambigüedad.

Existen dos tipos de algoritmos: cualitativo y cuantitativo:

Cualitativo	Cuantitativo
Se describen los pasos utilizando palabras.	Se describen los pasos utilizando palabras.
<b>Encender el auto</b>	<b>Obtener promedio de 10+9+8</b>
1. Abrir auto	1. suma=10+9+8
2. Meterme al auto	2. promedio=suma/3
3. Introducir llave	3. Fin
4. Girar llave	
5. Fin	

Un algoritmo debe contar con ciertas características para cumplir su propósito:




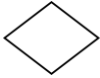
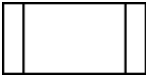



- **Preciso:** Debe indicar el orden estricto en cada paso y no debe presentar ambigüedad alguna; además, las operaciones a llevar deben ser especificadas de manera rigurosa.
- **Definido:** Se refiere a que cada vez que se siga, el algoritmo debe obtener el mismo resultado cada una de esas veces.
- **Finito:** Debe terminar después de un número finito (limitado) de pasos.
- **Correcto:** No debe presentar resultados erróneos.
- **Entrada:** Un algoritmo tendrá cero o más **entradas**. Una **entrada** es una cantidad dada antes de que el algoritmo comience, o dinámicamente mientras el algoritmo corre.
- **Salida:** Un algoritmo tendrá una o más **salidas**. Una **salida** es una cantidad que tiene una relación específica con las entradas. Los datos de salida serán los resultados de efectuar las instrucciones.

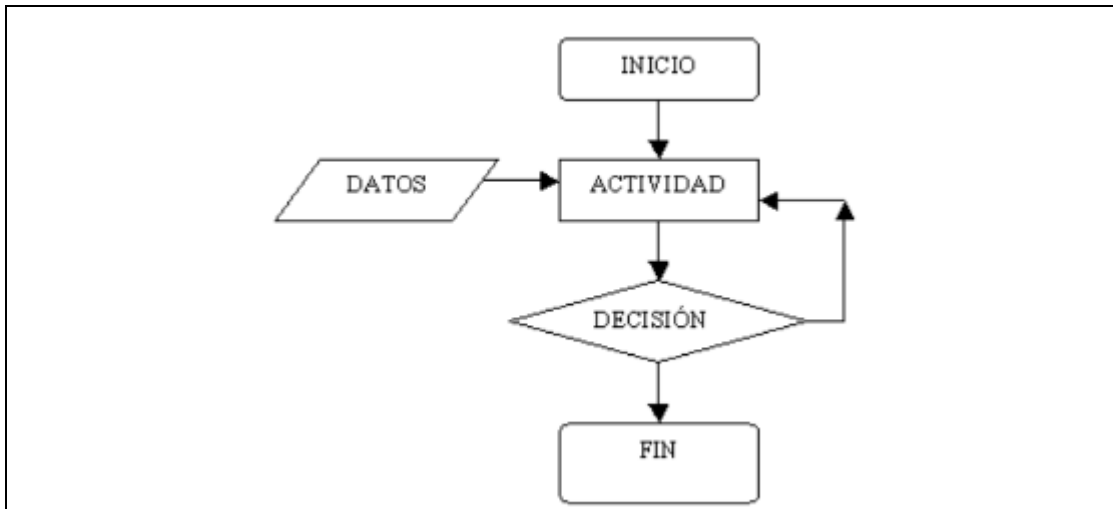
- **Eficacia:** Un aspecto muy importante, se espera que un algoritmo sea eficaz; que todas las operaciones a realizar en un algoritmo deben ser suficientemente básicas como para que en un principio puedan ser hechas por el usuario manualmente.
- **Solución:** El algoritmo debe llegar a un resultado específico.

Un algoritmo puede ser expresado o representado de las siguientes formas:

- **Gráficamente/Diagrama de flujo:** Es la representación gráfica de las operaciones que realiza un algoritmo. También se puede decir que es la representación detallada en forma gráfica de cómo deben realizarse los pasos para producir resultados. Esta representación gráfica se presenta mediante un conjunto de símbolos que se relacionan entre si a través de líneas que indican el orden en que se deben ejecutar cada uno de los procesos.

Los símbolos básicos utilizados en los diagramas de flujo son:

Símbolo	Descripción
	Indica el inicio y el final del diagrama de flujo.
	Indica la entrada y salida de datos.
	Símbolo de proceso. Indica la asignación de un valor y/o la ejecución de una operación aritmética.
	Símbolo de decisión. Indica la realización de una comparación de valores.
	Se utiliza para representar los llamados a los subprogramas o subprocesos.
	Conector dentro de página. Representa la continuidad del diagrama dentro de la misma página.
	Conector fuera de página. Representa la continuidad del diagrama en otra página.
	Líneas de flujo o dirección. Indican la secuencia en que se realizan las operaciones.



- **No gráficamente/Pseudocódigo:** Un pseudocódigo es una mezcla de lenguaje de programación y un idioma como el español, que se emplea dentro de la programación estructurada para especificar el diseño de un programa. También se puede definir como un lenguaje de especificaciones de algoritmos, utilizando palabras que indican el proceso a realizar.

Al realizar un pseudocódigo, las palabras empleadas más comunes son: *Inicio, fin, leer, escribir, si, sino, fin si, para, fin para, mientras que, fin mientras que, repita, hasta, regresar.*

El diseño de un algoritmo consta de varias etapas:

- Definición del problema.
- Análisis del problema.
- Selección de la mejor alternativa.
- Elaboración de diagramas y/o pseudocódigo.
- Prueba de escritorio.

En **entidades primitivas para el diseño de algoritmos** aprendimos que para comunicarnos con la computadora vamos a utilizar:

- **Tipos de datos:**
  - ***Simples***
    - *Númericos*

- Entero: Son números completos que no tienen componentes fraccionarios o decimales y pueden ser positivos o negativos.
- Real: Son números que siempre tienen un punto decimal y pueden ser positivos o negativos.

- **Alfanuméricos**

Permite representar valores identificables de forma descriptiva, como lo pueden ser nombres de personas, direcciones, características, etc.

- Caracteres alfabéticos (a...z y A...Z).
- Caracteres numéricos (0, 1, 2, 3, 4, 5, 6, 7, 8, 9)
- Caracteres especiales (+ - # \$ % & \* ( ) / < > : ; ' " )

- **Lógicos**

Son aquellos que sólo pueden tener dos valores (cierto o falso) ya que representan el resultado de una comparación entre datos (numéricos o alfanuméricos).

- **Operadores y operandos:**

Se les conoce operandos a los elementos que se relacionan (variables o constantes) y operadores a los elementos relacionadores.

- **Aritméticos**

Permiten la realización de operaciones matemáticas con los valores (variables y constantes), siguiendo la prioridad de los operadores aritméticos.

- + Suma
- - Resta
- \* Multiplicación
- / División
- Mod Módulo (residuo de la división entera)

- **Relacionales**

Se utilizan para establecer una relación entre dos valores. Comparan estos valores entre si y esta comparación produce un resultado de certeza o falsedad (verdadero o falso).

- > Mayor que
- < Menor que
- >= Mayor o igual que
- <= Menor o igual que
- <> Diferente
- == Igual

- **Lógicos**

Estos operadores se utilizan para establecer relaciones entre valores lógicos. Estos valores pueden ser resultado de una expresión relacional.

- and/y
- or/o
- not/no (negación)

- **Expresiones:**

Son conjuntos de constantes, variables, operadores y paréntesis. Normalmente se utilizan para definir operaciones matemáticas relacionales y/o lógicas. Consta de operandos y operadores.

- **Aritméticos:** Son análogas a las fórmulas matemáticas. Las variables y constantes son numéricas y las operaciones son las aritméticas.
- **Relacionales:** Permiten realizar comparaciones de valores de tipo numérico o carácter. Sirven para expresar las condiciones en algoritmos o programas.
- **Lógicos:** Operador lógico. No están permitidos dos operadores sucesivos y la expresión puede consistir en un solo identificador usado como constante o como variable.

- **Identificadores como localidades de memoria:**

Los identificadores representan los datos de un programa (constantes, variables, tipos de datos). Un identificador es una secuencia de caracteres que sirve para identificar una posición en la memoria de la computadora, que nos permite acceder a su contenido.

- **Variables:** Es un espacio en la memoria de la computadora que permite almacenar temporalmente un dato durante la ejecución de

un proceso, su contenido puede cambiar durante la ejecución del programa.

- Numéricas: Aquellas que solo permiten almacenar valores numéricos.
- Alfanuméricas: Aquellas que puede almacenar uno o más caracteres (dígitos, números, y símbolos).

Los tipos de variables más comunes son: *enteras*, *reales*, *carácter*, *cadena* y *lógicas*.

- **Constantes:** Una constante es un dato numérico o alfanumérico que no cambia durante la ejecución del programa.

Una vez vistos los términos fundamentales, entramos a la **programación en C**.

### Estructura de un programa en C:

```
/*inclusión de bibliotecas, definición de constantes y */  
/*macros, definición de prototipos de funciones*/  
  
main() {  
/*cuerpo de la función main  
/*En esta función escribiremos el código principal de nuestro  
programa*/  
}  
  
/*Definición de otras funciones*/
```

### Palabras reservadas:

Las palabras reservadas son identificadores predefinidos que tienen significados especiales y no pueden usarse como identificadores creados por el usuario en los programas: *auto const double float int short struct unsigned break continue else for long signed switch void case default enum goto register sizeof typedef volatile char do extern if return static union while*

Tipo	Significado	Tamaño en bytes
char	caracter	1
int	entero	2 – 4
short	entero corto	2
long	entero largo	4
unsigned char	caracter sin signo	1
unsigned	entero sin signo	2 – 4
unsigned char	entero corto sin signo	2
unsigned long	entero largo sin signo	4
float	coma flotante, real	4
double	coma flotante largo	8

- **Variables**

Identificador utilizado para representar un cierto tipo de información.

Cada variable es de un tipo de datos determinado.

Una variable puede almacenar diferentes valores en distintas partes del programa.

! Debe comenzar con una letra o el carácter guión bajo (\_).

! El resto puede contener letras, números o \_

*Ejemplos de variables válidas: numero, \_color, identificador\_1*

Una declaración asocia un tipo de datos determinado a una o más variables.

El formato de una declaración es:

**tipo\_de\_datos** var1, var2, ..., varN;

*Ejemplos:*

- **int** a, b, c;
- **float** numero\_1, numero\_2;
- **char** letra;
- **unsigned long** entero;



- **Expresiones y sentencias**

Una expresión representa una unidad de datos simple, tal como un número o carácter.

- **Operadores:**

- + Suma
    - - Resta
    - \* Multiplicación
    - / División
    - Mod Módulo (residuo de la división entera)

- **Punto y coma:**

Indica que la línea de instrucción termina por ejemplo al declarar variables.

*Ejemplo:*

```
int a,s,d,f; //indica terminación de variables
clrscr; //función limpiar pantalla
printf("hola"); //imprimir hola
```

- **Funciones**

- ***printf***

Permite imprimir información por la salida estándar (monitor).

**Formato:** printf(formato acompañado o no de cadenas, argumentos);

*Ejemplo:*

```
printf("Hola mundo\n");
printf("El numero 28 es %d\n", 28);
```

- ***scanf***

Permite leer datos del usuario: la función devuelve el número de datos que se han leído correctamente.

**Formato:** scanf(formato, argumentos);

El símbolo & indica que el valor leído se almacenará en la dirección de memoria de la variable correspondiente

*Ejemplo:*

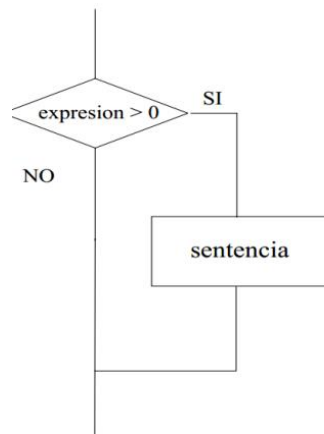
```
scanf("%f", &numero);  
scanf("%c", &letra);  
scanf("%f %d %c", &real, &entero, &letra);  
scanf("%ld", &entero_largo);
```

- **Secuencias de control**

Una secuencia de control es aquella por la cual los enunciados en un programa son ejecutados uno después del otro, en el orden en que aparecen escritos.

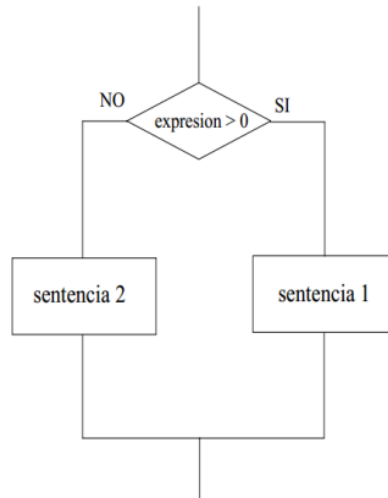
- **if**

Se utiliza para elegir entre cursos alternativos de acción. Es decir, si la condición definida por la expresión1 es verdadera, entonces se ejecuta la o las sentencias siguientes. Si la condición resulta falsa, se ignora la o las sentencias siguientes, y se ejecuta el siguiente fragmento de código en su orden.



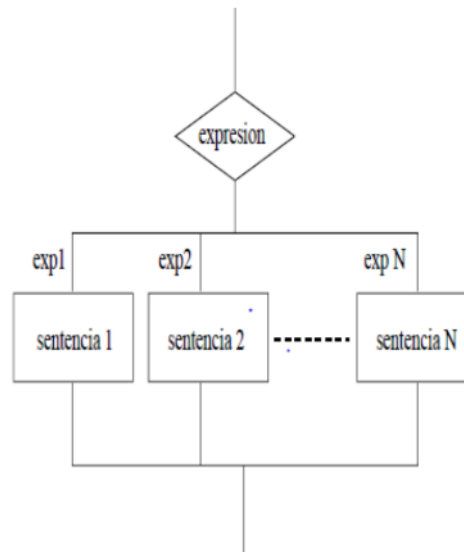
- **if-else**

Se utiliza de igual manera para elegir entre cursos alternativos de acción dependiendo de la condición definida por la expresión1; pero en este caso, si la condición resulta falsa, se pasa a ejecutar el siguiente fragmento de código indicado por else.



- **switch**

En los casos que se utiliza esta secuencia de control, las variables o expresiones se probarán por separado contra cada uno de los valores constantes enteros que puede asumir, y se tomarán diferentes acciones.

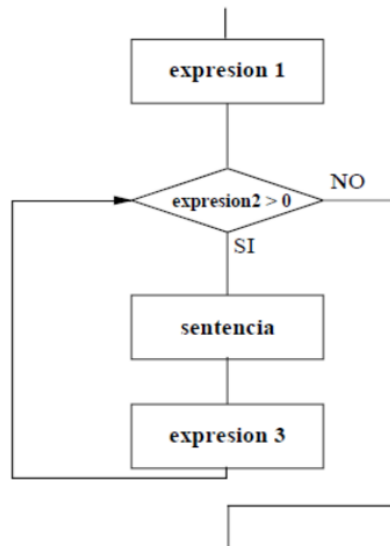


- **for/contador**

La repetición controlada por contador se denomina a veces repetición definida, porque con anticipación **se sabe** con exactitud cuántas veces se ejecutará el ciclo.

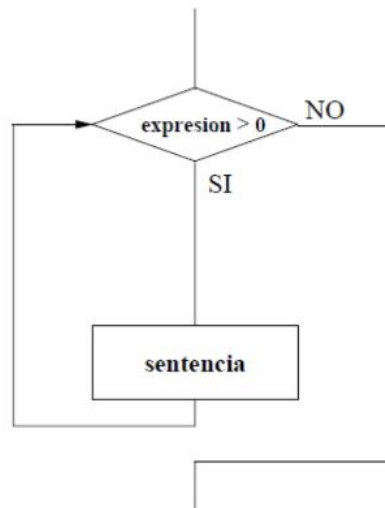
Se utiliza una variable de control para contar el número de repeticiones y es incrementada cada vez que se ejecuta el grupo de instrucciones; hasta que se ha ejecutado el número correcto de

repeticiones, se termina el ciclo y la computadora continúa ejecutando el enunciado siguiente al de la estructura de repetición.



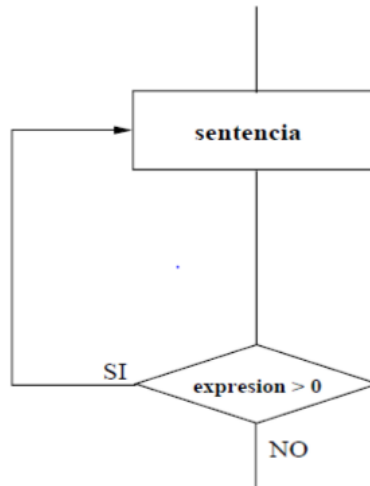
- **while/centinela**

La repetición controlada por centinela a veces se denomina repetición indefinida, porque **no se sabe** con anticipación cuantas veces el ciclo se ejecutará.



- **do-while**

En este caso, las sentencias se ejecutarán mientras el valor de la expresión sea verdadera, y al menos una vez después se evalúa la expresión. Normalmente las sentencias incluyen algún elemento que altera la expresión **proporcionando el fin de las repeticiones**.



Los temas más complejos que continuaron a lo largo y hasta el término del curso, de manera resumida son los siguientes:

- **Arreglos**

Un arreglo es una colección ordenada de elementos de un mismo tipo. Ordenada significa que cada elemento tiene una ubicación determinada dentro del arreglo y debemos conocerla para poder acceder a él.

Se definen de la siguiente manera:

```
<tipo>nombre variable [longitud];
```

Se asignan de la siguiente manera:

```
nombre variable[índice] = expresión del tipo <tipo >
```

Para tener acceso al contenido de un arreglo:

```
nombre variable[índice]
```

- **Apuntadores**

Los apuntadores también conocidos como **punteros** son variables que guardan direcciones de memoria. Proporcionan mucha utilidad al programador para acceder y manipular datos de maneras que no es posible en otros lenguajes, y también son útiles para pasarle parámetros a las funciones de tal modo que les permiten modificar y regresar valores a la rutina que las llama.

Los punteros, al igual que una variable común, pertenecen a un tipo, se dice que un puntero apunta a ese tipo al que pertenece. *Ejemplos:*

```
int* pint; //Declara un puntero a entero
char* pchar; //Puntero a char
fecha* pfecha; //Puntero a objeto de clase fecha
```

- **Estructuras**

Una estructura es un grupo de variables relacionadas de manera lógica, las cuales pueden ser de diferentes tipos y declaradas en una sola unidad, donde la unidad es la estructura. Se forma utilizando la palabra reservada **struct**, seguida por un campo etiqueta opcional, y luego una lista de miembros dentro de la estructura. La etiqueta opcional se utiliza para crear otras variables del tipo particular de la estructura.

```
struct campo_etiqueta{
    tipo_miembro miembro_1;
    tipo_miembro miembro_2;
    .
    .
    .
    tipo_miembro miembro_n;
};
```

- **Archivos**

Para el manejo de archivos tenemos que considerar primero la parte y el modo de apertura (lectura, escritura, creación, binario, etc.), el modo de trabajo y luego cerrar el archivo. .

```
FILE *archivo
/*Define una variable de tipo de archivo
la información de el archivo se encuentra
almacenada en un punto hacia él mismo. */

FILE *archivo
FILE *fopen (char*nombre, char*modo);
//archivo es un apuntador de archivo
```

- Argumentos de **fopen**

El primero es una cadena de caracteres con el nombre del archivo: *“datos.txt”*

El segundo es el tipo de acceso:

- **w** Escritura
- **r** Lectura
- **a** Agregación

➤ **BIBLIOGRAFÍA CONSULTADA**

Olivares, L. (2008). Manual de Programación en Lenguaje C++. Recuperado el 7 de marzo de 2021, de <https://paginas.matem.unam.mx/pderbf/images/mprogintc++.pdf>