75.40 Algortimos y Programación I Curso 4
Introducción a la Programación

Dr. Mariano Méndez - Martin Dardis[1]

[1]Facultad De Ingeniería. Universidad de Buenos Aires

19 de agosto de 2019

1. Agradecimientos

Es imposible poder dejar de agradecer a todos los Docentes Auxiliares, AdHorem, Colaboradores y alumnos que hicieron posible la existencia de este curso de Algoritmos y Programación I de la Facultad de Ingeniería de la Universidad de Buenos Aires. Este apunte es una guía de referencia de *mis* apuntes del curso y los utilizo como una ayuda memoria para recordar y recopilar lo que normalmente enseñamos en el mismo. El apunte no podría existir sin la colaboración de muchas personas a lo largo de los últimos 3 años. A continuación intentaré nombrar a todos aquellos que han colaborado en la redacción del mismo: Manuel Camejo, Melina V. Díaz, Fernando Gastón Caceres, Anibal Lovaglio, Juan Pablo Capurro, Carlos Talavera, Mauro Toscano, Martin Dardis, Gonzalo Diaz Mulligan, Delfina Garcia Villamor, Javier Mermet, Lorenzo Gimenez, Gonzalo Martinez Sastre, Gabriel Pucci, Damian Ramonas, Gisela Ramos, Nicolas Riedel, Tomas Rodriguez Dala, Miguel Toscano (alias mike amigorena), Nicolas Sanfilipo, Lucas Ballester, Brian Stanley, Agustin Teston, Andoni Zabala, Pablo Bucci, Pablo Zuleta, Yanina Faretta, Adriano Deganis, Lucas Bonfil, Mauricio González, Nicolás Garofalo, Matías Caceres, Benjamín Ortiz, Ramiro Chogri, Constanza Frutos Ramos, Matías Giménez, Jeremías Longo, Elvis Castro, Rodrigo Souto, Alejandro Daneri, Alejandro García, Iván Loyarte ... Gracias a todos !

2. Introducción

Esta sección se enfocará en resolver algunos temas conceptuales sobre la construcción de algoritmos y programas. Como es de esperar en primer lugar hay que tener muy en claro con qué conceptos se estará trabajando.

2.1. ¿Qué es un Algoritmo?

La etimología de la palabra algoritmo deriva del árabe y procede del sobrenombre del sabio y matemático árabe *Al-Khwarizimi* Mohamed ben Musa (780-840). En el medioevo el vocablo se contrajo con el sufijo griego "aritmo" que significa número. Pero ¿qué significa la palabra? Según Joyanes Aguilar: "Un algoritmo es un método para resolver un problema mediante una serie de pasos precisos, definidos y finitos" [1]. La RAE (Real Academia Española) un algoritmo es un "conjunto ordenado y finito de operaciones que permite hallar la solución de un problema" [7]. La mayoría de las definiciones de algoritmo tienen conceptos comunes entre ellas:

- **Método de resolución de problemas**
- Conjunto de Acciones
  - **Finito**
  - **Ordenado**
  - **Preciso**

En lo cotidiano usamos algoritmos muchas veces en el día. Por ejemplo, al seguir una receta de cocina. A continuación se muestran otros ejemplos ejemplos:

Cargar Un Celular:

1. Buscar el cargador
2. Agarrar el Cargador

3. Agarrar el celular
4. Poner El Enchufe en el Celular
5. Esperar a que se ponga a cargar
6. Ponerlo en un lugar seguro

Colocar la mesa para la comida:

1. Limpiar la mesa
2. Sacar los individuales
3. Poner los individuales
4. Sacar las servilletas
5. Poner las servilletas
6. Sacar los vasos
7. Poner los vasos
8. Sacar los platos
9. Servir los platos
10. Ponerlos en la mesa
11. Sentarse
12. Comer

Entonces, un algoritmo resuelve un problema mediante su ejecución. ¿Existe un método de resolución de problemas? Por supuesto que la respuesta es sí. Existen varios métodos de resolución de problemas uno de los más conocidos es el siguiente:

1. **Entender el problema**: ¿Cuál es la incógnita? ¿Cuáles son los datos?
2. **Crear un plan**: ¿Se ha encontrado con un problema semejante? ¿Conoce un problema relacionado con este? ¿El problema se podría enunciar de otra forma?
3. **Llevar a cabo el plan**: ¿Los pasos dados son correctos?
4. **Examinar la solución obtenida**: ¿El resultado puede ser verificado? ¿El razonamiento se puede verificar?

Problema

> **Definición**: Cuestión que se plantea para hallar un dato desconocido a partir de otros datos conocidos, o para determinar el método que hay que seguir para obtener un resultado dado.

Se puede decir que siempre que hay un salto entre donde estás y donde querés llegar y no sabés cómo encontrar el camino para salvar este salto, tenés un problema.

Resolución de Problemas con una Computadora

> "En la década de 1930, antes de la llegada de las computadoras, los matemáticos trabajaron con gran celo para formalizar y estudiar el concepto de algoritmo, que entonces se definía de manera informal como un conjunto claramente especificado de instrucciones sencillas a seguir para resolver un problema o calcular una función" [2]

Resolver un problema con una computadora va más allá de simplemente sentarse delante de ella y comenzar a teclear una solución. Así como existe un método general para la resolución de problemas, también existe un método para la resolución de problemas con una computadora. El método consta de los siguientes pasos:

1. **Especificación de los requerimientos del problema**: Se buscan y se describen cuales son los requerimientos del problema. Eta descripción puede ser un dibujo, un párrafo en lenguaje natural, cualquier herramienta que me permita comunicar y describir **QUÉ** hay que hacer.

2. **Análisis del problema**: El problema se analiza teniendo presente la especificación de los requerimientos, restricciones y datos dados del problema a solucionar.

3. **Diseño del algoritmo para la solución**: Una vez analizado el problema, se diseña una o varias soluciones y se elegirá aquella que por su naturaleza se considere la mejor que conducirá a un algoritmo que lo resuelva. Una técnica de diseño en la resolución de problemas consiste en la identificación de las partes (subproblemas) que lo componen y la manera en que se relacionan. Cada uno de estos *subproblemas* debe tener un objetivo específico, es decir, debe resolver una parte del problema original. Además se debe **Especificar el algoritmo**, para solucionar cada subproblema se plantea un algoritmo, éste debe ser especificado de alguna forma. Esta etapa busca obtener la secuencia de pasos a seguir para resolver el problema. La elección del algoritmo adecuado es fundamental para garantizar la eficiencia de la solución.

4. **Implementación de un programa**: La solución que plantea el algoritmo se transcribe a un lenguaje de programación.

5. **Ejecución, verificación y depuración**: El programa se ejecuta, se comprueba rigurosamente y se eliminan todos los errores (denominados "bugs", en inglés) que puedan aparecer.

6. **Mantenimiento**: El programa se actualiza y modifica, cada vez que sea necesario, de modo que se cumplan todas las necesidades de cambio de sus usuarios.

La resolución de un problema es un **proceso** que tiene una *metodología determinada* que hay que seguir. Este proceso puede ser estudiado desde dos puntos de vistas según la teoría de sistemas:

1. **Caja Negra**: Aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno.

2. **Caja Blanca**: Aquel elemento que es estudiado desde el punto de vista de las interacciones entre los componentes internos que lo integran.

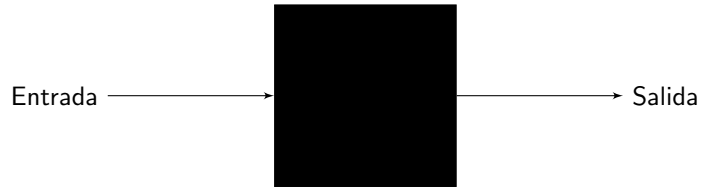


Figura 1: Esquema de Caja Negra

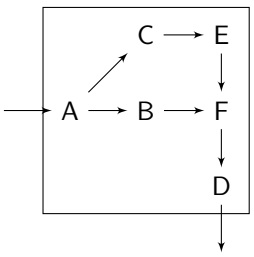


Figura 2: Esquema de Blanca

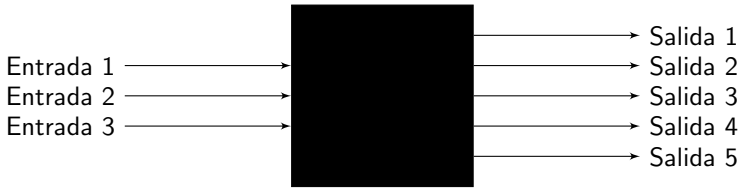


Figura 3: Esquema de Caja Negra Múltiples Entradas/Salidas

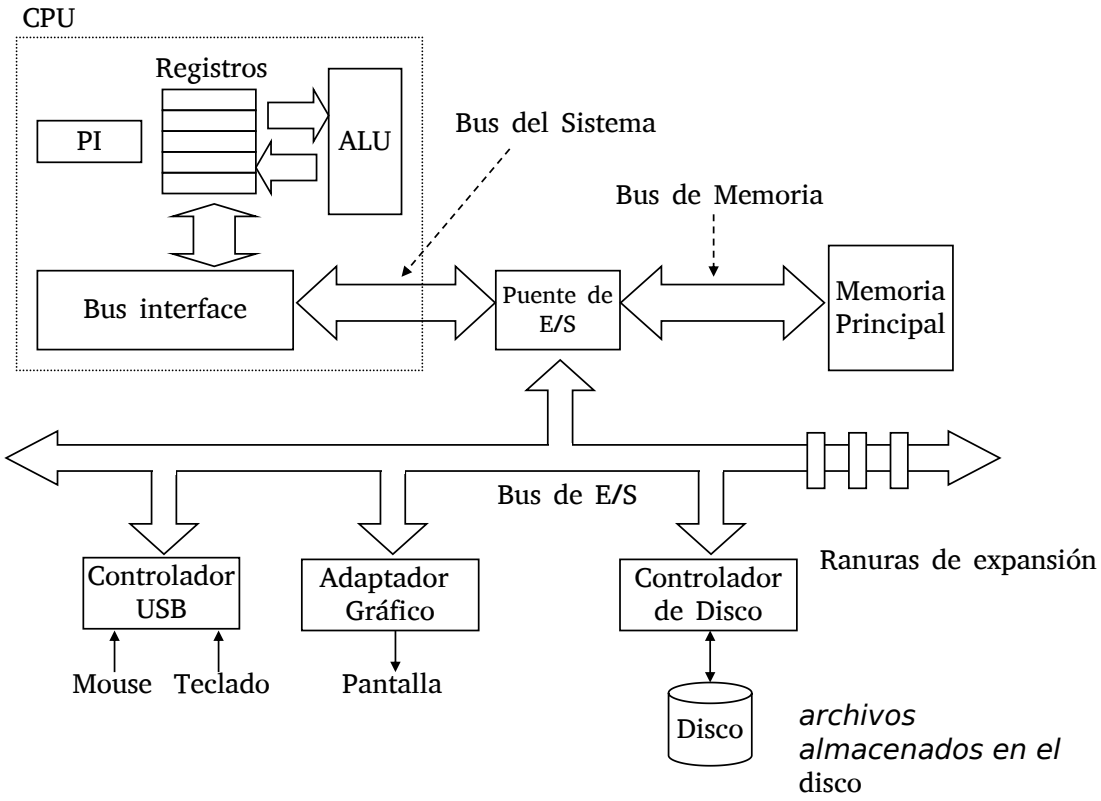


Figura 4: Diagrama Básico de la Estructura de una Computadora [4]

2.2. ¿Qué es una Computadora?

Según la RAE una computadora es "Máquina electrónica que, mediante determinados programas, permite almacenar y tratar información, y resolver problemas de diversa índole". Si bien esta definición es un poco pobre podemos considerarla como válida. Esta máquina se compone a groso modo de cuatro partes fundamentales [8]:

- **La Unidad Central de Procesamiento o CPU**: controla el funcionamiento de la computadora y lleva a cabo sus funciones de procesamiento de datos (Procesador).
- **La Memoria Principal**: almacena datos.
- **Los Periféricos de entrada y salida de datos**: transfieren datos entre la computadora y el entorno externo.
- **Las Inter-conexiones o Buses**: mecanismos que proporcionan la comunicación entre la CPU, la memoria ppal. y la E/S.

La unidad central de procesamiento es un dispositivo electrónico que determina qué, cuándo y con cuáles datos una instrucción o acción debe ser ejecutada. Las partes de la unidad central de procesamiento son [8]:

1. **Unidad de control**: controla el funcionamiento de la CPU y, por tanto lo, del computador.
2. **Unidad Aritmético Lógica (ALU)**: lleva a cabo las funciones de procesamiento de datos del computador.
3. **Registros**: proporcionan almacenamiento interno a la CPU.
4. L**as Interconexiones**: son mecanismos que proporcionan comunicación entre la unidad de control, la ALU y los registros.

3. ¿Cómo se Cuantifica la Información?

En los inicios de la era de la computación cada fabricante de computadoras construía su propio modelo, como un sastre construye un traje: a medida. Según fue desarrollándose las diferencias entre las computadoras de distintos fabricantes eran tan grandes que se tubo que adoptar un estándar para que todos los diseños de computadoras manejaran la información de la misma forma. para ello el trabajo de Frederick Brooks en [3] fue fundamental. En términos de información las computadoras únicamente manejan pulsos eléctricos, por ende podría pensarse que la ausencia de electricidad se representa con un 0. Por otro lado la presencia de electricidad se representa con un 1.

Bit a este par 0/1 se lo denomina bit ó **dígito binario o BInary digiT**. El **bit** representa la unidad de información más pequeña que una computadora maneja. Un bit solo puede representar dos valores o estados (encendido/apagado, está/no está, etc.).

Byte Brooks propuso en su artículo que una computadora accediera a la información en **"pedazos"** de **8 bits**, por motivos que no vienen al caso. Estos pedazos fueron denominados **bytes** (botar que byte es my parecido foneticamente a *bite* que en ingles significa pegar un mordizco), con lo cual formalmente un Byte equivale a 8 bits.

Unidad	Cantidad de Bytes	Equivalencia
1 byte	2^0= 1 byte	8 bits
1 KiloByte	2^{10} = 1024 bytes	1024 bytes
1 MegaByte	2^{20} = 1 048 576 bytes	1024 KiloBytes
1 GigaByte	2^{30} = 1 073 741 824 byes	1024 MegaBytes
1 TeraByte	2^{40} = 1 099 511 627 776 bytes	1024 Gigabytes
1 PettaByte	2^{50} = 1 125 899 906 842 624 bytes	1024 TeraBytes
1 ExaByte	2^{60} = 1 152 921 504 606 846 976 bytes	1024 PettaBytes
1 ZettaByte	2^{70} = 1 180 591 620 717 411 303 424 bytes	1024 ExaBytes
1 YottaByte	2^{80} = 1 205 925 916 614 629 174 706 176 bytes	1024 ZettaBytes

4. ¿Qué es un Programa?

Un programa es la traducción de un algoritmo a un lenguaje de programación determinado capaz de ser ejecutado por una computadora.

Listing 1: Algoritmo

```
escribir "hola Mundo!"
```

Listing 2: Assembler

```
DOSSEG
.MODEL TINY
.DATA
TXT DB "Hola mundo!$"
.CODE
START:
     MOV ax, @DATA
     MOV ds, ax

     MOV ah, 09h
     MOV dx, OFFSET TXT
     INT 21h

     MOV AX, 4C00h
     INT 21h
END START
```

Listing 3: C

```
#include<stdio.h>

int main()
{
   printf("\nHola mundo!");
   return 0;
}
```

Listing 4: Pascal

```
program byeworld;
begin
   writeln('Hola mundo!');
end.
```

Listing 5: Python

```
print "Hola mundo!"
```

5. ¿Qué es un Lenguaje de Programación?

Para poder entenderse con una máquina electrónica es necesario hablar su idioma, en este caso habría que mandarle impulsos eléctricos. La forma más sencilla que tiene una máquina de entender este tipo de señales sería

mediante encendido y apagado (*on/off*). De hecho el alfabeto de una computadora tiene sólo dos dígitos, que se representan como 0 y 1. Decirle a una computadora que sume dos números se traduciría en:

$$10000110001010000$$

La secuencia de bits equivaldría a la acción sumar A y B. Esta es la forma con la que los primeros programadores se comunicaban con las computadoras, esto puede verse en las Figuras 6 y 7. Cincuenta y dos minutos es el tiempo que tardaba Manchester "Baby" en ejecutar un programa de 17 instrucciones usando un "tubo de la memoria" ideado por Freddie Williams y Tom Kilburn. El 21 de junio de 1948, la computadora con programas almacenados en su memoria programa había nacido. Es evidente que esta forma de comunicación no era nada sencilla, no era posible hacer programas que escalaran, era tediosa, las acciones imposibles de memorizar, entre otras muchas desventajas que tenía. Por ello se decidió crear un lenguaje intermedio que pudiera ser fácilmente recordado por un ser humano y que sería procesado por otro programa para transformarlo en binario. El lenguaje se llama **assembly language** y el programa que traducía a binario se lo denominó **assembler**:

$$ADDA, B$$

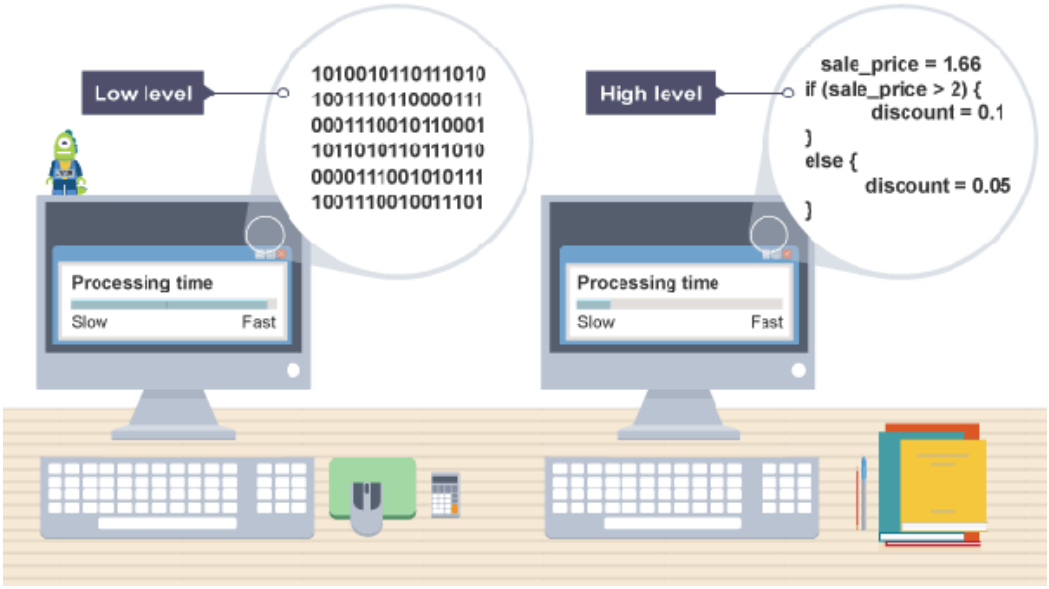


Figura 5: Lenguaje de Bajo Nivel vs Alto Nivel

Este sería el primer **lenguaje de programación**. Si bien el lenguaje aportaba una mejora y facilitaba la creación de programas todavía era tediosa y muy difícil, ya que realizar operaciones insumían un gran número de instrucciones en el lenguaje assembly. Por ello se avanzó con la idea de crear otros lenguajes en los cuales realizar ciertas acciones sea mucho menos complejo, por ejemplo, sumar dos números sea:

$$A + B$$

Este tipo de lenguaje se denomina lenguaje de programación de alto nivel. La idea es que un lenguaje de **bajo nivel** se *acerca más a los pulsos eléctricos que necesita una máquina electrónica* y un lenguaje de **alto nivel** se *acerca más a la idea de lenguaje que los seres humanos utilizamos*. Así en 1956, nacía el primer lenguaje de programación de alto nivel llamado FORTRAN I. Hacia fines del año 2012 la cantidad de lenguajes de programación existente era de aproximadamente unos **8512** y al día de hoy (23-07-2018) hay unos **8945** lenguajes y dialectos de programación. Entre los más importantes nos encontramos con: Cobol, Java, PHP, Lisp, C, C++, Python, por nombrar algunos. Cabe destacar que hasta el día de la fecha no hay una ley, teorema o motivo que indique que un lenguaje es mejor que otro, la elección y utilización de un determinado lenguaje de programación depende en gran medida del tipo de problema que se está intentando resolver y del programador.

Para tener una idea el veamos como cambia el programa hola mundo de bajo a alto nivel:

Listing 6: Binario

```
0101 0101
0100 1000 1000 1001 1110 0101
1011 1111 1100 0100 0000 0101 0100 0000 0000 0000
1011 1000 0000 0000 0000 0000 0000 0000 0000 0000
1110 1000 1100 0111 1111 1110 1111 1111 1111 1111
1011 1000 0000 0000 0000 0000 0000 0000 0000 0000
0101 1101
1100 0011
```

Listing 7: Código Máquina

```
55
48 89 e5
bf c4 05 40 00
b8 00 00 00 00
e8 c7 fe ff ff
b8 00 00 00 00
5d
c3
```

Listing 8: Assembly

```
push   rbp
mov    rbp,rsp
mov    edi,0x4005c4
mov    eax,0x0
call   400400 <printf@plt>
mov    eax,0x0
pop    rbp
ret
```

Listing 9: Lenguaje C

```c
#include<stdio.h>
int main()
{
printf("\nHola mundo!");
return 0;
}
```

Listing 10: Python

```python
print "Hola mundo!"
```

Indice Tiobe El índice programación TIOBE es un indicador de la popularidad de los lenguajes de programación (solo un indicador no es el santo grial!). El índice se actualiza una vez al mes. Las calificaciones se basan en el número de ingenieros calificados en todo el mundo, cursos y proveedores de terceros. Para ver como está seguir este link Tiobe.

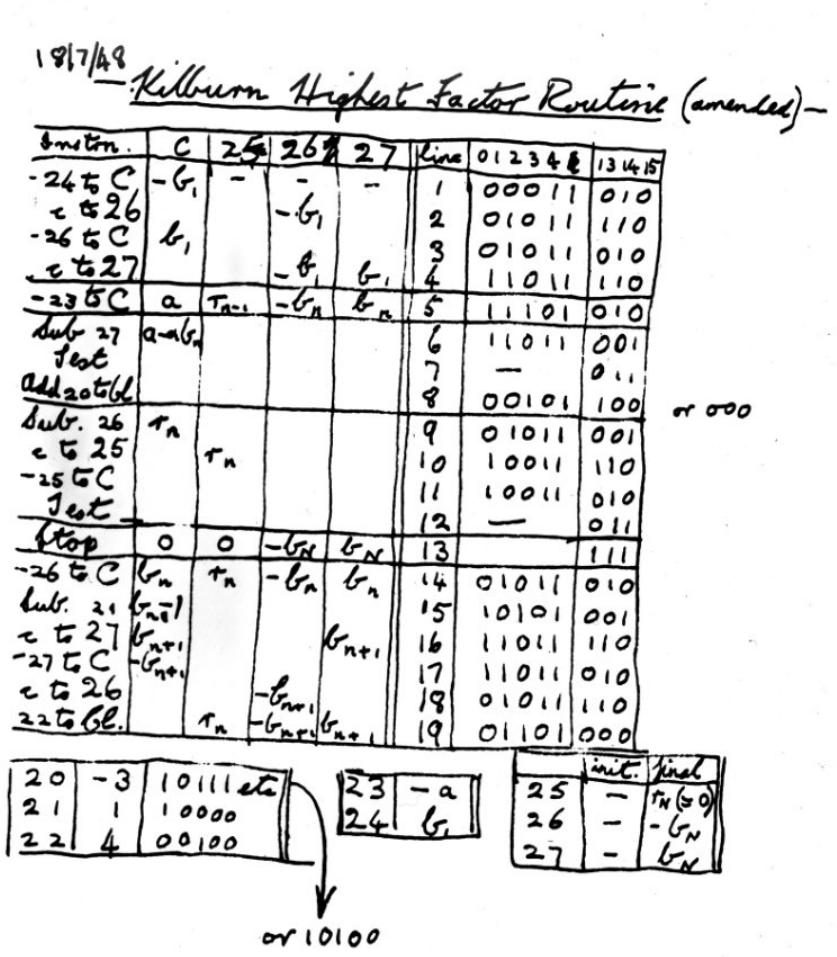


Figura 6: Programa de Baby Mancgester 1948

```
Fisrt HM1 ProgrAM
000 CI = S
001 A = A - S
010 A = - S
011 If A < 0, CI = A < 0, CI = CI + 1
100 CI = CI + S
101 A = A - S
110 S = A
111 HALT
```

Figura 7: El equivalente en Assembly del programa de Baby Mancgester 1948

6. ¿Cómo se Pasa de un Lenguaje de Alto Nivel a Binario?

Existen dos formas de hacer que una computadora puede entender un programa escrito en un lenguaje de alto nivel.

6.0.1. Compilación

El proceso de traducción entre un programa escrito en un lenguaje de alto nivel, más cerca del ser humano que del cpu, se realiza mediante un programa que traduce las acciones de alto nivel a un lenguaje de más bajo nivel, más cerca del procesador que del ser humano. Este programa se llama compilador, y hay que destacar que existe un compilador por cada lenguaje que necesite ser compilado. Por ejemplo el lenguaje de programación C posee su propio compilador, el lenguaje de programación Pascal a su vez tiene su propio compilador, y así sucesivamente.

6.0.2. Interpretación

El proceso de interpretación difiere con el de compilación, ya que en éste no se genera un programa ejecutable. El intérprete, es un programa que toma una línea del programa escrito y en el momento realiza la traducción y ejecución de la acción. Por ejemplo, Python es un lenguaje interpretado.

7. ¿Qué es programar?

Esta es una buena pregunta, muchos creen que programar es sentarse delante de una computadora y comenzar a escribir instrucciones indiscriminadamente hasta llegar a la solución del problema... Bueno justamente eso no es programar. Como dijo Martin Fowler:

> "**Any fool** can write code that a **computer can understand**. **Good programmers** *write code that* **humans can understand**"

Algo así como...

> "**Cualquier tonto** puede escribir código que una **computadora puede entender**. **Los buenos programadores** *escriben código que* **los humanos pueden entender**".

8. El Pseudo-Código

Uno de los pasos en la resolución de problemas con una computadora es la especificación de un algoritmo. Existen varias formas de especificar o describir un algoritmo. Para estar en concordancia con [1] usaremos pseudocódigo.

8.1. Palabras Reservadas

El pseudocódigo es un subconjunto del lenguaje natural, que permite describir un algoritmo sin lugar a ambigüedades. En pseudocódigo existe un conjunto de palabras que se llaman palabras reservadas que pueden ser utilizadas dentro de la descripción del algoritmo con un solo propósito. Éstas se listan a continuación:

```
si,para, mientras, sino, fin, repetir, hasta, hacer, entero, real,
caracter, algoritmo, const, var, tipo, inicio, fin, segun, caso
```

9. Variables

Desde el punto de vista de la algoritmia una variable es una entidad que puede almacenar un valor de un determinado tipo de dato, y dicho valor puede ser accedido y/o modificado a lo largo del algoritmo. Esta entidad se caracteriza por tener un identificador único. **Nota:** Vale destacar la importancia de tener claro el concepto de que una variable **NO ES UN VALOR en sí mismo** sino UN CONTENEDOR DE VALORES, Ver Figura 9.

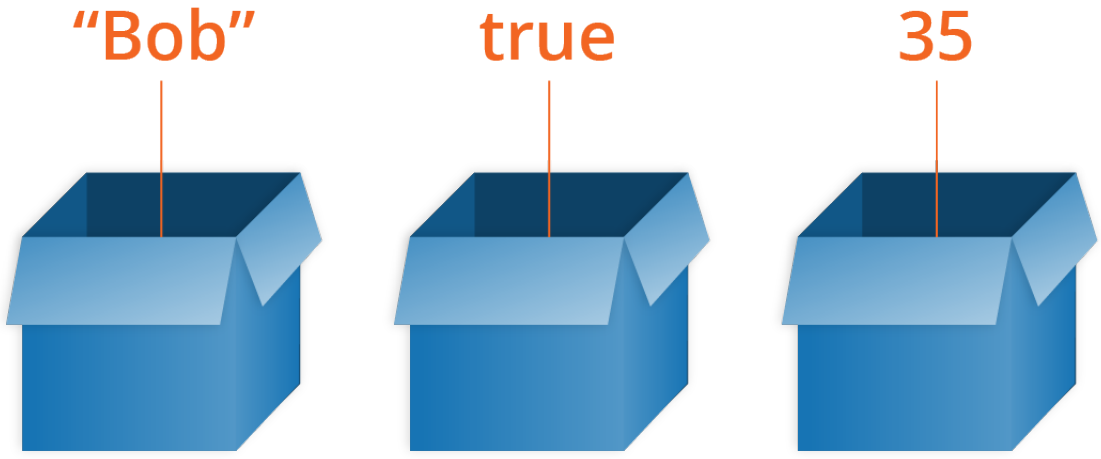


Figura 8: Un Variable. `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/Variables`

Desde el punto de vista de la programación como se ha visto en la Sección 2.2, uno de las componentes de una computadora es su memoria principal. Según [8] las memorias principales de una computadora en la actualidad se conforman por celdas semiconductoras que poseen tres propiedades:

1. Presentan dos estados estables que pueden emplearse para representar un 1 y/o un 0.
2. Se puede escribir en ellas para establecer su estado.
3. Puede leerse para determinar en qué estado se encuentra.

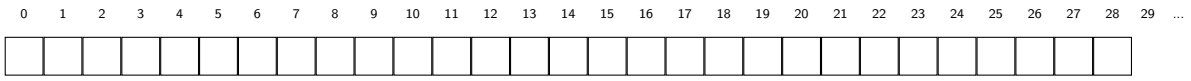


Figura 9: Bits que componen la memoria de una computadora

Dadas estas características, cada celda de memoria se debe poder acceder para determinar o establecer su estado, para ello cada una de estas celdas posee una dirección, que la identifica de las otras, esta dirección es un número. A lo largo de estos 60 años de evolución de las computadores se determinó hacia 1958, que la unidad mínima de acceso a la memoria principal de una computadora serían 8 celdas lo equivalente a 8 bits de información, esta medida se denominó byte (1byte = 8 bits), forjado por F. P. Brooks y otros en el artículo [3], ver Figura 10.

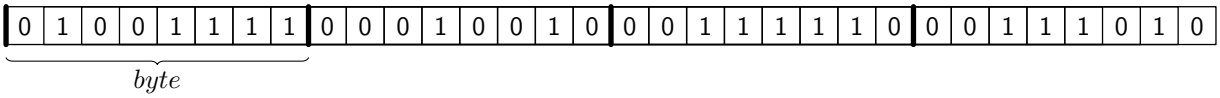


Figura 10: Bits, que componen la memoria de una computadora, agrupados en la unidad mínima de acceso ala información un $byte$

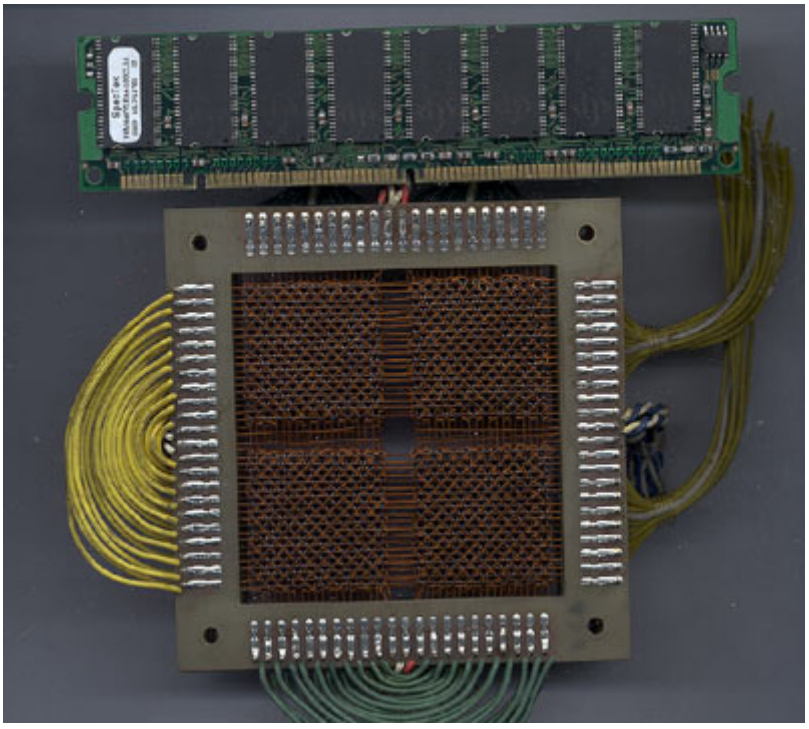


Figura 11: La Memoria. Este dispositivo está fechado en 1966. Este "módulo"de memoria se hacia en Japón por TDK. Se compone de 5 PCB con 4 "capas". Total de 8000 bits de RAM. ¡Es 1000 bytes enteros (0.9765625 kBytes)!.

Variable Es un conjunto de celdas de memoria asociado con un nombre simbólico (un identificador), que contiene alguna cantidad conocida o desconocida de **información** a la que comúnmente se refiere como **valor**. El nombre de la variable es la forma habitual de hacer referencia al valor almacenado; esta separación de nombre y contenido permite que el nombre sea utilizado de manera independiente de la información exacta que representa.

Asignación La asignación es una acción que permite al programador establecer el valor o contenido que debe almacenar dicha variable. Para ello se utiliza en pseudocódigo el símbolo $\leftarrow$, por ejemplo si se quisiera que la variable contador almacene el valor 1 se debe realizar de la siguiente manera:

Asignación	Resultado
$contador \leftarrow 1$	a la variable *contador* se le asigna el valor 1
$x \leftarrow 1$	a la variable *y* se le asigna el valor 1
$y \leftarrow 3$	a la variable *y* se le asigna el valor 3
$x \leftarrow (y + 1)$	a la variable *x* se le asigna el valor *y* más 1

La operación de asignación permite que la celda de memoria que está asociada a un nombre de una variable pueda contener un valor deseado. Esta operación debe evaluarse de derecha a izquierda, por ejemplo, si se desea incrementar en un el valor de la misma variable:

$$contador \leftarrow contador + 1$$

10. Tipos de Datos

Al solucionar un problema mediante un algoritmo no es posible utilizar cualquier tipo de información. Se utilizarán ciertos tipos de datos. Un tipo de dato puede definirse como

"Todos los valores posibles que una variable de ese tipo de dato puede tomar"

un tipo de dato es un conjunto de valores acotado por un valor máximo y un valor mínimo. Existen dos tipos de datos primitivos.

Clasificación los tipos de datos se clasifican en simples y compuestos o estructurados. Los llamados tipos de datos Simple se dividen a su vez en tipos de datos Ordinales, aquellos cuyos valores poseen sucesor y predecesor (ver Tabla 1) y tipos de datos No Ordinales, aquellos en los que no pueden determinarse un sucesor y un predecesor (ver Tabla 2). Aquellos llamados tipos de datos compuestos se verán más adelante.

Tipo de dato	Rango
Byte	0...255
Palabra	0...65535
Palabra Larga	0...4294967295
Entero Pequeño	-32768...32767
Entero Corto	-128...127
Entero Largo	-2147483648... 2147483647
Entero	Entero Pequeño o Entero Largo
Entero de 64	-9223372036854775808 .. 9223372036854775807
Booleano	verdadero, falso
Caracter	A,B,c,@,.., etc. (256 posibles)

Cuadro 1: Tipos de datos Simples Ordinales

Tipo de dato	Rango
Real	depende de la plataforma
Simple	1.5E-45 .. 3.4E38
Doble	5.0E-324 .. 1.7E308

Cuadro 2: Tipos de Datos Simples No Ordinales

Una de las acciones más importantes cuando se especifica un algoritmo es la definición de las variables.

Definir una variable implica reservar parte del espacio de la memoria de la computadora, asignarle a ese espacio un nombre y además especificar el tipo de información que se almacenará en ese espacio (número entero, número real, caracter, etc.). Para ello se utiliza la siguiente notación en pseudocódigo, en la sección de declaraciones de variables que más adelante se describe:

```
1        una_variable:un_tipo_de_dato;
2        otra_variable:otro_tipo_de_datos;
```

Algunos ejemplos reales de definición de variables en pseudocódigo:

```
1        un_valor:entero;
2        otro_valor:entero;
3        area_circulo:real;
```

La elección de nombres aunque parezca trivial la elección de nombre o identificador, como también suele llamarse, no es trivial. A continuación se enumeran algunas reglas empíricas para la elección de nombres, en un apunte posterior se verá el tema con mayor detalle [6]:

- Nombres Reveladores
- Evitar la Desinformación
- Nombres Distinguibles
- Nombres Fáciles de Buscar
- No Te Hagas el Canchero
- Una Sola Palabra por Concepto
- Evitar los Nombres Genéricos

11. Constantes

Desde el punto de vista de la algoritmia una constante es un valor que no cambia durante todo el algoritmo, este valor posee un identificador único.

Desde el punto de vista de la programación una constante es un conjunto de celdas de memoria que toman un valor al iniciar el algoritmo y posee un identificador único.

$$PI = 3{,}141592$$

$$e = 2{,}718228$$

12. Operadores

Un operador es un caracter o grupo de caracteres que actúa sobre una, dos o más variables y/o constantes para realizar una determinada operación con un determinado resultado.

12.1. Operadores Aritméticos

Permiten realizar operaciones aritméticas. Cabe destacar que en las Ciencias de la Computación la matemática no trabaja con elementos infinitos. Esto quiere decir que la recta real, como se la conoce de Análisis matemático NO EXISTE. Se verán más a delante con detalles ejemplos que demuestran estos dichos.

Operador	Descripción	Ejemplo
$*$	Multiplicación	(a$*$b)
$/$	División	(a/b)
$+$	Suma	(a+b)
$-$	Resta	(a$-$b)
$\%$	Módulo	(a$\%$b)

12.2. Operadores Relacionales

Los operadores relacionales son símbolos que se usan para comparar dos valores. Si el resultado de la comparación es correcto la expresión considerada es verdadera, en caso contrario es falsa.

Operador	Descripción	Ejemplo
<	Menor que	(a<b)
<=	Menor que o igual	(a <= b)
>	Mayor que	(a > b)
>=	Mayor o igual que	(a >= b)
==	Igual	(a == b)
!=	No igual	(a != b)

12.3. Operadores Lógicos

Los operadores booleanos devuelven tras realizar la operación un resultado booleano, es decir verdadero o falso.

Operador	Descripción	Ejemplo
expresion1 || expresion2	or	(x>2) || (y<4)
expresion1 && expresion2	and	(x>2) && (y<4)

12.3.1. Operadores Lógicos: Las Tablas

a	b	a AND b
1	1	1
1	0	0
0	1	0
0	0	0

a	b	a OR b
1	1	1
1	0	1
0	1	1
0	0	0

a	b	a XOR b
1	1	0
1	0	1
0	1	1
0	0	0

12.3.2. Operadores Lógicos: La Precedencia

Operadores	Prioridad	Asociatividad
{}, (), []	1	izquierda a derecha
++, –, !	2	Derecha a Izquierda
*, /, %	3	izquierda a derecha
+, -	4	izquierda a derecha
<, <=, >, >=, ==, ! =	5	izquierda a derecha
&&	6	izquierda a derecha
——	7	izquierda a derecha
?:	8	derecha a izquierda
=, +=, -=, *=, /=, %=	9	derecha a izquierda

13. Expresiones

"Las expresiones son combinaciones de constantes, variables, símbolos de operación, paréntesis y nombres de funciones especiales. Las mismas ideas son utilizadas en notación matemática tradicional. Una expresión consta de operandos y operadores. Según sea el tipo de objetos que manipulan, las expresiones se clasifican en:

- aritméticas
- relacionales
- lógicas
- caracter

El resultado de la expresión aritmética es de tipo numérico; el resultado de la expresión relacional y de una expresión lógica es de tipo lógico" [1]. Todas las expresiones devuelven un valor de algún tipo de dato, lo cual no significa que sea del tipo requerido.

Ejemplo de expresiones sean x e y dos variables enteras:

$$x + 1$$

$$x + y$$

$$x < y$$

$$(x >= 0)\&\&(x <= 10)$$

14. Acciones Primitivas

Una acción primitiva es aquella que no necesita ningún tipo de aclaración para ser ejecutada. Normalmente en pseudocódigo se utilizan dos acciones; una para la lectura de valores desde algún dispositivo de entrada (LEER()) y otra para la escritura de valores hacia algún dispositivo de salida (ESCRIBIR()).

14.1. Salida de Datos: Escribir()

Esta acción se utiliza para realizar operaciones de salida. La misma recibe una lista una lista de variables o expresiones y las muestra por el dispositivo estándar de salida, que normalmente es un monitor:

```
escribir (item1, item2,...itemj);
```

Los dispositivos de salida más utilizados son el monitor y la impresora.

14.2. Entrada de Datos: Leer()

De la misma forma que en pseudocódigo es posible asignarle un valor a una variable desde un dispositivo de entrada (teclado, por ejemplo). Para ello se utiliza la Acción LEER().

```
leer (item1, item2,...itemj);
```

La lista de variables o expresiones proporcionan los valores a escribir, teniendo en cuenta su orden relativo:

```
leer(gradosCentigrados);
```

Lo que obtendremos de esta acción es que se esperará que desde el dispositivo de entrada estándar se ingrese un valor, que le será asignado a la variable correspondiente.

Nota 1: Una regla no escrita pero muy útil a la hora de programar es recordar que **"siempre antes de una instrucción leer() tiene que ir una instrucción escribir()"**.

15. Estructuras de Control

En esta Sección se estudiarán las estructuras de control que proporciona Pseudocódigo. Estas son los componentes básicos que permiten al programador controlar el flujo del programa, el flujo del programa es la forma en que se van ejecutando las instrucciones. El flujo de control normal de un programa/algoritmo es **secuencial**, esto quiere decir que se ejecuta una acción o instrucción y luego la siguiente en orden de aparición, tal como fueron escritas. Las estructuras de control *permiten determinar el orden de ejecución de las acciones o instrucciones de un programa/algoritmo*, proporcionando a dichos algoritmos la capacidad de ejecutar distintas acciones/instrucciones según se cumplan situaciones diferentes [5].

Existen 3 tipos de estructuras de control:

- **Secuenciales**
- **Selectivas**
- **Iterativas**

El orden en que se ejecutan las distintas instrucciones en un algoritmo o programa se denomina **flujo de control**.

15.1. Bloque de Acciones

Un bloque de acciones representa una estructura de control secuencial. Los bloque de acciones son agrupaciones de acciones o declaraciones que sintácticamente son equivalentes a una acción sencilla, permitiendo asegurar que si el bloque se ejecuta todas las instrucciones se ejecutarán respetando la secuencia de las mismas dentro del bloque. Cabe recordar que las acciones se delimitan con el caracter ";". En el Pseudocódigo las palabras reservadas "inicio" y "fin" delimitan el inicio y el fin de un bloque.

```
inicio
        Acción1;
        Acción2;
        Acción3;
        Acción4;
fin
```

15.2. Estructuras de Control Selectivas

Estas estructuras de control permiten alterar el flujo normal del programa según se cumpla o no una determinada condición. Esta condición está contenida en la expresión que está entre paréntesis. El resultado de la evaluación de esta expresión puede ser **true** (verdadero) o **false** (falso).

15.2.1. si simple

```
si (expresion) entonces  accion;
```

En su defecto utilizando un bloque de acciones:

```
si (expresion) entonces
inicio
        accion1;
        accion2;
        accion3;
        ...
        accionj;
fin
```

15.2.2. Si - Sino

```
si (expresion) entonces
        accion1;
sino
        accion2;
```

En su defecto utilizando un bloque de acciones:

```
si (expresion) entonces
    inicio
        accion1;
        accion2;
        accion3;
        ...
        accionj;
    fin
sino
    inicio
        accionj+1;
        accionj+2;
        accionj+3;
        ...
        accionj+n;
    fin
```

15.2.3. si - sino si

La construcción de esta forma de la estructura de control si se la denomina **si múltiple** o **si anidados**;

```
si (expresion) entonces
    accion1
sino si (expresion2) entonces
    accion2
sino si (expresion3) entonces
    accion3
sino si (expresion4) entonces
    accion4
sino
    accion5;

```

Análisis: Se evalúa expresion1. Si el resultado es true(!= 0), se ejecuta accion1. Si el resultado es false (== 0), se evalúa expresion2. Si el resultado es true (!=0) se ejecuta accion2, mientras que si es false (== 0) se evalúa expresion3 y así sucesivamente. Si ninguna de las expresiones o condiciones es true se ejecuta accion5 que es la opción por defecto. Todas las acciones pueden ser simples o compuestas.

15.2.4. segun - caso

Esta estructura de control surge debido situaciones en las cuales se deben realizar un bloque de acciones, o al menos una acción diferente, para cada valor/es posible/es de una variable. Lógicamente se podría seguir utilizando un si - sino si, pero el código rápidamente perdería claridad al repetir múltiples veces siempre la misma pregunta: cual es el valor de la misma variable. Con un **segun** se pueden agrupar todos los **si** en una estructura clara que muestra sencillamente toda la información relevante de lo que está sucediendo.

Ejemplo de uso:

```
segun (variable)
inicio
        caso valor1:
             accion1; // O Bloque De Acciones
                      // Que Se Ejecuta
                      // Si Variable Es igual A Valor1

        caso valor2:
         accion2;   // O Bloque De Acciones
                    // Que Se Ejecuta
                    // Si Variable Es igual A Valor2

        ...
        ...

        caso valorN:
             accionN;   // O Bloque De Acciones
                        // Que Se Ejecuta
                        // Si Variable Es igual A ValorN

        default:
         accionDefault;  // Bloque De Acciones Que Se
                         // Ejecuta Si El Valor De La Variable
                         //Es Distinto A Todos
                         //Los Valores De Los Case Dados
fin segun
```

15.3. Estructuras de Control Iterativas

Las estructuras de control iterativas o repetitivas, repiten una acción o un bloque de acciones si se cumple una condición o mientras se cumple una condición.

15.3.1. Mientras

Esta estructura de control repite una acción o bloque de acciones **mientras** el resultado de la evaluación de la expresión de control de la iteración arroje el valor verdadero (true), esta evaluación se realiza antes de la ejecución del bloque. Este tipo de estructura de control suele denominarse en inglés loop y en español ciclo o bucle. Funciona de la siguiente manera:

1. Se evalúa la expresión booleana de control de la iteración.
2. Si la evaluación arroja un resultado verdadero se ejecuta el bloque de acciones y se vuelve al paso anterior.

3. Sino se continua con la ejecución de la acción siguiente al bloque de acciones.

Ejemplo del uso de la estructura de control mientras de una sola acción:

```
Mientras (expresion)  Hacer accion;

```

Ejemplo del uso de la estructura de control mientras repitiendo un bloque de acciones:

```
mientras (expresion)  hacer
inicio
      accion1;
      accion2;
      ...
      accionj;
fin

```

15.3.2. Hacer o Repetir - Hasta

Esta estructura de control repite una acción o bloque de acciones **hasta** que el resultado de la evaluación de la expresión de control de la iteración arroje el valor verdadero o true. Ejemplo del uso de la estructura de control Repetir - Hasta de una sola acción:

```
repetir
    accion
hasta (expresion);

```

Ejemplo del uso de la estructura de control Repetir - Hasta repitiendo un bloque de acciones:

```
repetir
        accion1;
        accion2;
        ...
        accionj;
hasta(expresion);

```

15.3.3. Para

Esta estructura de control se utiliza cuando se **conoce la cantidad de iteraciones previamente** y esta cantidad es un **número finito** de repeticiones. Ejemplo del uso de la estructura de control **para** de una sola acción:

```
para variable <- expresion1 hasta  expresion2  Hacer accion;

```

Ejemplo del uso de la estructura de control para repitiendo un bloque de acciones:

```
para Contador <- expresion1 hasta  expresion2  Hacer
inicio
        accion1;
        accion2;
        ...
        accionj;
fin

```

Dado que la estructura de control para puede parecer compleja a continuación se propone un pequeño ejemplo;

```
                para contador ← 1 hasta  8  Hacer
                inicio
                        suma ← suma +contador ;
                        tolal ← total +suma ;
                fin

```

16. Estructura de un Algoritmo

En esta sección se muestra la estructura de la especificación de un algoritmo y algunos ejemplos.

```
Algoritmo unNombre;

Const
                UnaConstante=UnValor;

Var
                UnaVariable:unTipoDeDato;
                OtraVariable:otroTipoDeDatos;

inicio
                Accion1;
                Accion2;
                Accion3;
                Accion4;
                Accion5;
Fin
```

17. Ejemplos y Ejercicios

17.1. Ejercicios

Determinar el tipo de dato del resultado de las siguientes expresiones:

1. (x>y)
2. 1.0 + sin(x)
3. (x=1)
4. (x>=2)&&(z ==3)
5. (x==1)
6. (10 - 3 % 8 + 6 / 4)
7. ((1 + 3 % 2 + 6 * 4)
   + 2)
8. (x=34) + 1

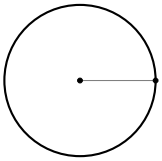
17.2. Ejemplos de Algoritmos

En esta sección se muestran algunos ejemplos de programas que contiene entrada y salida de datos. Además se presentan algunos ejemplos básicos en los que se utilizan también las estructuras de control.

Para poder validar los resultados de los ejercicios realizados en pseudocódigo se recomienda descargar PSEint un entorno de desarrollo hecho para especificar algoritmos en pseudocódigo, puede descargarse de : PSEint. Nota: existen muchas formas de solucionar cualquiera de los ejercicios dados, la que se propone es una opción de muchas.

17.2.1. Área de un Circulo

Diseñar un algoritmo que calcule la longitud de la circunferencia y el área del círculo de radio dado.



1. Análisis del problema:

   $Longitud = 2 * \pi * radio$
   $Area = \pi * radio^2$

   datos de entrada: radio del circulo.
   datos de salida: Longitud y Área del circulo.

   Ejemplo:

   $Radio = 3$
   $Longitud = 2 * \pi * radio = 2 * 3{,}1416 * 3 = 18{,}8496$
   $Area = \pi * radio^2 = 3{,}1416 * 3^2 = 28{,}2744$

2. Diseño del algoritmo:

```
1   Algoritmo circulo;
2   Const
3           pi=3.1416;
4   Var
5           radio:real;
6           longitud:real;
7           area:real;
8   inicio
9           longitud ← 0;
10          area ← 0;
11          escribir("ingrese el valor del radio del círculo");
12          leer(radio);
13          longitud ← 2*pi*radio;
14          area ← pi * radio * radio;
15          escribir("La longitud del circulo es:",longitud);
16          escribir("El area del circulo es:",area);
17  fin.
18
```

17.2.2. Ejemplo 1

Ingresar un valor entero desde el teclado y mostrar por pantalla el valor ingresado.

```
1       Algoritmo EjemploUno;
2
3       Var
4           unValor:entero;
5
6       inicio
7           escribir("Ingrese un valor entero por teclado");
8           leer(unValor);
9           escribir("el numero ingresado es ",unValor)
10      Fin
11
```

17.2.3. Ejemplo 2

Realizar un algoritmo que dados dos números enteros que se ingresan por teclado muestre el valor de la suma de ambos. Resolución versión1:

```
Algoritmo EjemploDos;

Var
    unValor:entero;
    otroValor:entero
    suma:entero largo;

inicio
    escribir("Ingrese un valor entero por teclado");
    leer(unValor);
    escribir("Ingrese otro valor entero por teclado");
    leer(otroValor);
    suma← unValor+otroValor;
    escribir("el numero ingresado es ",suma);
Fin

```

Resolución versión2:

```
Algoritmo EjemploDos;

Var
    unValor:entero;
    otroValor:entero

inicio
    escribir("Ingrese un valor entero por teclado");
    leer(unValor);
    escribir("Ingrese otro valor entero por teclado");
    leer(otroValor);
    escribir("el numero ingresado es ",unValor+otroValor);
Fin

```

17.2.4. Ejemplo 3

Desarrollar un algoritmo que permita leer dos números enteros por teclado y emitir los resultados de las siguientes operaciones:

1. La suma de ambos.
2. La resta del primero menos el segundo.
3. La multiplicación de ambos.

```
1  Algoritmo  suma
2  var
3      numeroUno:entero;
4      numeroDos:entero;
5
6  Inicio
7      escribir ("ingrese un numero entero");
8      leer (numeroUno);
9      escribir ("ingrese otro numero entero");
10     leer (numeroDos);
11     escribir ('La suma de los numeros ingresados es: ',numeroUno+numeroDos);
12     escribir ('La resta del primero menos el segundo es: ',numeroUno-numeroDos);
13     escribir ('La multiplicacion de ambos numeros es: ',numeroUno*numeroDos);
14 Fin
15
```

17.2.5. Ejemplo 4

Leer una secuencia de números reales que termina con un cero. Imprimir los datos a medida que se los ingresa junto con la suma parcial de los mismos.(por Lautaro Royan modificado por Mariano Méndez)

```
1  Algortimo  sumaparcial;
2
3  var
4      numero:real; // variable ingresada por el usuario
5      sumaParcial:real; // variable de la suma parcial
6
7  Inicio
8      sumaParcial←0;
9      escribir ('Ingresar un numero, para terminar ingrese 0');
10     leer (numero)
11     mientras (numero < > 0) Hacer // En pseudocodigo "< >" significa DISTINTO DE
12     inicio
13         escribir ('numero ingresado: ',numero);
14         sumaParcial←sumaParcial+numero;
15         escribir ('suma parcial: ' ,sumaParcial);
16         leer (numero);
17     finmientras
18 Fin.
19
20
```

17.2.6. Ejemplo 5

Dada una secuencia de números enteros terminada en cero, imprimir los tres mayores. (por Damián Ramonas /Sylvina Enriquez / Mariano Méndez)

```
1   Algoritmo LosTreMayores;
2
3   var
4       numero, mayor1, mayor2, mayor3 : Entero;
5
6   Inicio
7       escribir ("Ingrese un número:");
8       leer (numero);
9       mayor1 ← numero;
10      mayor2 ← numero;
11      mayor3 ← numero;
12      mientras (0 < >  numero) Hacer
13      inicio
14          si (mayor1 < = numero) Entonces
15          inicio
16              mayor3 ← mayor2;
17              mayor2 ← mayor1;
18              mayor1 ← numero;
19          fin
20          sino si (mayor2 < = numero) entonces
21              inicio
22                  mayor3 ← mayor2;
23                  mayor2 ← numero;
24              fin
25              sino si (mayor3 < numero) Entonces
26                      mayor3 ← numero;
27                  fin si
28              fin si
29          fin si
30
31          escribir ("Ingrese el siguiente número (la serie termina ingresando un 0):");
32          leer (numero);
33      fin
34      escribir ("Los tres números mayores son: ", mayor1,\", \", mayor2,", ", mayor3);
35  Fin.
36
```

17.2.7. Ejemplo 6

Leer un número N y calcular su factorial. (por Sylvina Enriquez).

```
1        Algoritmo CalcFactorial;
2        //definición de variables
3        Var
4        contador,numero:entero;
5        factorial      : entero largo;
6
7        // cuerpo principal
8        Inicio
9        Factorial ← 1;
10       Escribir ("Ingrese un numero para calcular su factorial: ");
11       Leer (numero);
12       Para i ← 1 hasta numero hacer
13       Factorial ← Factorial * i;
14
15       Escribir ("El factorial de ", num , " es  " , factorial);
16       Fin.
17
```

Nota: observar que para el caso de N=0 también cumple y no se trata como caso especial, pues si es cero, no entra en la iteración del "Para".

Referencias

[1] Luis Joyanes Aguilar. *Fundamentos de programación: algoritmos, estructuras de datos y objetos*. 2008.

[2] Sara Baase and Allen Van Gelder. *Algoritmos Computacionales: Introducción al análisis y diseño*. 2002.

[3] Frederick P Brooks Jr, Gerrit A Blaauw, and Wilfried Buchholz. Processing data in bits and pieces. *Electronic Computers, IRE Transactions on*, (2):118–124, 1959.

[4] Randal E Bryant, O'Hallaron David Richard, and O'Hallaron David Richard. *Computer systems: a programmer's perspective*, volume 2. Prentice Hall Upper Saddle River, 2003.

[5] Rafael Berlanga Llavori and José Manuel Iñesta Quereda. *Introducción a la programación con Pascal*. Number 2. Universitat Jaume I, 2000.

[6] Steve McConnell. *Code complete*. Pearson Education, 2004.

[7] Real Academia Española Rae. Diccionario de la lengua española. *Vigésima segunda Edición. Disponible en línea en http://www. rae. es/rae. html*, 2001.

[8] William Stallings. *Data and Computer Communications 7th Edition*. Prentice Hall, New Jersey, 2000.