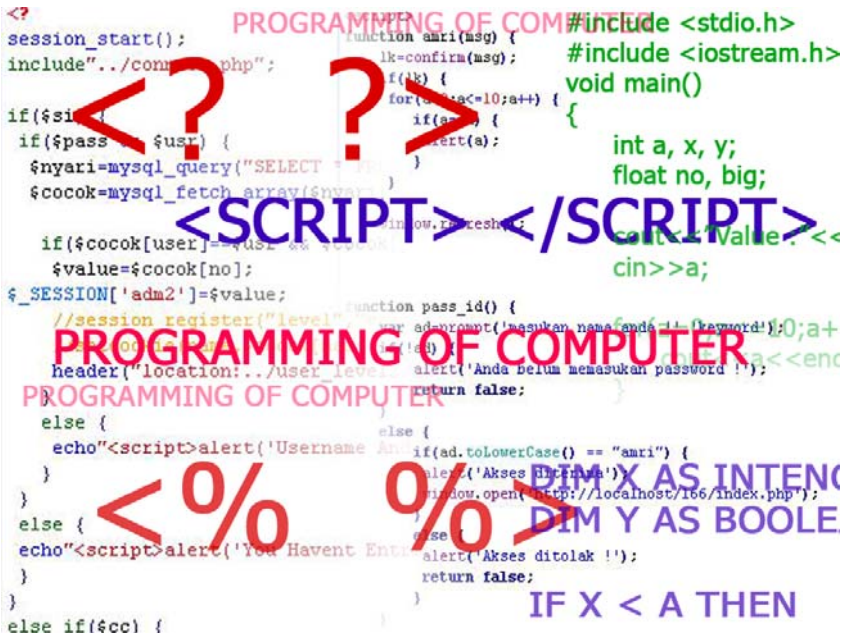
UNIVERSIDAD DISTRITAL
FRANCISCO JOSE DE CALDAS



Constructores Básicos

2013
Transversal de Programación Básica
Proyecto Curricular de Ingeniería de Sistemas

1. LOS DATOS EN PROGRAMACIÓN

1.1 DATOS

Un **conjunto** es una colección *'bien definida'* de objetos. Se dice *'bien definida'* si se sabe de manera exacta qué elementos están en la colección y qué elementos no están. Existen dos maneras de definir un conjunto: por extensión y por comprensión. Un conjunto es definido por **extensión** cuando se presentan todo sus elementos entre llaves **{}**.

Ejemplos.

A = {a, b, c}
B = {0, 1, 2, 3, 4}

Un conjunto es definido por **comprensión** cuando sus elementos se describen enunciando una propiedad que todos ellos cumplen.

Ejemplos.

C = {x | x es un número primo}
D = {x | x es una vocal}

A cada conjunto se le puede asignar uno o varios nombres; dichos nombres constituyen el **tipo** de los elementos del conjunto.

De manera informal, un **dato** es una pieza de información simple como un número, un código, un hecho o una edad. De manera formal, un **dato** es un elemento concreto de algún conjunto. El nombre del conjunto al que pertenece el dato constituye el tipo del mismo. Los tipos de datos más utilizados en programación son:

Entero:	El nombre asignado al conjunto de números enteros.	-123,...,-2,-1,0,1,2,...,85
Real:	El nombre asignado al conjunto de números reales.	-0.12,...,-1.0,0.0,..., 3.1416
Caracter:	El nombre asignado al conjunto de caracteres.	'a', '*', '+', ';'
Booleano:	El nombre asignado al conjunto de valores de verdad.	falso, verdadero
Cadena:	El nombre asignado al conjunto de cadenas de caracteres.	"Hola", "AbC123"

Como se puede advertir en los ejemplos anteriores, cuando se trabaja en programación es necesario:

- *Distinguir los números enteros de los números reales*. Por ejemplo el **2** (sin punto decimal), indica que el elemento es del conjunto de los enteros, mientras que el **2.0** (con punto decimal), indica que es un elemento del conjunto de los reales.

- *Distinguir los símbolos y palabras que forman parte del sistema de representación o codificación del algoritmo, de los usados para representar un dato concreto de los tipos caracter y cadena de caracteres respectivamente.* En este curso, los datos de tipo caracter son representados entre comillas simples ('), mientras que los datos de tipo cadena de caracteres son representados entre comillas dobles ('').

1.2 VARIABLES

Una variable es un símbolo que permite referenciar (señalar o demarcar) un espacio en memoria en el que se puede almacenar un dato. Toda variable posee tres características: nombre, tipo y estado.

a. **Nombre o identificador**: El nombre de una variable es una secuencia de caracteres alfanuméricos (letras y dígitos) y caracteres de subrayado, la cual siempre empieza por una letra. En algunos lenguajes de programación (C++, C, Java), se hace distinción entre mayúsculas y minúsculas mientras que en otros no (Pascal, Basic). Por ejemplo, las variables *VelMax* y *velmax* son diferentes en *C* mientras que son la misma variable en *Pascal*.

Una buena técnica de programación es asignarle el nombre a una variable de tal manera que indique por un lado el papel que desempeña dicha variable en el algoritmo y por otro los posibles valores que almacena.

Ejemplos de nombre de variable.

- velocidad
- x
- valor1
- exponente
- valMaximo

b. **Tipo**. El tipo de una variable es el mismo tipo de los datos que se pueden almacenar en el espacio de memoria que la variable referencia. Por ejemplo, una variable de tipo entero, es decir, que se ha declarado como entera, sólo se puede usar para referenciar datos de tipo entero. La cantidad de memoria, es decir, el espacio en memoria en bits, asignada para almacenar un dato de un tipo depende del sistema operativo y del compilador del lenguaje.

En este curso una variable es declarada de la siguiente manera (seudo-código):

<nombre> : **<tipo>**

Donde **<nombre>** es el nombre que se le dará a la variable y **<tipo>** es el tipo de datos que la variable puede referenciar.

Ejemplos de declaración de variables[1].

- valMaximo : real
- contador : entero
- nombre : cadena
- letra : caracter
- bandera : booleano

c. **Estado o valor**. El estado o valor de una variable es el dato almacenado en el espacio que referencia la variable. El valor de una variable puede cambiar en el tiempo, es decir, conforme el algoritmo se va ejecutando, pero solamente puede tomar valores que pertenezcan al tipo de datos declarado para la variable.

1.3 LITERALES

Un **literal** es una secuencia de caracteres que representa un valor concreto. Los literales son clasificados de acuerdo al tipo de datos que representan. Las siguientes secciones describen la forma de los literales usada en este capitulo (esta representación es específica para cada lenguaje de programación).

a. ***Literales enteros****:* Un literal entero es una secuencia de dígitos (0,1,..,9), posiblemente precedida por el signo menos (-).

Ejemplos de literales enteros.

- 12345
- 4768
- –138
- 2609
- 10

b. ***Literales reales****:* Un literal real es una secuencia de dígitos, posiblemente precedida por el signo menos, y en la cual debe aparecer el punto decimal (.), que permite representar un dato de tipo real. El punto decimal separa la parte entera de la parte decimal del número; si no existe punto decimal el literal es considerado como un literal entero y generalmente, si después del punto no hay dígitos, el literal se considera incorrecto[2]. Los siguientes son ejemplos de literales reales:

Ejemplos de literales reales.

- 3465.98
- 29.0
- -123.78
- 23.7e-4 (en notación científica 23.7e-4 equivale a $23.7 * 10^{-4)}$
- –3.9876

[1] Declarar una variable es asignarle un nombre y definirle su tipo.
[2] En muchos casos se puede utilizar notación científica.

c. ***Literales de caracteres****:* Un literal de tipo carácter es una símbolo delimitado por comillas simples (') que permite representar al carácter ubicado entre las comillas simples.

Ejemplos de literales de carácter.

- 'a' representa el caracter **a**
- ' ' representa el caracter espacio en blanco
- '3' representa el caracter dígito **3**
- '$' representa el caracter de pesos.
- '?' representa el caracter interrogación de cierre.

d. ***Literales Cadenas****:* Un literal de cadena es una secuencia de símbolos, delimitada por comillas dobles, que sirve para representar la cadena de caracteres delimitada por las comillas dobles.

Ejemplos de literales de caracter.

- "Pepito va al colegio", representa la cadena ***Pepito va al colegio***.
- "El área de la circunferencia (2*pi*r) es : " , representa la cadena ***El área de la circunferencia (2*pi*r) es :***.

e. ***Literales booleanos****:* Son las palabras: falso y verdadero, las cuales son usadas para representar los dos valores de verdad.

1.4 CONSTANTES

Una constante es un símbolo que permite referenciar un espacio en memoria en el que hay un dato almacenado que **NO** se puede cambiar. Las constantes se declaran de la siguiente manera:

<nombre> = **<literal>**

Donde, **<nombre>** es el nombre de la constante y <literal> es el literal que representa el valor de la constante. Una buena técnica de programación es usar letras mayúsculas para los nombres de las constantes.

Ejemplos de constantes.

- PI = 3.1415926
- VELOCIDAD_LUZ = 300000000.0
- SALUDO_BASICO = "Hola, Buenos días"
- TAMANO_MAXIMO = 1000
- ESPACIO = ' '

1.5 EXPRESIONES

En muchas ocasiones, la ejecución de una tarea del algoritmo implica la realización de un cálculo matemático (operaciones aritméticas y/o lógicas). Una **expresión** es una serie de términos (constantes, literales, variables y funciones) posiblemente agrupados mediante paréntesis y

conectados mediante operadores (aritméticos como +, - y lógicos como &, I), que representan un cálculo matemático. El proceso que permite determinar el valor de la expresión, es decir el resultado del cálculo, es conocido como evaluación de expresión. Según el tipo del resultado de la expresión, el cual es conocido como tipo de la expresión, las expresiones se clasifican en:

- Expresiones numéricas: Si el resultado de la expresión es un entero o un real.
- Expresiones lógicas: Si el resultado de la expresión es un valor de verdad.

a. **Expresiones numéricas**: Son expresiones en las que solamente aparecen operadores aritméticos (+, -, *, /, %3) , conectando términos de tipo numérico exclusivamente. Por ejemplo, una expresión como:

Ejemplo:

(A + B) – (5 * C) + 8

Donde A, B y C son variables de tipo entero. En las expresiones se pueden utilizar funciones de tipo numérico.

Ejemplo:

(A + 5) * (Y + piso(X+2.5))

Es una expresión numérica si A es una variable de tipo entero (pues A esta siendo sumada con un literal de tipo entero 5), Y es una variable de tipo entero y X es una variable de tipo real (la función piso(X) calcula el entero más cercano al real X por debajo, por ejemplo, piso(3.45) = 3 y piso(-4.678) = -5).

Entre las funciones numéricas y de manejo de cadenas más usadas en programación y definidas en el seudo-código se cuentan[3]:

Funciones nombre_funcion(<parametros>): <tipo>	**Descripción de la Función**
raiz2(x:real):real	Calcula raíz cuadrada
ln(x:real):real	Calcula logaritmo natural
exp(x:real):real	Calcula $\mathbf{e^x}$
sen(x:real):real	Calcula seno (x)
cos(x:real):real	Calcula coseno(x)
tan(x:real):real	Calcula tangente(x)
asen(x:real) :real	Calcula arc seno(x)
acos(x:real) :real	Calcula arc coseno(x)

[3] El operador %, retorna el módulo (o residuo) de la división entera de dos números, por ejemplo, 11 % 3 = 2 y 11 % 4 = 3.

atan(x:real) **:**real	Calcula arc tangente(x)
aleatorio(x:entero) **:**real	Produce un numero aleatorio entero entre [0,x)
elevar(x:real,exp:real) **:**real	Eleva x^{exp}
abs(x:real) **:**real	Calcula valor absoluto de x
piso(x:real) **:**real	devuelve el entero menor más cercano a x
techo(x:real) **:**real	devuelve el entero mayor más cercano a x
enteroAreal(x:entero) **:**real	convierte un literal entero a real, o una variable de tipo entero a real
realAentero(x:real) **:**entero	convierte un literal real a entero, o una variable de tipo real a entero.
caracterAentero(c:caracter)**:**entero	convierte un literal carácter a su representación en ASCII, o una variable de tipo caracter a su representación en ASCII
enteroAcaracter(x:entero)**:**entero	convierte un literal entero a su carater equivalente, o una variable de tipo entero a su caracter equivalente
longitudCadena(cad1: arreglo[] de caracter)**:**entero	devuelve la longitud de la cadena.cad1
compararCadenas(cad1: arreglo[] de **caracter**, cad2: arreglo[] de caracter)**:entero**	Compara las cadenas cad1 y cad2 en orden lexicográfico. devuelve un entero <0 si (cad1 < cad2) devuelve un entero >0 si (cad1 > cad2) devuelve 0 si (cad1 = cad2)
compararCadenasM(cad1: arreglo[] de **caracter**, cad2: arreglo[] de caracter)**:entero**	Compara las cadenas cad1 y cad2 en orden lexicográfico, sin importar las letras en mayúscula. devuelve un entero <0 si (cad1 < cad2) devuelve un entero >0 si (cad1 > cad2) devuelve 0 si (cad1 = cad2)

b. ***Expresiones lógicas****:* Una expresión lógica es aquella en la cual el resultado se da en términos de **verdadero** o **falso**. Generalmente una expresión lógica se construye a partir de expresiones comparativas (dos expresiones numéricas relacionadas mediante algún operador relacional), de variables y/o literales booleanos, los cuales se conectan mediante operadores lógicos.

Los operadores de comparación usados en programación son los siguientes: mayor que (>)**,** menor que (<)**,** igual (=), mayor o igual que (>=)**,** menor o igual que (<=) y diferente (<>); mientras que los operadores lógicos usados son: o (ı), y (**&**) y la negación (~).

En los siguientes cuadros se resumen las tablas de verdad de estos operadores lógicos.

TABLAS DE VERDAD

Negación: (~)

p	~(p)
V	F
F	V

Conjunción lógica (&) y disyunción lógica (•)

p	q	p | q	p&q
V	V	V	V
V	F	V	F
F	V	V	F
F	F	F	F

Por ejemplo, una expresión como:

(A + 5 >=3) & (Y + piso(X + 2.5) <=3 * X) ı ~bandera

Es una expresión lógica si **A** y **Y** son variables de tipo entero, **X** es una variable de tipo real y **bandera** es una variable de tipo booleano.

1.5. 1 Evaluación de expresiones

Una expresión sea del tipo que sea se evalúa mediante el siguiente algoritmo:

Inicio
PASO 1. Reemplazar todas las variables de la expresión por su valor.
PASO 2. Desde los paréntesis más internos hacia los más externos mientras existan paréntesis y/o operadores hacer:
2.1. Si una función no tiene todos sus argumentos evaluados, evaluar cada uno de los mismos.
2.2. Evaluar toda función que tenga sus argumentos evaluados y reemplazarla por su valor resultado.
2.3. Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.
2.4. Si sólo un número y/o valor de verdad se encuentra entre paréntesis eliminar los paréntesis.
PASO 3. El número o valor de verdad que resulta es el valor de la expresión.
Fin

Para que una expresión sea correcta, los términos que se conectan mediante un operador deben ser del mismo tipo. Por ejemplo, si en una expresión se suma una variable A con un literal entero, la variable debe ser tipo entero.

• *Precedencia de operadores*

La precedencia de operadores es un orden de evaluación estándar, que se les ha asignado a los operadores para evitar excesivo uso de paréntesis. Evitan las posibles ambigüedades en el proceso de evaluación, es decir, evitan que una expresión pueda tomar más de un valor. Por ejemplo la expresión 3 + 4 * 5, es ambigua pues puede ser interpretada como: (3 + 4) * 5 = 35 o como 3 + (4 * 5)=23, las cuales al ser evaluadas producen resultados diferentes. La precedencia de operadores está definida en la siguiente tabla:

Precedencia de operadores	
Precedencia	**Operadores**
8	**( ) Paréntesis**
7	**- (signo menos)**
6	***, /, %**
5	**+, - (substracción)**
4	**>, <, = ,>= , <=, <>**
3	**~**
2	**&**
1	**|**

Entre más alta es la precedencia más rápido se debe evaluar el operador. Nótese que los paréntesis tienen la máxima precedencia, es decir, lo que está entre los paréntesis es lo primero que se evalúa; mientras que la disyunción lógica (|) tiene la más baja precedencia, lo que indica que será el último en ser evaluado. Cuando una expresión esta formada por operadores de la misma precedencia, dichos operadores son evaluados de izquierda a derecha, salvo unas pocas excepciones.

Ejemplos de evaluación de expresiones

Si los valores de las variables enteras **A**, **B** y **C** son **5**, **3** y **9** respectivamente, la siguiente expresión **-(3 + 4) * A + B * C** se evalúa de la siguiente forma:

-(3 + 4) * 5 + 3 * 9	Reemplazando las variables por sus valores
-(7)*5+3*9	Paréntesis prioridad más alta (8)
-7*5+3*9	Eliminación paréntesis
-35+27	Multiplicación, prioridad seis (6)
-8	Suma, prioridad cinco (5)

Si los valores de las variables enteras **A**, **B** y **C** son **5**, **3** y **9** respectivamente, la siguiente expresión **-(3 + 4) * A / B * C + B * C** se evalúa de la siguiente forma:

-(3 + 4) * 5 / 3 * 9 + 3 * 9	Reemplazando las variables por sus valores
-(7) * 5 / 3 * 9 + 3 * 9	Paréntesis prioridad más alta (8)
-7 * 5 / 3 * 9 +3 * 9	Eliminación paréntesis
-35 / 3 * 9 +27	Multiplicación prioridad seis (6) de izquierda a derecha.
-11 * 9 +27	División prioridad seis (6) de izquierda a derecha.
-99 +27	Multiplicación prioridad seis (6).
-72	Suma prioridad cinco (5)

Si **A** y **Y** son variables de tipo **entero** y **X** de tipo **real**, con valores **6**, **8** y **–1.8** respectivamente, la expresión **(A + 5) * (Y+ piso(X+2.5))** se evalúa de la siguiente manera:

Reemplazar todas las variables por su valor	**(6+5)*(8+piso(-1.8+2.5))**
Aplicar toda función	**(6 + 5) * (8+ piso(0.7))**
Aplicar toda función que tenga todos sus argumentos evaluados y reemplazar la función por el valor resultado de la función.	**(6 + 5) * (8+ 0)**
Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.	**(11)*(8)**
Eliminar los paréntesis si sólo un número o un valor de verdad se encuentra entre ellos	**11*8**
Realizar las operaciones indicadas según la precedencia de los operadores que actúan sobre números y/o valores de verdad, es decir, términos ya evaluados.	**88**
El número o valor de verdad que resulta es el valor de la expresión.	**88**

Sean **A** y **B** variables de tipo entero con valores **6** y **8** respectivamente, la expresión siguiente **(A > B+4)** | **((A = 3) & ~(B < 4))** es evaluada como se muestra en la siguiente página.

Reemplazar todas las variables por su valor.	**(6>8+4) | ((6=3) & ~(8<4))**
Evaluar la suma	**(6>12) | ((6=3) & ~(8<4))**
Evaluar los operadores >, = y <	**(falso) | ((falso) & ~ (falso))**
Eliminar los paréntesis que encierran un solo valor.	**falso |(falso &~falso)**
Evaluar la negación (~).	**falso |(falso &verdadero)**
Evaluar la conjunción (&).	**falso |(falso)**
Eliminar los paréntesis que encierran un solo valor.	**falso | falso**
Evaluar la disyunción (|)	**falso**

Como se puede apreciar en el segundo ejemplo, la división de enteros da como resultado un número entero y no un número real.

Ejercicios de expresiones

1. Si ***tiempo***, ***velocidad*** y ***peso*** son variables de tipo **real**; ***contador***, ***itera*** y ***suma*** son de tipo **entero**; ***letra*** y ***primo*** son de tipo **carácter** y ***bandera*** es de tipo **booleano**, determinar cuáles de las siguientes expresiones son válidas y cuales no; se debe justificar la respuesta.

a. (tiempo + itera) / (velocidad + peso)
b. **piso(**tiempo**)** + contador < itera ***** suma & ~bandera
c. tiempo ***** velocidad < bandera & peso >=0.0
d. letra + contador ***** suma
e. **techo(**velocidad**) *** suma + **piso(**tiempo**) *** itera

2. Si ***tiempo***, ***velocidad*** y ***peso*** son variables de tipo **real**; ***contador***, ***itera*** y ***suma*** son de tipo **entero**; ***letra*** y ***primo*** son de tipo **carácter** y ***bandera*** es de tipo **booleano**, con valores **3.0, -4.5, 8.0, 5, -2, 30, 'p', 't' y falso** respectivamente, evaluar las siguientes expresiones

a. itera + **piso(**tiempo + velocidad**)** / (suma + itera)
b. letra <>primo **& techo(**tiempo**)** / **piso(**velocidad**)** + cuenta < 5
c. peso ***** velocidad **/ (** tiempo + 5.6 **)**
d. contador + (**(**itera**)*(itera)*** suma**)** / 4 – itera
e. bandera **&**(**raiz2(**tiempo**)** <=peso ***** velocidad)
f. contador < suma ı tiempo > 1.0 **&** ~(suma = itera)

2. INSTRUCCIONES SOBRE VARIABLES, LITERALES Y CONSTANTES

Las dos operaciones que se pueden realizar sobre una variable son modificar su valor y mostrar su valor. Existen dos instrucciones diferentes que permiten modificar el valor de una variable: asignación y lectura, mientras que existe sólo una forma de mostrar el valor: escritura.

La única operación que se puede realizar sobre constantes y/o literales es mostrar su valor y se usa de la misma manera como se muestra el valor de una variable.

2.1 ASIGNACIÓN

La **asignación** es la instrucción por medio de la cual se puede modificar el valor de una variable utilizando el resultado de una expresión. Para que la asignación sea válida, es decir, se pueda realizar, la variable y la expresión deben ser del mismo tipo.
La asignación tiene la siguiente forma:

<**nombre**>:=<**expresión**>

Donde, **<nombre>** es el nombre de la variable, que debe ser previamente declarada, y <**expresión**>es una expresión del mismo tipo de la variable **<nombre>**.

El símbolo := se usará para indicar que el valor de la expresión <**expresión**> se almacena en la variable <**nombre**>.

Para realizar una asignación (ejecutar), se sigue el siguiente algoritmo:

1. Evaluar la expresión.
2. Almacenar el resultado de la expresión en la dirección de memoria referenciada por la variable.

Ejemplos de asignación

Si **n** es una variable de tipo **entero** que tiene el valor **5**, la asignación

n :=n + 1, se ejecuta de la siguiente manera:

1.	Evaluar la expresión (1. reemplazar todas la variables por su valor)	**n :=5 + 1**
2.	Evaluar la expresión (2. realizar las operaciones indicadas según su precedencia)	**n :=6**
3.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**n = 6**

La variable **n** queda finalmente con el valor de 6. Si **n**, **m** y **k** son variables de tipo **entero** y tienen los valores **5**, **4** y **–3** respectivamente, la asignación **m :=k + n * 5**, se ejecuta de la siguiente manera:

1.	Evaluar la expresión (1. Reemplazar todas la variables por su valor)	**m := -3 + 5 * 5**
2.	Evaluar la expresión (2. realizar las operaciones indicadas según su precedencia)	**m :=22**
3.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**m = 22**

La variable **m** queda finalmente con el valor de **22**.

Si **A** y **Y** son variables de tipo **entero** y **X** de tipo **real**, con valores **6**, **8** y **–1.8** respectivamente, la asignación **A :=(A + 5) * (Y+ piso(X+2.5))**, es ejecutada de la siguiente manera:

1.	Evaluar la expresión (La evaluación fue realizada en los ejemplos de la sección anterior)	**A :=88**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**A = 88**

La variable **A** queda con el valor **88**.

Sean **A** y **B** variables de tipo **entero** con valores **6** y **8** respectivamente, **bandera** de tipo **booleano** con valor **verdadero**, la asignación: **bandera :=(A > B+4) ı ((A = 3) &~(B < 4))** es ejecutada como se muestra en la página siguiente.

1.	Evaluar la expresión (La evaluación fue realizada en los ejemplos de la sección anterior)	**bandera :=falso**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**bandera = falso**

La variable **bandera** queda con el valor de verdad **falso**.

Suponga que **y** es una variable de tipo **real** que tiene el valor **–3.0**, la asignación **y :=3 + y * y** es ejecutada como se muestra en la siguiente página.

1.	Evaluar la expresión	**y :=12.0**
2.	Almacenar el valor en el espacio de memoria referenciado por la variable.	**y = 12.0**

La variable **y** queda finalmente con el valor de **12.0**.

Como se puede apreciar en los ejemplos anteriores, la variable a la que se le está asignando el resultado de la expresión puede aparecer en la expresión, sin que ello sea un error. Se debe recordar que primero se evalúa la expresión (tomando el valor que tengan las variables en ese momento), y posteriormente se almacena el resultado en la variable.

2.2 LECTURA Y ESCRITURA

La instrucción **lectura** es la que le permite al usuario del algoritmo modificar el valor de una variable, durante la ejecución del mismo. Esta instrucción espera que el usuario ingrese un valor (mediante el teclado el usuario escribe el valor a almacenar en la variable) y después, garantizando que sea del tipo apropiado, lo asigna a la variable. La lectura tiene la siguiente forma:

En este curso una variable es leída de la siguiente manera:

leer (<nombre>)

donde **<nombre>** es el nombre de la variable que se está leyendo.

Ejemplo de lectura

- leer (n)

Si **n** es variable de tipo **entero** y si el usuario ingresa el valor de **30**, después de ejecutar la instrucción de lectura la variable **n** tendrá el valor de **30.**

La **escritura** es la instrucción que permite mostrar, normalmente en la pantalla, el valor de una variable, una constante o un literal durante la ejecución. La escritura tiene la siguiente forma:

escribir(<nombre>)

donde, <nombre> es el nombre de la variable, constante o literal que se está mostrando.

Ejemplo de escritura

- escribir (n)

Si **n** es variable de tipo **entero** y si la variable **n** tiene el valor de **30**, en la pantalla se presentará dicho valor.

Ejemplo en seudo-código

```
procedimiento principal()
variables
   // declaración de una variable de tipo booleano
   miBooleano: booleano
   // declaración de una variable de tipo entero
   miEntero: entero
   // declaración una variable de tipo real
   miReal: real
   // declaración de una variable de tipo caracter
   miCaracter: caracter
   // declaración de una variable de tipo cadena
   miCadena: cadena

  inicio
   // inicializar miBooleano
   miBooleano := verdadero
   // inicializar miEntero
   leer (miEntero)
    // inicializar miReal
   miReal := 0.0
   // inicializar miCaracter
   miCaracter 'c'
   // inicializar miCadena
   miCadena := "Respuesta: "
   escribir("Valores de mis variables: ")
   cambioLinea()
   escribir("variable miBooleano:  ")
   escribir(miBooleano)
   cambioLinea()
   escribir("variable miEntero:  ")
   escribir(miEntero)
   cambioLinea()
   escribir("variable miReal:  ")
   escribir(miReal)
   cambioLinea()
   escribir("variable miCaracter:  ")
   escribir(miCaracter)
   cambioLinea()
   escribir("variable miCadena:  ")
   escribir(miCadena)
   cambioLinea()
fin-procedimiento
```

Ejercicios de instrucciones sobre variables

1. Si ***x, v*** y ***p*** son variables de tipo **real**; ***cont, i*** y ***k*** son variables de tipo **entero**; ***letra*** y ***c*** son variables de tipo **carácter**, ***band*** y ***terminar*** son variables de tipo **booleano**; ***MAX*** es una constante de tipo **entero** y ***PI*** es una constante de tipo **real**, determinar cuáles de las siguientes instrucciones son válidas. Se debe justificar la respuesta.

- leer (letra)
- escribir (MAX)
- leer (cont)
- escribir (x)
- leer (MAX)
- leer (v)
- x := enteroAreal(k) + 2.0 * PI
- letra :='p'
- letra :="p"
- letra :=p
- letra :=c
- p :=v / x * raiz2(p) – band
- x :="3.2444" + "1.4e-4"
- x :=3.2444 + 1.4e-4
- x :='v' + 'p'
- terminar :=p > v >=x
- terminar :=verdadero
- terminar :="falso"
- terminar :='f' <=letra
- x + 3.0 :=v
- v :=x * p + enteroAreal(caracterAentero(letra))
- band :=(x+v)*p I terminar I cont <= k + i
- k := caracterAentero (letra + caracterAentero (c)) + k

2. Si ***x, v*** y ***p*** son variables de tipo **real**, con valores **3.5**, **1.4** y **6.0** respectivamente, determinar el valor de la variable ***z*** de tipo **real** al realizar cada una de las asignaciones siguientes:

- z :=v - p / (x + p / x – v)
- z :=p / x / v + p / x * v
- z:= (v – 3.0 * x / p) / (4.0 – v / (5.0 + p / x))
- z :=2.0*p – 4.0/v + 5.0*x/(3.0+v*p – x)

3. Suponga que ***x, v*** y ***p*** son variables de tipo **real**, con valores **3.5**, **1.4** y **6.0** respectivamente; ***i, k*** y ***cont*** son variables **enteras** con valores **5**, **2** y **–4**, respectivamente; ***letra*** y ***c*** son variables de tipo **carácter** con valores **'p'** y **'t'**; y ***bandera*** y ***terminar*** de tipo **booleano** con valores **falso** y **verdadero** respectivamente. Determine el valor de cada una de las variables, después de ejecutar las siguientes instrucciones, en el orden en que aparecen.

- x :=v + enteroAreal(caracterAentero(letra))
- k :=(k–2*cont)* (k–2*cont) + i*(cont – k / 2.0)/cont
- v :=x + (p – raiz2(x) * (v – x) + 3.0) * p
- bandera :=(terminar ı~(i+2 <>cont ıp>=v)) ı(x+v) = p
- cont :=cont + 1
- i:= cont * realAentero(x * v – p/2.0) + k – i
- p :=p + x – enteroAreal(piso(p+x) + 5)
- letra :=enteroAcaracter(k)

- k :=(k + cont) mod (piso(p) + techo(v))
- v :=x + (p – raiz2(x) * (v – x) + 3.0) * p
- k :=(k–2*cont) (k–2*cont) + i*(cont – k / 2.0)/cont

3. ESTRUCTURAS DE CONTROL

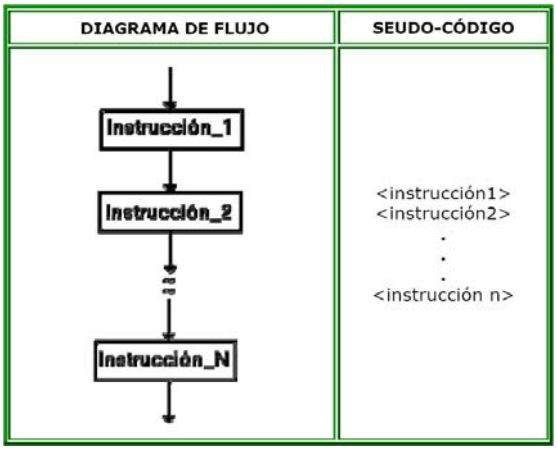
Las estructuras de control tienen como finalidad ir señalando el orden en que tienen que sucederse los pasos de un algoritmo.

Ejemplo
Suponga que acaba de mostrar un mensaje en la pantalla que pregunte al usuario: ***"¿desea salir? - Si / No".***
Obviamente, de la respuesta del usuario va a depender la siguiente acción del programa. Por ejemplo, este mensaje se nos presenta tras haber seleccionado la opción de cerrar el programa, si nosotros digitamos "Sí", el programa finalizará, y si digitamos "No", el programa continuara con su ejecución. *El programador tiene que haber escrito código para las dos posibilidades, aunque cuando el programa esté funcionando, sólo se elegirá una.*

Las estructuras de control pueden clasificarse en tres tipos: Secuenciales, Selectivas y Repetitivas

3.1 ESTRUCTURAS SECUENCIALES

Dada una lista de instrucciones **<instrucción 1> <instrucción 2> ... <instrucción n>** las estructuras **SECUENCIALES** permite la ejecución de dicha lista en el orden en que aparecen las instrucciones, es decir, se ejecuta primero la instrucción **<instrucción 1>** luego la instrucción **<instrucción 2>**, y por último se ejecuta la instrucción **<instrucción n>.** Una secuencia de instrucciones es llamada generalmente **BLOQUE DE INSTRUCCIONES**.
La forma general de la secuencia es:



Ejemplos de Estructuras Secuenciales.

Aplicando la metodología de programación presentada en este curso, encontrar los algoritmos que permiten resolver los siguientes problemas:

1. Dado el radio de un círculo, calcular su área.

Análisis Del Problema

VARIABLES CONOCIDAS	Un número que representa el radio de un círculo
VARIABLES DESCONOCIDAS	Un número que representa el área de un círculo.
CONDICIONES	El número buscado corresponde al área del círculo con radio igual al número dado.

Especificación

ENTRADAS	r ∈ Reales, (**r** es el radio del círculo).
SALIDAS	a ∈ Reales, (**a** es el área del círculo).
CONDICIONES	$a = pi * r^2$

Diseño

Primera división:

Inicio
Paso 1. leer el valor del radio del círculo
Paso 2. calcular el área del círculo
Paso 3. escribir el área del círculo
Fin

División Final:

```
PI = 3.1415926 /* se define la constante pi */
r: real    /*se define la variable para el radio del círculo*/
a : real /* se define la variable para el área del círculo */
leer (r) /* se lee el radio */
a := PI * cuadrado( r ) /* se calcula el área */
escribir (“El área es: ” ),
escribir (a)  /* se escribe el resultado */
```

Prueba De Escritorio

Este algoritmo cuenta con siete (7) líneas, las tres primeras, son para definir las variables y constantes usadas y las últimas cuatro, son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las cuatro últimas líneas, teniendo en cuenta los valores para las constantes y las variables.

LINEA	r	a	ENTRADA	SALIDA
4	5.0		5.0	
5		78.53981		
6				El área es:
7				78.53981

2. Un granjero tiene una cantidad **X** de animales de dos tipos: conejos y gansos. Si la cantidad de patas total de los animales **Y** es conocida, ¿cuántos conejos y cuantos gansos tiene el granjero?

Análisis Del Problema

VARIABLES CONOCIDAS	La cantidad total de animales **X** y la cantidad de patas totales **Y**.
VARIABLES DESCONOCIDAS	La cantidad de conejos y la cantidad de gansos.
CONDICIONES	La suma de los conejos y los gansos es igual a **X**. La suma de los patas de los conejos (cuatro por cada uno) y de los gansos (dos por cada uno) es igual a **Y**.

Especificación

ENTRADAS	X, Y ∈ Naturales, (**X** es el total de animales, **Y** es el total de patas).
SALIDAS	C, G ∈ Naturales, (**C** es el número de conejos, **G** es el número de gansos).
CONDICIONES	X = G + C, Y = 2*G + 4*C

Aplicando los conocimientos de álgebra a las condiciones anteriores, para expresar las variables de salida **C** y **G** en términos de las variables de entrada **X** y **Y**, se tiene que:

G= (4*X - Y)/2, C = (Y - 2*X)/2

Diseño

Primera división:

Inicio
Paso 1. Leer las entradas
Paso 2. Realizar el cálculo
Paso 3. Imprimir los resultados
Fin

Segunda división:

Inicio
Paso 1. Leer las entradas
Paso 1.1. Leer el número total de animales
Paso 1.2. Leer el número total de patas
Paso 2. Realizar el cálculo
Paso 2.1. Calcular el número de qansos
Paso 2.2. Calcular el número de conejos
Paso 3. Imprimir los resultados
Paso 3.1. Imprimir el número de gansos
Paso 3.2. Imprimir el número de conejos
Fin

Segunda División:

```
1  x: entero /* se define la variable para el número total de animales */
2  y: entero /* se define la variable para el número total de patas de los animales */
3  g: entero /*se define la variable para el número de gansos*/
4  c: entero /*se define la variable para el número de conejos*/
5  leer (x) /* lee el número total de animales */
6  leer (y) /* lee el número total de patas de los animales */
7  g := (4*x - y)/2 /* calcula el número de gansos */
8  c := (y - 2*x)/2 /* calcula el número de conejos */
9  escribir ("Cantidad de gansos:" )
10 escribir (g)
11 escribir ("Cantidad de conejos:" )
12 escribir (c)
```

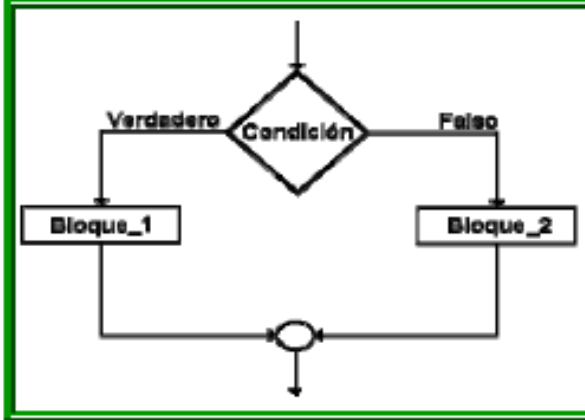
Prueba De Escritorio

Este algoritmo cuenta con doce (12) líneas, las cuatro primeras, son para definir las variables usadas y las últimas ocho, son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas (5-12), teniendo en cuenta los valores para las variables.

LINEA	X	y	g	c	ENTRADA	SALIDA
5	50				50	
6		140			140	
7			30			
8				20		
9						Cantidad de gansos:
10						30
11						Cantidad de conejos:
12						20

3.2 ESTRUCTURAS SELECTIVAS

Las **ESTRUCTURAS SELECTIVAS** o de **SELECCION** permiten la ejecución de un bloque de instrucciones o de otro, dependiendo del valor de una expresión lógica denominada **CONDICION**. La forma general de la selección es:

DIAGRAMA DE FLUJO	SEUDO-CÓDIGO
	**si** <condición> **entonces** <bloque 1> **sino** <bloque 2> **finsi**

Donde, **<condición>** es la expresión lógica que se evalúa para determinar el bloque de instrucciones a ejecutar, **<bloque 1>** es el conjunto de instrucciones que se ejecuta si la condición evalúa Verdadero y **<bloque 2>** es el conjunto de instrucciones que se ejecuta si la condición evalúa Falso.

Ejemplo de selección.

Dados un número entero entre 100 y 999 determinar si es capicúa o no lo es. Un número es capicúa si se lee igual al derecho y al revés. Por ejemplo, 101 es capicúa (al derecho se lee 101 y al revés se lee 101), mientras que 203 no lo es (al derecho se lee 203 y al revés se lee 302).

Análisis Del Problema

VARIABLES DESCONOCIDAS	Un texto.
VARIABLES CONOCIDAS	Un número entero entre 100 y 999.
CONDICIONES	El texto debe indicar si el número ingresado es capicúa o no lo es.

Especificación

ENTRADAS	n ∈ Enteros, (n es el número ingresado).
SALIDAS	Texto ∈ Cadenas, (Texto es una cadena de caracteres indicando si el número es o no capicúa).
CONDICIONES	n entre 100 y 999 Texto = "Es capicúa" si u = c "No es capicúa" en otro caso donde u representa las unidades de n, y c las centenas de n

Diseño
Primera división:

```
Inicio
Paso 1.  Leer el número.
Paso 2.  Determinar si es capicúa o no.
Paso 3.  Imprimir el texto resultado.
Fin
```

Segunda División:

```
Inicio
Paso 1.   Leer el número.
Paso 2.   Determinar si es capicúa o no
Paso 2.1. Calcular las unidades del número
Paso 2.2. Calcular las decenas del número.
Paso 2.3.    Calcular el texto resultado.
Paso 3. Imprimir el texto resultado.
Fin
```

División Final:

```
1  n: entero /* se define la variable para el número */
2  u: entero /* se define la variable unidades del número*/
3  c: entero /*se define la variable centenas del número*/
4  texto: cadena /* define la variable para el texto */
5  leer (n)  /* lee el número */
6  u := n mod 10 /* calcula las unidades del número */
7  c :=  ( n / 100 ) mod 10 /* calcula las centenas del número */

8  si u = c entonces
9    texto :=  "es capicúa" /* el número es capicúa */
10 sino
11  texto :=  "no es capicúa" /* el número no es capicúa */
12 finsi

13 escribir ( "El número es:")

14 escribir (texto)
```

Prueba De Escritorio

Este algoritmo cuenta con catorce (14) líneas, las cuatro primeras, son para definir las variables usadas y las últimas diez son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 5 - 14, teniendo en cuenta los valores para las variables.

Primer prueba de escritorio

LINEA	n	u	c	Texto	ENTRADA	SALIDA
5	404				404	
6		4				
7			4			
8	La condición es evaluada a verdadero, por lo tanto se pasa a la línea 9, la siguiente línea al **entonces** de la selección en ejecución.					
9				es capicúa		
10	Se salta hasta el **finsi** de la selección en ejecución, es decir, hasta la línea 12					
12	Se salta a la siguiente línea.					
13						El número es:
14						capicúa

Segunda prueba de escritorio

LINEA	n	u	c	texto	ENTRADA	SALIDA
5	725				725	
6		7				
7			5			
8	La condición es evaluada a falso, por lo tanto se pasa a la línea 11, la siguiente línea al **sino** de la selección en ejecución.					
11				no es capicúa		
12	Se salta a la siguiente línea.					
13						El número es:
14						no capicúa

Dos bloques de instrucciones), lecturas, escrituras, asignaciones u otras selecciones.

3.3 ESTRUCTURAS DE SELECCIÓN ANIDADAS

En los bloques de instrucciones de una estructura de selección el programador puede utilizar otras estructuras de selección. Cuando esto sucede se dice que las estructuras de selección están anidadas.

```
si<condición1> entonces
<variable> := <expresión 1>
sino
  si <condición 2>entonces

      <variable> := <expresión 2>

  sino
 ..............
  sino
   si <condición n>entonces

        <variable>:= <expresión n>

   sino

        <variable> := <expresión final>

   finsi
 ..............
  finsi

finsi
```

Ejemplo de selección anidada.

Dados tres números enteros calcular el máximo.

Análisis Del Problema

VARIABLES DESCONOCIDAS	Tres números enteros.
VARIABLES CONOCIDAS	Un número entero.
CONDICIONES	El número buscado es el máximo de los tres números dados.

Especificación

ENTRADAS	A, B, C e Enteros, (A es el primero número, B es el segundo número y C es el tercer número).
SALIDAS	Mayor e Enteros, (Mayor es el máximo de los tres números dados).
CONDICIONES	$Mayor = \begin{cases} A \text{ si } A \geq B \wedge A \geq C \\ B \text{ si } B \geq A \wedge B \geq C \\ C \text{ si } C \geq A \wedge C \geq B \end{cases}$

Diseño
Primera división:

Inicio
Paso 1. Leer los tres números
Paso 2. Determinar cual es el máximo
Paso 3. Imprimir el máximo
Fin

Segunda División:

Inicio
Paso 1. Leer los tres números
Paso 1.1. Leer el primer número
Paso 1.2. Leer el segundo número
Paso 1.3. Leer el tercer número
Paso 2. Determinar cual es el máximo
- Si el primer número es mayor o igual a los otros dos números el primero es el máximo.
- Si no es así y si el segundo número es mayor o igual a los otros dos números el segundo es el máximo.
- De otra manera, se tiene que el tercer número es el máximo.

Paso 3. Imprimir los resultados
Fin

División Final:

```
1  a : entero /* se define la variable para el primer número */
2  b : entero /* se define la variable para el segundo número */
3  c : entero /*se define la variable para el tercer número */
4  mayor : entero /*se define la variable para el máximo */

5  leer (a)  /* lee el primer número */
6  leer (b)  /* lee el segundo número */
7  leer (c)  /* lee el tercer número */

8  si a >= b & a >= c entonces
9   mayor :=a  /* el máximo es a pues es mayor que b y c */
10 sino
11  si b >= a & b >= c entonces
12     mayor := b  /* el máximo es b pues es mayor que a y c */
13  sino
14     mayor := c /* sino es ni a ni b entonces es c */
15  finsi

16 finsi

17 escribir  ("El máximo de los tres es:  ")
18 escribir (mayor)
```

Prueba De Escritorio

Este algoritmo cuenta con dieciocho (18) líneas, las cuatro primeras, son para definir las variables usadas y las últimas catorce son las instrucciones que son aplicadas sobre dichos datos. De esta manera la prueba de escritorio se debe realizar solamente sobre las líneas 5-18, teniendo en cuenta los valores para las variables.

Primera prueba de escritorio

LINEA	a	b	C	mayor	ENTRADA	SALIDA
5	50				50	
6		140			140	
7			30		30	
8	La condición es evaluada a falso, por lo tanto se pasa a la línea 11, la siguiente al **sino** de la selección en ejecución					
11	La condición es evaluada a verdadero, por lo tanto se pasa a la línea 12, la siguiente línea al **entonces** de la selección en ejecución.					
12				140		
13	Se salta hasta el **finsi** de la selección en ejecución, es decir, hasta la línea 15					
15	Se salta a la siguiente línea.					
16	Se salta a la siguiente línea.					
17						El máximo de los tres es:
18						140

Segunda prueba de escritorio

LINEA	a	b	C	Mayor	ENTRADA	SALIDA
5	90				90	
6		-50			-50	
7			70		70	
8	La condición es evaluada a verdadero, por lo tanto se pasa a la línea 9, la siguiente al **entonces** de la selección en ejecución.					
12				90		
13	Se salta hasta el **finsi** de la selección en ejecución, es decir, hasta la línea 16					
16	Se salta a la siguiente línea.					
17						El máximo de los tres es:
18						90

Tercera prueba de escritorio

LINEA	a	b	C	mayor	ENTRADA	SALIDA
5	20				20	
6		10			10	
7			30		30	
8	La condición es evaluada a falso, por lo tanto se pasa a la línea 11, la siguiente al **sino** de la selección en ejecución.					
11	La condición es evaluada a falso, por lo tanto se pasa a la línea 14, la siguiente línea al **sino** de la selección en ejecución.					
14				30		
15	Se salta a la siguiente línea.					
16	Se salta a la siguiente línea.					
17						El máximo de los tres es:
18						30

3.4 ESTRUCTURA DE SELECCIÓN MÚLTIPLE

La **Estructura de Selección Múltiple** proporciona una manera muy práctica para seleccionar entre un conjunto de opciones predefinidas. Si el valor de la <opción> coincide con <constante_i> se ejecutará el <bloque de instrucciones i>. Si el valor de <opción> no coincide con ninguna <constante_i> se ejecutara el <bloque de instrucciones> establecido en **otro caso**.

```
seleccionar  <opción> hacer

caso <constante_1>:
                    <bloque instrucciones 1>
caso <constante  2>:
                    <bloque instrucciones 2>

---------------

caso <constante  n>:
                    <bloque instrucciones n>
otrocaso :
                    <bloque instrucciones>
finseleccionar
```

Ejemplo de Selección múltiple

Una persona selecciona una transacción en un cajero automático

Análisis Del Problema

VARIABLES DESCONOCIDAS	Que transacción desea realizar el usuario
VARIABLES CONOCIDAS	Los números correspondientes a las transacciones disponibles en el cajero.
CONDICIONES	El cajero debe ejecutar la transacción seleccionada por el usuario

Especificación

ENTRADAS	n e Enteros entre 1 y 5, (n es el número ingresado por el usuario para seleccionar la transacción).
SALIDAS	Ejecución de una Transacción
CONDICIONES	El numero n seleccionado por el usuario debe corresponder a la transacción seleccionada por el usuario

Diseño

Primera División:

Inicio
Paso 1. Leer el número.
Paso 2. Determinar la transacción y Ejecutarla.
Fin

Segunda División:

Inicio
Paso 1. Leer el número.
Paso 2. Determinar la transacción
Paso 2.1. si es 1
Paso 2.1.1. Ejecutar la transacción 1
Paso 2.1.2. Salir
Paso 2.2. si es 2
Paso 2.2.1. Ejecutar la transacción 2
Paso 2.2.2. Salir
Paso 2.3. si es 3
Paso 2.3.1. Ejecutar la transacción 3
Paso 2.3.2. Salir
Paso 2.4. si es 4
Paso 2.4.1. Ejecutar la transacción 4
Paso 2.4.2. Salir
Paso 2.5. si es 5
Paso 2.5.1. Ejecutar la transacción 5
Paso 2.5.2. Salir
Paso 2.5. si no esta en el rango
Paso 2.5.1. Salir
Fin

División Final:

```
n: entero /* se define la variable para la selección del usuario*/
leer (n)  /* lee el número de la transacción seleccionada*/
seleccionar  n hacer
caso 1:
        <consignación>
caso 2:
        <retiro>
caso 3:
        <pago de servicios>
caso 4:
        <cambio de clave>
caso 5:
        <consulta de saldo>
finseleccionar
```

4. CODIFICACIÓN DE ALGORITMOS EN C++

TRADUCCIÓN DE ESTRUCTURAS BÁSICAS

Para traducir un algoritmo representado en UN pseudocódigo, al lenguaje de programación C++, se deben seguir las siguientes reglas:

	SEUDOCODIGO	C++
Definición Variables	x : tipo	tipo x;
Definición Constante	PI = 3.1415926	PI = 3.1415926
Asignación	=	=
Operadores Aritméticos		
Suma	+	+
Resta	-	-
Multiplicación	*	*
División	/	/
Módulo	**mod**	%
Lectura	**leer** (a)	**cin** >>a;
Impresión	**escribir(a)**	**cout** << a;
Cambio de línea	cambio_linea	"\n"
Cadena Caracteres	"cadena"	"cadena"

Selección	**si** (condición)
	entonces
	bloque_instrucciones1
	sino
	bloque_instrucciones2
	fin_si
		if (condición)
		{
		bloque_instrucciones1;
		}
		else
		{
		bloque_instrucciones2
		;
		}

Selección Múltiple	**Seleccionar** (variable) *d e*
	caso constante 1:
	bloque_instrucciones_1
	.
	.
	.
	caso constante n:
	bloque_instrucciones_n
	
	fin_seleccionar
		switch(opción)
		{
		case constante_1:
		{
		
		bloque_instrucciones_1;
		break;
		{
		
		.
		.
		case constante_n:
		{
		
		bloque_instrucciones_n;
		{
		break;
		}

Comentarios	/* comentario */	/* comentario */
Operadores Lógicos		
negación	~	!
y lógico	&	&&
o lógico	|	||
Oper. Relacionales		
Menor que	<	<
Mayor que	>	>
Igual a	=	==
Menor o igual que	<=	<=
Mayor o igual que	>=	>=
Diferente a	<>	!=

Ejemplos.

1. El siguiente algoritmo que imprime el mayor de dos números enteros.

SEUDOCODIGO	*C++*

procedimiento principal()
variables
a:**entero**
b:**entero**
inicio
escribir("Digite numero:")
leer (a)
escribir "Digite numero"
leer (b)
si a<b **entonces**
escribir ("El mayor es:")
escribir (b)
 	
sino
escribir ("El mayor es:")
escribir (a)
fin_si
fin-procedimiento
	/* para mostrar en pantalla */
	#include <iostream.h>
	void main()
	{
	int a; /* a es entera */
	int b; /* b es entera */
	
	cout << "Digite numero:";
	cin >> a;
	cout << "Digite numero:";
	cin >>b;
	if(a<b)
	{
	cout << "El mayor es:";
	cout << b;
	}
	else{
	cout <<"El mayor es:";
	cout << a;
	}
	}

Traducción de tipos de datos:

Pseudo Código	*C++*
x : real	***float*** x; ***double*** x;
contador : entero	***int*** contador;
letra : carácter	***char*** letra;
bandera : booleano	***bool*** bandera;

Lecturas de Profundización:

- El texto fue tomado de:
http://aplicaciones.virtual.unal.edu.co/drupal/files/Constructores%20basicos%20-%20programacion%20de%20computadores.pdf

Imágenes:
Las imágenes fueron tomadas de www.google.com

Referentes:

- http://teleformacion.edu.aytolacoruna.es/PASCAL/document/vars.htm
- http://www.frro.utn.edu.ar/repositorio/catedras/sistemas/1_anio/algoritmo_estructura_datos/SORRIBAS.pdf

- http://www.taringa.net/posts/ciencia-educacion/9085903/Programacion-1-constantes-variables-tipos-de-datos-operadore.html
- http://mimosa.pntic.mec.es/~flarrosa/pseudoco.pdf
- http://wwwdi.ujaen.es/asignaturas/mtp1/Tema02-concepto_de_algoritmo-2dpp.pdf