TEMA 3

Estructuras de control. Desarrollo de programas

1. Introducción.

Como ya sabemos, un algoritmo consta de la descripción de los objetos que manipula y de la descripción de las acciones o instrucciones que constituyen un método de solución del problema.

Como estrategia para el diseño de algoritmos, usaremos en el tema de acciones y funciones el diseño descendente por refinamientos sucesivos. Al final, nuestra forma de desarrollar algoritmos se apoya en un resultado que establece que cualquier algoritmo puede describirse utilizando solamente tres composiciones lógicas o estructuras de control: secuencial, condicional y repetitiva. Desde el punto de vista del lenguaje podemos pensar que nuestra notación algorítmica sólo tiene tres formas de componer frases a partir de frases bien construidas.

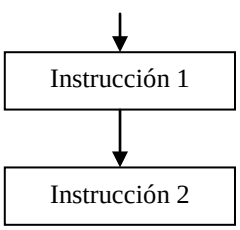
2. Estructura secuencial.

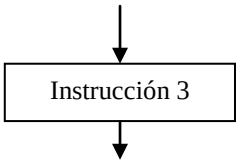
La composición secuencial es el mecanismo de composición de acciones más elemental y consiste en descomponer una acción compleja en una secuencia de acciones (instrucciones) que deben ser ejecutadas en un orden determinado.

Sintaxis:

```
principio
   <instrucción1>
   <instrucción2>
        .
        .
        .
   <instrucciónN>
fin
```

Esquema:





Las instrucciones se ejecutan una y sólo una vez de forma secuencial (una tras otra).

Ejemplos:

1- Dados dos números enteros, calcular su suma.

```
Especificación:
Interfaz:
 Entrada entero x, y
 Salida  entero suma
Efecto:
 Condiciones de entrada: x, y con valores cualesquiera
 Efecto producido: suma = x + y

algoritmo sumados
variables
 entero x, y, suma
principio
 leer(x,y)
 suma = x+y
 escribir(suma)
fin
```

2- Determinar la cantidad anual producida por una inversión de dinero a interés fijo conocidos el capital invertido y el interés de rendimiento (constante=8%).

```
Especificación:
Interfaz:
 Entrada   real capital
 Salida    real cantidad
Efecto:
 Condiciones de entrada: capital >= 0
 Efecto producido: cantidad = capital * 0.08

algoritmo inversión
variables
      real capital,cantidad
principio
 leer(capital)
 cantidad = capital * 0.08
 escribir( cantidad)
```

```
fin
```

3- Especificar y diseñar un algoritmo que calcule las raíces de la ecuación de segundo grado ax+bx+c=0 suponiendo que las soluciones son reales.

Especificación:

Interfaz:
  Entrada real a, b, c
  Salida real r1, r2
Efecto:
  Condiciones de entrada: b*b-4*a*c>=0, a≠0
  Efecto producido: r1= $\frac{-b+\sqrt{b^2-4ac}}{2a}$, r2= $\frac{-b-\sqrt{b^2-4ac}}{2a}$

```
Algoritmo ecuación
variables
  real a,b,c,r1,r2
  real x1,x2
principio
  leer(a, b, c)
  x1=-b/(2*a)
  x2=sqrt(b*b-4*a*c)/(2*a)
  r1=x1+x2
  r2=x1-x2
  escribir(r1, r2)
fin.
```

4.- Dados los valores de los coeficientes a, b, c y d del polinomio ax+bx+cx+d, especificar y diseñar un algoritmo que evalúe el polinomio en un valor x.

Especificación:

Interfaz:
  Entrada real a,b,c,d,x
  Salida real p
Efecto:
  Condiciones de entrada: a, b, c, d, x cualesquiera
  Efecto producido: p=ax+bx+cx+d

```
Algoritmo Horner;
//Se utiliza el método de Horner para evaluar un polinomio
variables
  real a, b, c, d, x, p
principio
  leer(a, b, c, d, x)
  p=(d+x*(c+x*(b+x*a)))
  escribir(p)
fin
```

3. Estructura alternativa (condicional o selectiva)

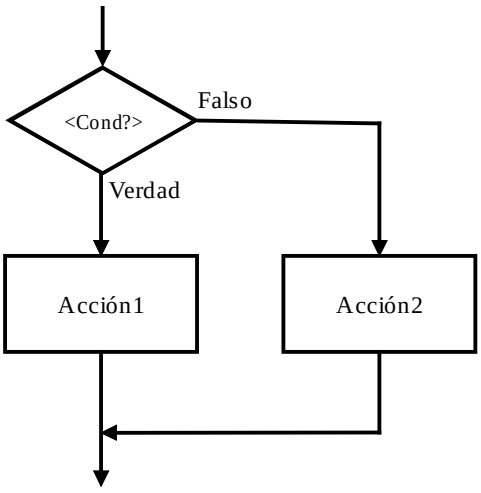
La estructura condicional se utiliza para tomar decisiones lógicas. Es necesaria cuando la resolución de un problema requiere elegir entre dos o más alternativas posibles. Dependiendo de si una determinada condición se satisface o no, se ejecutarán unas acciones u otras.

3.1 Esquema básico

Sintaxis:

```
si condición entonces
      acciones 1
si no
      acciones 2
fsi
```

Esquema:



La semántica (funcionamiento) es:

1) Evalúa *condición*, expresión lógica que devuelve un valor lógico (verdad o falso).

2) Si el valor anterior es verdad, ejecuta el bloque de acciones *acciones 1*, en caso contrario ejecuta *acciones 2*. En ambos casos, continúa con la siguiente instrucción al fsi.

Ejemplos

1- Especifica y diseña un algoritmo que divida dos números enteros introducidos por teclado, teniendo en cuenta que no es posible dividir por cero.

Especificación:

```
Interfaz:
 Entrada entero dividendo, divisor
 Salida  entero cociente
Efecto:
 Condiciones de entrada: dividendo y divisor con valor cualquiera
 Efecto producido: cociente = dividendo DIV divisor, si divisor≠0,no calculable si divisor=0
```

```
algoritmo división
variables
 entero dividendo, divisor,cociente
principio
 leer(dividendo, divisor)
 si divisor==0 entonces
      escribir("No es posible realizar esta división")
 si no
      cociente= dividendo DIV divisor
      escribir(cociente)
 fsi
fin
```

2- Algoritmo que calcule el máximo de dos números enteros distintos.

Especificación:

```
Interfaz:
 Entrada entero x1, x2
 Salida entero max
Efecto:
 Condiciones de entrada: x1 ≠ x2
 Efecto producido: max es x1 si x1>x2, max es x2 en caso contrario
```

```
algoritmo máximo
variables
 entero x1,x2,max
principio
 leer(x1,x2)
 si x1>x2 entonces
      max =x1
 si no
      max =x2
 fsi
 escribir(max)
```

```
fin
```

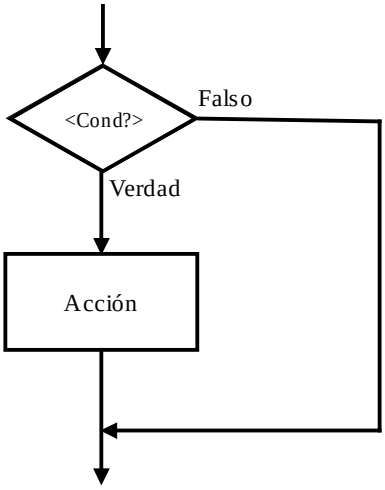
3.2 Condicional incompleta.

Surge al eliminar de la anterior estructura la ramificación **si no**

Sintaxis:

```
si condición entonces
   acciones
fsi
```

Esquema:



La semántica es:

1) Evalúa *condición*.

2) En caso de ser cierta, ejecuta *acciones*, si es falsa las ignora y continúa tras el fsi.

La secuencia de acciones se ejecutará sólo si la condición es cierta. Si la condición es falsa, la secuencia de acciones será ignorada.

Una vez evaluada la condición y ejecutada, si procede, la secuencia de acciones, el control del programa pasa a la siguiente instrucción al fsi.

3.3 Alternativas anidadas.

Muchos problemas necesitan de alternativas más complejas, pero siempre se pueden construir a partir de alternativas simples. Por ejemplo, la siguiente es una sentencia válida, ya que cada secuencia de acciones puede ser una única acción, varias o una estructura de control (condicional o repetitiva).

Ejemplo:

```
si condición 1 entonces
      acciones 1
si no
      si condición 2 entonces
           acciones 2
      si no
           acciones 3
      fsi
fsi
```

Por supuesto, puede haber estructuras anidadas en cualquier ramificación y en cualquier nivel de profundidad.

Ejemplo:

Calcular las soluciones de la ecuación de segundo grado $ax^2+bx+c=0$.

Especificación:

Interfaz:
Entrada real a, b, c
Salida real r1, r2
Efecto:
Condiciones de entrada: a≠0; b, c cualesquiera
Efecto producido: r1= $\frac{-b+\sqrt{b^2-4ac}}{2a}$, r2=$\frac{-b-\sqrt{b^2-4ac}}{2a}$

```
algoritmo ecuación
variables
 real a,b,c,r1,r2,disc
principio
 leer(a,b,c)
 disc = b*b - 4*a*c
 si disc==0 entonces
      r1=-b/(2*a)
      escribir("Raíz doble ",r1)
 si no
      si disc>0 entonces
           r1 = (-b+sqrt(disc))/(2*a)
           r2 = (-b-sqrt(disc))/(2*a)
           escribir(r1, r2)
      si no
           escribir("No existen raíces reales")
      fsi
 fsi
fin
```

3.4 Selección múltiple

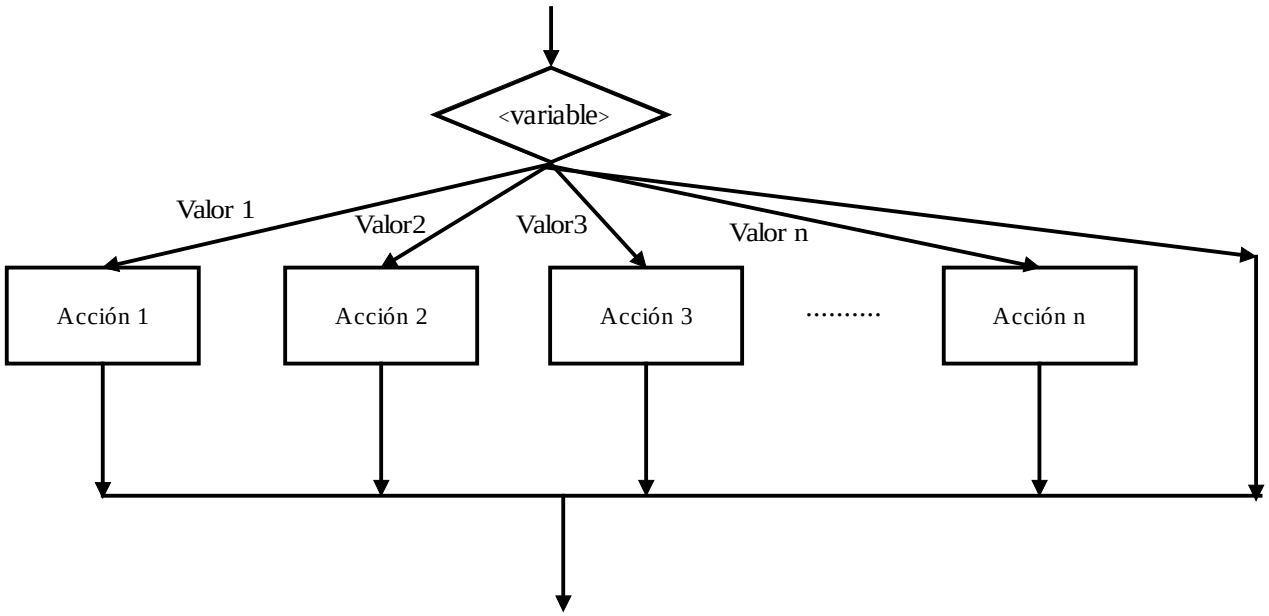
La estructura de selección múltiple permitirá elegir entre un conjunto finito de acciones, que deben ejecutarse de forma excluyente, **dependiendo del valor que pueda tomar una variable**. La estructura de selección múltiple constituye, por tanto, un caso particular del esquema condicional exclusivo.

Sintaxis:

```
según variable sea
   caso <valor1>: acciones 1
   caso <valor2>: acciones 2
         :
   caso <valorN>: acciones N
   otros: acciones M
fsegún
```

La cláusula otros es opcional.

Esquema:



Semántica:

1) Evaluar *variable* obteniendo un valor (la variable debe ser de tipo escalar: entero, carácter).

2) **Localizar** el valor coincidente con el valor obtenido. En caso de no localizar el valor, seleccionar **otros**.
3) **Ejecutar** la secuencia de instrucciones asociada al valor seleccionado en el paso anterior.

Nota: Los valores que etiquetan cada acción pueden ser también intervalos además de valores concretos.

Ejemplos:

1- Dado un número entre 1 y 12, escribir el nombre del mes correspondiente.

```
  Especificación:
  Interfaz:
   Entrada entero mes
   Salida ---
  Efecto:
   Condiciones de entrada: 1 <= mes <= 12
   Efecto producido: muestra el nombre del mes
correspondiente al número introducido

  algoritmo meses
  variables
   entero mes
  principio
   leer(mes)
   según mes sea
        caso 1:    escribir(“enero”)
        caso 2:    escribir(“febrero”)
        caso 3:    escribir(“marzo”)
        caso 4:    escribir(“abril”)
        caso 5:    escribir(“mayo”)
        caso 6:    escribir(“junio”)
        caso 7:    escribir(“julio”)
        caso 8:    escribir(“agosto”)
        caso 9:    escribir(“septiembre”)
        caso 10:  escribir(“octubre”)
        caso 11:  escribir(“noviembre”)
        caso 12:  escribir(“diciembre”)
   fsegún
  fin
```

2- Escribir un algoritmo que presente en pantalla un menú con tres opciones, permita elegir una y escriba un mensaje distinto para cada una de ellas.

```
Especificación:
Interfaz:
 Entrada carácter opcion
 Salida: ---
Efecto:
 Condiciones de entrada: { }
 Efecto producido: {muestra un mensaje....}
```

```
algoritmo menu
variables
  carácter opcion
principio
  escribirln("A. Primera opción")
  escribirln("B. Segunda opción")
  escribirln("C. Tercera opción")
  escribirln
  escribir("Elige una opción: ")
  leer(opcion)
  según opcion sea
    caso 'A': escribir("Opción A")
    caso 'B': escribir("Opción B")
    caso 'C': escribir("Opción C")
  otros:
    escribir("No has elegido ninguna de las tres opciones")
  fsegún
fin
```

4. Estructura repetitiva o iterativa.

Muchos programas precisan que un grupo de instrucciones se ejecute repetidamente, ya sea un número de veces *determinado* de antemano o un número de veces *indeterminado*.

Veremos varios esquemas iterativos.

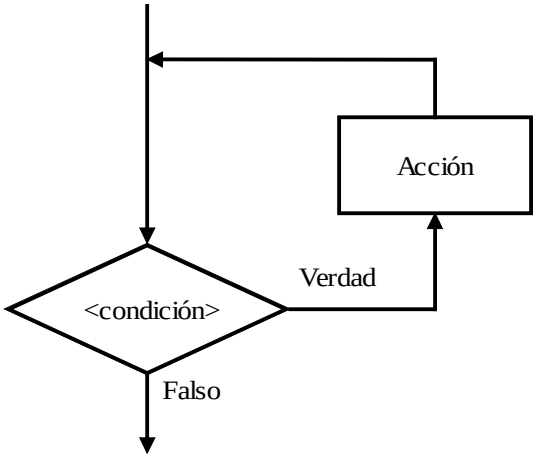
4.1 Esquema iterativo básico

Repite la ejecución de una secuencia de acciones **mientras** la condición sea *cierta*. Este esquema es el caso general, con él podemos construir cualquier estructura repetitiva. No es necesario conocer de antemano cuántas iteraciones serán necesarias, incluso contempla el caso de que no se ejecute ninguna vez el cuerpo del ciclo.

Sintaxis:

```
mientras que condición hacer
   acciones
fmq
```

Esquema:



Semántica :

1) Evaluar *condición* obteniendo como resultado un valor lógico (*verdad* o *falso*).

2) Si el valor anterior es *verdad*, se ejecutan las acciones del cuerpo del ciclo *acciones* y se vuelve al punto anterior, es decir, a 1). En caso contrario, valor *falso*, no se ejecuta ninguna acción.

Hay dos aspectos a considerar al utilizar una estructura repetitiva:

a) Si inicialmente no se satisface la condición, las acciones del cuerpo del ciclo *acciones* no se ejecutan ninguna vez, por tanto, *acciones* pueden ejecutarse 0 a n veces.

b) La *condición* debe, en algún momento, devolver el valor *falso*, para no entrar en un bucle infinito. Para ello es imprescindible que dentro del ciclo repetitivo haya al menos una acción que tenga efecto sobre la condición. A la condición se le llama condición de permanencia en el ciclo, puesto que se permanece en el ciclo mientras la condición es cierta.

Aplicación a la resolución de problemas

Mediante la estructura repetitiva podemos conseguir que un bloque de instrucciones se ejecute varias veces, obviamente sin tener que escribirlas repetidas veces en nuestro programa.

Ejemplo1. Programa que escriba en pantalla una vez el mensaje “Hola”

…

```
principio
      escibir (“Hola”)
fin
```

Ejemplo2. Programa que escriba en pantalla 10 veces el mensaje “Hola”

```
…
principio
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
      escibir (“Hola”)
fin
```

¿Si en vez de 10 queremos escribir 100 o 1000 saludos? Parece que la solución anterior no es la mejor. Solución: utilizar la estructura repetitiva.

```
…
principio
      n=0
      mientras que (n<10) hacer
           escribir (“Hola”)
           n=n+1
      fmq
fin
```

Obviamente la estructura repetitiva sirve para escribir programas más interesantes que el anterior. Imagina que debemos escribir un programa para hacer un determinado proceso para cada uno de los empleados de una empresa cuyos datos están almacenados en un fichero o BD. La estructura de ese programa se parecería bastante al programita anterior, sería la siguiente:

```
…
principio
      …
      acceder al primer empleado
      mientras que queden empleados hacer
           procesar empleado
           acceder al siguiente
      fmq
fin
```

En otras ocasiones se utiliza la estructura repetitiva para llegar a la solución de un problema *por pasos*, o dicho de una manera más formal, aplicando el método de inducción. Esta aproximación se puede esquematizar de la forma:

Resolución del problema en un caso trivial; {Solución de P}

```
mientras que no esté resuelto el problema P en el tamaño deseado hacer

  resolver el problema Pa partir de la solución del problema P
fmq
```

Los puntos claves a tener en cuenta al construir una estructura repetitiva son:

1) Identificar un caso trivial del problema P y dar su solución.
2) Encontrar la ley de recurrencia que permita resolver el problema Pa partir de la solución del problema P.
3) Decidir cuándo se logra el objetivo. Estado final solución de problema P. Asegurarnos que la instrucción iterativa finaliza.

Ejemplos:

1- Especificar y diseñar un algoritmo que calcule el factorial de un entero positivo introducido por teclado.

Especificación:

```
Interfaz:
 Entrada entero n
 Salida entero fact
Efecto:
 Condiciones de entrada: n>=0
 Efecto producido: fact=n!
algoritmo factorial
variables
 entero n,fact,i
principio
 leer(n)
 fact=1  {solución inicial, caso trivial, 0!=1}
 i=1
 mientras que i<=n hacer
      fact= fact*i
      i=i+1
 fmq
 escribir(fact)
fin
```

2- Especificar y diseñar un algoritmo que calcule la suma de los números pares hasta 100.

Especificación:

```
Interfaz:
  Entrada: ---
  Salida: entero suma
Efecto:
  Condiciones de entrada: { }
  Efecto producido: suma=2+4+...+100
algoritmo suma_pares
variables
  entero suma, n
principio
  n=2
  suma=0                  // Solución inicial
  mientras que n<=100 hacer
      suma=suma+n
      n=n+2
  fmq
```

```
  escribir(suma)        //suma=2+...+100
fin
```

3- Especificar y diseñar un algoritmo que calcule el valor del número e a partir del desarrollo en serie:

$$e = 1 + 1/1! + 1/2! + ... + 1/n! + ...,$$

cortando después de sumar un término cuyo valor sea inferior a una cierta cota introducida desde el teclado.

cota ----> ----> e

Especificación:

```
Interfaz:
  Entrada: real cota
  Salida:  real e
Efecto:
  Condiciones de entrada: cota>0
  Efecto producido: e = 1 + 1/1! + 1/2! + ... + 1/n! + ...
```

```
algoritmo calcular_e
variables
  real       cota, e, termino
  entero     i
principio
  leer(cota)
  término=1
  e=0 { Solución del problema trivial }
  i= 1
  mientras que término>cota hacer
      e=e + término         //Ley de recurrencia
      término= término/i         //término=1/i!
      i= i+1
  fmq
  escribir(e)
fin
```

4.2 Esquema de repetición condicional

Repite la ejecución de una secuencia de acciones **hasta que** la condición sea *cierta*. La diferencia fundamental con la anterior es que siempre se ejecutan al menos una vez las instrucciones del ciclo.

Sintaxis:

```
repetir
   acciones
mientras que condición
```

Semántica:

1) **Ejecutar** <secuencia de acciones>
2) **Evaluar** <condición> y si el resultado es *cierto*, vuelve al punto anterior, si no, termina la instrucción.

Notas:

- Con este esquema, siempre se ejecuta al menos una vez las acciones comprendida entre repetir ...mientras, es decir se ejecutará **1 a n** veces.
- La repetición acaba cuando la condición es falsa (**condición de permanencia**).

Es equivalente a las instrucciones:

```
acciones
mientras que condición hacer
   acciones
fmq
```

Ejemplos:

1- factorial

```
Interfaz:
 Entrada entero n
 Salida entero fact
Efecto
 Condiciones de entrada: n>=0
 Efecto producido: fact=n!
```

```
algoritmo factorial
variables
 entero  n, fact, i
principio
 leer(n)
 fact=1
 i=0
 repetir
      fact= fact*(i+1)
      i=i+1
 mientras que i<n
 escribir(fact)
fin
```

2- Escribir un algoritmo que presente en pantalla un menú con tres opciones, permita elegir una y escriba un mensaje distinto para cada una de ellas. El algoritmo repetirá el proceso hasta que la opción elegida sea válida.

```
algoritmo menu2
variables
  entero opcion
principio
  repetir
    escribirln("1. Primera opción")
    escribirln("2. Segunda opción")
    escribirln("3. Tercera opción")
    escribirln
    escribir("Elige una opción: ")
    leer(opción)
    según opción sea
      caso 1: escribir("Has elegido la opción 1ª")
      caso 2: escribir("Has elegido la opción 2ª")
      caso 3: escribir("Has elegido la opción 3ª")
    fsegún
  mientras que (opción !=1) AND (opción != 2) AND (opción
!= 3)
fin
```

4.3 Esquema de repetición incondicional.

Repite la ejecución de una secuencia de acciones **un número de veces determinado** de antemano.

Sintaxis:

```
para variableIndice = valorInicial hasta valorFinal hacer
        acciones
fpara
```

La variable `variableIndice`, llamada **variable de control** del ciclo, ha de ser de tipo escalar enumerable y su valor no puede modificarse durante la ejecución del mismo (dentro del cuerpo del ciclo).

Funcionamiento:

Cuando se ejecuta por primera vez la instrucción **para**... la `variableIndice` toma el valor `valorInicial`. El valor de la `variableIndice` se compara con `valorFinal`, si es menor o igual se ejecutará a continuación la secuencia de acciones, en caso contrario se continúa en la instrucción siguiente al **fpara**.

Si se ha ejecutado el cuerpo del ciclo, al alcanzar el **fpara** se vuelve de nuevo a la línea **para**... Ahora se incrementa automáticamente la variable índice y se repite de nuevo el proceso explicado anteriormente.

De esta forma, la repetición acabará cuando la `variableIndice` supere el `valorFinal.`

Notar que si inicialmente `valorInicial` > `valorFinal`, el cuerpo del ciclo no se ejecuta ninguna vez.

Una variante de este esquema es:

```
para varInd = valorInic descendiendo hasta valorFin hacer
        acciones

fpara
```

Notar que si inicialmente `valInic` < `valFin`, el cuerpo del ciclo no se ejecuta ninguna vez.

Semántica:

1) Hace `variableIndice` igual a `valorInicial`

2) Si el valor de `variableIndice` es menor o igual que `valorFinal` (o mayor o igual en el caso de usar la cláusula descendiendo) ejecuta el cuerpo del ciclo , en caso contrario termina el bucle.

3) Hacer `variableIndice` igual a Sucesor(`variableIndice`) Iincrementar) (o a Predecesor(`variableIndice`) (decrementar) en el caso de usar la cláusula descendiendo) y volver al punto anterior.

Ejemplo:

1- Factorial.

```
Interfaz:
 Entrada entero n
 Salida entero fact
Efecto:
 Condiciones de entrada: n>=0
 Efecto producido: fact=n!
```

```
algoritmo factorial
variables
 entero  n, fact, i
principio
 leer(n)
 fact=1
 para i=1 hasta n hacer
      fact= fact*i
 fpara
 escribir(fact)
fin
```

5. Tratamiento secuencial. Introducción

Vamos a introducir de una manera intuitiva los conceptos de acceso secuencial y acceso directo.

Un programa puede recibir sus datos de entrada desde diferentes dispositivos físicos: teclado, ficheros de diverso tipo y en diferentes soportes, líneas de comunicación, etc. En ocasiones, el acceso será directo y en otras secuencial, algunas veces impuesto por el sistema físico que proporciona los datos.

Hablamos de acceso (o proceso) secuencial cuando para acceder (o procesar) un dato es necesario haber accedido (o procesado) los anteriores. Por el contrario, hablamos de acceso directo cuando para acceder a un dato NO es necesario haber accedido a los anteriores.

La estrategia a seguir por los programas que necesitan procesar un conjunto de datos, todos del mismo tipo, organizados secuencialmente consiste en tomar un dato, procesarlo, tomar el siguiente, procesarlo,...,hasta acabar con los datos de entrada. Fíjate en dos detalles: el primero es que el programa va tomando datos de forma secuencial, uno tras otro, sin tener la capacidad de tomar un dato concreto en cada lectura. La segunda cuestión es que, si no se conoce de antemano el número de elementos a procesar, deberá detectar el final de alguna manera. En este caso, se utiliza un dato especial, no procesable, que marca el final de los datos.

La clave del diseño de programas que realizan un tratamiento secuencial radica en diseñar correctamente la o las estructuras repetitivas apropiadas poniendo especial énfasis en la terminación de los mismos, dependiendo del tipo de esquema necesario: recorrido o búsqueda. Lo tratamos a continuación.

5.2. Esquemas de recorrido y busqueda

Veamos ahora unos problemas sencillos que permiten esquematizar los modelos fundamentales de algoritmos de proceso secuencial.

Ejemplo

1) Dado un texto terminado en punto,

a) calcular el número de caracteres que tiene.
b) contar el número de apariciones de la letra 'a'.
c) comprobar si está la letra 'w'.

2) Dada una secuencia de enteros no negativos acabada en cero,

a) sumar todos los valores de la secuencia.
b) contar el número de unos.
c) indicar si todos los términos son pares.

Si comparamos las soluciones de cada apartado de uno de estos ejemplos con las correspondientes del otro observaremos ciertas similitudes:

- para resolver el apartado a) del primer ejemplo hay que ir contando los caracteres según se van leyendo del primero hasta el último, el anterior al punto, (decimos que recorremos la secuencia de principio a fin). De forma similar, en el apartad a) del segundo ejemplo hay que ir acumulando cada número a medida que se vayan leyendo, del primero al último (el anterior al cero).

- para resolver el apartado b) también recorremos toda la secuencia aunque contamos únicamente las aes en el primer caso y los unos en el segundo.

- por el contrario, en el último apartado de ambos ejemplos la estrategia correcta debe considerar que si se encuentra el objeto deseado, la letra 'w' en el primer caso, un número impar en el segundo, ya no se ha de continuar la búsqueda, no es necesario seguir leyendo elementos, de modo que sólo se agota la entrada de los datos (se llega al final de la secuencia) si dicho elemento no aparece.

Por tanto, según la naturaleza del problema, tenemos dos estrategias diferenciadas que generan esquemas algorítmicos distintos. Estos esquemas, que pueden aplicarse a un gran número de problemas, se denominan, respectivamente, **esquema de recorrido** y **esquema de búsqueda.**

Recorrido

La aplicación de un esquema u otro viene determinada por la necesidad, o no, de recorrer toda la secuencia. Aplicaremos el esquema de *recorrido* si la estrategia adecuada para el problema exige acceder a todos los elementos de la secuencia.

Resolución de los ejemplos

1) Contar las apariciones de la letra 'a' en un texto acabado en punto.

La manera intuitiva de conseguirlo consiste en plantear una iteración donde en cada ciclo se tome un carácter del texto, se compruebe si es una 'a', y en caso afirmativo se actualice num_a (variable contador) .

```
Especificación:
  Interfaz:
   Entrada secuencial de caracteres acabada con un punto
   Salida entero num_a
  Efecto:
   Condiciones de entrada: { }
   Efecto producido: num_a es el número de aes de la
secuencia de entrada
algoritmo contar_a
variables
  entero num_a
  carácter c
principio
  num_a=0
  leer(c)
  mientras que (c≠'.') hacer
    si c=='a' entonces
      num_a= num_a+1
    fsi
    leer(c)
  fmq
```

```
  escribir(num_a)
fin
```

2) Sumar los términos de una secuencia de enteros positivos marcada con el cero.

La misma discusión del ejemplo anterior sirve aquí. Recorremos la secuencia y almacenamos en la variable suma la suma (variable acumulador) de los elementos leidos hasta el momento.

```
Especificación:
  Interfaz:
   Entrada secuencial de enteros que acaba con un cero
   Salida entero suma
  Efecto:
   Condiciones de entrada: { }
   Efecto producido:suma es la suma de todos los valores
de la secuencia de enteros de entrada
algoritmo sumar
variables
  entero suma, n
principio
  suma=0
  leer(n)
  mientras que n≠0 hacer
    suma=suma+n
    leer(n)
  fmq
  escribir(suma)
fin
```

Podemos abstraer los elementos básicos de un algoritmo de **recorrido** a partir de estos dos ejemplos. Por tanto, en un caso general,

```
Especificación:
Entrada secuencial de tipo de dato que acaba en un
elemento especial (marca)
{ }
{tratada toda la secuencia de entrada menos la marca}
```

la solución seguirá el siguiente esquema:

```
leer primer elemento
tratamiento inicial
mientras que elemento ≠ marca hacer
  tratar elemento
  leer siguiente elemento
fmq
tratamiento final
```

donde hemos añadido un *tratamiento final* que en los ejemplos anteriores no ha hecho falta pero que en algún otro problema puede ser necesario.

La característica básica en el esquema de **recorrido** es que **debe tratarse toda la secuencia**, necesariamente se deben leer y procesar todos los elementos para dar la solución al problema, por tanto, la condición de permanencia en el ciclo deja de cumplirse al alcanzar la marca.

Búsqueda

Aplicaremos el esquema de búsqueda cuando se trata de encontrar el primer elemento de una secuencia de datos de entrada que verifique una determinada propiedad.

Ejemplo

1) Determinar si un texto acabado en punto contiene la letra 'w'.

Es suficiente encontrar la primera aparición del carácter 'w' que buscamos. Para ello, basta escribir un bucle que vaya leyendo caracteres y se detenga si el elemento en curso coincide con el buscado.

Especificación:

```
  Interfaz:
   Entrada secuencial de caracteres que acaba con un
punto
   Salida: booleano encontrado
  Efecto:
   Condiciones de entrada: { }
```

```
   Efecto producido:{encontrado = verdad si 'w' aparece
en el texto
    encontrado = falso en caso contrario}
algoritmo buscar_w
variables
  booleano encontrado
  carácter c
principio
  encontrado=falso
  leer(c)
  mientras que (c≠'.') AND (NOT encontrado) hacer
    si c=='w' entonces
      encontrado=verdad
    si_no
      leer(c)
    fsi
  fmq
  escribir(encontrado)
fin
```

Este ejemplo presenta un caso de búsqueda pura en el que sólo se determina la existencia, dentro de la secuencia, de un elemento que verifique una propiedad P y no hay ningún tratamiento de los elementos leidos.

Basándonos en lo visto, especificamos y diseñamos el esquema de búsqueda en una secuencia de un tipo de dato que cumpla una propiedad *P*.

Especificación:

```
  Entrada secuencial de tipo de dato que acaba en marca
  Salida booleano encontrado
  { }
    búsqueda
  {encontrado = verdad si existe elemento Œ S que cumple
la propiedad P}
    encontrado = falso si no existe}
```

Esquema:

```
  encontrado=falso
  leer primer elemento
  tratamiento inicial
  mientras que ((elemento ≠ marca) AND (NOT encontrado))
hacer
    si elemento cumple P entonces
      encontrado=verdad
      tratamiento de elemento
    si_no
      leer siguiente elemento
    fsi
  fmq
  tratamiento final
```

La característica básica en el esquema de **búsqueda** es que **NO necesariamente deben leerse todos los elementos de la secuencia de entrada para dar la solución al problema**, por tanto, la condición de permanencia en el ciclo deja de cumplirse al alcanzar la marca o al encontrar el elemento buscado, de ahí que la condición sea compuesta.

Esquema mixto. Composición de esquemas.

A partir de los esquemas básicos de recorrido y búsqueda podemos construir algoritmos que consideren variantes más o menos complicadas de uno de ellos o combinaciones de los dos. Veamos los siguientes ejemplos:

1) Contar el número de caracteres siguientes a la primera aparición de la letra 'a' en un texto acabado en punto.

Esto nos sugiere realizar primero una búsqueda de la letra 'a' contando a continuación los caracteres leidos hasta encontrar el punto. Se trata por tanto de realizar primero una búsqueda y a continuación un recorrido.

```
Especificación:
  Interfaz:
   Entrada secuencial de caracteres que acaba con un
punto
   Salida entero num_car
  Efecto:
    Condiciones de entrada: {}
  Efecto producido: {num_car = número de caracteres del
texto de entrada después de la aparición de la primera
'a'}
algoritmo búsqueda_con_conteo
variables
  entero num_car
  carácter c
principio
  num_car=0
  leer(c)
  mientras que (c≠'.') AND (c≠'a') hacer    {búsqueda}
    leer(c)
  fmq
  si c≠'.' entonces
    leer(c)
    mientras que (c≠'.') hacer   {recorrido}
      num_car= num_car+1
      leer(c)
    fmq
  fsi
  escribir(num_car)
fin
```

2) Algoritmo que tome como entrada un texto acabado en punto y escriba ese mismo texto pero sin los espacios en blanco iniciales.

La estrategia adecuada consiste en **buscar** la primera letra del texto y, a partir de ahí, escribir cada carácter leido, hasta alcanzar el punto.

Especificación:

Interfaz:

Entrada: secuencial de caracteres que acaba con un punto

Salida: secuencial de caracteres que acaba con un punto

Efecto:

Condiciones de entrada: { }

Efecto producido: {Escribe el mismo texto de entrada pero sin los espacios en blanco iniciales}

```
algoritmo copia
variables
  carácter c
principio
  { Búsqueda de la primera letra }
  leer(c)
  mientras que  c == ' '   hacer
    leer(c)
  fmq
  { Recorrido }
  mientras que c≠'.' hacer
    escribir(c)
    leer(c)
  fmq
fin
```

Ejercicios propuestos

Estructura secuencial:

Especificar y escribir los siguientes algoritmos.

1. **algoritmo** cuadrado
   { Leer un número y calcular su cuadrado }

2. **algoritmo** hipotenusa triángulo
   { Determinar la hipotenusa de un triángulo rectángulo conocidas las longitudes de sus catetos }

3. **algoritmo** inversión
   { Determinar la cantidad anual producida por la inversión de dinero a interés fijo, conocidos el capital invertido y el interés del rendimiento }

4. **algoritmo** funciones trigonométricas
   { Determinar el valor de las funciones trigonométricas (seno, coseno y tangente) de un valor cualquiera }

5. **algoritmo** aceleración
   { Conocida la fuerza que actúa sobre una partícula y su masa calcular su aceleración }

6. **algoritmo** volumen cilindro
   { Conocido el radio de la base y la altura de un cilindro, calcular su volumen }

7. **algoritmo** resto cociente
   { Dadas dos variables n y m (n>m) de tipo entero, escribir un algoritmo que permita calcular el resto y el cociente de la división entera }

8. **algoritmo** Fahrenheit
   { Transformar una temperatura dada en grados centígrados a su equivalente en fahrenheit. La fórmula de conversión es: Fahrenheit = 9/5 Centígrado+ 32 }

9. **algoritmo** posición letra mayúscula
   { Leer una letra mayúscula del alfabeto y escribir su posición (en el alfabeto) }

10. **algoritmo** minúscula mayúscula
    { Leer una letra minúscula y escribir la mayúscula correspondiente }

11. **algoritmo** convertir horas minutos segundos
    { Conversión de una cantidad positiva de segundos a su equivalente en horas, minutos y segundos }

12. **algoritmo** cambio óptimo
    { Programa que obtenga el cambio óptimo (mínimo número de monedas posible) de una cantidad de euros, en monedas de 50, 20, 10, 5, 2 y 1 céntimo }

13. **algoritmo** invertir entero
    { Leer un número entero de dos cifras y presentarlo con sus cifras invertidas }

14. **algoritmo** distancias
    { Convertir una medida dada en pies a sus equivalentes en: a) yardas: b) pulgadas; c) centímetros, y d) metros (1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 m = 100 cm.) }

15. **algoritmo** intercambio valores
    { Dadas dos variables X e Y, escribir un algoritmo que permita intercambiar los contenidos de las dos variables }

16. **algoritmo** descodificar fecha
    { Descodificar una fecha expresada como un entero de 6 dígitos. El año está representado en las unidades y decenas, el mes en las centenas y millares y el día en las decenas y centenas de millar }

Estructura alternativa

Especificar y escribir los siguientes algoritmos.

1. **algoritmo** par impar
   { Dado un número entero deducir si es par o impar }

2. **algoritmo** divisibles
   { Dados dos números enteros positivos determinar si son divisibles }

3. **algoritmo** llamada telefónica
   { Determinar el total a pagar por una llamada telefónica de acuerdo a las premisas:
   - Toda llamada que dure menos de tres minutos tiene un coste de 15 céntimos.
   - Cada minuto adicional a partir de los tres primeros es un paso de contador y cuesta 10 céntimos. }

4. **algoritmo** detectar carácter
   { Dado un carácter detectar si es a) letra minúscula, b) letra mayúscula, c) dígito. d) otro carácter }

5. **algoritmo** ordinal mes
   { Dado el número ordinal de un mes escribe el nombre del mes y el número de días que tiene suponiendo que el año no es bisiesto }

6. **algoritmo** nómina semanal
   { Se quiere pagar la nómina semanal de los empleados de una empresa cuyo trabajo se paga por horas de la siguiente forma:
   - Si trabajan 40 horas o menos se paga la hora a una cantidad determinada, que se introducirá por el teclado: *precio_hora*.
   - Cada hora por encima de las 40 horas semanales se paga multiplicando por 1.7 la cantidad base.

   Las retenciones que se aplican son las siguientes:
   - Si Sueldo <= 1000 € al 10 %.
   - Si 1000 €< Sueldo <= 2000 € al 20 %.
   - Si Sueldo > 2000 € al 25 %.}

7. **algoritmo** factura
   { Calcular el importe de la factura de electricidad de un abonado sabiendo que se paga:
   - 10 euros de gastos fijos, aunque no haya consumo.
   - el consumo según una tarifa por tramos:
     - 6 céntimos por Kw/h para los primeros 100 Kw/h
     - 3,5 céntimos por Kw/h para los siguientes 200 Kw/h
     - 2,5 céntimos por Kw/h para los siguientes Kw/h }

8. **algoritmo** descuento
   { En un comercio se realiza descuento en las compras en función del importe total. Calcular el descuento según los siguiente criterios: 1) Si el importe < 100€ no se aplica descuento, 2) si está entre 100 <= importe <= 400 el descuento es del 10%, 3) si el importe > 400 el descuento es del 15%. }

9. **algoritmo** precio billete
   { Determinar el precio del billete de ida y vuelta en ferrocarril, conociendo la distancia a recorrer y sabiendo que si el número de días de estancia es superior a 7 y la distancia superior a 800 km. el billete tiene una reducción del 30 por 100. El precio por km. es de 7,5 céntimos }

10. **algoritmo** año bisiesto
    { Escribir un programa que determine si un año es bisiesto. Un año es bisiesto si es múltiplo de 4 pero no es múltiplo de 100, salvo si es múltiplo de 400 (2000 es bisiesto, 1800 no lo es) }

11. **algoritmo** calificaciones
    { Escribir un programa que tome como entrada la nota de un examen (real entre 0 y 10) y muestre la calificación textual (nota <5, “suspenso”; 5<=nota<7, “aprobado”; 7<= nota <9, “notable”; nota >=9, “sobresaliente”) }

Estructura repetitiva

Especificar y escribir los siguientes algoritmos:

1. **algoritmo** tabla de multiplicar
   { Dado un número n entero positivo, construir su tabla de multiplicar }

2. **algoritmo** potencia
   { Dados dos números enteros x e y , con y no negativo, diseñar un algoritmo que calcule el valor de x}

3. **algoritmo** impares
   { Escribir un algoritmo que calcule la suma de los números impares menores que un entero positivo dado por el usuario }

4. **algoritmo** divisores
   { Dado un número entero mayor que cero calcular sus divisores }

5. **algoritmo** número perfecto
   { Un *número perfecto* es un entero positivo, que es igual a la suma de todos los enteros positivos (excluidos el mismo) que son divisores del número. El primer número perfecto es 6, ya que los divisores son 1, 2, 3 y 1 + 2 + 3 = 6 .
   Escribir un programa que encuentre los nº perfectos menores que 1000 }

6. Especificar y diseñar un algoritmo que muestre por pantalla los números perfectos comprendidos entre n y m (n y m son números enteros leídos por el teclado).
   Nota: Un número es perfecto si es igual a la suma de sus divisores

7. **algoritmo** número primo
   { Diseñar un algoritmo para determinar si un entero es un número primo }

8. **algoritmo** suma serie
   { Dado un número entero n calcular la suma
   suma =1 + 2+3 + ... + n }

9. **algoritmo** número términos

{ Obtener el número de términos de la serie que es necesario tomar para satisfacer la desigualdad:

1 + 1/2 + . . . + 1/n> límite }

}

10. **algoritmo** dividir

{ Dados dos números enteros realizar un programa que calcule el cociente y el resto sin utilizar las funciones mod y div }

11. **algoritmo** número dígitos

{ Dado un número entero determinar el número de dígitos que tiene }

12. Especificar y diseñar un algoritmo que, dado un entero positivo leído desde el teclado, determine si la suma de sus cifras es un múltiplo de 11.

13. **algoritmo** Fibonacci

{ Calcular el valor de la función de Fibonacci de un número dado n >1

F(n) = F(n-1) + F(n-2) con F(0) = 0, F(1) = 1 }

14. **algoritmo** mcd

{ Calcular el máximo común divisor de dos números enteros positivos según el siguiente algoritmo:

m.c.d.(a, 0) = a

m.c.d.(a,b) = m.c.d.(b, resto(a/b)) }

15. Especifica y diseña un algoritmo que dado un entero positivo escriba el número que se obtiene sumando los dígitos que lo componen. Mientras este resultado sea mayor o igual que 10, se repetirá el cálculo para este valor y así sucesivamente.

Ejemplos:

Entrada	Salida por pantalla
7	7
13	4
492	15 6
9767	29 11 2

16. Practicando con bucles anidados. Escribe algoritmos para conseguir las siguientes salidas por pantalla:

    NOTA: Escribe un único carácter de cada vez (escribir('*') o escribir('+')). Utiliza la instrucción *escribirln* para saltar de línea

a) Una línea con 5 asteriscos: *****

b) 5 líneas de 5 asteriscos

```
*****
*****
*****
*****
*****
```

c) 5 líneas de asteriscos en número decreciente

```
*****
****
***
**
*
```

d) 5 líneas como las que se muestran a continuación

```
****+
***++
**+++
*++++
+++++
```

Tratamiento secuencial

1. Sea una entrada secuencial de caracteres acabada en un punto. Especifica y diseña algoritmos que resuelvan:

   a) Contar el número de caracteres.
   b) Contar el número de espacios en blanco.
   c) Comprobar si está la letra 'p'.
   d) Número de caracteres anteriores a la primera aparición de la letra 'a'.
   e) Determinar si en el texto aparecen más letras 'd' que 'c'.
   f) Calcular el número de caracteres después de la primera aparición de la letra 'c'.
   g) Determinar si un texto contiene al menos 5 apariciones de la letra 'a'.
   h) Calcular la longitud de la primera palabra de un texto.

2. Sea una entrada secuencial de enteros no negativos acabada en -1. Especifica y diseña algoritmos que resuelvan:

   a) Contar el número de enteros.
   b) Calcular el máximo de la secuencia (si secuencia vacía el máximo es 0).
   c) Calcular la media.
   d) Indicar si todos los términos son pares.

3. Dada una secuencia de números enteros positivos acabada en -1, se dice que está en espiral si cada término se calcula multiplicando el anterior por un número entero llamado razón, es decir , siendo la razón. Por ejemplo: la secuencia 2 4 8 16 32 64 128 -1 está en espiral. Se pide especificar y realizar un algoritmo que determine si una secuencia está o no en espiral.

4. Especificar y realizar un algoritmo que calcule la longitud media de las palabras de un texto acabado en un punto. Por simplicidad se supondrá que las palabras se encuentran separadas unas de otras únicamente por espacios en blanco. En la secuencia no aparecen comas ni otros signos de puntuación, salvo el punto final.

5. Una secuencia de números enteros positivos se dice *melchoriforme* si uno de sus elementos es *rubio*. Un elemento de una secuencia se dice *rubio* si es la suma del resto de elementos de la secuencia. Escribir un programa que diga si una secuencia es *melchoriforme*.

6. Se desea analizar la diversidad de palabras que componen un texto, para lo que entre otras cosas se desea conocer cuántas palabras comienzan por “qu”, como “que” o “quien”. El texto a analizar acaba en un punto (‘.’) y está compuesto por palabras separadas únicamente por espacios en blanco (puede haber varios espacios en blanco entre las palabras y no existen signos de puntuación en el texto, salvo el ‘.’ final).

a) Especifica y diseña un algoritmo que indique el número de palabras que comienzan por “qu”.

b) Especifica y diseña un algoritmo que indique el número máximo de palabras consecutivas que comienzan por las letras “qu”.

7. Sea un texto acabado en un punto y compuesta por palabras separadas únicamente por espacios en blanco (puede haber varios espacios en blanco entre las palabras y no existen signos de puntuación en el texto, salvo el punto final). El texto es el de un telegrama. Especificar y diseñar un algoritmo que calcule el coste de enviar el telegrama.

Los precios de las palabras según su longitud son los siguientes: 1 letra 5 céntimos, 2 letras 7 céntimos, 3 letras 11 céntimos, 4 letras 17 céntimos, 5 letras 25 céntimos y más de 5 letras, 30 céntimos.

8. Sea un texto acabado en un punto y compuesto por palabras separadas únicamente por espacios en blanco (puede haber varios espacios en blanco entre las palabras y no existen signos de puntuación en la secuencia, salvo el ‘.’ final). Especificar y diseñar un algoritmo que dado el texto anterior escriba otro texto formado por la última letra de cada palabra del texto de entrada. Por ejemplo, si en la entrada es *Hoy tenemos viento de poniente* , la salida será *ysoee*.

9. Especifica y diseña un algoritmo que lea un texto acabado en '*' que contiene un texto con errores tipográficos. Concretamente, en el texto faltan los puntos y el número de espacios en blanco entre palabras es variado. El algoritmo escribirá e texto corregido, es decir, al que se le han incluido los puntos y con un único espacio en blanco entre palabras. El programa pondrá puntos delante de aquellas palabras que empiecen por mayúscula así como al final del texto.

Ejm:
Entrada: Este es un texto con errores Faltan los puntos Hay demasiados espacios en blanco *

Salida: Este es un texto con errores. Faltan los puntos. Hay demasiados espacios en blanco.*

10. Dada una secuencia de números enteros acabada en 0. Especificar y diseñar un algoritmo que compruebe si los elementos que ocupan las posiciones múltiplos de cinco dentro de la secuencia, suman igual que el resto de los elementos. Si el número de elementos de la secuencia no es múltiplo de cinco, los elementos comprendidos entre el último que ocupa una posición múltiplo de cinco y el final de la secuencia no son tenidos en cuenta.

Ejemplo:

Dada la secuencia 1 2 5 5 22 4 7 8 5 15 8 7 0, los elementos que no ocupan posiciones múltiplos de cinco suman 1+2+5+5+4+7+8+5=37 y los elementos que ocupan posiciones múltiplos de cinco suman 22+15=37 luego la propiedad se cumple, notar que como el número de elementos de la secuencia no es múltiplo de cinco los valores 8 y 7 no son tenidos en cuenta.

11. Especifica y diseña un algoritmo que lea texto escrito en minúsculas y acabado en punto y muestre por pantalla el número de palabras que comienzan y terminan con el mismo carácter.

Ejemplo:

Dada:

y vimos que allá estaban ellos solos .

Escribiría 3. (y, allá solos)

12. Especifica y diseña un algoritmo que lea un texto escrito en minúsculas y acabado en punto, y escriba un texto que contenga las mismas palabras que el texto de entrada pero separando las palabras encadenadas por guiones. Decimos que dos palabras están encadenadas si la primera acaba en el mismo carácter con la que comienza la siguiente.

Ejemplo:

Entrada:

el día anterior resolvimos el mismo problema acertadamente .

Salida:

El *día-anterior-resolvimos el mismo problema-acertadamente.*

13. Se parte de un texto que finaliza con un punto y que sólo contiene letras (mayúsculas y minúsculas) y espacios en blanco. Especificar y diseñar un algoritmo que calcule el porcentaje de palabras del texto que contienen la letra 'b' (o la letra 'B').

14. Especifica y diseña un algoritmo que, dado un texto que finaliza con un punto, decida si alguna palabra de dicho texto empieza y acaba por la letra ‘a’.

15. Dada una entrada secuencial de caracteres acabada con la marca ’#’ que contiene una expresión aritmética, especifica y diseña un algoritmo que indique si los paréntesis están correctamente dispuestos. Es correcto si siempre el número de paréntesis abiertos es mayor o igual que el de paréntesis cerrados y, al acabar, el número de paréntesis abiertos es igual que el de paréntesis cerrados.

16. Dada una secuencia de números enteros positivos acabada en 0, calcular la longitud del mayor tramo de valores ordenados de forma decreciente.