Estructura general de un programa

INTRODUCCION

Un programa puede considerarse como una secuencia de acciones (instrucciones) que manipulan un conjunto de objetos (datos). Contendrá por tanto dos bloques para la descripción de los dos aspectos citados:

- **Bloque de declaraciones**: en él se especifican todos los objetos que utiliza el programa (constantes, variables, tablas, registros, archivos, etc.).
- **Bloque de instrucciones**: constituido por el conjunto de operaciones que se han de realizar para la obtención de los resultados deseados.

Partes principales de un programa

Dentro del bloque de instrucciones de un programa podemos diferenciar tres partes fundamentales. En algunos casos, estas tres partes están perfectamente delimitadas, pero en la mayoría sus instrucciones quedan entremezcladas a lo largo del programa, si bien mantienen una cierta localización geométrica impuesta por la propia naturaleza de las mismas.

- **Entrada de datos**: la constituyen todas aquellas instrucciones que toman datos de un dispositivo externo, almacenándolos en la memoria central para que puedan ser procesados.
- **Proceso o algoritmo**: está formado por las instrucciones que modifican los objetos a partir de su estado inicial hasta el estado final, dejando éstos disponibles en la memoria central.
- **Salida de resultados**: conjunto de instrucciones que toman los datos finales de la memoria central y los envían a los dispositivos externos.

CLASIFICACION DE LAS INSTRUCCIONES

Una instrucción se caracteriza por un estado inicial y final del entorno. El estado final de una instrucción coincide con el inicial de la siguiente. No siempre una instrucción modifica el entorno, pues su cometido puede limitarse a una mera observación del mismo o a un cambio en el orden de ejecución de otras. Las instrucciones pueden ser:

Instrucciones de declaración

Se utilizan en aquellos lenguajes de programación que no tienen declaración explícita de los objetos. Su misión consiste en indicar al procesador que reserve espacio en la memoria para un objeto del programa, indicando asimismo su nombre, tipo y características.

Instrucciones primitivas

Son aquellas que ejecuta el procesador de modo inmediato. Las principales son asignación, entrada y salida.

- **Instrucción de asignación**: consiste en calcular el valor de una expresión y almacenarlo en una variable. En algún lenguaje es preciso calcular previamente el resultado de la expresión, pues la instrucción de asignación sólo permite el movimiento de un valor simple.
- **Instrucción de entrada**: toma un dato de un dispositivo de entrada y lo almacena en un objeto. En algún lenguaje, los datos de entrada no provienen de un dispositivo externo, sino que han sido colocados previamente en el mismo programa.
- **Instrucción de salida**: toma el valor de una expresión u objeto y lo lleva a un dispositivo externo.

Instrucciones compuestas

Son aquellas que el procesador no puede ejecutar directamente, sino que realiza una llamada a un subprograma, subrutina o párrafo.

Instrucciones de control

Son aquellas de controlar la ejecución de otras instrucciones.

- **Instrucción alternativa**: controla la ejecución de unas u otras instrucciones según una condición. Puede ser simple o doble (SI y SINO).
- **Instrucción de salto incondicional**: altera la secuencia normal de ejecución de un programa, continuando la misma en la línea indicada en la propia instrucción (IR A).
- **Instrucción de salto condicional**: altera la secuencia normal de ejecución de un programa únicamente en el caso de cumplimiento de una condición asociada a la propia instrucción (SI ... IR A).
- **Instrucción repetitiva**: hace que se repitan una o varias instrucciones un número determinado o indeterminado de veces (PARA, MIENTRAS, HASTA e ITERAR).

ELEMENTOS AUXILIARES DE UN PROGRAMA

Son variables que realizan funciones específicas dentro de un programa, y por su gran utilidad, frecuencia de uso y peculiaridades, conviene hacer un estudio separado de las mismas. Las más importantes son:

Contadores

Un contador es un campo de memoria cuyo valor se incrementa en una cantidad fija, positiva o negativa, generalmente asociado a un bucle. Toma un valor inicial antes de comenzar su función, y cada vez que se realiza el suceso, incrementa su valor. Se utiliza en los siguientes casos:

- Para contabilizar el número de veces que es necesario repetir una acción (variable de control de un bucle).
- Para contar un suceso particular solicitado por el enunciado del problema (asociado a un bucle o independientemente).

Acumuladores

Un acumulador es un campo de memoria cuyo valor se incrementa sucesivas veces en cantidades variables. Se utiliza en aquellos casos en que se desea obtener el total acumulado de un conjunto de cantidades, siendo preciso inicializarlo con el valor 0. También en las situaciones en que hay que obtener un total como producto de distintas cantidades se utiliza un acumulador, debiéndose inicializar con el valor 1.

Interruptores (switches)

Un interruptor es un campo de memoria que puede tomar dos valores exclusivos (0 y 1, -1 y 1, FALSO y CIERTO, etc.). Se utiliza para:

- Recordar en un determinado punto de un programa la ocurrencia o no de un suceso anterior, para salir de un bucle o para decidir en una instrucción alternativa qué acción realizar.
- Para hacer que dos acciones diferentes se ejecuten alternativamente dentro de un bucle.

TIPOS DE PROGRAMAS

Un programa, por lo general, estará compuesto por una secuencia de acciones, algunas de las cuales serán alternativas o repetitivas. En determinados programas sencillos, no se da esta mezcla de acciones, en cuyo caso los

podemos clasificar como sigue:

- **Programas lineales**: consisten en una secuencia de acciones primitivas (su ejecución es lineal en el orden en que han sido escritas).
- **Programas alternativos**: consisten en el anidamiento de acciones alternativas (las tablas de decisión se realizan mediante programas alternativos).
- **Programas cíclicos**: son aquellos en los que un conjunto de acciones se repiten un número determinado o indeterminado de veces (un programa de este tipo se denomina bucle).

Otra clasificación relativa a la aplicación desarrollada por el programa es:

- **Programas de gestión**: se caracterizan por el manejo de gran cantidad de datos con pocos cálculos (resuelven problemas de gestión).
- **Programas técnicos-científicos**: al contrario que los anteriores, realizan gran cantidad de cálculos con pocos datos (revuelven problemas matemáticos, físicos, etc.).
- **Programas de diseño (CAD)**: se caracterizan por la utilización de técnicas gráficas para resolver problemas de diseño.
- **Programas de simulación**: intentan reflejar una situación real, para facilitar su estudio.
- **Programas educativos (EAO)**: utilizan las ventajas del ordenador para la docencia.
- **Programas de inteligencia artificial**: se utilizan para simular el razonamiento humano.

LENGUAJES DE PROGRAMACION

Un lenguaje de programación es una notación para escribir programas, es decir, para describir algoritmos dirigidos al computador. Un lenguaje viene dado por una gramática o conjunto de reglas que se aplican a un alfabeto. El primer lenguaje de programación que se utilizó fue el lenguaje máquina, el único que entiende directamente el computador, cuyo alfabeto es el binario, formado por los símbolos 0 y 1. El lenguaje ensamblador, resultó de la evolución del lenguaje máquina, al sustituir las cadenas de símbolos binarios por nemotécnicos. Posteriormente surgieron los lenguajes de alto nivel, cuya evolución es:

- 1955: FORTRAN.
- 1960: LISP, ALGOL, COBOL.
- 1965: APL, SNOBOL, PL/1, BASIC.
- 1970: PROLOG, ALGOL68, SIMULA67.
- 1975: C, PASCAL.
- 1980: MODULA-2, LIS, EUCLID.
- 1985: ADA.

Los lenguajes de programación pueden clasificarse de la siguiente manera:

Según su parecido con el lenguaje natural

- **Bajo nivel**: lenguajes máquina y ensambladores.
- **Alto nivel**: todos los demás.

Según la estructura de los programas

- **Convencionales o línea a línea**: ensambladores, FORTRAN, BASIC, COBOL, etc.
- **Estructurados**: Algol, PL/I, Pascal, Ada, COBOL estructurado, etc.

Según la realización de los programas

- **Funcionales**: Lisp, Prolog, APL, etc.

- **Imperativos**: la mayoría.

Según el tipo de proceso

- **Interactivos o conversacionales**: BASIC, Pascal, APL, etc.
- **Orientados al proceso por lotes (batch)**: COBOL, FORTRAN, PL/I, etc.

Notación pseudocodificada

INTRODUCCION

Diremos que una notación es un pseudocódigo si mediante ella podemos describir la solución de un problema en forma de algoritmo dirigido al computador, utilizando palabras y frases del lenguaje natural sujetas a unas determinadas reglas. Todo pseudocódigo debe posibilitar la descripción de:

- Instrucciones de entrada/salida.
- Instrucciones de proceso.
- Sentencias de control del flujo de ejecución.
- Acciones compuestas, que hay que refinar posteriormente.

Asimismo, tendrá la posibilidad de describir datos, tipos de datos, constantes, variables, expresiones, archivos y cualquier otro objeto que sea manipulado por el programa.

Acciones simples

Las acciones simples, también denominadas instrucciones primitivas, son aquellas que son ejecutadas de forma inmediata por el procesador.

- **Asignación**: almacena en una variable el resultado de evaluar una expresión.
- **Entrada**: Toma un dato del dispositivo estándar de entrada y lo almacena en una variable. Si se leen varias variables, se pueden colocar éstas en una misma instrucción separándolas por comas.
- **Salida**: imprime en el dispositivo estándar de salida el resultado de evaluar una expresión. Al igual que en la lectura, se pueden imprimir varias expresiones en una sola instrucción de escritura.

Sentencias de control

También se denominan sentencias estructuradas y controlan el flujo de ejecución de otras instrucciones.

- **Secuencia**: se ejecutan las instrucciones en el mismo orden en que aparecen escritas. Utilizamos el punto y coma como separador de instrucciones que están en la misma línea.
- **Alternativa**: en esta instrucción la condición es una expresión booleana, si su evaluación produce el resultado CIERTO se ejecutarán las instrucciones asociadas, y en caso contrario otras.
- **Repeticiones o bucles**: en todo bucle hay una o varias acciones que se han de repetir y una condición que determina el número de repeticiones de las mismas. Es fundamental que el valor de la condición sea afectado por las acciones para asegurar la terminación del bucle en algún momento.

Acciones compuestas

Una acción compuesta es aquella que ha de ser realizada dentro del algoritmo, pero que aún no está resuelta en

términos de acciones simples y sentencias de control.

Comentarios

Son líneas explicativas cuyo objetivo es facilitar la comprensión del programa a quien lo lea. Estas líneas serán ignoradas por el procesador cuando ejecute el programa. Los comentarios se utilizan para aclarar:

- El significado o cometido de un objeto del programa.
- El objetivo de un bloque de instrucciones.
- La utilización de una determinada instrucción.
- Siempre que sea necesario aclarar algún aspecto del programa.

Programa

Un programa es la solución final de un problema. En esta notación consiste en la descripción de los objetos (entorno) y de las instrucciones (algoritmo). Tendr una cabecera con el nombre del programa y dos bloques precedidos por las palabras "entorno" y "algoritmo".

PASO A PSEUDOCODIGO

Asignación

variable <- expresion

Entrada

LEER variable

Salida

ESCRIBIR expresion

Secuencia

I1 ; I2 ; ... ; In

Alternativa simple

```
SI condicion ENTONCES
  I1 ; ... ; In
FINSI
```

Alternativa doble

```
SI condicion ENTONCES
  I1 ; ... ; In
SINO
  J1 ; ... ; Jn
FINSI
```

Bucle MIENTRAS

MIENTRAS condicion HACER

```
  I1 ; ... ; In
FINMIENTRAS
```

Bucle REPETIR

```
REPETIR
  I1 ; ... ; In
HASTA condicion
```

Bucle PARA

```
PARA variable DESDE inicial HASTA final INCREMENTO paso HACER
  I1 ; ... ; In
FINPARA
```

Bucle ITERAR

```
ITERAR
  I1 ; ... ; In
  SI condicion ENTONCES
    SALIR
  FINSI
  J1 ; ... ; Jn
FINITERAR
```

Comentario

```
** comentario
```

Programa

```
PROGRAMA nombre
ENTORNO:
  ** descripción de los objetos
  ...
ALGORITMO:
  ** descripción de las acciones
  ...
FINPROGRAMA
```

PASO A LENGUAJE DE PROGRAMACION

Asignación

- **Pseudocódigo**: variable <- expresion
- **BASIC**: LET variable = expresion
- **COBOL**: COMPUTE variable = expresion . | MOVE expresion TO variable .
- **Pascal**: variable := expresion

Entrada

- **Pseudocódigo**: LEER variable
- **BASIC**: INPUT variable | READ variable

- **COBOL**: ACCEPT variable .
- **Pascal**: READ(variable)

Salida

- **Pseudocódigo**: ESCRIBIR expresion
- **BASIC**: PRINT expresion | LPRINT expresion | WRITE expresion
- **COBOL**: DISPLAY expresion .
- **Pascal**: WRITE(expresion)

Secuencia

- **Pseudocódigo**: I1 ; I2 ; ... ; In
- **BASIC**: Nº_línea I1 : I2 : ... : In
- **COBOL**: I1 I2 ... In .
- **Pascal**: I1 ; I2 ; ... ; In

Alternativa simple

- **Pseudocódigo**: SI condicion ENTONCES I1 ; ... ; In FINSI
- **BASIC**: IF condicion THEN I1 : ... : In
- **COBOL**: IF condicion I1 ... In .
- **Pascal**: IF condicion THEN BEGIN I1 ; ... ; In END

Alternativa doble

- **Pseudocódigo**: SI condicion ENTONCES I1 ; ... ; In SINO J1 ; ... ; Jn FINSI
- **BASIC**: IF condicion THEN I1 : ... : In ELSE J1 : ... : Jn
- **COBOL**: IF condicion I1 ... In ELSE J1 ... Jn .
- **Pascal**: IF condicion THEN BEGIN I1 ; ... ; In ELSE BEGIN J1 ; ... ; Jn END

Bucle MIENTRAS

- **Pseudocódigo**: MIENTRAS condicion HACER I1 ; ... ; In FINMIENTRAS
- **BASIC**: WHILE condicion I1 : ... : In WEND
- **COBOL**: PERFORM I1 ... In UNTIL condicion .
- **Pascal**: WHILE condicion DO BEGIN I1 ; ... ; In END

Bucle REPETIR

- **Pseudocódigo**: REPETIR I1 ; ... ; In HASTA condicion
- **BASIC**: I1 : ... : In IF NOT condicion THEN GOTO línea
- **COBOL**: BUCLE . I1 ... In . IF NOT condicion GO TO BUCLE .
- **Pascal**: REPEAT I1 ; ... ; In UNTIL condicion

Bucle PARA

- **Pseudocódigo**: PARA variable DESDE inicial HASTA final INCREMENTO paso HACER I1 : ... : In FINPARA
- **BASIC**: FOR variable = inicial TO final STEP paso I1 ; ... ; In NEXT variable
- **COBOL**: PERFORM I1 ... In VARYING variable FROM inicial BY paso UNTIL variable > final .
- **Pascal**: FOR variable := inicial TO | DOWNTO final DO BEGIN I1 ; ... ; In END

Bucle ITERAR

- **Pseudocódigo**: ITERAR I1 ; ... ; In SI condicion ENTONCES SALIR FINSI J1 ; ... ; Jn FINITERAR
- **BASIC**: I1 : ... : In IF condicion THEN GOTO línea-fin J1 : ... : Jn GOTO línea
- **COBOL**: BUCLE . I1 ... In . IF condicion GO TO FIN-BUCLE . J1 ... Jn . GO TO BUCLE . FIN-BUCLE .
- **Pascal**: I1 ; ... ; In ; WHILE NOT condicion DO BEGIN J1 ; ... ; Jn ; I1 ; ... ; In END

Comentario

- **Pseudocódigo**: ** comentario
- **BASIC**: REM comentario
- **COBOL**: * comentario
- **Pascal**: (* comentario *)

Programa

- **Pseudocódigo**: PROGRAMA nombre ... FINPROGRAMA
- **BASIC**: REM nombre ... END
- **COBOL**: IDENTIFICATION DIVISION . PROGRAM-ID . nombre . ... STOP RUN . ...
- **Pascal**: PROGRAM nombre ; ... BEGIN ... END .

Alternativas

Los programas, para un mejor funcionamiento y poder realizar un número mayor de tareas, deben permitir emplear acciones alternativas para poder elegir una de ellas cuando la situación lo requiera. Las instrucciones condicionales o tomas de decisión permiten realizar acciones alternativas. Por tanto, la ejecución de una línea o grupos de líneas del programa depende de si cumplen o no una o varias condiciones.

TOMAS DE DECISION

Para preguntar se utiliza la instrucción o sentencia SI (IF). La contestación sólo puede ser verdadero o falso, es decir, sí o no.

Ejemplo: Si llueve, coge el paraguas

La realización de la acción está supeditada a que se cumpla la condición. El formato de las tomas de decisión es:

Instrucción SI

```
SI condicion
  instrucciones
FINSI
```

El ordenador primero examina la condición. Pueden suceder dos cosas:

- **La cumple**: realiza todas las instrucciones que hay dentro del SI, luego continua ejecutando las que están fuera del SI.
- **No la cumple**: no entra en el SI. Sólo realiza las instrucciones siguientes al SI.

Es decir, las instrucciones del SI sólo las realiza cuando cumple la condición. Las instrucciones que están fuera las realiza siempre, se cumpla o no la condición. Se puede poner más de una condición, siempre y cuando estén unidas por los operadores lógicos (AND, NOT, OR).

Ejemplo: introducir un número por teclado. Que nos diga si es negativo.

```
PROGRAMA negativo
Borrar_pantalla( )
num <-0
ESCRIBIR "Introduce un número: "
LEER num
SI num < 0 ENTONCES
  ESCRIBIR "es negativo"
FINSI
FINPROGRAMA
```

Instrucción SI - SINO

```
SI condicion_1 operador_logico condicion_2 ENTONCES
  instrucciones_1
SINO
  instrucciones_2
FINSI
```

A menudo necesitamos realizar dos procesos completamente distintos, dependiendo de si cumple o no la(s) condición(es) de entrada del SI.

Ejemplo: si hace frío, ponte el abrigo; en caso contrario, ven en camisa

- **Cumple la(s) condición(es)**: realiza las instrucciones que hay entre el SI y el SINO (instrucciones_1).
- **No las cumple**: ejecuta las instrucciones que hay entre el SINO y el FINSI (instrucciones_2).

Todo lo que se encuentre fuera del SI siempre lo va a realizar. SINO significa "en caso contrario".

Ejemplo: introducir un número por teclado. Que nos diga si es par o impar.

```
PROGRAMA par
Borrar_pantalla( )
num <- 0
ESCRIBIR "Introduce un número: "
LEER num
SI num = int(num / 2) * 2 ENTONCES
  ESCRIBIR "es par"
SINO
  ESCRIBIR "es impar"
FINSI
FINPROGRAMA
```

Instrucciones SI - SINO anidadas

```
SI condición_1 ENTONCES
  instrucciones_1
  SI condición_2 ENTONCES
    instrucciones_2
  SINO
    instrucciones_3
```

```
  FINSI
SINO
  instrucciones_4
  SI condición_3 ENTONCES
    instrucciones_5
  SINO
    instrucciones_6
  FINSI
FINSI
```

En el formato general para la sentencia SI, las instrucciones 1 y 2 no están limitadas a ser instrucciones imperativas; pueden ser expresiones condicionales y surge la posibilidad de usar instrucciones SI anidadas.

Bucles

BUCLE O CICLO

En informática, la mayoría de las veces la tarea que debe realizar el ordenador es la misma: lo único que varía son los valores de los datos con los que está operando. Llamamos bucle o ciclo a todo proceso que se repite un número de veces dentro de un programa.

```
MIENTRAS condicion HACER
  instruccion_1
FINMIENTRAS
instruccion_2
```

Lo primero que hace el ordenador es examinar la condición. Da como resultado dos posibilidades:

- **Se cumple**: va a realizar todas las instrucciones que están dentro del ciclo, instruccion_1. Las estará repitiendo hasta que deje de cumplirse la condición. Entonces sale del ciclo y continúa ejecutando las instrucciones que hay fuera de él, instruccion_2.
- **No se cumple**: no entrará en el ciclo. Ejecuta las instrucciones que están fuera de él, instruccion_2.

La condición del bucle no tiene por qué ser única: puede haber más de una, siempre y cuando estén unidas por los operadores lógicos (AND, NOT, OR). Lo expresamos:

```
MIENTRAS condicion_1 operador_logico condicion_2 HACER ...
```

CONTADOR

Un contador es una variable destinada a contener diferentes valores, que se va incrementando o decrementando cada vez que el ordenador realiza la instrucción que lo contiene. El incremento o decremento, si es negativo, llamado también paso de contador, es siempre constante.

```
variable = variable + | - constante
```

El ordenador primero evalúa la expresión situada a la derecha del signo igual, realiza la suma o la resta y su resultado lo asigna a lo que hay a la izquierda del igual. El valor de la constante no tiene por qué ser la unidad, puede ser cualquier número, pero en todo el programa se debe conservar siempre dicho valor.

SUMADOR O ACUMULADOR

Es una variable que nos va a permitir guardar un valor que se incrementa o decrementa de forma no constante durante el proceso. En un instante determinado tendrá un valor y al siguiente tendrá otro valor igual o distinto.

sumador = sumador + | - variable

Ejecuta en primer lugar lo que hay a la derecha del signo igual. Realiza la operación y el resultado lo guarda en la variable sumador.

BUCLES ANIDADOS

Un ciclo puede estar formado por otro u otros ciclos. Al igual que sucedía con la instrucción SI, que dentro de un SI podíamos tener todos los SI que fueran necesarios, dentro de un bucle MIENTRAS pueden ir otro u otros bucles MIENTRAS, de tal forma que el último de todos, el situado más interiormente, es el primero en cerrarlo, en acabar. El primero de todos, situado más fuera, es el último en terminar.

```
MIENTRAS condicion_1 HACER
  ...
  MIENTRAS condicion_2 HACER
    ...
    MIENTRAS condicion_3 HACER
      ...
      ...
    FINMIENTRAS
    ...
  FINMIENTRAS
  ...
FINMIENTRAS
```

Los puntos suspensivos serán las distintas instrucciones a realizar.

Menús

Instrucción CASO

Las operaciones a realizar se están presentando por separado sin ningún nexo de unión entre ellas. Generalmente esto no sucede así, es preciso realizar una serie de operaciones que pueden estar más o menos relacionadas entre sí, indicando cuál es la que deseamos ejecutar en ese instante. Un menú consiste en presentar en pantalla una ventana con una serie de operaciones u opciones a realizar, cada una de las cuales realiza una función determinada. Cuando termine de ejecutar cada una de ellas, mediante subprogramas, el programa vuelve de nuevo al menú del que había partido. A veces los menús se presentan anidados, es decir, alguna de las opciones del menú, al ser seleccionada, hace que aparezca otro menú, dando lugar a nuevas posibilidades de elección. Los menús permiten ejecutar más de un programa, sin necesidad de tener que escribir su nombre, cada vez que se desea ejecutarlo. Simplemente, le indicaremos mediante una variable la opción deseada. La selección del programa a realizar se puede hacer mediante la instrucción SI. Pero hay una más rápida y más fácil de manejar: es la instrucción condicional múltiple CASO.

```
HACER CASO
  CASO condicion_1 HACER
```

```
    subprograma_1
  CASO condicion_2 HACER
    subprograma_2
    ...
  CASO CONTRARIO
    ...
FINCASO
```

Funciona bifurcando la ejecución del programa a las instrucciones que siguen a la evaluación verdadera de una condicion CASO. La ejecución del programa continua hasta encontrarse la próxima orden CASO, CASO CONTRARIO o FINCASO. A continuación se ejecuta la primera línea que sigue a la sentencia CASO. Si ninguna de las condiciones del CASO es verdadera, el siguiente conjunto de instrucciones que sigue a la sentencia CASO CONTRARIO, si existe, se ejecuta hasta la próxima orden FINCASO. En las condiciones puede haber más de una condición, siempre y cuando estén relacionadas mediante los operadores lógicos. Esta instrucción también se puede utilizar cuando sabemos que los valores de las variables sólo pueden ser unos fijos y determinados.

Ejemplo: introducir dos números por teclado y mediante un menú calcular su suma, su resta, su multiplicación y su división.

```
PROGRAMA aritm,tica
op = 0
Borrar_pantalla()
EN 10,20 ESCRIBIR "1§ Número: "
EN 10,29 LEER n1
EN 12,20 ESCRIBIR "2§ Número: "
EN 12,29 LEER n2
MIENTRAS op <> 5 HACER
  op = 0
  Borrar_pantalla( )
  EN 6,20 ESCRIBIR "Menú de opciones"
  EN 10,25 ESCRIBIR "1.- Suma"
  EN 12,25 ESCRIBIR "2.- Resta"
  EN 14,25 ESCRIBIR "3.- Multiplicación"
  EN 16,25 ESCRIBIR "4.- División"
  EN 18,25 ESCRIBIR "5.- Salir del programa"
  EN 22,25 ESCRIBIR "Elige opción: "
  EN 22,39 LEER op
  Borrar_pantalla( )
  HACER CASO
    CASO op = 1
      EN 10,20 ESCRIBIR "Su suma es: "
      EN 10,33 ESCRIBIR n1 + n2
    CASO op = 2
      EN 10,20 ESCRIBIR "Su resta es: "
      EN 10,33 ESCRIBIR n1 - n2
    CASO op= 3
      EN 10,20 ESCRIBIR "Su multiplicación es: "
      EN 10,33 ESCRIBIR n1 * n2
    CASO op = 4
      EN 10,20 ESCRIBIR "Su división es: "
      EN 10,33 ESCRIBIR n1 / n2
  FINCASO
FINMIENTRAS
FINPROGRAMA
```

Subprogramas

Los pseudocódigos realizados hasta ahora están todos ellos descritos en un único programa, llamado programa principal. Cuando se habla de modularidad, consiste en estructurar el programa principal en módulos más pequeños llamados subprogramas o subrutinas. Un subprograma es un conjunto de sentencias de un programa que realizan una determinada tarea y que pueden ser ejecutadas desde más de un punto del programa principal. Cuando termine de ejecutarse el subprograma continuan procesándose las instrucciones siguientes del programa. Una subrutina es como un boomerang: va, realiza lo que tenga que hacer y regresa al punto de partida. Su estructura básicamente es la de cualquier programa, con las diferencias lógicas en la parte inicial y final. A su vez, un subprograma puede estar compuesto por varios subprogramas. Están descritos fuera del programa principal.

HACER subprograma

Ejemplos de llamadas a subprogramas en distintos lenguajes:

- DO
- GOSUB
- PERFORM
- CALL

Los nombres de los subprogramas los escribiremos en minúsculas con la inicial en mayúsculas, para una mayor legibilidad del programa. A la hora de nombrarlos, deben cumplir la normativa de las variables, es decir, deben empezar con una letra y el resto de los caracteres, hasta una longitud de 8, puede ser una combinación de números, letras y el guión, pero no puede contener espacios en blanco. El número de subprogramas que habrá dentro de cada programa será elección nuestra en función de la complejidad del ejercicio. La función que deben cumplir los subprogramas es la de conseguir la estructuración del programa y facilitar la tarea en su construcción y simplificar al máximo las posibles modificaciones posteriores en el programa. Otras de las misiones de los subprogramas es la de evitar la repetición de instrucciones dentro de un programa, estas instrucciones se escriben en un subprograma. Este subprograma es llamado, ejecutado, las veces que haga falta. Así, las instrucciones que lo componen sólo están escritas una vez, necesitando menos cantidad de memoria para almacenar el programa.

Ejemplo: calcular el factorial de un número, mediante subprogramas.

```
PROGRAMA factorial
res = "S"
MIENTRAS res = "S" HACER
  Borrar_pantalla()
  factorial = 1
  ESCRIBIR "Número: "
  LEER numero
  SI numero < 0 ENTONCES
    ESCRIBIR "No tiene factorial"
  SINO
    HACER Calculos
  FINSI
  HACER Mas
FINMIENTRAS
FINPROGRAMA

Calculos.
MIENTRAS numero > 1 HACER
  factorial = factorial * numero
  numero =numero - 1
```

```
FINMIENTRAS
HACER Imprimir

Mas.
res = " "
MIENTRAS res <> "S" AND res <> "N" HACER
   ESCRIBIR "Deseas calcular m s factoriales (S/N): "
   LEER res
   res = Convertir_mayusculas(res)
FINMIENTRAS

Imprimir.
ESCRIBIR "Su factorial es: "
ESCRIBIR factorial
```