TEMA 4: ALGORITMOS Y PROGRAMAS

Introducción

Un algoritmo es una sucesión finita de pasos no ambiguos, que se ejecutan en un tiempo finito, que le dicen al ordenador lo que hay que hacer en cada momento para llegar a la resolución del problema.

Resaltar que:

... es una cantidad **finita** de pasos que se llevan a cabo en un **tiempo finito** (los bucles infinitos no son algoritmos.)

... es una sucesión de pasos **no ambiguos**, es decir cada paso especifica una tarea determinada a ser realizada en cada momento por el ordenador.

Los pasos a seguir en la resolución de cualquier problema mediante un ordenador son los siguiente:

1. Análisis del problema.
2. Búsqueda del algoritmo.
3. Programación de algoritmo.
4. Traducción y comprobación del programa.

Análisis del problema

a.- Acotar y especificar el problema con total precisión (obtener el máximo de información acerca de lo que debemos resolver y las soluciones a determinar.)

b.- Definir los datos iniciales o de partida (que datos necesitamos proporcionar del problema para resolverlo.)

c.- Definir que datos o resultados debe proporcionar el algoritmo.

Búsqueda del algoritmo

Búsqueda de una sucesión finita de pasos no ambiguos que nos lleven a la resolución del problema:

a.- Selección del mejor algoritmo.

b.- Mejora del algoritmo.

La selección y la mejora se hacen, habitualmente, en función del tiempo de ejecución (aproximadamente la cantidad de instrucciones que se tienen que ejecutar para resolver correctamente el problema.)

Ejemplo 1: Encontrar al alumno de más edad de la clase

1.- Análisis del problema. ¿Qué entradas tenemos? ¿Qué queremos obtener como resultado?

2.- Búsqueda del algoritmo.

.1. Preguntar a un alumno su nombre y edad y guardarlas

.2. Preguntar a otro alumno su edad

.3. Si su edad es mayor que la edad apuntada entonces sustituir el nombre y edad guardados por las de este último alumno.

.4. Mientras queden alumnos por preguntarles la edad ir al paso 2

.5. Escribir el nombre y edad guardados

Para la obtención de los algoritmos utilizaremos normalmente dos métodos:

Diseño descendente o modular: El problema se divide en subprogramas de más fácil resolución.

Catalogar los libros de la Universidad de Valencia. La universidad de Valencia tiene sus libros distribuidos por varias Bibliotecas y departamentos, por ello abarcar este problema como un todo resulta de difícil solución. Es más sencillo catalogar por separado cada una de las bibliotecas y departamentos y luego juntar todos los resultados en una única catalogación.

Diseño por refinamiento por pasos: Se va buscando la solución y se va especificando hasta llegar a una solución concreta que sí que sabemos resolver.

Poner una nota final a un alumno:

Solución parcial: nota de prácticas

Solución parcial: nota de teoría

Solución final: nota de practicas * 0.3 + nota de teoría * 0.7

Generalmente la utilización de los dos métodos es complementaria.

Programación del algoritmo

Una vez determinado el algoritmo hay que escribirlo en un lenguaje de alto nivel. Por eso lo mejor es escribir el algoritmo en un lenguaje restringido que sea fácil de traducir a un lenguaje de alto nivel.

El **pseudocódigo** es una manera de escribir algoritmos de forma poco estricta (con una sintaxis relajada) o estructuras de datos poco detalladas, pero intentando acercar las ideas del algoritmos a estructuras y sintaxis parecidas a las de los lenguajes de alto nivel en los que vamos a programar el algoritmo.

Ejercicios:

1. Calcular la tabla del dos.
2. Calcular el cuadrado de los 10 primeros números.
3. Calcular el factorial de un número.

Estructura de un programa en pseudocódigo

Algoritmo *nombre_de_algoritmo*

Constantes

constante = valor

Tipos

tipo *nombre_tipo*

campo1: tipo

campo2: tipo

fin_tipo

Variables

variable1, *variable2:* tipo

Inicio

Sentencias

Fin

Las sentencias serán únicamente de tres tipo:

1. Instrucciones de entrada/salida

   Leer(variable). Pide un valor al usuario y lo almacena en la variable pasada como parámetro.

   Ejemplo:

   Leer(a) --> Pedirá al usuario que introduzca un valor y lo almacene en la variable a.

   Escribir(variable ó literal). Escribe en pantalla el valor que contenga la variable especificada o el literal. Los literales serán números (enteros o reales) o cadenas de caracteres.

   Ejemplos:

   A ← **3**

   Escribir (A) ---> Escribirá en la pantalla un 3.

   Escribir ("Hola") ---> Escribirá en la pantalla *Hola*

2. Asignaciones

   Instrucción simple compuesta por una variable, el símbolo de la asignación (**←)** y una expresión o literal. Tras ejecutar la instrucción a la variable se le asignará el resultado de la expresión.

   Variable ← **expresión**

   Ejemplo:

   a ← **2 + 4 * (8+a)**

cad ← "Soy una cadena"

2.1. Operadores Aritméticos

+	suma
-	resta
*	producto
/	division
mod	módulo (resto de una división)

3. Estructuras de control. Alguna de las estructuras de control vistas a continuación.

Estructuras de control

Llamaremos estructuras de control a las acciones que tienen como objeto marcar el orden de ejecución de las instrucciones y que van a servirnos para escribir concisamente y sin ambigüedades los algoritmos.

Todas las estructuras de control que estudiaremos estarán compuestas de unos elementos básicos (léxico) y una estructura (sintaxis.)

Estructuras secuenciales

En una estructura secuencial una instrucción sigue a otra en una secuencia lineal.

Pseudocódigo

```
Inicio
        tarea1
        tarea2
        ...
        tarean
Fin
```

Ejemplo: Calcular la nota media de un alumno .

nota_teoria * 0.7 + nota_practica *0.3

Pseudocódigo

```
Algoritmo calcula_nota
variables
real: nota_teoria, nota_practica, nota_final
Inicio
        Leer (nota_teoria)
        Leer (nota_practica)
        nota_final ← nota_teoria * 0.7 + nota_practica * 0.3
        Escribir (nota_final)
Fin
```

Estructuras selectivas

Son las que toman una cierta dirección dentro del flujo del programa en función de una condición o el valor de una variable.

Una condición es una expresión lógica que se evalúa a Verdadero o Falso.

Operadores Lógicos

= Igual

<> Distinto

< menor que

> mayor que

<= menor o igual que

>= mayor o igual que

Alternativas simples

Se realiza una acción o conjunto de acciones si se cumple una determinada condición.

Pseudocódigo

```
...
Si ( condición ) entonces
        acciones
Fin_si
...
```

Ejemplo: Ordenar dos números (Leídos dos números escribir por pantalla primero el menor y luego el mayor)

Pseudocódigo

```
Algoritmo Ordenar
Variables
 a, b, aux: entero
Inicio
        Leer (a)
        Leer (b)
        Si (a > b) Entonces
                aux ← a
                a ← b
                b ← aux
        Fin_si
        Escribir (a)
        Escribir (b)
Fin
```

Alternativas dobles

Si una condición se cumple se realizan unas acciones, si no se cumple la condición se realizan otras.

Pseudocódigo

```
...
Si ( condición ) entonces
        acciones1
sino
        acciones2
Fin_si
...
```

Ejemplo sencillo: Dado un número, decir si es positivo o negativo.

Pseudocódigo

```
Algoritmo Positivo_Negativo
Variables
        x: Entero
Inicio
        Leer (x)
        Si (x<0) entonces
                Escribir ('Numero negativo')
        sino
                Escribir ('Numero positivo')
        Fin_si
Fin
```

Alternativas múltiples

Dependiendo del valor de una variable se realizan unas acciones u otras.

Pseudocódigo

```
...
Según_sea (variable) hacer
        Caso valor1: acciones1
        Caso valor2: acciones2
        …
        Caso valorn: accionesn
        Default: accionesx
Fin_según_sea
...
```

Este tipo de estructuras se utiliza especialmente cuando se programan menús de selección.

Ejemplo: Calculadora infantil. Realizar un programa que pida dos números y una operación (suma, resta, multiplicación o división) y nos de el resultado de operar los números con esa operación.

Pseudocódigo

```
Algoritmo Calculadora
Variables
    x, y, op: Entero
    res: Real
Inicio
    Escribir ( 'Dame números')
    Leer (x)
    Leer (y)
    Escribir ('Dame operación:')
    Escribir ('(1-Suma/2-Resta')
    Escribir ('(3-Multiplic./4-División')
    Leer (op)
    Según_sea (op) hacer
        Caso 1: res  a + b
                Escribir (res)
        Caso 1: res  a + b
                Escribir (res)
        Caso 1: res  a + b
                Escribir (res)
        Caso 1: res  a + b
                Escribir (res)
        Default: Escribir ('Operación no válida')
    Fin_según_sea
Fin
```

Estructuras repetitivas (o bucles)

Un bucle es un conjunto de instrucciones del programa que se ejecutan repetidamente o bien un número determinado de veces, o bien mientras se cumpla una determinada condición (*hay que tener cuidado con los bucles infinitos*)

Todo bucle contiene los siguientes elementos (aunque no necesariamente en ese orden):

- Iniciación de las variables referentes al bucle.
- Decisión (seguimos con el bucle o terminamos.)
- Cuerpo del bucle.

Existen tres tipos de bucles en programación estructurada:

Bucle *Desde…Hasta*

Este bucle se utiliza cuando sabemos el número de veces que queremos que se realice una cierta tarea.

Pseudocódigo

```
...
Desde variable ← v_ini hasta v_fin hacer
        acciones
Fin_desde
...
```

Bucle *Hacer…Mientras*

Este bucle lo utilizaremos si sabemos la condición que hace que se repita la tarea varias veces. Las acciones se realizan al menos una vez, antes de realizar la comprobación de la condición

Pseudocódigo

...

Hacer

acciones

Mientras (condición)

...

Bucle Mientras…Hacer

Es muy parecido al anterior, pero en este caso la comprobación de la condición se realiza antes de ejecutar la tarea, de manera que la tarea puede no llegar a hacerse.

Pseudocódigo

...

Mientras (condición) *Hacer*

acciones

Fin_Mientras

...

Bucles anidados e independientes.

Existen dos maneras básicas de utilizar varios bucles: De forma anidada y de forma independiente.

De forma independiente nos limitaremos a ir haciendo los bucles de manera que al finalizar uno empezará el siguiente. De esta forma las tareas entre bucles son independientes (cálculo del número combinatorio)

Otra forma es mediante la utilización de bucles anidados. Los bucles anidados son bucles que están dentro de otros bucles de manera que la ejecución de los bucles internos depende de la ejecución de los bucles externos (algoritmo que muestre las tablas de multiplicar del 1 al 10.)

Programación modular

Cuando realizamos ciertos programas (por ejemplo el programa de números combinatorios) podemos darnos cuenta que existen ciertas partes del programa que se repiten de forma muy similar, de manera que cambiando pocos elementos quedarían realmente iguales.

Estas partes del programa que se repiten podríamos hacerlas depender de unos valores para que sirviesen para cualquier propósito. A estas partes de los programas es a lo que llamaremos módulos o subprogramas. Con esta separación podemos reutilizar ciertas partes del código de forma más sencilla, y podemos escribir más fácilmente los programas utilizando el diseño descendente para la resolución de algoritmos.

Las características básicas que deben cumplir los subprogramas o módulos son:

a.- Realización de una tarea específica.

b.- Parametrización para caracterizar la actuación de los módulos (parámetros)

c.- Existen básicamente dos tipos distintos de subprogramas: las funciones y los procedimientos (las funciones devuelven uno o más valores, mientras que los procedimientos devuelven ninguno o más.)

Ambito de las variables

Variables Locales

Son propias de cada módulo y sólo existen mientras dura la ejecución del módulo. Cuando la ejecución del módulo desaparece, las variables locales desaparecen.

Variables Globales

Son variables que existen durante toda la ejecución del algoritmo, de manera que se puede acceder a ellas en cualquier momento de la ejecución del algoritmo.

Paso de parámetros

Por valor

Sólo se tiene en cuenta el valor del parámetro pasado al módulo, de manera que durante la ejecución del módulo se reserva un espacio para ese parámetro y se copia el valor pasado en ese nuevo espacio. Cuando acaba la ejecución del módulo, el espacio para el parámetro desaparece.

Por referencia

Lo que pasamos a la función es una referencia al parámetro que se pasa, de manera que no existe un espacio reservado para ese parámetro durante la ejecución del módulo. Cualquier modificación que realicemos del parámetro quedará reflejado en el punto desde el que se hizo la llamada al módulo.

Programación estructurada

Diremos que estamos realizando una programación estructurada si…

…empleamos para el diseño de los algoritmos el diseño descendente y/o modular.

…descomponemos, en función del diseño descendente, el programa en módulos independientes (prohibida la utilización de variables globales.)

…en cualquier caso, sólo utilizamos en la escritura de los algoritmos (y los programas) los tres tipos de estructuras de control vistas.

Recurrencias (o recursividad)

Es la propiedad de llamar a una función desde sí misma o desde otra que ha sido llamada por ésta.

En matemáticas la recursividad es una forma sencilla y habitual de definir una función a partir de ella misma (por ejemplo la definición de factorial o de los números de Fibonacci.)

Una característica muy importante de la recursividad es que, igual que en las estructuras repetitivas, debe de existir un punto o condición de fin, que en algún momento detenga la recursividad.