David Bernal Navarrete – 1º DAM

1. La universalidad se refiere a que sea un programa portable, que los programas escritos en un lenguaje se puedan llevar a otro y sigan funcionando correctamente, además de que dicho lenguaje se pueda usar en cualquier máquina, puesto que es independiente del procesador.

2. Se refiere a la capacidad de crear y usar las instrucciones del lenguaje de una forma más "humana", menos estricta, y con un uso más abierto de las instrucciones.

3. Los mantenimientos adaptativos y perfectivos reinician el ciclo de vida. El mantenimiento adaptativo consiste en actualizar el programa para adaptarlo a nuevas condiciones o necesidades que surjan. El perfectivo trata de corregir errores ya existentes, pero que pueden poner en peligro el funcionamiento correcto del programa. Ambos reinician el ciclo de vida ya que de ellos resultará un programa diferente del que se tenía, y deben pasar por el mismo ciclo que el original, para resultar en un programa pulido y bien construido.

4. Depende de cómo se traduzca dicho programa:

- Si es ensamblado: un ensamblador traducirá el código fuente a código máquina, para posteriormente poder ejecutarlo. Si existe algún error de sintaxis lo notificará.

- Si es interpretado: un intérprete leerá cada línea del código fuente, la traducirá y la ejecutará si no existen errores de sintaxis. En caso contrario notificará del error.

- Si es compilado: un compilador leerá todo el código fuente y lo compilará si no existen errores de sintaxis, en caso contrario notificará del error. El programa resultante todavía no es el final, es el programa objeto, al cual se le aplicará el enlazado de librerías y módulos necesarios, y de él resultará el programa ejecutable.

5. Para que un algoritmo sea correcto debe: estar ordenado; contemplar todas las posibles situaciones; estar diseñado independientemente del lenguaje de programación en el que se vaya a implementar; resolver el problema para el que se ha diseñado.

6. Los valores necesarios para definir zonas de datos en un programa son: el tipo de dato, el nombre de la variable en la que se va a guardar el dato, y el dato en cuestión. Los obligatorios son el tipo de dato y el nombre de la variable. Esto es así porque es necesario decirle al ordenador que tiene que reservar una parte de la memoria, esperando una variable de un tipo determinado, y que responda a un identificador concreto. Sin saber dónde está ni cómo se llama, el ordenador no puede ni guardar ni escoger información almacenada en su memoria. Un ejemplo en C para ilustrarlo sería:

int _variablenum; ← declaramos el tipo y el nombre de la variable.

int _variablenum = 12; ← declaramos el tipo, el nombre, y la inicializamos.

Int 12; ← daría error, puesto que no le hemos dado ningún nombre para declarar la variable.

_variablenum = 12; ← daría error, ya que el ordenador no sabe qué tipo esperar de la variable.

7. Los tipos de datos elementales son: entero, real, carácter, y booleano. Los enteros son números enteros; los reales son números con coma flotante; los caracteres son letras, símbolos o números únicos; y los booleanos son valores de Verdadero o Falso. Su implementación en C sería la siguiente:

Entero: int num = 2;
Real: float numf = 2.5;
Carácter: char letra = "a";

Booleano: bool interruptor = false;

8.

a. Podría contener un error sintáctico en el caso de que: faltara un paréntesis; faltara el ; al final de la sentencia; “printf” estuviera mal escrito; no estuviera la coma que separa en el printf la cadena del dato formateado; faltara alguna de las comillas dobles (“) de la cadena en el interior del printf; en vez de %d estuviera &d (o cualquier otro símbolo que no sea %); la variable ‘a’ no estuviera definida, o no esperara una suma con un entero (por ejemplo, si ‘a’ fuera un booleano o un carácter).

b. Podría dar un error semántico si: esperamos que se formatee un número real, pero ‘a’ está definida como otro tipo de dato diferente; si a+7 no es el resultado que esperamos, aunque el tipo sea correcto.

9.

bool bisiesto = (year % 100 == 0 && year % 400 == 0) || (year % 4 == 0);

10. Un programa que muestra en pantalla la cantidad de veces que hemos pulsado cualquier tecla cada diez segundos. Al finalizar el programa muestra los tecleos totales.

```
ALGORITMO muestra_tecleo
INCLUIR obtener_tiempo_actual, leer_golpe, leer_escape
DECLARAR golpes_dados = 0
DECLARAR tiempo_inicial, golpe
DECLARAR ejecutando = true
MIENTRAS ejecutando == true:
    SI leer_escape == true:
        ejecutando = false
        BREAK
    tiempo_inicial = obtener_tiempo_actual()
    MIENTRAS (tiempo_inicial – obtener_tiempo_actual() < 10) && (ejecutando == true):
        SI leer_escape == true:
            ejecutando = false
            BREAK
        golpe = leer_golpe
        golpes_dados = golpes_dados + golpe

    ESCRIBIR golpes_dados
FIN muestra_tecleo
```

ACLARACIONES:

-obtener_tiempo_actual() es una función que obtiene el tiempo actual en segundos.

-leer_golpe() es una función que detecta una tecla pulsada. Si detecta cualquiera, devolverá 1.

-leer_escape() es una función que detecta si la tecla escape ha sido pulsada. Si se ha pulsado, devolverá true.

-Si se presiona escape mientras se está ejecutando el MIENTRAS que calcula los golpes dados cada 10 segundos, se mostrarán en pantalla los golpes_dados. Esto es intencional, para mostrar el número de golpes totales dados al final de la ejecución del programa.