Sumario

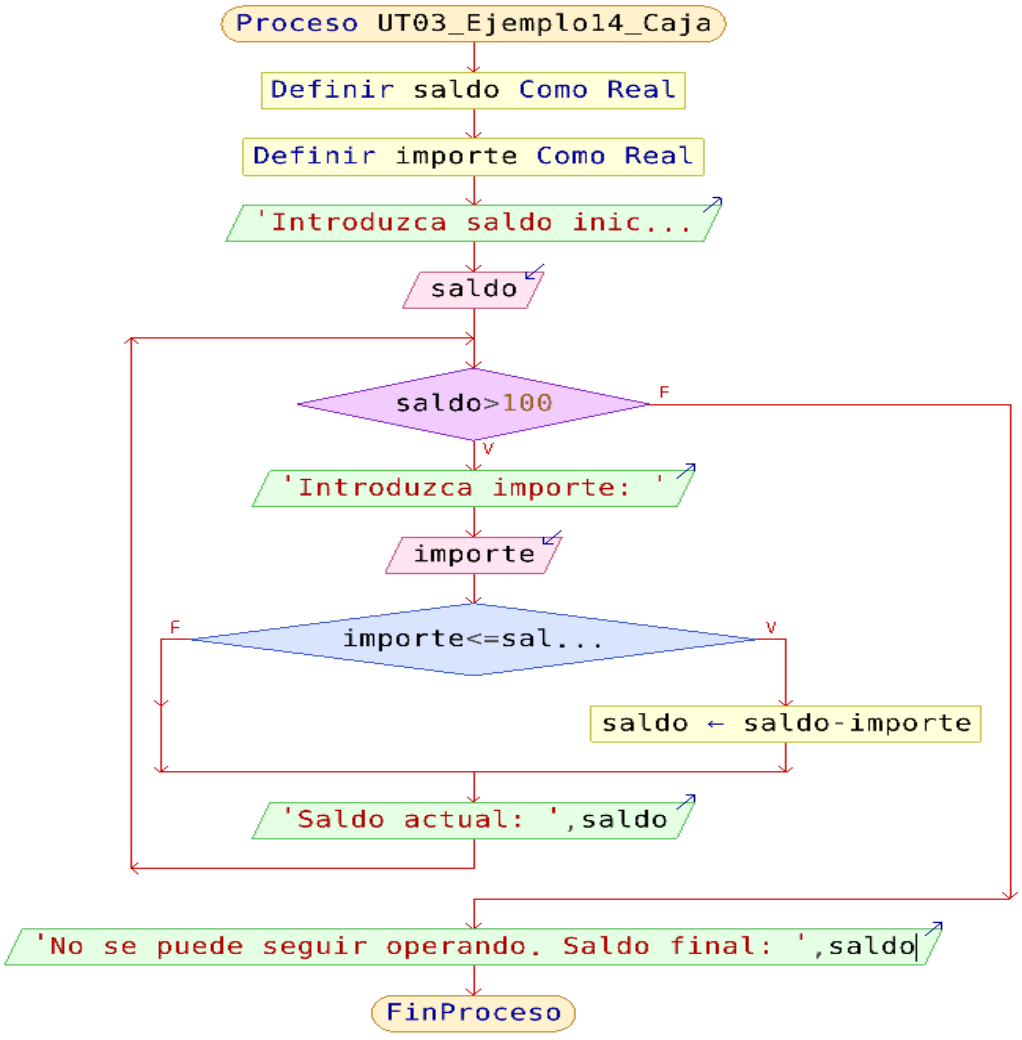
UT 03: Elementos de un programa informático

1 Ejercicios avanzados con PSEInt

A partir de los ejercicios desarrollados hasta ahora con PSEInt vamos a introducir algunas mejoras:

1.1 Caja

Realizar un programa que simule el funcionamiento de un cajero. Este sería el diagrama de flujo del programa:



Completar el código a partir de este esqueleto:

```
Proceso UT03_Ejemplo14_Caja

   //El programa parte de un saldo inicial en Caja
   //El usuario introduce el valor del importe que quiere sacar
   //El programa lo resta mientras el saldo sea mayor a 100 euros
   Definir saldo Como Real;
   Definir importe Como Real;

   Escribir "Introduzca saldo inicial: ";
   Leer saldo;
```

```
   Escribir "No se puede seguir operando. Saldo final: ",saldo;

FinProceso
```

1.2 Caja mejorada

Mejorar el programa anterior para que introduzca controles:

- El saldo inicial introducido por el usuario no puede ser inferior a 0 (el programa debe solicitar que se introduzca de nuevo)
- El importe solicitado por el usuario debe ser siempre positivo (en caso contrario, el programa preguntará de nuevo)
- Si el importe solicitado por el usuario deja menos de 100€ en la cuenta, no se podrá hacer la operación y aparecerá el mensaje "No tienes suficiente dinero".

1.3 Subprocesos

A partir del programa UT03_Ejemplo13_Edades_Para_Mejorado, vamos a desarrollar una nueva versión que realice las siguientes operaciones:

- Además de la edad de cada alumno, el programa preguntará la nota obtenida por cada alumno (entre 0 y 10) y la guardará en un array.
- El programa obtendrá la edad media y la nota media de la clase. Al final imprimirá por pantalla un listado con todas las edades en una línea, y todas las notas en otra línea.

A continuación se presenta el programa de partida (desarrollado en clase) al que hemos añadido un par de modificaciones, consistentes en extraer parte del código para que ciertas operaciones sean realizadas como subprocesos: uno para leer los datos del usuario y otro para calcular las medias

NOTA: Para obtener la nota media podremos reutilizar el subproceso que ya tenemos para calcular las edades medias de los alumnos.

```
Proceso UT03_Ejemplo13_Edades_Para_Mejorado

   //El programa escuchará una serie de valores (edades)
   //hasta que el usuario introduzca un valor negativo o cero
   //y a continuación se mostrará el promedio

   //Esta versión permitirá almacenar las edades en un array
   //y calculará la media con un Subproceso

   Definir contador Como Entero;
   Definir alumnos Como Entero;
   Definir edad Como Entero;

   //Array para guardar las edades. Debe tener un tamaño fijo.
   //Lo dimensionamos a 30, que sería el máximo de alumnos
   //permitido por nuestro programa.
   Definir edad_alumno como Entero;
   Dimension edad_alumno[30];

   alumnos<-0;
   Mientras alumnos<=0 o alumnos>30 Hacer
      Escribir "¿Cuántos alumnos hay en clase? " Sin Saltar;
      Leer alumnos;
      si alumnos<=0 o alumnos>30 Entonces
         Escribir "Número de alumnos incorrecto. Inténtalo otra
vez.";
      FinSi
   FinMientras
```

```
   //A continuación llamamos al subproceso que recoge datos
   entradaDatos(edad_alumno,alumnos);

   //A continuación llamamos al subproceso que calcula la media
   Escribir "Edad media: ", CalculoMedio(edad_alumno,alumnos);

   Escribir "Listado de edades:";
   Para contador<-0 hasta alumnos-1 Hacer
      Escribir edad_alumno[contador]," - " Sin Saltar;
   FinPara
   Escribir "";

FinProceso

//Sacamos el código de cálculo de la media a un Subproceso
SubProceso mediaEdades <- CalculoMedio (edad_alumno,alumnos)
   Definir contador Como Entero;
   Definir mediaEdades como Real;
   mediaEdades<-0;

   Para contador<-0 hasta alumnos-1 Hacer
      mediaEdades<-mediaEdades+edad_alumno[contador];
   FinPara
   mediaEdades<-mediaEdades/alumnos;

FinSubProceso

//Sacamos el código de entrada de datos a un subproceso
Subproceso entradaDatos(edad_alumno,alumnos)
   Definir contador,edad Como Entero;

   Para contador<-0 hasta alumnos-1 Con Paso 1 Hacer
      Escribir "Introduzca edad: " Sin Saltar;
      edad<-0;
      Mientras edad<=0 Hacer
         leer edad;
         si edad<=0 Entonces
            Escribir "Edad incorrecta, introdúzcala de nuevo: " Sin
Saltar;
         FinSi
      FinMientras
      edad_alumno[contador]<-edad;
   FinPara

FinSubProceso
```

1.4 Programa "Encuesta de Franjas Salariales"

Realizar un programa que pregunte la edad y el sexo a 10 personas mayores de edad, guardando los datos en un array, y a continuación realice los cálculos oportunos para generar un desglose de salarios medios, divididos por franjas de edad y sexo.

Notas para la resolución:

Los datos de entrada de los encuestados se guardarán en un array denominado "Salarios", que tendrá 10 filas (una por cada encuestado) y tres columnas:

- La primera columna guardará el salario del encuestado.
- La segunda columna guardará el sexo en formato binario (0-Hombre ; 1-Mujer)
- La segunda columna contendrá la edad del encuestado

Los datos de salida se guardarán en un array denominado "CalculoSueldos" que tendrá 6 filas, una correspondiente a cada franja de edad según los siguientes tramos:

- TRAMO 1 – 18-19
- TRAMO 2 – 20-25
- TRAMO 3 – 26-35
- TRAMO 4 – 36-45
- TRAMO 5 – 46-60
- TRAMO 6 - +60

Este array tendrá 4 columnas que nos servirán para almacenar los datos de salida:

- Columna 1: Sueldo total hombres
- Columna 2: Número total hombres
- Columna 3: Sueldo total mujeres
- Columna 4: Número total mujeres

Sugerencia:

Para afrontar este problema, la idea es que los datos se vayan acumulando en el array de salida a medida que se introducen por consola, de manera que al final se puedan realizar los cálculos medios solo dividiendo la columna que corresponda entre el número de individuos. Por ejemplo: El salario medio de las mujeres entre 36 y 45 años sería el resultado de dividir la celda de la fila 4, columna 3, por el número total de mujeres, que sería la celda en la fila 4, columna 4.

1.5 Programas de ayuda para el ejercicio de Bases de Datos.

Se añaden varios programas de ayuda para dar estrategias y sugerencias con el objetivo de afrontar la tarea de bases de datos que se debe entregar.

Los archivos son:

- UT03_BBDD_Ayudas_Menu.psc
- UT03_BBDD_Ayudas_Menu2.psc
- UT03_BBDD_Ayudas_Menu3.psc
- UT03_BBDD_Ayudas_Menu4.psc

Estos programas aportan estrategias para programar, cómo manejar los bucles para hacer un menú repetitivo, cómo usar arrays en función de los tipos de datos a usar (al tener un conjunto de elementos con datos de diferente tipo, como cadenas para nombres y enteros para las edades, no podemos usar tablas bidimensionales, sino unidimensionales, una para cada tipo de datos) y cómo usar funciones pasando parámetros por valor o por referencia.

1.6 Algoritmo de ordenación

El programa UT03_Ejemplo17_Ordenacion.psc añade un ejemplo de algoritmo de ordenación alfabética de una serie de nombres.

Estrategias para la elaboración de un algoritmo de ordenación:

(1) Partimos de un conjunto de elementos que necesitamos ordenar de mayor a menor. Supongamos que tenemos 4 monedas de diferente valor:
Array `monedas[4]`, en el que disponemos de los elementos:

- `monedas[0] <- 1 €`
- `monedas[1] <- 2 €`
- `monedas[2] <-10 c`
- `monedas[3] <- 20 c`

(2) Intuitivamente, los humanos realizaríamos una observación rápida de los elementos. Una vez observados, extraeríamos el mayor. A continuación, observaríamos los restantes, y de ahí extraeríamos el mayor (que sería el segundo elemento de la lista ordenada). Después ya solo nos quedarían dos, por lo que volveríamos a extraer el mayor entre ellos y ya no sería necesario ordenar el último, porque solo nos quedaría uno.

(3) Veamos cómo haríamos esto en términos informáticos. En primer lugar un bucle que recorre los elementos para quedarse con el mayor de ellos:

- Utilizaremos una variable "i" como puntero para recorrer los elementos. Lo podríamos hacer con un bucle "Para":

```
elemento_mayor<-0; // Suponemos que el primero es el mayor
Para i<-1 hasta 3 Hacer  //Recorremos los elementos 1-3
   Si monedas[i]>monedas[elemento_mayor] Entonces
        elemento_mayor<-i;  //Aquí hemos encontrado un valor mayor
   FinSi
FinPara
```

- Al llegar a este punto, ya sabemos que el elemento mayor sería el 1 (monedas[1] tenía el valor 2).

(4) A continuación, recorreríamos los tres restantes elementos (todos menos el 1). Pero ¿cómo hacer esto en un programa? Nos encontramos con una pequeña dificultad, y es que tendríamos que recorrer los elementos 0, 2 y 3. No podemos hacer un bucle que se salte uno cualquiera de los elementos.

(5) Se nos ocurre otra idea mejor. En lugar de simplemente quedarnos con encontrar el elemento mayor de la lista, vamos a pensar en hacer un INTERCAMBIO de posiciones, es decir, una vez que detectamos que el elemento mayor es el de la posición 1, lo que hacemos es un intercambio de posiciones, es decir, traemos el valor de la posición 1 a la posición 0, y el de la posición 0 a la posición 1. Para poder hacer un intercambio de variables vamos a necesitar una variable auxiliar:

```
elemento_mayor=0; //Para empezar decimos que el primero es el mayor
Para i<-1 hasta 3 Hacer
   Si monedas[i]>monedas[elemento_mayor] Entonces
        elemento_mayor<-i;
   FinSi
FinPara
aux<-monedas[0];
monedas[0]<-monedas[elemento_mayor];  //Elemento mayor a la pos. 0
monedas[elemento_mayor]<-aux;
```

(6) Una vez ordenado el primer valor, nos quedan tres por ordenar, en este caso monedas[1], monedas[2] y monedas[3]. Para ordenarlas, haríamos un bucle muy similar al anterior, pero en este caso solo recorrería los valores 2 y 3. Algo así:

```
elemento_mayor=1; //Para empezar decimos que el primero es el mayor
Para i<-2 hasta 3 Hacer
   Si monedas[i]>monedas[elemento_mayor] Entonces
        elemento_mayor<-i;
   FinSi
```

```
FinPara
aux<-monedas[1];
monedas[1]<-monedas[elemento_mayor];  //Elemento mayor a la pos. 1
monedas[elemento_mayor]<-aux;
```

(7) Y ya que tenemos ordenadas las posiciones 0 y 1, no necesitaríamos más bucles, sino simplemente comparar los elementos 3 y 4, e intercambiarlos si fuera preciso:

```
Si monedas[3]>monedas[2] Entonces
   aux<-monedas[2];
   monedas[2]<-monedas[3];  //Elemento mayor a la pos. 2
   monedas[3]<-aux;
FinSi
```

(8) Los pasos anteriores están basados en un conjunto de 4 elementos. Nos han salido 3 iteraciones con un bucle para cada una de ellas. En lugar de escribir todo el código seguido, lo podríamos simplificar, metiendo el bucle inicial en un bucle anidado:

```
Para iteracion<-0 hasta 2
   elemento_mayor<-iteracion;
   Para i<-iteracion+1 hasta 3 Hacer
      Si monedas[i]>monedas[elemento_mayor] Entonces
           elemento_mayor<-i;
      FinSi
   FinPara
   aux<-monedas[iteracion];
   monedas[iteracion]<-monedas[elemento_mayor];
   monedas[elemento_mayor]<-aux;
FinPara
```

(9) Si tenemos un número indeterminado (n) de elementos, generalizamos y quedarían n-1 iteraciones:

```
Para iteracion<-0 hasta numero_elementos-2
   elemento_mayor<-iteracion;
   Para i<-iteracion+1 hasta numero_elementos-1 Hacer
      Si monedas[i]>monedas[elemento_mayor] Entonces
           elemento_mayor<-i;
      FinSi
   FinPara
   aux<-monedas[iteracion];
   monedas[iteracion]<-monedas[elemento_mayor];
   monedas[elemento_mayor]<-aux;
FinPara
```

(10) Por último, puesto que estamos utilizando la estrategia de intercambiar posiciones, podemos pensar en hacerlo directamente dentro del bucle, sin tener que calcular el valor del "elemento_mayor". Este algoritmo integrado sería el que se da como solución al ejercicio 17:

```
Para iteracion<-0 hasta numero_elementos-2
   Para i<-iteracion+1 hasta numero_elementos-1 Hacer
      Si monedas[i]>monedas[iteracion] Entonces
         aux<-monedas[iteracion];
         monedas[iteracion]<-monedas[elemento_mayor];
         monedas[elemento_mayor]<-aux;
      FinSi
   FinPara
FinPara
```