UNIVERSIDAD DE CANTABRIA

Ejercicios de Ada

1. Estructura de un programa

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con el entorno GPS
  - usar el editor para crear el fichero `hola.adb`
  - compilar y enlazar el programa
  - ejecutar el programa
- Compilar y ejecutar el programa desde línea de comandos
  - compilar y enlazar con `gnatmake`
  - ejecutar el programa

Descripción:

- Escribir, compilar y ejecutar el programa que escribe "hola" en la pantalla

2. Declaraciones

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con declaraciones de variables y constantes

Descripción:

- Crear un programa que declare varias variables y constantes enteras y reales
- El programa debe hacer operaciones simples con estas variables (+,-,*,/).
- Al final, mostrar los resultados

3. Strings y estructuras de control

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con strings de longitud variable
- Practicar con estructuras de control sencillas

Descripción:

- Declarar un string de hasta 25 caracteres
- Leer una palabra introducida por teclado y almacenarla
- Encontrar el número de veces que se repite la vocal ’a’ y sacar el resultado por pantalla

4. Arrays

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con arrays de números reales

Descripción:

- Queremos desarrollar una aplicación que calcule la nota final obtenida por los alumnos de una clase y la muestre por pantalla
- La clase contiene 25 alumnos y la nota numérica de cada alumno tomará un valor entre 0.0 y 10.0
- Crear variables de tipo array para almacenar las notas obtenidas por cada alumno de una asignatura (examen y prácticas)
- Inicializar las variables con valores aleatorios (Utilizar la librería estándar Ada.Numerics.Float_Random)

4. Arrays (cont'd)

UNIVERSIDAD DE CANTABRIA

- Calcular la nota final del alumno sabiendo que la nota de prácticas constituye el 40% y la nota del examen el 60% de dicha nota
- Solicitar al usuario que indique el número de alumno que desea consultar y mostrar las notas del alumno (examen, prácticas y nota final) por pantalla

5. Registros o Estructuras

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con datos almacenados en un array de registros

Descripción:

- Queremos manejar los datos personales de una clase de alumnos. Cada alumno tiene los siguientes datos.
  - Nota examen
  - Nota prácticas
  - Nota final
- La clase contiene un número variable de alumnos
  - Podemos acotar el número máximo a 25
- El programa debe permitir consultar los datos del alumno indicado por el usuario

5. Registros o Estructuras (cont'd)

UNIVERSIDAD DE CANTABRIA

Diseño:

- Definir un tipo de datos para almacenar la información de un alumno
  - Nota examen (número real)
  - Nota prácticas (número real)
  - Nota final (dato enumerado)
- Implementar *un subprograma* que permita consultar los datos de un alumno:
  - El número de alumno se pasará como argumento de entrada
  - Muestra los datos de ese alumno por pantalla
- Crear un array con las notas de cada alumno e inicializarlo
- Crear un programa principal que solicite el número de alumno que se desea consultar y muestre sus datos

5. Registros o Estructuras (cont'd)

UNIVERSIDAD DE CANTABRIA

Para leer y escribir un enumerado en Ada:

- **Utilizar un paquete que sea una instancia del Ada.Text_IO.Enumeration_IO con el tipo `nota final` que se haya creado**
- **Usar las operaciones `put` y `get` de este paquete**

6. Modularidad

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Crear un módulo de programa con interfaz separada del cuerpo

Descripción:

- Programa que gestiona las notas de una lista de alumnos

6. Análisis de requerimientos

UNIVERSIDAD DE CANTABRIA

Queremos manejar las notas obtenidas por una clase de alumnos

Cada alumno tiene los siguientes datos:

- **Nota exámen**
- **Nota prácticas**
- **Nota final**

La clase contiene un número variable de alumnos

- **Podemos acotar el número máximo a 25**

El programa debe permitir insertar alumnos nuevos y ver los datos del alumno indicado por el usuario

6. Diseño arquitectónico

UNIVERSIDAD DE CANTABRIA

El programa está dividido en las siguientes partes

- **paquete "*alumno*"**
  - **datos: notas obtenidas por cada alumno**
  - **operaciones: leer datos por teclado, escribir los datos en pantalla**
- ***programa principal* que crea un array de alumnos y gestiona las llamadas a las operaciones del paquete *alumno***

6. Diseño detallado

UNIVERSIDAD DE CANTABRIA

Paquete `Alumno`

- **datos: registro con los siguientes campos:**

```
nota examen (número real)
nota prácticas (número real)
nota final (dato enumerado)
```

- **solicitar los datos del alumno por teclado**

```
solicitar los dos primeros datos y almacenarlos.
calcular la nota final.
```

- **mostrar los datos del alumno en pantalla**

```
mostrar el nombre de cada dato y su valor.
```

6. Diseño detallado (cont.)

Programa principal. Pseudocódigo:

```
procedure Principal is
    type Clase is ...;
    Tercero_B : Clase;  -- Vacia
begin
    loop
       Pedir una opcion
       case la opcion es
          when insertar
              introducir datos del alumno
          when mirar
              solicitar el numero de alumno
              mostrar los datos de ese alumno
          when salir
              finalizar
    end loop;
end Principal
```

7. Tratamiento de errores

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar los métodos de tratamiento de errores

Descripción:

- Tratar distintos errores en el ejercicio anterior

7. Diseño arquitectónico

UNIVERSIDAD DE CANTABRIA

El programa está dividido en las siguientes partes

- **paquete "*alumno*"**
  - **datos: notas obtenidas por cada alumno**
  - **operaciones: leer datos por teclado, escribir los datos en pantalla**
- ***programa principal* que crea un array de alumnos y gestiona las llamadas a las operaciones del paquete *alumno***

7. Diseño detallado

UNIVERSIDAD DE CANTABRIA

Paquete `Alumno`

- datos: registro con los siguientes campos:

```
nota examen (número real)
nota prácticas (número real)
nota final (dato enumerado)
```

- solicitar los datos del alumno por teclado

```
solicitar los dos primeros datos y almacenarlos.
tratar errores de I/O
calcular la nota final
```

- mostrar los datos del alumno en pantalla

```
mostrar el nombre de cada dato y su valor
```

7. Diseño detallado (cont.)

Programa principal. Pseudocódigo:

```
procedure Principal is
    type Clase is ...;
    Tercero_B : Clase;  -- Vacia
begin
    loop
       Pedir una opcion
       case la opcion es
          when insertar
              introducir datos del alumno
          when mirar
              solicitar el numero de alumno
              mostrar los datos de ese alumno
          when salir
              finalizar
    end loop;
end Principal
```

8. Concurrencia

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Desarrollar un programa concurrente sincronizado

Descripción:

- Se propone desarrollar un programa que implemente la búsqueda concurrente de palabras en una sopa de letras
  - El fichero ***Letras*** dispone de una matriz de letras mayúsculas en el que se han incluido repetidamente y en múltiples direcciones los nombres JUAN, INES, FELIPE, LAURA, CARLOS y CARMEN

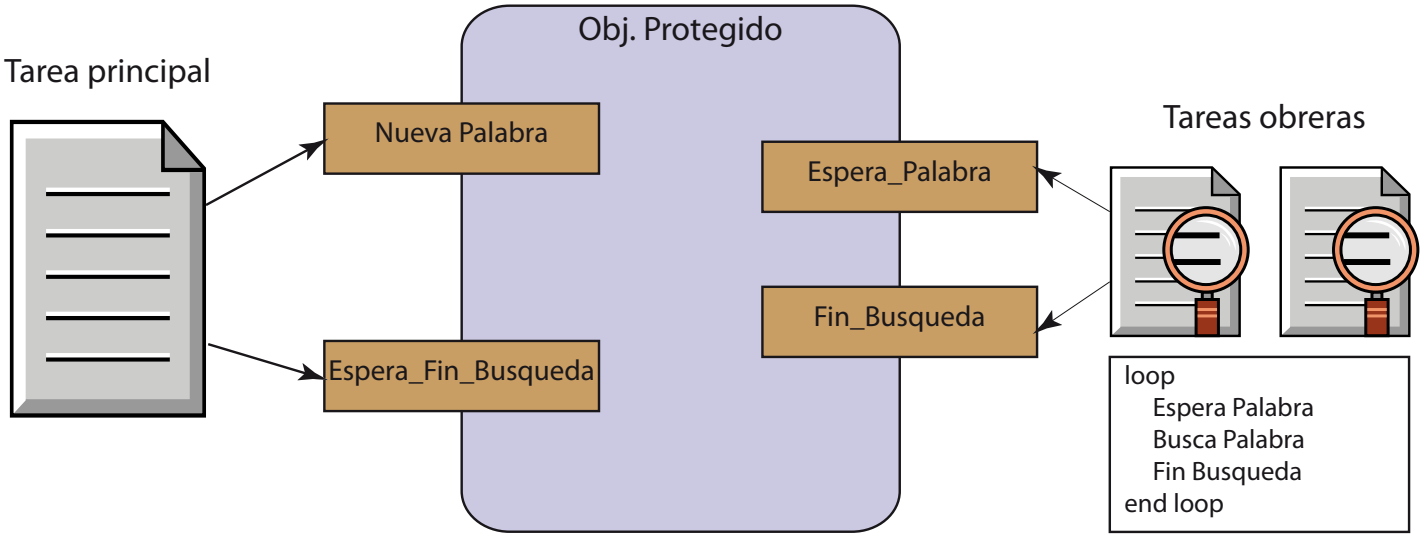
8. Diseño arquitectónico

UNIVERSIDAD DE CANTABRIA

- Paquete "Sopa"
  - Proporciona los recursos para la búsqueda y salida de los resultados obtenidos
- Paquete "Controlador"
  - Un objeto protegido que nos permita la sincronización de las tareas y el acceso mutuamente exclusivo a la información
    - Con procedimientos/entries para introducir nuevas palabras a buscar, esperar la llegada de nuevas palabras y la señalización de que la búsqueda ha finalizado
- Procedimiento principal
  - Creará las tareas obreras para la búsqueda de palabras
  - La tarea principal será la encargada de solicitar por teclado las palabras a buscar y transferirlas a las tareas obreras

8. Planteamiento

- Las tareas obreras se encolan en el objeto protegido a la espera de palabras.
- La tarea principal recibe nuevas palabras, las transfiere a las tareas obreras y éstas comenzarán la búsqueda.



8. Sopa

UNIVERSIDAD DE CANTABRIA

Este paquete se da ya implementado. Su funcionalidad se puede consultar en la especificación del paquete.

Sus procedimientos más importantes son:

- **Procedimiento *Pide_Datos* para solicitar por teclado una serie de palabras y devolverlas almacenadas en una lista**

```
procedure Pide_Datos (NumPal  : out NumeroPalabras;
                      Palabra : out LstPalabras);
```

- **Procedimiento *Busca* para la búsqueda de una palabra en la sopa de letras y realizar una acción determinada con ellas**

```
procedure Busca (Palabra : Unbounded_String;
                 Accion  : ProcAccion);
```

- **Procedimiento *Escribe* para la impresión por pantalla de los resultados de la búsqueda**

8. Programa Secuencial

UNIVERSIDAD DE CANTABRIA

```
with Sopa;
procedure Secuencial is

   Numero_Palabras : Sopa.NumeroPalabras;
   Palabras        : Sopa.LstPalabras;

begin
   Sopa.Pide_Datos (Numero_Palabras, Palabras);
   for p in 1..Numero_Palabras loop
    Sopa.Busca(Palabras(p), Sopa.Escribe'Access);
   end loop;
end Secuencial;
```

8. Controlador

UNIVERSIDAD DE CANTABRIA

El objeto protegido ofrece cuatro procedimientos:

- Nueva_Palabra
  - A través de la que el programa principal añade una nueva palabra para que sea buscada por alguna tarea obrera
- Espera_Palabra
  - Donde las tareas obreras quedan suspendidas hasta la llegada de una nueva palabra
- Fin_Busqueda
  - En el que las tareas obreras señalan que han terminado la búsqueda de la palabra señalada
- Espera_Fin_Busqueda
  - Donde la tarea principal queda suspendida hasta que todas las tareas obreras hayan terminado su búsqueda

8. Programa Principal

UNIVERSIDAD DE CANTABRIA

El programa principal deberá crear varias tareas con la misma funcionalidad:

```
task type Worker is
end Worker;

task body Worker is
     --  Declaraciones
begin
     --  Instrucciones de la tarea
exception
     --  Gestion de excepciones
end Worker;

--  Declaracion de variables
T1,T2,T3 : Worker; --  Definimos tres tareas
...
```

8. Programa Principal (cont'd)

UNIVERSIDAD DE CANTABRIA

Lazo principal. Pseudocódigo

```
begin
   --  Lazo principal
   loop
      Pide palabras para buscar;

      for Cada palabra leída loop
         Añadirla al Controlador;
      end loop;

      Espera a que las tareas obreras terminen la búsqueda;
   end loop;

exception
   ...
end;
```

9. Concurrencia (II)

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Finalizar correctamente un programa concurrente

Descripción:

- Modificar el ejercicio anterior para que el programa termine correctamente cuando el usuario no desee buscar nuevas palabras en la sopa de letras

9. Planteamiento

UNIVERSIDAD DE CANTABRIA

Se pide modificar el ejercicio anterior para que el procedimiento principal *finalice el programa de forma controlada cuando se requiera*. Para ello deberá modificarse:

- **Procedimiento principal**
  - **Establecer una condición de salida del lazo principal (por ejemplo, que no se introduzcan nuevas palabras para buscar)**
  - **Informar al resto de tareas que el programa va a finalizar**
- **Paquete "Controlador"**
  - **Modificar el objeto protegido para que permita la sincronización de las tareas cuando el programa vaya a terminar**

10. Tiempo Real

UNIVERSIDAD DE CANTABRIA

Objetivos:

- Practicar con los mecanismos de tiempo real incluidos en Ada

Descripción:

- Se propone configurar el programa concurrente del ejercicio anterior de tal forma que su comportamiento sea predecible.
- Con cada modificación introducida, ejecutar la aplicación y observar las diferencias con el caso concurrente.

10. Política de planificación

UNIVERSIDAD DE CANTABRIA

Configurar la aplicación para que utilice una política de planificación basada en prioridades fijas

- **Debe asignarse una prioridad diferente a la tarea principal y a cada una de las tareas obreras**

10. Política de sincronización

Modificar el objeto protegido para que utilice el protocolo de sincronización de techo de prioridad.

- **Configurar el techo de prioridad de tal forma que se evite el problema de la inversión de prioridad**
- **Utilizar una política de encolado basada en prioridades**

10. Compilación y Enlazado para Sistemas de Tiempo Real

UNIVERSIDAD DE CANTABRIA

Para garantizar el correcto funcionamiento de las políticas de tiempo real seleccionadas

- Deben modificarse las opciones de compilación para utilizar el runtime de MaRTE OS
- Los pragmas de configuración deben especificarse en el fichero `gnat.adc`
- Compilación desde el intérprete de comandos:

```
gnatmake --RTS=marte -gnatec=./gnat.adc nombre.adb
```

- Compilación desde el entorno GPS:
  - Incluir las opciones `--RTS=marte` y `-gnatec=./gnat.adc` a las propiedades del proyecto
  - *Project->Edit Project Properties->Switches->Gnatmake*