

Ejercicios de Tipos Abstractos de Datos: Unidades, Registros, Funciones, Procedimientos y Punteros

Información del Proyecto

Descripción	Detalles
Profesores	Sergio Cavero y Salvador Sanchez
Asignatura	Estructuras de Datos
Universidad	Universidad Rey Juan Carlos
Licencia	CC BY-NC-SA 4.0

Ejercicio 1: Números Complejos

En este ejercicio, trabajaremos con números complejos. Para ello, se proporcionan dos archivos Pascal:

1. [uComplejo.pas](#)
2. [tad_ej1_num_complejo.pas](#)

El archivo [tad_ej1_num_complejo.pas](#) utiliza la unidad [uComplejo.pas](#), la cual deberán implementar. A continuación, se detallan los registros, funciones y procedimientos que SE deben implementar en [uComplejo.pas](#).

Registros y variables

Define un registro [TComplejo](#) que represente un número complejo. Este registro debe tener dos campos:

- [r](#): de tipo [Double](#), que representa la parte real del número complejo.
- [i](#): de tipo [Double](#), que representa la parte imaginaria del número complejo.

Funciones y Procedimientos a Implementar

1. [CrearComplejo](#)

```
procedure CrearComplejo(var c: TComplejo; r: Double; i: Double);
```

Este procedimiento crea un número complejo a partir de sus partes real e imaginaria.

2. [SetParteReal](#)

```
procedure SetParteReal(var c: TComplejo; r: Double);
```

Este procedimiento establece la parte real de un número complejo.

3. SetParteImaginaria

```
procedure SetParteImaginaria(var c: TComplejo; i: Double);
```

Este procedimiento establece la parte imaginaria de un número complejo.

4. GetParteReal

```
function GetParteReal(c: TComplejo): Double;
```

Esta función devuelve la parte real de un número complejo.

5. GetParteImaginaria

```
function GetParteImaginaria(c: TComplejo): Double;
```

Esta función devuelve la parte imaginaria de un número complejo.

6. Sumar

```
procedure Sumar(var res: TComplejo; c1, c2: TComplejo);
```

Este procedimiento suma dos números complejos y devuelve el resultado. Para sumar dos números complejos, se suman las partes reales y las partes imaginarias por separado. El resultado se almacena en el número complejo **res** que se pasa por referencia.

7. Restar

```
procedure Restar(var res: TComplejo; c1, c2: TComplejo);
```

Esta función resta dos números complejos y devuelve el resultado. Para restar dos números complejos, se restan las partes reales y las partes imaginarias por separado. El resultado se almacena en el número complejo **res** que se pasa por referencia.

8. Multiplicar

```
procedure Multiplicar(var res: TComplejo; c1, c2: TComplejo);
```

Esta función multiplica dos números complejos y devuelve el resultado. Para multiplicar dos números complejos, se utilizan las siguientes fórmulas:

$$(a + bi) * (c + di) = (a*c - b*d) + (a*d + b*c)i$$

El resultado se almacena en el número complejo `res` que se pasa por referencia.

9. Dividir

```
procedure Dividir(var res: TComplejo; c1, c2: TComplejo);
```

Esta función divide dos números complejos y devuelve el resultado. Para dividir dos números complejos, se utilizan las siguientes fórmulas:

$$(a + bi) / (c + di) = ((a*c + b*d) / (c^2 + d^2)) + ((b*c - a*d) / (c^2 + d^2))i$$

El resultado se almacena en el número complejo `res` que se pasa por referencia.

10. Potencia

```
procedure Potencia(var res: TComplejo; c: TComplejo; exponente: Integer);
```

Esta función eleva un número complejo a una potencia entera y devuelve el resultado. Para elevar un número complejo a una potencia entera... investiga!!.

El resultado se almacena en el número complejo `res` que se pasa por referencia.

Implementación

Deben implementar todas estas funciones y procedimientos en el archivo `uComplejo.pas`. Una vez implementados, ejecuta el programa `tad_ej1_num_complejo.pas` para verificar su correcto funcionamiento.

Ejercicio 2: Métodos de Pago

En este ejercicio, trabajaremos con diferentes métodos de pago. Para ello, se proporcionan cuatro archivos Pascal:

1. `uPagoTarjeta.pas`
2. `uPagoTransferencia.pas`
3. `uPagoBizum.pas`

4. `tad_ej2_pagar.pas`

El archivo `tad_ej2_pagar.pas` utiliza las unidades `uPagoTarjeta.pas`, `uPagoTransferencia.pas` y `uPagoBizum.pas`, las cuales deberán implementar. A continuación, se detallan los procedimientos y funciones que deben implementar en cada una de estas unidades.

Procedimientos y Funciones a Implementar

1. `IniciarPago`

```
procedure IniciarPago;
```

Este procedimiento muestra un mensaje de inicio de pago y solicita la información necesaria para realizar el pago. La información solicitada varía según el método de pago:

- Para `uPagoTarjeta.pas`, solicita el número de tarjeta.
- Para `uPagoTransferencia.pas`, solicita el número de cuenta IBAN.
- Para `uPagoBizum.pas`, solicita el número de teléfono.

2. `RealizarPago`

```
procedure RealizarPago(<<tipo-de-pago>>: string; monto: real);
```

Este procedimiento realiza el pago por el monto especificado. La implementación varía según el método de pago:

- Para `uPagoTarjeta.pas`, muestra un mensaje indicando que se está realizando el pago con tarjeta. En este caso `<<tipo-de-pago>>` es "tarjeta".
- Para `uPagoTransferencia.pas`, muestra un mensaje indicando que se está realizando una transferencia bancaria con el IBAN proporcionado. En este caso `<<tipo-de-pago>>` es "iban".
- Para `uPagoBizum.pas`, muestra un mensaje indicando que se está realizando un pago por Bizum. En este caso `<<tipo-de-pago>>` es "telefono".

3. `ValidarPago`

```
function ValidarPago(<<tipo-de-pago>>: string): boolean;
```

Esta función valida la información proporcionada para el pago. La validación varía según el método de pago:

- Para `uPagoTarjeta.pas`, valida que el número de tarjeta tenga 16 dígitos. En este caso `<<tipo-de-pago>>` es "tarjeta".
- Para `uPagoTransferencia.pas`, valida que el número de cuenta IBAN tenga 24 caracteres. En este caso `<<tipo-de-pago>>` es "iban".

- Para **uPagoBizum.pas**, valida que el número de teléfono tenga 9 dígitos. En este caso <<tipo-de-pago>> es "telefono".

Implementación

Deben implementar todas estas funciones y procedimientos en los archivos **uPagoTarjeta.pas**, **uPagoTransferencia.pas** y **uPagoBizum.pas**. Una vez implementados, ejecuta el programa **tad_ej2_pagar.pas** para verificar su correcto funcionamiento.

El archivo **tad_ej2_pagar.pas** contiene el programa principal que utiliza las tres unidades de pago. Este programa solicita al usuario que introduzca una cantidad a pagar y seleccione un método de pago. Luego, realiza el pago utilizando el método seleccionado y valida si el pago fue exitoso.

Ejercicio 3: Gestión Académica

En este ejercicio, trabajaremos con la gestión académica de asignaturas y personas. Para ello, se proporcionan cuatro archivos Pascal:

1. **uPersona.pas**
2. **uAsignaturaEvalContinua.pas**
3. **uAsignaturaEvalFinal.pas**
4. **tad_ej3_gestion_academica.pas**

El archivo **tad_ej3_gestion_academica.pas** utiliza las unidades **uPersona.pas**, **uAsignaturaEvalContinua.pas** y **uAsignaturaEvalFinal.pas**, las cuales deberán implementar. A continuación, se detallan los registros, funciones y procedimientos que deben implementar en cada una de estas unidades.

Unidad **uPersona.pas**

Registros y variables

Define un registro **TPersona** que represente una persona. Este registro debe tener los siguientes campos:

- **Nombre**: de tipo **String**, que representa el nombre de la persona.
- **Apellido**: de tipo **String**, que representa el apellido de la persona.
- **DNI**: de tipo **String**, que representa el DNI de la persona.
- **FechaNacimiento**: de tipo **String**, que representa la fecha de nacimiento de la persona.
- **Rol**: de tipo **TRol**, que representa el rol de la persona (puede ser **rProfesor** o **rAlumno**).

Crea un tipo puntero **PPersona** que apunte a variables de tipo registro **TPersona**.

Funciones y Procedimientos a Implementar

1. **CrearPersona**

```
procedure CrearPersona(var persona: TPersona; Nombre, Apellido, DNI,  
FechaNacimiento: String; Rol: TRol);
```

Este procedimiento crea una persona a partir de los datos proporcionados. La persona es inicializada con los datos proporcionados y el rol especificado. Para ello la persona es pasada por referencia.

2. MostrarPersona

```
procedure MostrarPersona(P: TPersona);
```

Este procedimiento muestra la información de una persona.

Unidad uAsignaturaEvalContinua.pas

Registros y variables

Define un registro **TAsignatura** que represente una asignatura con evaluación continua. Este registro debe tener los siguientes campos:

- **Profesor**: de tipo **PPersona**, que representa el profesor de la asignatura.
- **Alumnos**: un array de **PPersona** que representa los alumnos de la asignatura. Puedes crear una constante **MAX_ALUMNOS** para definir el tamaño máximo del array.
- **NotasPrimerParcial**: un array de **Real** que representa las notas del primer parcial.
- **NotasSegundoParcial**: un array de **Real** que representa las notas del segundo parcial.
- **NumAlumnos**: de tipo **Integer**, que representa el número de alumnos en la asignatura.

Funciones y Procedimientos a Implementar

1. InicializarAsignatura

```
procedure InicializarAsignatura(var A: TAsignatura; Prof: PPersona);
```

Este procedimiento inicializa una asignatura con el profesor proporcionado.

2. AnadirAlumno

```
function AnadirAlumno(var A: TAsignatura; Alumno: PPersona): Boolean;
```

Esta función añade un alumno a la asignatura.

3. Evaluar

```
procedure Evaluar(var A: TAsignatura; AlumnoDNI: string; Nota: Real; Parcial: Integer);
```

Este procedimiento evalúa a un alumno en un parcial específico y deberás poner la nota que nos pasan en el array correspondiente. Dado que hay dos parciales, el parámetro **Parcial** puede ser 1 o 2. No nos

preocuparemos por la validación de los parámetros en este ejercicio. Vamos a asumir que los parámetros son correctos.

4. CalcularNotaFinal

```
function CalcularNotaFinal(A: TAsignatura; AlumnoDNI: string): Real;
```

Esta función calcula la nota final de un alumno en la asignatura. En este caso, la nota final se calcula como el promedio de las notas de los dos parciales. No nos preocuparemos por determinar si un alumno ha sido evaluado en ambos parciales. Vamos a asumir que todos los alumnos han sido evaluados en ambos parciales.

Unidad uAsignaturaEvalFinal.pas

Registros y variables

Define un registro **TAsignatura** que represente una asignatura con evaluación final. Este registro debe tener los siguientes campos:

- **Profesor**: de tipo **PPersona**, que representa el profesor de la asignatura.
- **Alumnos**: un array de **PPersona** que representa los alumnos de la asignatura.
- **Notas**: un array de **Real** que representa las notas finales de los alumnos.
- **NumAlumnos**: de tipo **Integer**, que representa el número de alumnos en la asignatura.

Funciones y Procedimientos a Implementar

1. InicializarAsignatura

```
procedure InicializarAsignatura(var A: TAsignatura; Prof: PPersona);
```

Este procedimiento inicializa una asignatura con el profesor proporcionado.

2. AnadirAlumno

```
function AnadirAlumno(var A: TAsignatura; Alumno: PPersona): Boolean;
```

Esta función añade un alumno a la asignatura.

3. Evaluar

```
procedure Evaluar(var A: TAsignatura; AlumnoDNI: string; Nota: Real);
```

Este procedimiento evalúa a un alumno con una nota final y deberás poner la nota que nos pasan en el array correspondiente.

4. CalcularNotaFinal

```
function CalcularNotaFinal(A: TAsignatura; AlumnoDNI: string): Real;
```

Esta función calcula la nota final de un alumno en la asignatura. En este caso, la nota final de su único examen es la nota final del alumno.

Implementación

Deberás implementar todas estas funciones y procedimientos en los archivos `uPersona.pas`, `uAsignaturaEvalContinua.pas` y `uAsignaturaEvalFinal.pas`. Una vez implementados, ejecuta el programa `tad_ej3_gestion_academica.pas` para verificar su correcto funcionamiento.

El archivo `tad_ej3_gestion_academica.pas` contiene el programa principal que utiliza las tres unidades. Este programa crea un profesor y dos alumnos, inicializa las asignaturas, añade los alumnos a las asignaturas, evalúa a los alumnos y muestra los resultados finales.

Cuestiones adicionales

1. ¿Qué ventajas tiene utilizar registros para representar a las personas y asignaturas en lugar de variables individuales?
2. ¿Qué ventajas tiene utilizar funciones y procedimientos para realizar operaciones sobre las personas y asignaturas en lugar de realizar las operaciones directamente en el programa principal?
3. ¿Qué otros campos o funcionalidades podrían añadirse a las unidades `uPersona.pas`, `uAsignaturaEvalContinua.pas` y `uAsignaturaEvalFinal.pas` para mejorar la gestión académica?
4. ¿Qué complejidad algorítmica tienen las funciones implementadas?