Práctico 11 - Arreglo con Tope y Registros Variantes

Programación 1
InCo - Facultad de Ingeniería, Udelar

1. Dadas las siguientes declaraciones:

```
const
   CANT_PERS = ... ;{valor entero mayor estricto a 0}
   MAX_CADENA = ... ;{valor entero mayor estricto a 0}
type
   Cadena = record
                letras : array [1..MAX_CADENA] of char;
                largo : 0..MAX_CADENA;
            end;
   Persona = record
                nombre : Cadena;
                edad : 0..120;
                estado : (casado, soltero, divorciado);
                salario : Real;
                exenciones : 0..maxint;
             end;
var
   juanita : Persona;
   grupo : array [1..CANT_PERS] of Persona;
```

   Determine cuáles de las siguientes instrucciones son válidas:

   I) `grupo[1] := juanita`

   II) `grupo[1].nombre := 'juanita'`

   III) `read (grupo[1].estado)`

   IV) `with grupo do writeln (nombre)`

   V)
```
with grupo[100] do
begin
   read (edad)
end
```

   VI)
```
with juanita do
begin
   nombre := grupo[50].nombre;
   salario := grupo[1].salario
end
```

2. Una sociedad genealógica tiene un arreglo de registros con datos de personas. Los datos son, para cada persona, su nombre, la fecha de su nacimiento y los índices en el arreglo de los registros de los datos de su padre y de su madre. Cada nombre de persona aparece una única vez. Se definen los siguientes tipos de datos para representar esta realidad:

```
const
    MAXPERSONAS = ...; {valor entero mayor estricto a 0}
    MAXCAD      = ...; {valor entero mayor estricto a 0}

type
   Cadena = record
               letras : array [1..MAXCAD] of char;
```

```
            largo : 0..MAXCAD;
         end;
Fecha = record
            dia : 1..31;
            mes : 1..12;
            anio : 0..maxint;
        end;
Persona = record
                nombre : Cadena;
                fechNac : Fecha;
                indMadre, indPadre : 0..MAXPERSONAS;
            end;
Familia = record
             pers : array[1..MAXPERSONAS] of Persona;
             tope : 0..MAXPERSONAS;
          end;
```

Los campos `indMadre` e `indPadre` de una persona contienen el índice de los registros de la madre y del padre en el arreglo `pers` de la familia, o cero en caso de no disponer de la información correspondiente. En caso de ser distinto de cero, se asume que es un índice válido del arreglo.

(a) Escriba la función `cadenasIguales` que, dadas dos cadenas, determina si son iguales.

```
function cadenasIguales (cad1, cad2 : Cadena): Boolean;
```

(b) Escriba el procedimiento `desplegarCadena` que, dada una cadena, la despliega en la salida.

```
procedure desplegarCadena (cad: Cadena);
```

(c) Escriba el procedimiento `antepasados` que, dado el nombre de una persona en el parámetro `usted` y una familia en el parámetro `historia`, despliegue en la salida los nombres y fechas de nacimiento del padre y de la madre de la persona de nombre `usted` (si es que se dispone de la información correspondiente). En caso de que la persona de nombre usted no esté registrada, no se desplegará nada.

```
procedure antepasados (usted : Cadena; historia : Familia);
```

(d) Escriba un programa principal que permita probar los subprogramas de las partes anteriores, declarando toda variable que sea necesaria. También debe definir cualquier otro subprograma auxiliar que necesite para carga y/o exhibición de datos.

3. Dado el tipo `Cadena` definido para almacenar hasta `MAX` caracteres:

```
const
    MAX = ...; {valor mayor estricto a 0}

type
   Cadena = record
               letras : array [1..MAX] of char;
               largo : 0..MAX;
            end;
```

(a) Escriba un procedimiento llamado `cargarCadena` que tenga como parámetro una cadena de caracteres y la cargue con caracteres leidos de la entrada estándar. Al ingresar los caracteres, se utilizará un punto (.) para marcar el fín de la cadena (el cual no forma parte de la misma, solo será tipeado para marcar su finalización). En caso de que se ingresen más de `MAX` caracteres, solamente se cargarán los primeros `MAX`, descartando los restantes.

(b) Escriba una función llamada `contarOcurrencias` que tenga como parámetro una cadena de caracteres llamada `frase` y una variable de tipo carácter llamada `letra`. La función devuelve el número de apariciones del carácter `letra` en la cadena `frase`.

(c) Escriba una función llamada `existeVocal` que tenga como parámetro una cadena de caracteres y determine si en la cadena hay o no alguna letra vocal. La función devuelve `true` en caso afirmativo y `false` en caso negativo.

4. Se desea implementar un procedimiento que calcule las raíces de la ecuación cuadrática $ax^2 + bx + c = 0$ donde a, b, y c son coeficientes reales. El procedimiento debe determinar si la ecuación tiene dos raíces reales

y distintas, una raíz real doble o dos raíces complejas conjugadas. Para devolver el resultado, se definirá un tipo de datos `TipoRaices` mediante una estructura de registro con variante, de modo tal que contemple los tres casos posibles. El cabezal del procedimiento es el siguiente:

```
procedure raices (a,b,c : real; var r : TipoRaices);
```

(a) Defina el tipo `TipoRaices` utilizando la estructura de registro con variante. Debe definir también cualquier tipo de datos auxiliar que pueda necesitar.

(b) Implemente el procedimiento `raices`, de acuerdo al comportamiento descrito.

5. Se considera el tipo de datos `Nerr` que es la unión de los números naturales (N) con el conjunto `Err`. El conjunto `Err` se define como `{diverr, reserr, argerr}`, donde `diverr` es el error de la división por cero, `reserr` es el error de la resta con resultado negativo y `argerr` es el error de cualquier operación donde alguno de sus argumentos no es natural.

   Se definen las siguientes operaciones sobre elementos del tipo `Nerr`:

   `division: Nerr x Nerr -> Nerr`

   - Si a pertenece a N y b pertenece a N$-\{0\}$ $\Rightarrow$ `division(a,b) = a DIV b`;
   - Si a pertenece a N y $b = 0$ $\Rightarrow$ `division(a,b) = diverr`;
   - Si a pertenece a Err o b pertenece a Err $\Rightarrow$ `division(a,b) = argerr`;

   `resta: Nerr x Nerr -> Nerr`

   - Si a pertenece a N, b pertenece a N y $a \geq b$ $\Rightarrow$ `resta(a,b) = a - b`;
   - Si a pertenece a N, b pertenece a N y $a < b$ $\Rightarrow$ `resta(a,b) = reserr`;
   - Si a pertenece a Err o b pertenece a Err $\Rightarrow$ `resta(a,b) = argerr`;

   `suma: Nerr x Nerr -> Nerr`

   - Si a pertenece a N y b pertenece a N $\Rightarrow$ `suma(a,b) = a + b`;
   - Si a pertenece a Err o b pertenece a Err $\Rightarrow$ `suma(a,b) = argerr`;

   `producto: Nerr x Nerr -> Nerr`

   - Si a pertenece a N y b pertenece a N $\Rightarrow$ `producto(a,b) = a * b`;
   - Si a pertenece a Err o b pertenece a Err $\Rightarrow$ `producto(a,b) = argerr`;

   (a) Defina el tipo de datos `Err`.

   (b) Defina el tipo de datos `Nerr`. Defina también cualquier tipo de datos auxiliar que pueda necesitar.

   (c) Escriba los siguientes procedimientos que implementan, respectivamente, las operaciones division, resta, suma y producto del tipo Nerr.

```
procedure division (a, b: Nerr; var resu: Nerr);
procedure resta (a, b: Nerr; var resu: Nerr);
procedure suma (a, b: Nerr; var resu: Nerr);
procedure producto (a, b :Nerr; var resu: Nerr);
```

6. Se desea implementar el producto de matrices de elementos de tipo `Nerr` (del ejercicio anterior). Las matrices a operar podrán tener diferentes dimensiones. Se sabe que si una matriz tiene dimensión `mxn`, entonces `m` y `n` son enteros positivos que nunca superarán constantes conocidas `Y` y `X` respectivamente.

   El producto entre matrices de tipo `Nerr` se define de manera análoga al producto de matrices de números naturales con la suma y el producto para el tipo `Nerr` dado en el ejercicio anterior.

   Sea la matriz `m1` de dimensión `m x n` y la matriz `m2` de dimensión `p x q`. Si `n = p`, entonces el producto `m1 x m2` tendrá dimensión `m x q`, en caso contrario diremos que el producto falla.

   (a) Defina el tipo `MNerr`, que representa las matrices de dimensiones `m x n` de tipo `Nerr`, para cualquier `m` entre 1 e Y, y para cualquier `n` entre 1 y X. Puede asumir que X e Y son constantes con valores mayores o iguales que 1. Además `MNerr` debe tener un valor de error `merr` para el caso en que el producto de matrices falle. Defina también cualquier tipo enumerado auxiliar que pueda necesitar.

   (b) Implemente el procedimiento `mprod`, el cual recibe dos matrices `m1` y `m2`, retornando en `resu` el producto `m1 x m2` o `merr` en caso de que dicho producto falle.

```
procedure mprod (m1, m2: MNerr; VAR resu: MNerr);
```

7. En una isla del Caribe una banda de piratas se gana la vida asaltando barcos mercantes. Anualmente en la isla se realiza un evento multitudinario llamado "la entrega de los premios Calavera". Para este año los piratas están pensando en entregar el premio *Calavera de oro* al pirata que haya conseguido más dinero asaltando barcos para la banda. Como usualmente los piratas discuten a quién le corresponden los premios en trifulcas interminables y sangrientas, este año la comisión directiva de la banda ha decidido informatizar el registro de logros de los piratas en los distintos asaltos, con la esperanza de terminar así con algunas de las discusiones sobre los créditos que le corresponden a cada pirata.

   Los logros de los piratas durante el año se representan mediante las siguientes declaraciones:

```
const
    MAXPIRATAS   = ...; {valor entero mayor estricto a 0}
    MAXASALTOS   = ...; {valor entero mayor estricto a 0}
    MAXDIGITOSCI = ...; {valor entero mayor estricto a 0}
    MAXCADENA    = ...; {valor entero mayor estricto a 0}

type
    TipoCadena = record
        letras: array [1..MAXCADENA] of char;
        tope:   0 .. MAXCADENA
    end;

    TipoCI = array [1..MAXDIGITOSCI] of '0'..'9';

    TipoFecha  = record
        dia: 1..31;
        mes: 1..12;
        anio: 0..maxint;
    end;

    TipoAsalto = record
        nombre_barco: TipoCadena;
        fecha: TipoFecha;
        botin: integer;
    end;

    ConjuntoAsaltos = record
        asaltos: array [1..MAXASALTOS] of TipoAsalto;
        tope: 0..MAXASALTOS
    end;

    TipoCausaMuerte = (asesinato, enfermedad, accidente);

    TipoPirata = record
        nombre: TipoCadena;
        ci: TipoCI;
        case estaVivo: boolean of
            true:  (asaltos: ConjuntoAsaltos);
            false: (causaMuerte: TipoCausaMuerte; fechaMuerte: TipoFecha)
    end;

    Banda = record
        pirata: array [1..MAXPIRATAS] of TipoPirata;
        tope: 0..MAXPIRATAS
    end;
```

   (a) Implemente la función `dineroObtenidoPorPirata` que calcula la suma de dinero obtenida por el pirata de cedula `CI`, en el año `anio` por la banda `b`. En caso de que el pirata se encuentre muerto o no se encuentre en la banda debe retornar **0**.

```
function dineroObtenidoPorPirata(pirata: TipoCI; anio: integer; b:Banda) : integer;
```

Se sugiere implementar primero las siguientes funciones auxiliares:

```
function ciIguales (ci1, ci2: TipoCI): boolean;
(* Retorna true si ci1 y ci2 son iguales y false en caso contrario*)

function contarDinero (ca: ConjuntoAsaltos; anio:integer): integer;
(* Retorna la suma del dinero obtenido en los asaltos del conjunto ca realizados
durante el año anio. *)
```

(b) Implemente un procedimiento el cual, dada una banda de piratas `piratas` y un año `anio`, retorna en el parámetro `piratasMerecedores` las cédulas de los piratas vivos merecedores del premio *Calavera de Oro*. Tener en cuenta que varios piratas pueden coincidir en la cantidad de dinero obtenido para la banda.

```
procedure hallarGanadores (piratas:Banda; anio:integer; var piratasMerecedores: ConjuntoCIs);
```

El tipo ConjuntoCIs se declara como sigue:

```
type
    ConjuntoCIs = record
        cedulas: array [1..MAXPIRATAS] of TipoCI;
        tope: 0..MAXPIRATAS
    end;
```

8. Se desea trabajar con una aritmética de naturales de hasta 100 dígitos. Los enteros de *Pascal* no soportan dicha aritmética, por lo que se piensa utilizar la siguiente representación de naturales basada en arreglos con tope, de tal manera que las unidades queden en el primer lugar del arreglo, las decenas en el segundo, y así hasta que el dígito más significativo quede en el tope:

```
const
    MaxDig = ...; {valor entero mayor estricto a 0};
type
    Digito = 0..9;
    Natural = record
        digitos : array[1..MaxDig] of Digito;
        tope : 0..MaxDig;
    end;
```

(a) Implemente la suma de Naturales representados en términos de la estructura anterior. Utilice el siguiente cabezal:

```
procedure sumaNaturales (a, b : Natural; var c : Natural);
```

(b) Analice cuál sería la dificultad si la representación fuera al revés, es decir que el dígito menos significativo quede en el tope y el más significativo en el primer lugar del arreglo.

9. Deseamos hallar la descomposición de un número natural ($n <= N$) en factores primos.

Ej: $8 = 2 * 2 * 2$, $36 = 2 * 2 * 3 * 3$, $37 = 37$, $20 = 2 * 2 * 5$, $105 = 3 * 5 * 7$, $600 = 2 * 2 * 2 * 3 * 5 * 5$

Vamos a suponer que dicha descomposición no involucra más de `M` números primos, donde `M` se supone una constante previamente definida en el valor adecuado. Deseamos almacenar la descomposición en un arreglo con tope:

```
type Descomp = record
                  factores : array [1..M] of Factor;
                  tope : 0..M;
               end;
```

el cual contiene los factores ordenados en forma ascendente y sus respectivos exponentes.

```
type Factor = record
              primo : 1..N;
              case multiple : boolean of
                true: (exponente : 2..N);
                false : ();
              end;
```

Los factores se representan mediante un registro con variante (record-case). Si el exponente de un factor es 1, entonces se almacena simplemente el número primo. Si por el contrario, el exponente del factor es mayor que 1, entonces se almacena el número primo y el valor del exponente.

(a) Implemente un procedimiento que reciba como entrada un número cualquiera mayor que 1 y retorne su descomposición en factores primos.

Utilizar el siguiente cabezal:

```
Procedure factorizacion (num : Integer; var listaFac : Descomp);
```

(b) Escriba un programa principal que permita probar el subprograma de la parte anterior, declarando toda variable que sea necesaria. También debe definir cualquier otro subprograma auxiliar que necesite para carga y/o exhibición de datos en la salida estándar.