EJEMPLOS

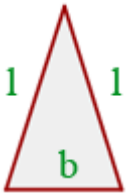
Un triángulo es un polígono de 3 lados que, en función de sus lados, puede clasificarse en equilátero, isósceles o escaleno. Considerando esto, defina e implemente:

Triángulo Equilátero

Perímetro

$$p = 3 \times lado$$

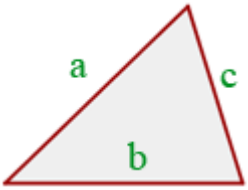
Triángulo Isósceles



Perímetro

$$p = 2 \times lado + b$$

Triángulo Escaleno



Perímetro

$$p = a + b + c$$

Semiperímetro $sp = \frac{p}{2}$

Área del triángulo $a = \sqrt{sp \times (sp - lado_1) \times (sp - lado_2) \times (sp - lado_3)}$

(fórmula de Herón)

a) el TDA triángulo utilizando arreglos,

b) el TDA triángulo utilizando registros y

c) las operaciones *crear triángulo*, *calcular perímetro* y *calcular área* para las definiciones de los ítems a) y b).

Especificación del TDA Triángulo

El TDA triángulo comprende 3 valores reales que representan las longitudes de los lados de un triángulo. Sobre este tipo pueden definirse las operaciones:

- Crear triángulo: dados 3 valores reales, devuelve un triángulo.
- Calcular perímetro: dado un triángulo, devuelve el valor de su perímetro.
- Calcular área: dado un triángulo, devuelve el valor de su superficie.

Implementación del TDA Triángulo con Arreglos

De acuerdo a la especificación, el tipo triángulo se representa mediante 3 valores reales que corresponden a las longitudes de los lados de la figura. Para implementar el concepto es necesario elegir una estructura de datos que permita almacenar y manipular estos 3 valores.

A continuación se presenta la implementación del tipo triángulo realizada mediante arreglos. En este caso, se utiliza un arreglo de 3 posiciones reales que representan los lados del triángulo; asimismo se desarrollan las operaciones correspondientes a la implementación elegida.

Definición de la estructura de datos que representa al tipo triángulo:

En Pseudodcódigo

```
t_triangulo=ARREGLO [1..3] de REALES
```

En C/C++

```
typedef float t_triangulo[3];
```

Operaciones definidas sobre el tipo triángulo:

En Pseudodcódigo

```
PROCEDIMIENTO crear(E/S t:t_triangulo)
INICIO
  ESCRIBIR "Ingrese lados: "
  LEER t[1],t[2],t[3]
FIN
```

En C/C++

```
void crear(t_triangulo &t)
{ cout << "Ingrese lados: ";
  cin >> t[0] >> t[1] >> t[2];
}
```

La operación *crear*, según se indicó en la especificación, permite crear un dato triángulo a partir de 3 valores. Por ello, esta operación se implementa mediante un *procedimiento* que permite almacenar los valores de longitud ingresados por el usuario.

```
FUNCIÓN perimetro(E t:t_triangulo):REAL
INICIO
  perimetro←t[1]+t[2]+t[3]
FIN
```

```
float perimetro(t_triangulo t)
{
  return t[0]+t[1]+t[2];
}
```

La operación *perímetro*, según se indicó en la especificación, permite calcular el perímetro de un triángulo (un valor real). Por ello, esta operación se implementa mediante una *función* real que recibe como entrada un dato tipo triangulo. La función utiliza los valores almacenados en la estructura para realizar el cálculo (fórmula general) y generar el resultado de salida.

```
FUNCIÓN area(E t:t_triangulo):REAL
VARIABLES
  sp:REAL
  a:REAL
INICIO
  sp←perimetro(t)/2
 a←sp*(sp-t[1])*(sp-t[2])*(sp-t[3])
 a←a^0.5
 area←a;
FIN
```

```
float area(t_triangulo t)
{ float sp;
  float a;
  sp=perimetro(t)/2;
  a=sp*(sp-t[0])*(sp-t[1])*(sp-t[2]);
  a=pow(a,0.5);
  return a;
}
```

La operación *area*, según se indicó en la especificación, permite calcular la superficie de un triángulo (un valor real). Por ello, esta operación se implementa mediante una *función* real que recibe como entrada un dato tipo triangulo. La función utiliza los valores almacenados en la estructura para realizar el cálculo (fórmula de Herón) y generar el resultado de salida.

Implementación del TDA Triángulo con Registros

Como ya se mencionó, para implementar el concepto triángulo es necesario elegir una estructura de datos que permita almacenar y manipular estos 3 valores. Una alternativa a la implementación presentada en el ejemplo anterior, puede plantearse con estructuras de registro.

A continuación se muestra la definición del tipo triángulo y, a modo ilustrativo, las operaciones *crear* y *area* adaptadas a la nueva implementación.

Definición de la estructura de datos que representa al tipo triángulo:

```
En Pseudodcódigo
t_triangulo=REGISTRO
              a:REAL
              b:REAL
              c:REAL
            FIN_REGISTRO
```

```
En C/C++
typedef struct t_triangulo {
                              float a;
                              float b;
                              float c;
                              };
```

Operaciones definidas sobre el tipo triángulo:

```
En Pseudodcódigo
PROCEDIMIENTO crear(E/S t:t_triangulo)
INICIO
  ESCRIBIR "Ingrese lados: "
 LEER t.a,t.b,t.c
FIN
```

```
En C/C++
void crear(t_triangulo &t)
{ cout << "Ingrese lados: ";
  cin >> t.a >> t.b >> t.c;
}
```

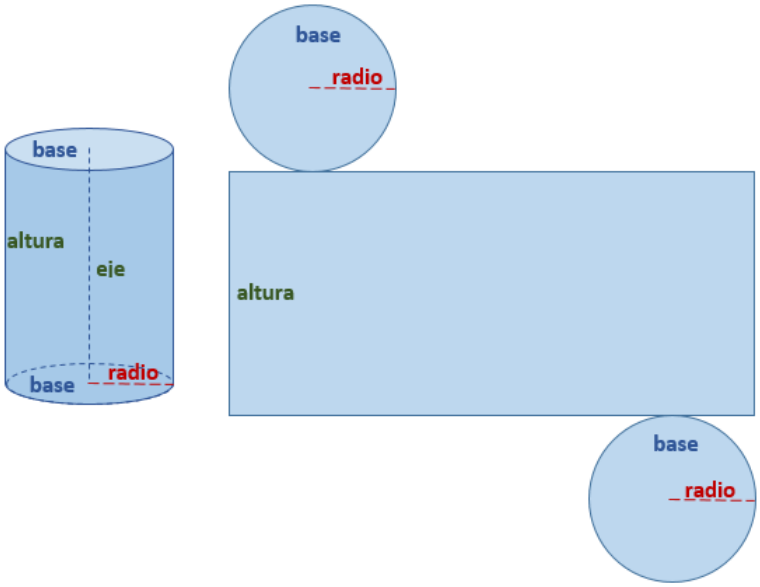
```
FUNCIÓN area(E t:t_triangulo):REAL
VARIABLES
  sp:REAL
  a:REAL
INICIO
  sp←perimetro(t)/2
  a←sp*(sp-t.a)*(sp-t.b)*(sp-t.c)
  a←a^0.5
  area←a;
FIN
```

```
float area(t_triangulo t)
{ float sp;
  float a;
  sp=perimetro(t)/2;
  a=sp*(sp-t.a)*(sp-t.b)*(sp-t.c);
  a=pow(a,0.5);
  return a;
}
```

Como puede observarse en las operaciones no se modificaron los datos que se reciben o los resultados que se generan, ya que sin importar la estructura de datos elegida para la implementación el concepto y las operaciones no varían. Los cambios sólo se reflejan en la forma en la que se accede a los datos almacenados, ya sean las posiciones de un arreglo o los campos de un registro.

EJERCICIOS

1) Un cilindro circular recto es un cuerpo geométrico cuya superficie se forma por los puntos a una distancia fija de un determinado segmento de línea (eje del cilindro) y 2 planos perpendiculares a este eje. Sabiendo esto, implemente el *TDA cilindro* y las operaciones: crear cilindro, calcular área, calcular volumen y mostrar cilindro.



Realice la implementación mediante

a) ARREGLOS

b) REGISTROS.

Nota: CONSIDERE QUE LA OPERACIÓN *CREAR CILINDRO* COMPRUEBA QUE LOS VALORES INGRESADOS PARA DEFINIR EL *CILINDRO* NO SEAN NEGATIVOS O CERO.

2) En geometría, los ángulos se miden en grados, minutos y segundos. Los grados oscilan entre 0° y 360°, mientras que minutos y segundos oscilan entre 0 y 59. En función de estos valores, los ángulos se clasifican en: agudos (menores a 90°), rectos (90° 0' 0''), obtusos (menores a 180°), llanos (180° 0' 0'') y completos (mayores a 180°). Sabiendo esto, implemente el *TDA ángulo* y las operaciones *crear_angulo*, *validar_angulo*, *tipo_angulo* y *radianes*. La operación *crear_angulo* almacena los datos correspondientes a un ángulo, *validar_angulo* verifica que los componentes del TDA tengan valores correctos, *tipo_angulo* indica de qué tipo de ángulo se trata y *radianes* obtiene el equivalente en radianes del ángulo especificado. Realice la implementación utilizando a) ARREGLOS y b) REGISTROS.

3) Un triángulo es un polígono de 3 lados que, en función del valor de sus lados, puede clasificarse en equilátero, isósceles o escaleno. En particular si el triángulo tiene un ángulo recto se denomina triángulo rectángulo y verifica el conocido teorema de Pitágoras. Defina e implemente el TDA triángulo con las siguientes operaciones:
   - Crear triángulo: dados 3 valores reales, devuelve un triángulo.
   - Validar triángulo: dado un triángulo, determina que el valor de sus lados sea correcto. Téngase en cuenta que para que un triángulo exista (el valor de sus lados sea válido) debe verificarse que la suma de cualquier par de sus lados sea mayor que el lado restante.

- Tipo: identifica el tipo de triángulo (equilátero, isósceles o escaleno).
- Pitágoras: determina si el triángulo es un triángulo rectángulo.

Defina el *TDA triángulo* utilizando para su implementación a) ARREGLOS y b) REGISTROS.

4) Considerando que el concepto *hora* tiene 3 componentes (horas, minutos y segundos), implemente el *TDA hora* con las siguientes operaciones:
   - Crear hora: dados 3 valores enteros, devuelve una hora.
   - Validar hora: dada una hora, determina si ésta contiene datos válidos.
   - Calcular minutos: dada una hora, obtiene su equivalente en minutos.
   - Calcular segundos: dada una hora, obtiene su equivalente en segundos.
   - Comparar horas: dadas 2 horas, determina si la primera es mayor que la segunda, la segunda es mayor que la primera o si son iguales. La comparación se realiza componente a componente.
   - Sumar segundos: dada una hora, suma a ésta una cantidad de segundos. La operación debe garantizar que el resultado sea válido.
   - Sumar minutos: dada una hora, suma a ésta una cantidad de minutos. La operación debe garantizar que el resultado sea válido.
   - Mostrar fecha: dada una fecha, presenta su contenido por pantalla.

   Defina el *TDA fecha* utilizando para su implementación a) ARREGLOS y b) REGISTROS.

5) Un pixel ("*picture element*") es el componente más pequeño que forma una imagen digital (gráfico, fotografía o fotograma de vídeo). Un pixel se forma de 3 puntos de color (rojo, azul y verde) cuya intensidad determina el color resultante. Defina e implemente el TDA pixel con las siguientes operaciones:
   - Crear pixel: dados 3 valores enteros, devuelve un pixel.
   - Validar pixel: dado un pixel, determina que el valor de sus puntos sea correcto. Considere que los puntos asumen valores sólo en el rango [0, 255].
   - Mostrar pixel dado un pixel, muestra el valor de sus componentes.
   - Crear blanco: dado un pixel, asigna a sus puntos los valores apropiados para generar blanco.
   - Crear negro: dado un pixel, asigna a sus puntos los valores apropiados para generar negro.

   Defina el *TDA pixel* utilizando para su implementación a) ARREGLOS y b) REGISTROS.

6) Un número racional puede comprenderse como el cociente de 2 números enteros, donde el primero de ellos representa el numerador y el segundo el denominador, siendo el denominador siempre distinto de cero. Sobre números racionales pueden aplicarse las siguientes operaciones:
   - Crear Racional: dados 2 números enteros, devuelve un racional.
   - Igualdad de Racionales: dados 2 racionales retorna verdadero solamente si los racionales son iguales.
   - Sumar Racionales: dados 2 racionales, devuelve otro racional que es la suma de los dados.
   - Restar Racionales: dados 2 racionales, devuelve otro racional que es la diferencia de los dados.
   - Producto Racionales: dados 2 racionales, devuelve otro racional que es el producto de los dados.
   - Cociente Racionales: dados 2 racionales, devuelve otro racional que es el cociente de los dados.
   - Potencia Racionales: dado un valor racional y un exponente entero positivo o cero, devuelve otro racional a cuyo numerador y denominador se aplicó la potencia.
   - Decimal: dado un número racional, devuelve el valor decimal correspondiente.
   - Mostrar Racional: dado un valor racional, presenta su contenido por pantalla.

   Considerando la definición y operaciones anteriores, implemente el *TDA racional* utilizando a) ARREGLOS y b) REGISTROS.

7) El concepto de polinomio se puede entender como una expresión algebraica entera, donde las letras y coeficientes están relacionados únicamente por adición, sustracción, multiplicación y potenciación con exponente natural. Las operaciones definidas para polinomios son:
   - Crear Polinomio: dados 3 números enteros, devuelve un polinomio.
   - Sumar Polinomios: dados 2 polinomios, devuelve otro polinomio que es la suma de los dados
   - Restar Polinomios: dados 2 polinomios, devuelve otro polinomio que es la diferencia de los dados.
   - Producto Polinomio/escalar: dados un polinomio y un escalar, devuelve otro polinomio que es el producto de los dados.
   - Cociente Polinomio/escalar: dados un polinomio y un escalar, devuelve otro polinomio que es el producto de éstos.
   - Grado de Polinomio: dado un polinomio, devuelve el grado del mismo.
   - Raíces de Polinomio: dado un polinomio, devuelve el valor de sus raíces si éstas existen.
   - Mostrar Polinomio: dado un polinomio, presenta su contenido por pantalla.

Considerando la definición y operaciones anteriores, implemente el TDA polinomio teniendo en cuenta que sólo se trabajará con polinomios de la forma:

$$P = a \times x^2 + b \times x + c$$

Para la implementación utilice a) ARREGLOS y b) REGISTROS.

8) Un número complejo puede ser definido como un par ordenado de números reales, donde la primera componente representa la parte real del número complejo mientras que la segunda componente corresponde a la parte imaginaria. Sobre números complejos pueden aplicarse las siguientes operaciones:
   - Crear Complejos: dados 2 números reales, devuelve un complejo.
   - Igualdad de Complejos: dados 2 complejos devuelve un valor booleano que es verdadero solamente si los complejos son iguales (la comparación se realiza componente a componente).
   - Sumar Complejos: dados 2 complejos, devuelve otro complejo que es la suma de los dados.
   $$(a, b) + (c, d) = (a + c\,, b + d)$$
   - Restar Complejos: dados 2 complejos, devuelve otro complejo que es la diferencia de los dados.
   $$(a, b) - (c, d) = (a - c, b - d)$$
   - Producto Escalar: dados un complejo y un valor escalar, devuelve otro complejo cuyas componentes se multiplicaron por el escalar.
   $$(a, b) \times N = (a \times N, b \times N)$$
   - Producto Complejos: dados 2 complejos, devuelve otro complejo que es el producto de los dados.
   $$(a, b) \times (c, d) = (a \times c - b \times d, a \times d + b \times c)$$
   - Conjugado Complejo: dado un complejo, devuelve su conjugado (la parte imaginaria cambia de signo).
   $$\overline{(a, b)} = (a, -b)$$
   - Módulo Complejo: dado un complejo, devuelve el valor de su módulo.
   $$|(a, b)| = \sqrt{a^2 + b^2}$$
   - Mostrar Complejo: dado un valor complejo, presenta su contenido por pantalla.

   Implemente el *TDA complejo* y sus operaciones utilizando a) ARREGLOS y b) REGISTROS.

9) Un conjunto se define como una colección homogénea de elementos (no repetidos) sobre el que pueden aplicarse las siguientes operaciones:
   - Crear Conjunto: genera un conjunto vacío.
   - Agregar elementos: dado un conjunto y un elemento permite agregar éste último al conjunto (sin repeticiones).
   - Intersección de Conjuntos: dados 2 conjuntos, devuelve otro conjunto que contiene sólo los elementos comunes a los conjuntos dados.

- Unión de Conjuntos: dados 2 conjuntos, devuelve otro conjunto que contiene todos los elementos (no repetidos) de los conjuntos dados.
- Pertenencia: dados un conjunto y un elemento, determina si el valor se encuentra presente o no en el conjunto.
- Diferencia de Conjuntos: dados 2 conjuntos, devuelve otro que contiene los elementos del primer conjunto que no se encuentran en el segundo.
- Diferencia Simétrica de Conjuntos: dados 2 conjuntos, devuelve otro que contiene los elementos del primer y segundo que éstos no tengan en común.
- Mostrar Conjunto: dado un conjunto, presenta su contenido por pantalla.

Considerando esto, implemente el *TDA conjunto* utilizando a) ARREGLOS y b) LISTAS ENLAZADAS.

10) Un programa es, básicamente, un conjunto de instrucciones y datos que permiten procesar información para resolver diversas situaciones o problemas. Entre los tipos de datos más utilizados en programación se encuentran las cadenas de caracteres que permiten almacenar colecciones de valores alfanuméricos. Considerando esto, defina el *TDA tcadena* e implemente las siguientes operaciones:

- Generar Cadena Vacía: genera una cadena vacía o nula.
- Leer Cadena: permite cargar una cadena de caracteres con la entrada del usuario.
- Longitud de Cadena: permite determinar la cantidad de caracteres almacenados en una cadena.
- Comparar Cadenas: dadas 2 cadenas de caracteres determina si éstas son iguales (valor de retorno cero), si la primera es mayor que la segunda (valor de retorno 1) o si la segunda es la mayor (valor de retorno -1).
- Vacía: determina si una cadena está vacía o es nula.
- Mostrar Cadena: dada una cadena de caracteres, muestra su contenido en pantalla.
- Alfabético: permite determinar si una cadena contiene exclusivamente símbolos alfabéticos.
- Mayúsculas: permite convertir el contenido de una cadena de caracteres a mayúsculas.
- Minúsculas: permite convertir el contenido de una cadena de caracteres a minúsculas.
- Convertir en número: convierte una cadena que sólo contiene dígitos en su correspondiente valor numérico. Por ejemplo, la cadena “1995” se convierte en 1995.
- Convertir en cadena: convierte un valor entero en una cadena de caracteres. Por ejemplo, dado el valor 2008 se convierte en “2008”.

Realice la implementación utilizando LISTAS ENLAZADAS. Considere que la función *getchar()* permite leer caracteres individuales y que el símbolo ‘\n’ identifica la entrada ENTER o INTRO del teclado. Alternativamente, puede usar la función *getche*() para leer caracteres individuales y el símbolo ‘\r’ para identificar la entrada ENTER o INTRO.

11) Teniendo en cuenta que una imagen digital se compone de un conjunto de píxeles organizados en filas y columnas, utilice la definición del ejercicio 5 como base para construir el *TDA imagen* y diseñe las siguientes operaciones:

- Crear imagen: dados los valores de ancho y alto, se inicia en “negro” todos los píxeles de la imagen. Considere que el tamaño máximo de la imagen será de 800x600 (ancho x alto).
- Crear lienzo: crea una imagen en blanco.
- Generar imagen aleatoria: asigna valores aleatorios a los píxeles de una imagen.
- Suavizar imagen: asigna a cada píxel de una imagen el valor promedio de sus 3 componentes.
- Oscura: indica el porcentaje de píxeles negros de una imagen. Por ejemplo, si una imagen 96x96 tiene 580 píxeles negros entonces será un 6% oscura (580/9216 = 0,06).
- Clara: indica el porcentaje de píxeles blancos de una imagen. Por ejemplo, si una imagen 128x128 tiene 4320 píxeles blancos entonces será un 26% clara (4320/16384 = 0,06).

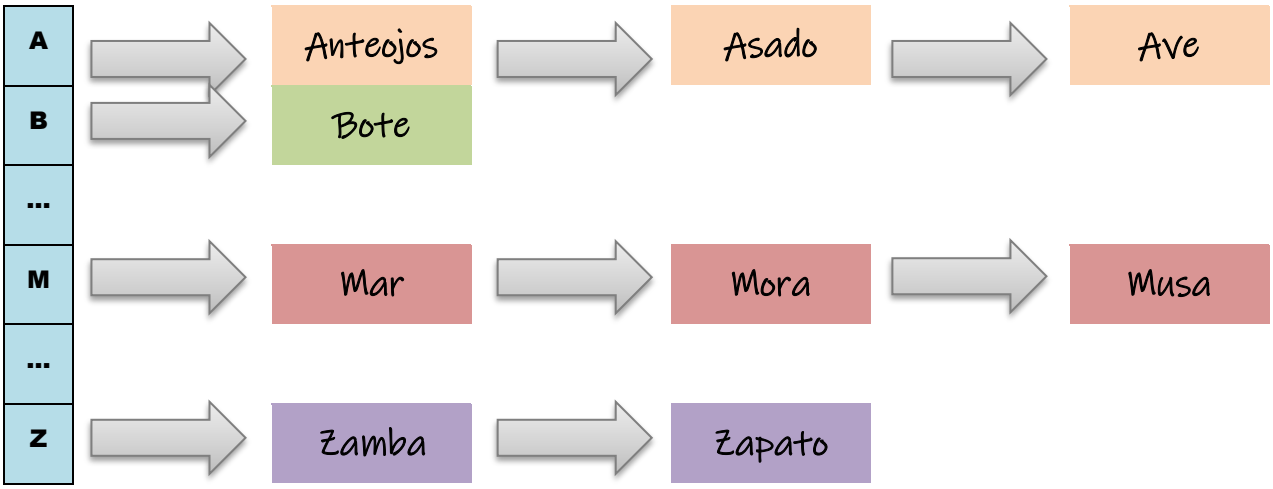
Realice la implementación utilizando ARREGLOS bidimensionales.

12) La tabla periódica de los elementos clasifica, organiza y distribuye los 118 elementos químicos conocidos, conforme a sus propiedades y características; su función principal es establecer un orden específico agrupando elementos. Cada elemento de la tabla se describe mediante los siguientes atributos: nombre del elemento, símbolo del elemento, número de protones, número de electrones, número de neutrones y grupo.

Considerando esto, implemente el *TDA tabla periódica* y las operaciones *crear elemento*, *mostrar tabla*, *mostrar grupo* y *calcular número de masa de un elemento específico* (suma del número de protones y el número de neutrones), utilizando a) ARREGLOS y b) LISTAS ENLAZADAS.

13) Un diccionario es una colección de palabras que se organizan de acuerdo a un orden alfabético de modo que la búsqueda se realice rápidamente.

En informática, un diccionario puede definirse como una colección de elementos (clave, valor) que se organizan según un criterio de orden preestablecido. En este caso, la clave hace referencia a la primera letra de una palabra, mientras que valor refiere a la palabra (palabra, tipo, definición) propiamente dicha. A fin de agilizar el proceso de búsqueda, las estructuras de datos que implementan diccionarios agrupan los valores por clave. Por ejemplo, las palabras "ANTEOJOS", "ASADO", "AVE" se agrupan bajo la clave "A", mientras que "MAR", "MORA", "MUSA" se agrupan bajo la clave "M". La siguiente figura muestra un ejemplo de organización de un diccionario.



Considerando la definición del *TDA diccionario* se especifican las siguientes operaciones:

- Tamaño: retorna el número de palabras de la agenda.
- Vacío: determina si el diccionario está vacío.
- Existe: determina si una palabra está registrada o no en la agenda.
- Consulta palabra: muestra el tipo (sustantivo, verbo, adjetivo, etc.) y definición de una palabra.
- Palabras por clave: muestra todas las palabras que corresponden a una clave determinada (letra).
- Agregar palabra: agrega una palabra al diccionario de acuerdo a la clave que corresponda. Por ejemplo, "NUBE" se guardará en la letra "N". En cada letra (categoría) los datos se disponen en orden.
- Eliminar palabra: remueve una palabra, retornando la palabra extraída.
- Listar: muestra todas las palabras de la agenda.
- Palabras por clave: determina la cantidad de palabras por cada clave del diccionario.

Teniendo en cuenta lo descripto, combine registros, arreglos y listas enlazadas para implementar el *TDA diccionario*.