Práctica 6

Estructura de Datos Lista I

Algoritmos y Programación 1
Ciencia de Datos en Organizaciones

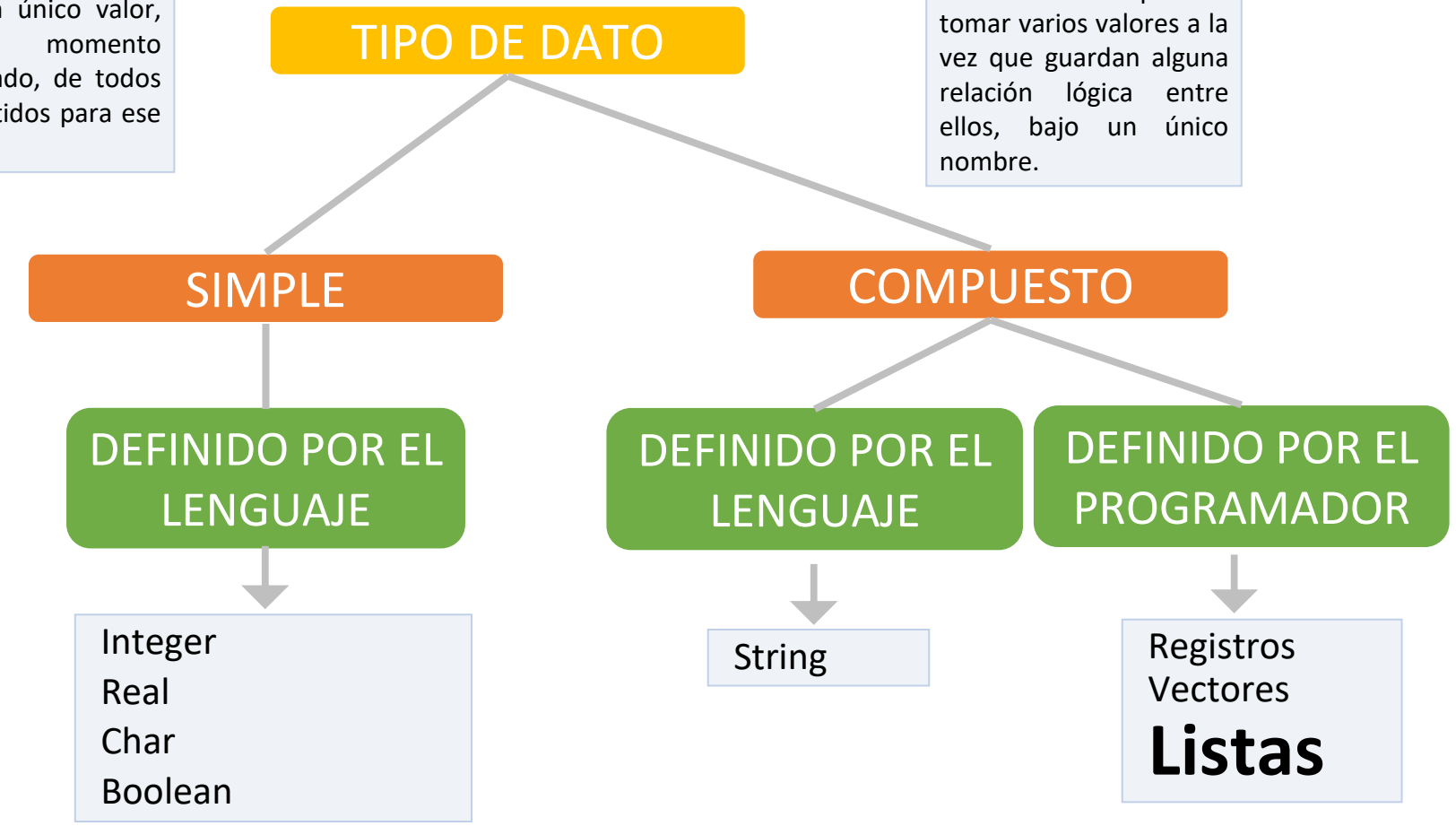
2025

Temas de la Práctica 6

- Contenidos
  - **Definición**
  - **Sintaxis (set de instrucciones)**
  - **Operaciones con listas**
    - Crear lista
    - Agregar al final
    - Recorrido de lista

TIPOS DE DATOS

SIMPLE: aquellos que toman un único valor, en un momento determinado, de todos los permitidos para ese tipo.



COMPUESTO: pueden tomar varios valores a la vez que guardan alguna relación lógica entre ellos, bajo un único nombre.

Listas en Pascal

```
Program uno;

uses GenericLinkedList;

type
    Lista = specialize LinkedList<TIPO>;

Var
    L: Lista;
```

Necesitamos incluir el tipo Lista y sus operaciones

Cualquiera de los tipos vistos hasta ahora

Declara una variable del tipo de la lista

Sintaxis: Instrucciones en Pascal

sintaxis	semántica
Lista = specialize LinkedList <TIPO>;	Declaración del tipo Lista (va en la sección type)
L: Lista	Declaración de variable del tipo Lista (en la sección var)
L:= Lista.create()	Creación de lista vacía asignada a la variable L
L.reset()	Se posiciona al principio de la lista L, se debe hacer siempre antes de recorrer una lista.
L.eol()	Devuelve True si no hay más nodos en la lista L o False en caso contrario.
L.current ()	Devuelve el nodo actual de la lista L.
L.next()	Avanza al siguiente nodo de la lista L o a / si no tiene más nodos.
L.add(elemento)	Agrega un nodo con el contenido de elemento al final de la lista L.

Ejercicio 3

- **Indicar qué hace el programa**.
- Indicar cómo queda conformada la lista si se lee la siguiente secuencia de números: 80, 1, 63, 120, 0
- Implementar un módulo que imprima los números enteros guardados en la lista generada.
- Implementar un módulo que reciba la lista y un valor x, e informe los números de la lista que son múltiplos de x.

El programa lee números y los almacena
en una lista hasta que llega el valor 0

```
program Listas;
type
    ListaEnteros = specialize LinkedList <integer>;
Procedure armarLista (var le:ListaEnteros);
var
  num: integer;
begin
      le:= ListaEnteros.create(); // crea la lista vacía
      read(num);
      while (num <> 0) do begin
            le.add(num);
            read(num);
      end;
end;
Var {declaración de variables del programa principal}
   le : ListaEnteros;
   x: integer;
Begin    {cuerpo del programa principal}
  armarLista(le);
  //imprimir lista
  // modificar lista
end.
```

Ejercicio 3

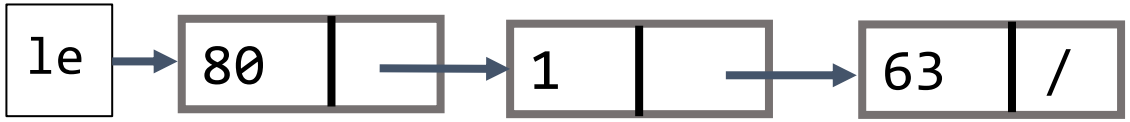
- **Indicar cómo queda conformada la lista si se lee la siguiente secuencia de números: 80, 1, 63, 120, 0**

```
program Listas;
Uses GenericLinkedList;
type
    ListaEnteros = specialize LinkedList <integer>;
Procedure armarLista (var le:ListaEnteros);
var
  num: integer;
begin
    le:= ListaEnteros.create(); // crea la lista vacía
    read(num);
    while (num <> 0) do begin
        le.add(num);
        read(num);
    end;
end;
```

Agrego el 80

Agrego el 1

Agrego el 63



Agrego el 120

Ejercicio 3

- Implementar un módulo que imprima los números enteros guardados en la lista generada.

```
Procedure Imprimir (le:ListaEnteros);
begin
   le.reset();
   while (not (le.eol()) ) do begin
     writeln(le.current());
     le.next ();
   end;
end;
```

```
program Listas;
Uses GenericLinkedList;
type
     ListaEnteros = specialize LinkedList <integer>;
Procedure armarLista (var le:ListaEnteros);
var
  num: integer;
begin
       le:= ListaEnteros.create(); // crea la lista vacía
       read(num);
       while (num <> 0) do begin
                le.add(num);
                read(num);
       end;
end;
Var {declaración de variables del programa principal}
   le : ListaEnteros;
   x: integer;
Begin     {cuerpo del programa principal}
  armarLista(le);
  imprimir (le);
  // modificar lista
end.
```

Ejercicio 3

- Implementar un módulo que reciba la lista y un valor x, e informe los números de la lista que son múltiplos de x.

```
Procedure multiplosX (le:listaEnteros, x: integer);
begin
   le.reset();
   while (not (le.eol()) ) do begin
      if (le.current() mod x = 0) then
            write (le.current());
      le.next ();
   end;
end;
```

```
program Listas;
Uses GenericLinkedList;
type
     ListaEnteros = specialize LinkedList <integer>;
Procedure armarLista (var le:ListaEnteros);
var
  num: integer;
begin
       le:= ListaEnteros.create(); // crea la lista vacía
       read(num);
       while (num <> 0) do begin
                le.add(num);
                read(num);
       end;
end;
Var {declaración de variables del programa principal}
    le : ListaEnteros;
    x: integer;
Begin     {cuerpo del programa principal}
   armarLista(le);
   imprimir (le);
   read(x);
   multiplosX(le,x);
end.
```

Ejercicio 6

Realizar un programa que lea y almacene la información de productos de un supermercado. De cada producto se lee: código, descripción, stock actual, stock mínimo y precio. La lectura finaliza cuando se ingresa el código 0. Una vez leída y almacenada toda la información, calcular e informar:

a) Porcentaje de productos con stock actual por debajo de su stock mínimo.
b) Descripción de aquellos productos con código impar.
c) Código de los dos productos más económicos.

```
program Listas;
Uses GenericLinkedList;
type
    producto = record
        código: integer;
        descripcion: string;
        stockActual: integer;
        stockMinimo: integer;
        precio: real;
    end;
    ListaProductos = specialize LinkedList <producto>;

Procedure armarLista (var lp:ListaProductos);
 var
   prod: producto;
 begin
      lp:= ListaProductos.create();
      leerProducto(prod);
      while (prod. codigo <> 0) do
      begin
            le.add(prod);
            leerProducto(prod);
      end;
end;
```

```
Var variables del programa principal}
    lp : ListaProductos;
Begin     {programa principal}
   armarLista(lp);
   recorrerInformar(lp)
 end.
```

```
Procedure LeerProducto (var p: producto);
 Begin
   readln(p.codigo);
   if (p. codigo <> 0) do begin
     readln(p.descripcion);
     readln(p.stockActual);
     readln(p.stockMinimo);
     readln(p.precio);
  end;
 end.
```

Ejercicio 6

Una vez leída y almacenada toda la información, calcular e informar:

a) Porcentaje de productos con stock actual por debajo de su stock mínimo.
b) Descripción de aquellos productos con código impar.
c) Código de los dos productos más económicos.

```
Type
  mínimo = record
      precio: real;
      código: integer;
End;
```

```
Procedure recorrerInformar ( lp:ListaProductos);
var  total, cant: integer;
      min1, min2: mínimo; porcentaje: real;
begin
   cant:=0;
   total:= 0;
   min1.precio: = 9999;
   min2.precio := 9999;
   min1. código := -1; // valor invalido en caso de lista vacia
   min2. código := -1;
   lp.reset();
   while (not (lp.eol()) ) do begin
     total:= total + 1; //a
     if (lp.current().stockActual < lp.current().stockMin) then //a
         cant:= cant + 1;
      if (lp.current().codigo mod 2 = 1) then //b
                  writeln ('Producto código impar : ', lp.current().descripcion);
      minimos(lp.current(), min1, min2);
      lp.next ();
   end;
   if total > 0 then
         writeln ('Porcentaje por debajo del stock mínimo: ', cant*100/total) ;
   writeln('Los dos códigos más económicos: ', min1.codigo, min2. codigo);
end;
```

Ejercicio 6

Una vez leída y almacenada toda la información, calcular e informar:

a) Porcentaje de productos con stock actual por debajo de su stock mínimo.
b) Descripción de aquellos productos con código impar.
c) Código de los dos productos más económicos.

```
Type
  mínimo = record
      precio: real;
      código: integer;
End;
```

```
Procedure minimos ( p: produto; var m1, m2: minimo);
Begin
      if (p.precio < m1.precio) then begin
          m2:= m1
          m1. precio:= p.precio;
          m1. codigo := p.codigo;
       end
        else
            if (p.precio < m2.precio) then begin
                m2. precio:= p.precio;
                m2. codigo := p.codigo;
            end;
end;
```