Introducción a Pascal: Estructura y Conceptos Básicos

Principios de lenguajes de programación

2025

Comparación de Pascal y lenguajes conocidos

- **Pascal:** Lenguaje imperativo/procedimental. Se centra en la ejecución secuencial de instrucciones, el uso de variables en representación de localizaciones de memoria y el uso de la asignación para cambiar el valor de las variables.

Comparación de Pascal y lenguajes conocidos

- **Pascal:** Lenguaje imperativo/procedimental. Se centra en la ejecución secuencial de instrucciones, el uso de variables en representación de localizaciones de memoria y el uso de la asignación para cambiar el valor de las variables.
- **Java:** Lenguaje orientado a objetos. Se basa en la creación de objetos y clases, haciendo énfasis en la encapsulación, herencia y polimorfismo.

Comparación de Pascal y lenguajes conocidos

- **Pascal:** Lenguaje imperativo/procedimental. Se centra en la ejecución secuencial de instrucciones, el uso de variables en representación de localizaciones de memoria y el uso de la asignación para cambiar el valor de las variables.
- **Java:** Lenguaje orientado a objetos. Se basa en la creación de objetos y clases, haciendo énfasis en la encapsulación, herencia y polimorfismo.
- **Smalltalk:** Lenguaje puramente orientado a objetos, donde **todo es un objeto** y la comunicación se efectúa mediante el envío de mensajes.

Comparación entre Pascal y lenguajes conocidos

Aspecto	**Pascal**	**Java**	**Smalltalk**
Paradigma	Imperativo, Procedural	OOP, Multiparadigma	OOP puro
Sintaxis	Simple, estricta	Basada en C, llaves	Minimalista, mensajes
Portabilidad	Limitada	Alta	Alta
Tipado	Estático	Estático	Dinámico
Estructura de Código	Procedimientos	Clases y métodos	Objetos y mensajes
Uso Actual	Educación, embebido	Empresarial, Web	Académico, Simulaciones
Compilación	Compilado	Compilado/Interpretado	Interpretado

Estructura de un Programa en Pascal

Un programa Pascal es un conjunto de instrucciones que siguen la sintaxis y la estructura del lenguaje Pascal.

Palabra Clave	Componentes de Programa	Ejemplo
Program	Cabecera	`PROGRAM Ejemplo;`
Const	Declaración de Constantes	`CONST Maximo = 234;`
Type	Declaración de Tipos	`TYPE Entero = integer;`
Var	Declaración de Variables	`VAR Anio: Real;`

Procedure	Declaración de Subprogramas	`PROCEDURE P1(A:T2; B,C:T1);`
		`...`
		`BEGIN`
		`...`
		`END;`

Function	Declaración de Subprogramas (Funciones)	`FUNCTION F1(N:T2): integer;`
		`...`
		`BEGIN`
		`...`
		`END;`

Begin	Bloque de Programa (sentencias)	`BEGIN`
		`bloque del programa;`
		`END.`

Modularización y Divide y Vencerás

Beneficios de Modularizar

- **Organización:** Permite dividir el programa en módulos o subprogramas, facilitando la comprensión y el mantenimiento.

Modularización y Divide y Vencerás

Beneficios de Modularizar

- **Organización:** Permite dividir el programa en módulos o subprogramas, facilitando la comprensión y el mantenimiento.
- **Reutilización:** Los módulos bien definidos se pueden reutilizar en distintos proyectos o partes del programa.

Modularización y Divide y Vencerás

Beneficios de Modularizar

- **Organización:** Permite dividir el programa en módulos o subprogramas, facilitando la comprensión y el mantenimiento.
- **Reutilización:** Los módulos bien definidos se pueden reutilizar en distintos proyectos o partes del programa.
- **Depuración:** Al trabajar en componentes independientes, es más sencillo localizar y corregir errores.

Divide y Vencerás

- Se basa en descomponer un problema complejo en partes más simples y manejables.

Modularización y Divide y Vencerás

Beneficios de Modularizar

- **Organización:** Permite dividir el programa en módulos o subprogramas, facilitando la comprensión y el mantenimiento.
- **Reutilización:** Los módulos bien definidos se pueden reutilizar en distintos proyectos o partes del programa.
- **Depuración:** Al trabajar en componentes independientes, es más sencillo localizar y corregir errores.

Divide y Vencerás

- Se basa en descomponer un problema complejo en partes más simples y manejables.
- Cada módulo se centra en una tarea específica, lo que permite abordar problemas grandes de manera escalable y eficiente.

Declaraciones

- **CONST:** La característica principal de una constante es que su valor no puede ser cambiado a lo largo del programa. Ejemplo: `CONST PI = 3.1416;`

Declaraciones

- **CONST:** La característica principal de una constante es que su valor no puede ser cambiado a lo largo del programa. Ejemplo: `CONST PI = 3.1416;`
- **VAR:** Al igual que una constante se almacena en una posición de memoria, pero al contrario de las constantes, el valor de una variable cambiará durante el transcurso de la ejecución de un programa.. Ejemplo: `VAR contador: integer;`

Tipos de Datos por Defecto

- **Numéricos:** `integer`, `real`, `longint`, `double`, `etc.`

Tipos de Datos por Defecto

- **Numéricos:** `integer`, `real`, `longint`, `double`, `etc.`
- **Alfanuméricos:** `char`, `string`, `shortstring`.

Tipos de Datos por Defecto

- **Numéricos:** `integer`, `real`, `longint`, `double`, `etc.`
- **Alfanuméricos:** `char`, `string`, `shortstring`.
- **Booleanos:** `boolean` (`TRUE`/`FALSE`).

Operadores Aritméticos

Las operaciones aritméticas siven para operar términos numéricos.

Operadores Unarios

- `-` : Negación del operando. Ejemplo: si `x = 100`, entonces `-x = -100`.
- **abs** : Valor absoluto de un número.
- **sqr** : Cuadrado de un número.
- **sqrt** : Raíz cuadrada (para números reales).
- **pred** : Predecesor de un entero.
- **succ** : Sucesor de un entero.
- **sin, cos, arctan** : Funciones trigonométricas.
- **log** : Logaritmo neperiano.
- **exp** : Exponencial con base e.

Operadores Aritméticos

Operadores Binarios

- + : Suma. Ejemplo: `a + b`.
- - : Resta. Ejemplo: `a - b`.
- * : Multiplicación. Ejemplo: `a * b`.
- / : División real. Ejemplo: `a / b`.
- **div** : División entera (solo para enteros). Ejemplo: `a div b`.
- **mod** : Módulo (resto de la división). Ejemplo: `a mod b`.

Operadores Relacionales

Una relación consiste de dos operandos separados por un operador relacional. Si la relación es satisfecha, el resultado tendrá un valor booleano verdadero (TRUE) ; si la relación no se satisface, el resultado tendrá un valor valso (FALSE).

Operador	**Descripción**
=	Igual a
<>	Distinto a
<	Menor que
<=	Menor o igual que
>	Mayor que
>=	Mayor o igual que

Operadores Lógicos

Al igual que las relaciones, en las operaciones con operadores lógicos se obtienen resultados cuyo valor de verdad toma uno de los valores booleanos verdadero (TRUE) o falso (FALSE).

Operador	**Descripción**
AND	Conjunción
OR	Disyunción
NOT	Negación

Operadores del tipo Char

No existen operaciones internas entre caracteres definidas por el lenguaje. Existen funciones que permiten convertir entre los valores de los tipos “char” y “entero”:

Símbolo	**Significado**
ord	Número de orden del caracter en el juego de caracteres adoptado.
chr	caracter asociado a un número de orden dado

Instrucciones de Entrada y Salida

Las instrucciones de Entrada y Salida se especifican dentro del cuerpo principal o bloque de instrucciones de un programa Pascal, y permiten que el programa se comunique con un periférico de manera que se pueda transmitir información desde el exterior a la memoria de la computadora o viceversa.

Instrucciones de Entrada y Salida

- READ y READLN (contracción de READ LINE) constituyen las instrucciones básicas para las operaciones de entrada estándar.

Instrucciones de Entrada y Salida

- READ y READLN (contracción de READ LINE) constituyen las instrucciones básicas para las operaciones de entrada estándar.
- WRITE y WRITELN (contracción de WRITE LINE) constituyen las instrucciones básicas para las operaciones de salida estándar.

***No se pueden leer variables de tipo boolean**

Funciones vs Procedimientos

Procedimiento

Un procedimiento en el lenguaje Pascal es un subprograma que realiza alguna tarea del programa, y no devuelve valor al subprograma que lo haya invocado.

```
1 PROCEDURE MostrarMensaje;
2 BEGIN
3   writeln('Hola mundo');
4 END;
```

Funciones vs Procedimientos

Función

Se utilizan de una manera similar a los procedimientos, si bien su principal diferencia es que el nombre o identificador de la función asume un valor, y cuando se ejecutan las acciones de una función, se devuelve el valor de salida al subprograma o módulo que lo invocó.

```
1 FUNCTION Sumar(a, b: integer): integer;
2 BEGIN
3   Sumar := a + b;
4 END;
```

Pasaje por Parámetro en Pascal

Por Valor vs Por Referencia

En Pascal existen dos formas de pasar parámetros a funciones o procedimientos:

- **Por Valor (sin `var`):** Se envía una copia del valor, de modo que las modificaciones internas no afectan a la variable original.

Pasaje por Parámetro en Pascal

Por Valor vs Por Referencia

En Pascal existen dos formas de pasar parámetros a funciones o procedimientos:

- **Por Valor (sin `var`):** Se envía una copia del valor, de modo que las modificaciones internas no afectan a la variable original.
- **Por Referencia (con `var`):** Se envía una referencia directa al valor original, permitiendo que cualquier cambio dentro del subprograma se refleje en la variable de llamada.

Pasaje por Parámetro en Pascal

Ejemplo

```
1 PROCEDURE IncrementarValor(x: integer);
2 BEGIN
3   x := x + 1; // La variable original no cambia
4 END;
5
6 PROCEDURE IncrementarReferencia(var x: integer);
7 BEGIN
8   x := x + 1; // La variable original se incrementa
9 END;
```

Ámbito de las Variables y de los Identificadores

Variables Globales vs. Locales

- **Globales:** Declaradas en la zona `VAR` del programa principal, su ámbito abarca todo el programa.

Ámbito de las Variables y de los Identificadores

Variables Globales vs. Locales

- **Globales:** Declaradas en la zona `VAR` del programa principal, su ámbito abarca todo el programa.
- **Locales:** Declaradas en el interior de un subprograma (procedimiento o función), solo son accesibles en ese bloque y sus subbloques.

Ámbito de las Variables y de los Identificadores

Variables Globales vs. Locales

- **Globales:** Declaradas en la zona `VAR` del programa principal, su ámbito abarca todo el programa.
- **Locales:** Declaradas en el interior de un subprograma (procedimiento o función), solo son accesibles en ese bloque y sus subbloques.
- Si un identificador local comparte nombre con uno global, el local **oculta** al global dentro de su ámbito.

Ámbito de las Variables y de los Identificadores

Ejemplo: Ocultamiento de Variables

```
1  PROGRAM verAlcance;
2  VAR
3    a: integer;
4
5  PROCEDURE cambia(a: integer);
6  BEGIN
7    a := 10;              // Modifica la copia local de 'a'
8    writeln('Dentro a: ', a)
9  END;
10
11 BEGIN
12   a := 3;
13   writeln('Antes a: ', a);
14   cambia(a);
15   writeln('Despu s a: ', a)
16 END.
```

Ámbito de las Variables y de los Identificadores

Salida:
Antes a: 3
Dentro a: 10
Después a: 3

Acceso a Variables Globales

Si en un subprograma se utiliza un identificador que no ha sido declarado localmente, se accede al elemento global correspondiente.

Estructuras de Control

Sentencia IF

La sentencia de control simple tiene la siguiente sintaxis

```
1 IF datoBoolean
2 THEN
3 accion1
```

Mientras que para una acción compuesta, la sintaxis:

```
1 IF <expr. booleana>
2 THEN BEGIN
3 <instrucci n-1>;
4 <instrucci n-2>;
5 ...
6 END;
```

Sentencia IF

También se destaca la instrucción If simple en la que se permite ejecutar una acción para el caso de que la condición sea falsa.

```
1 IF a > b THEN
2   writeln('a es mayor')
3 ELSE
4   writeln('b es mayor');
```

Sentencia CASE

La instrucción cuenta con una expresión (llamada selector) y una lista de sentencias para cada una de las constantes del mismo tipo del selector, de forma que cuando se evalúa la expresión, se ejecuta la instrucción que contenga el o los valores correspondientes

Estructuras de Control

```
1  PROGRAM EjemploCase;
2  VAR
3  Mes, CantDias : integer ;
4  BEGIN
5  writeln(  Ingresa   el mes (De 1 a 12):      );
6  readln(Mes);
7  CASE Mes OF :
8  1, 3, 5, 7, 8, 10, 12 : CantDias := 31;
9  4, 6, 9, 11 : CantDias := 30;
10 2 : CantDias := 28;
11 ELSE
12 writeln('Error en la entrada');
13 END;
14 write('El mes numero ', Mes:2);
15 writeln(' tiene ', CantDias:2 , ' dias');
16 END.
```

Bucle WHILE

La instrucción WHILE se utiliza para especificar una acción que se repite mientras es verdadera una determinada condición.

```
1 WHILE contador <= n DO BEGIN
2   suma := suma + contador;
3   contador := contador + 1;
4 END;
```

Estructuras de Control

La instrucción REPEAT también se utiliza para construir bucles a partir de una determinada condición, pero a diferencia del While si o si ejecuta una vez el cuerpo y luego evalua la condición para saber si debe continuar o no.

Bucle REPEAT

```
1 REPEAT
2   writeln('Iterando');
3   contador := contador + 1;
4 UNTIL contador > n;
```

Estructuras de Control

La instrucción FOR se utiliza para especificar un número predeterminado de repeticiones para una acción.

Bucle FOR

```
1 FOR i := 1 TO 10 DO
2   writeln('Numero: ', i);
```

Tipos de datos definidos por el usuario

Enumerados

Los enumerados son tipos de datos que se definen como un conjunto de valores válidos para un determinado tipo. Para declarar un tipo de datos enumerado, simplemente se enumeran los posibles valores que puede tener una variable para un tipo de datos.

Ejemplo

```
1 VAR
2 ayer, hoy, laboral : tipoDiasSemana;
3 contador: integer; ...
4 BEGIN
5 ...
6 ayer := miercoles;
7 hoy := succ(ayer);
8 ...
9 END.
```

Tipos de datos definidos por el usuario

Subrango

Los datos de tipo subrango se definen tomando como límites, el valor mínimo y el valor máximo del rango, como dos constantes. Las constantes puede ser del tipo entero, caracter o enumerado. Por ejemplo:

```
1  CONST
2  MaximoM = 10 ;
3  VAR
4  valores : 1 .. MaximoM
5  ...
6  BEGIN
7  ...
8  valores := 1;
9  write(valores); ...
10 END.
```

Tipos de datos definidos por el usuario

Conjunto

El tipo de datos definido por el usuario conjunto permite agrupar los los elementos de un conjunto de valores, que pueden pertenecer a cualquier tipo ordinal, incluido los enumerados. No poseen un orden entre ellos. Por ejemplo:

```
1  Estacion = Set Of Meses;
2  VAR
3  n : Integer;
4  mes : Meses;
5  otogno, invierno, primavera, verano: Estacion;
6  BEGIN
7  verano := [dic, ene .. mar ];
8  otogno := [mar .. jun ];
9  invierno := [jun .. sep ];
10 primavera := [sep .. dic ];
11 ...
```

Tipos de datos definidos por el usuario

Operaciones sobre los conjuntos

Operación	**Notación Pascal**	**Resultado**
Pertenencia	`IN`	*a IN Conjunto* es verdadero si el valor del elemento a está presente entre los valores del conjunto `Conjunto`.
Intersección	`*`	*A * B* es el conjunto cuyos elementos pertenecen a `A` y `B` simultáneamente.
Unión	`+`	*A + B* es el conjunto que contiene todos los elementos que están en `A`, en `B` o en ambos.
Diferencia	`-`	*A - B* es el conjunto cuyos elementos son de `A` pero no de `B`.

Tipos de datos definidos por el usuario

Arreglos (Matrices)

Un arreglo está formado por un número fijo de elementos contíguos de un mismo tipo. Pueden ser:

- arreglos unidimensionales, llamados también vectores, constituidos por un conjunto ordenado de elementos del tipo base.

Tipos de datos definidos por el usuario

Arreglos (Matrices)

Un arreglo está formado por un número fijo de elementos contíguos de un mismo tipo. Pueden ser:

- arreglos unidimensionales, llamados también vectores, constituidos por un conjunto ordenado de elementos del tipo base.
- arreglos multidimensionales, que según la cantidad de dimensiones pueden llamarse matrices, cubos, hipercubos, etc, y que poseen un ordenamiento en varias dimensiones.

Tipos de datos definidos por el usuario

Ejemplo

```
1 TYPE
2 tipoA = ARRAY [ 1 .. MaxElementos] OF real;
3 tipoM = ARRAY [ 1 .. 10 , ene .. abr ] OF integer
4 ...
5 VAR
6 arrM, arrN : tipoA;
7 matA, matB : tipoM;
8 cubo : ARRAY [1 .. 3 , 1 .. 4 , -2 .. 2] OF char;
```

Tipos de datos definidos por el usuario

Registros

Un registro es una estructura de datos que consiste de un número fijo de componentes llamados campos. Los campos pueden pertenecer a diferentes tipos y requieren de un identificador de campo.

```
1  TYPE
2    Persona = RECORD
3      nombre: string;
4      edad: integer;
5    END;
6  VAR
7    p: Persona;
8  BEGIN
9    p.nombre := 'Juan';
10   p.edad := 30;
11   writeln(p.nombre, ' tiene ', p.edad, ' anios');
12 END.
```

Tipos de datos definidos por el usuario

Registros Variante

El lenguaje Pascal provee además la alternativa de trabajar con registros con variantes, que constan de dos partes: la primera, llamada parte fija, está formada por aquellos campos del registro que forman parte de todos los ejemplares; la segunda parte, llamada parte variable, está formada por aquellos campos que sólo forman parte de algunos ejemplares.

Tipos de datos definidos por el usuario

Registros Variante

```
1  TYPE
2    Dato = RECORD
3      tipo: char;
4      CASE tipo OF
5        'N': (num: integer);
6        'C': (cad: string);
7    END;
8  VAR
9    d: Dato;
10 BEGIN
11   d.tipo := 'N';
12   d.num := 100;
13   writeln('N mero: ', d.num);
14 END.
```

Ejemplo completo en Pascal

```
1  program EjemploPascal;
2  type
3    TEstudiante = record
4      nombre: string[50];
5      edad: integer;
6      nota: real;
7    end;
8  var
9    estudiantes: array[1..2] of TEstudiante;
10   i: integer;
11 function calcularPromedio(): real;
12 var
13   total: real;
14   j: integer;
15 begin
16   total := 0;
17   for j := 1 to 2 do
18     total := total + estudiantes[j].nota;
19   calcularPromedio := total / 2;
20 end;
```

Ejemplo completo en Pascal

```
1  procedure ingresarDatos();
2  begin
3    for i := 1 to 2 do
4    begin
5      writeln('Ingrese los datos del estudiante ', i);
6      write('Nombre: ');
7      readln(estudiantes[i].nombre);
8      write('Edad: ');
9      readln(estudiantes[i].edad);
10     write('Nota: ');
11     readln(estudiantes[i].nota);
12   end;
13 end;
```

Ejemplo completo en Pascal

```
1  begin
2    ingresarDatos();
3    writeln('Datos ingresados:');
4    for i := 1 to 2 do
5    begin
6      writeln('Estudiante ', i, ':');
7      writeln('Nombre: ', estudiantes[i].nombre);
8      writeln('Edad: ', estudiantes[i].edad);
9      writeln('Nota: ', estudiantes[i].nota:0:2);
10   end;
11   writeln('El promedio de las notas es: ',
12   calcularPromedio():0:2);
13   if calcularPromedio() >= 6.0 then
14     writeln('El promedio alcanza o supera el 6.')
15   else
16     writeln('El promedio del curso es menor a 6.');
17   readln;
18 end.
```