Introducción

El lenguaje de programación PASCAL, fué diseñado por Niklaus Wirth en Zurich entre 1970 y 1971, con la finalidad de ser utilizado para la enseñanza de los conceptos de programación y para facilitar el desarrollo de aplicaciones comerciales y científicas. El Pascal es un lenguaje estructurado y permite realizar una programación modular.

El Pascal se usa en un ambiente Pascal estructurado, en un ambiente de objetos como el Object Pascal ó en un ambiente visual como el DELPHI 2.

En el curso usaremos el Object Pascal en el ambiente del Delphi 2.

Para trabajar con el Pascal en el ambiente del Delphi 2, debemos realizar lo siguiente:

1.− Ingresar al Delphi 2

2.− Elija Project/Options del menú

principal.

3.− Elija Linker

4.− Marcar `Generate Console

Application'

5.− Elija Projec, View Source.

Puede trabajar su programa en la ventana del Project1.

Para ejecutar un programa, presione la tecla F9.

Para salvar su programa elija la opción File del menú Principal y luego la opción Save As.

Para compilar el programa, elija Project/Compile ó presione Ctrl−F9.

Estructura de un programa Pascal

Program Nombre programa;

Uses Identificadores;

Const {Definición de constantes;}

Type {Declaración de tipos de datos

definidos por el usuario;}

Var {Declaración de variables;}

Procedure Definición de procedimiento

Function Definición de funcion

Begin {Inicio del cuerpo del programa principal}

Sentencias

End {Fin del programa principal}

Ejemplo de un programa en Pascal

```
Program Ejemplo1;

Uses Forms;

Const A=2;

Var B:Integer;

C:Integer;

Begin

Writeln(`ingrese un numero'); readln(B);

C:=b*a;

Writeln(`El doble del numero es = `,c);

Readln;

End.
```

Program.− Indica el nombre del programa. En Delphi no podemos cambiar el nombre directamente, sino a través del **Save As.** Si cambiamos directamente, al momento de salvarlo nos da un error. Este nombre no puede tener espacios en blanco intermedio.

Ejemplo de nombres : Programa, Serie_01, Calcular_area, etc.

Uses.− Identifica todas las unidades utilizadas en el programa. Ejemplo: Forms, Wincrt, etc.

Const.− Permite definir nombres de constantes que tienen un valor único, que no puede ser modificado mientras se ejecuta el programa.

Ejemplo

```
Const A=20;

B= A*10;

C= `Curso';

D= 20*5;
```

E,F,G=100;

Var.− Esta sección permite declarar las variables utilizadas en el programa. Una variable es una zona de memoria con un nombre y un tipo de dato, que almacena un valor que puede variar en cualquier parte del programa. La sintaxis es la siguiente:

Var variable1 : tipo de dato 1;

. . . . .

Variable n : tipo de dato n;

En una línea puede ir mas de una variable del mismo tipo.

Ejemplo : Var M:Integer;

N,O:Real;

P : Variant;

Const, Var y Type pueden aparecer en cualquier orden y mas de una vez en la sección de declaraciones.

Procedure y Function.− Son segmentos de programas ó sub programas.

Begin y End.− EL cuerpo del programa principal debe empezar siempre con un Begin y terminar con un End, este End debe terminar con un punto.

Identificadores.− Representan los objetos de un programa, tales como variables, constantes, procedimiento, etc.

El identificador ó nombre:

- Debe empezar con una letra. No puede contener espacios en blanco.
- Puede tener letras, dígitos y caracteres Subrayados después del primer carácter.
- No es case sensitivy, es decir no hay diferencia entre mayúsculas y minúsculas.
- El nombre debe tener una estrecha relación con el dato que almacena.

Tipos de Datos.− Todos los datos tienen un tipo de dato asociado a ellos. Cuando definimos un variable es necesario indicar el tipo de dato de la variable

Entero.− Permite representar números enteros. Existen varios tipos de números enteros.

Byte.− Son números enteros comprendidos entre 0 y 255. Se almacena en memoria como un Byte (8 bits). El tipo Byte no puede ser negativo.

Integer.− Son enteros sin decimales, están dentro del rango de −32768 y +32767. Se almacena en memoria como 2 bytes. Tienen un bit que guarda el signo (0 si es positivo y 1 si es negativo). Es posible separar los 2 bytes usando las funciones HI y LO.

Longint.− Son números enteros que están dentro del rango de −2147483648 y +22147483647. Ocupan 4 bytes.

Word.− Representa solo valores positivos que están dentro del rango de 0 a 65535. Ocupa 2 bytes de memoria.

Tipo Bytes Rango Signo

Byte 1 0 a 255 No

Integer 2 −32768 a +32767 Si

Longint 4 −2147483648 a Si

+2147483647

Word 2 0 a 65535 No

Real.− Permite almacenar números que tiene decimales. Cualquier número se puede representar como real.

Tipo Bytes Rango

Real 6 2.9 x 10−39 a 1.7 x 10+38

Single 4 1.5 x 10−45 a 3.4 x 10+38

Double 8 5 x 10−324 a 1.7 x 10+308

Extended 10 3.4x10−4951 a 1.1x10+4932

Los enteros ocupan menos memoria que los reales.

Las operaciones con enteros son siempre precisas.

Normalmente las operaciones con enteros son mas rápidas que con números reales.

Char.− Es un tipo de dato que puede contener un solo carácter ASCII, es decir puede almacenar una letra, un dígito ó un carácter especial. Ejemplo `A', 'X', '*', ' `, '1'.

Ejemplo de programa:

```
Program Ejemplo01;

Uses Forms;

Var letra : char;

Begin

letra:='S';

Writeln(`El carácter es',letra);

letra:='5';
```

```
Writeln(`El carácter es',letra);

Readln;

End.
```

String.− Representa cadenas de caracteres que tienen 0 ó mas caracteres tipo ASCII y están encerradas entre apóstrofes (` `).

Una cadena nula ó vacía, no tiene nada entre los apóstrofes.

La longitud de una cadena es el número de caracteres encerrados entre apóstrofes.

Ejemplo :

```
Program Ejemplo02;

Uses Forms;

Var letras1: String[20];

letras2: String[25];

Begin

letras1:='Colegio';

letras2:='Colegio Peruano Británico';

Writeln(`Nombre : ',letras1,letras2);

Readln;

End.
```

Variant.− Este tipo permite representar valores que cambian dinámicamente, es decir una variable de este tipo puede asumir valores de diferentes tipos en tiempo de ejecución, tales como valores integer, reales, string, boolean, etc.

Una variable tipo variant puede ser combinado con otros valores variant, integer, reales, string y boolean.

Este tipo de dato consume mayor memoria que los otros tipos de datos, además las operaciones son mas lentas.

Ejemplo :

```
Program Ejemplo03;

Uses Forms;

Var numero1:integer;
```

```
numero2:real;

numero3:variant;

Begin

numero1:=125;

numero2:=12.50;

numero3:=numero1*10;

Writeln(`Resultado : ',numero3);

numero3:=numero2*10;

Writeln(`Resultado : ',numero3);

Readln;

End.
```

Boolean.− Permite almacenar valores lógicos. Los valores lógicos pueden tomar solo dos valores : verdadero (true) ó falso (false)

Ejemplo :

```
Program Ejemplo04;

Uses Forms;

Var indicador:boolean;

numero2:real;

Begin

indicador := True;

Writeln(`Resultado : ',indicador);

indicador := False;

Writeln(`Resultado : ',indicador);

Readln;

End.
```

Sentencias.− Describen acciones que pueden ser ejecutadas. Las sentencias en Pascal terminan en punto y coma (;).

La sentencia de asignación se usa para almacenar valores a las variables. Tiene el sgte. Formato:

variable := expresión

Donde variable recibe el valor de la expresión.

El operador de asignación es :=

expresión es un valor constante, una variable ó una operación.

La expresión y la variable deben de ser del mismo tipo de datos.

Ejemplo:

A1:= 16; {A1 recibe el valor de 16}

A2:= A1*5 {A2 recibe el valor del producto}

N1:='USIL'; {N1 recibe el valor de USIL}

Una sentencia de asignación es una operación destructiva porque el valor almacenado en la variable se pierde ó se sustituye por el valor de la expresión. Ejemplo:

A1 := 10; {A1 guarda el valor de 10}

A1 := 20; {A1 reemplaza el 10 por el 20

Contador.− Es una operación de asignación que al ser ejecutada incrementa el valor de una variable en una unidad ó en una cantidad constante. Tiene el siguiente formato :

contador := contador + 1;

Ejemplo: numero := 21;

numero := numero + 1;

El nuevo valor de la variable número es 22, si ejecutamos varias veces la segunda sentencia del ejemplo, la variable número se incrementa en una unidad cada vez.

Acumulador.− Es una operación de asignación que al ser ejecutada incrementa el valor de una variable en una cantidad variable. Tiene el siguiente formato :

acumulador := acumulador + n ;

n es una variable que puede tener diferentes valores. Ejemplo :

B:= 10;

suma:=0;

suma := suma + b ;

b := 15;

suma := suma + b ;

El valor de la variable suma es de 25 al finalizar el ejemplo.

Los acumuladores se usan para realizar la suma de diferentes valores.

Comentarios .− Son textos explicativos que ayudan a la comprensión de un programa. Pueden ir en cualquier parte del programa, puede ocupar varias líneas. Los comentarios van encerrados entre llaves { } ó entre (* *).

Ejemplo:

A1:= 16; (*A1 recibe el valor de 16*)

A2:= A1*5; {A2 recibe el valor del producto}

N1:='USIL'; {N1 recibe el valor de USIL}

Expresiones.− Una expresión es un conjunto de datos ó funciones unidos por operadores aritméticos.

En Pascal podemos trabajar con diferentes tipos de operadores, tales como: operadores aritméticos, operadores de relación, operadores lógicos.

Operadores Aritméticos.− Permiten realizar operaciones aritméticas, tales como suma, resta, etc.

Operador Significado Ejemplo

+ Suma a + b

− Resta a − b

* Multiplicación a * b

/ División a / b

div División Entera a div b

mod Módulo a mod b

La división (/) produce un resultado real, independiente del tipo de operando a usar.

Los operadores Div y Mod solo se usan con números enteros

Div .− Calcula el cociente entero de la división de 2 números enteros.

Mod .− Calcula el resto ó residuo de dicha operación.

Ejemplo :

13 div 4 resultado 3 13 4

13 mod 4 resultado 1 12 3 div

1 mod

La prioridad de evaluación es de la siguiente manera:

1.− Se ejecutan todas las expresiones entre

paréntesis.

2.− Se ejecutan *, /, div, mod

3.− Se ejecutan +, −

4.− Los operadores de igual nivel en una misma

expresión se ejecutan de izquierda a

derecha.

Ejemplo: resolver 10*2/4 + (7−3) * 2

a).− Paréntesis 10*2/4 + 4 * 2

b).− Multiplicación 20 /4 + 8

c).− División 5 + 8

d).− Suma 13

Este mismo ejemplo en programa:

```
Program Ejemplo05;

Uses Forms;

Var Var1:Integer;

numero2:real;

Begin

Var1 := 10*2/4+(7−3)*2;

Writeln(`Resultado : ',var1);

Readln;

End.
```

Al ejecutar este programa, obtenemos un error por incompatibilidad de tipos de datos en la operación de asignación ya que Var1 es integer y en la operación existe una división por lo tanto el resultado es real. Para

que este programa se ejecute debemos definir var1 como real ó variant.

Operaciones de Entrada y Salida

Las operaciones de entrada y salida permiten ingresar datos a la computadora desde diferentes dispositivos, tales como el teclado, un archivo de disco, etc. y mostrar datos ó resultados.

La lectura se realiza por medio de la sentencia Read ó Readln.

La salida se realiza por medio de la sentencia Write ó Writeln.

Entrada de datos

Permite ingresar datos durante la ejecución de un programa.

El formato es el siguiente:

Read (var1, var2, .. varN);

Readln(var1, var2, .varN);

Var1, varN son las variables que van a almacenar los valores a ingresar.

Una sentencia Read ó Readln es posible que no tenga ninguna variable, que tenga solo una variable ó que tenga muchas variables.

Cuando se ejecuta la sentencia Read ó Readln, el proceso se detiene hasta ingresar un valor y presionar la tecla Enter ó hasta presionar solo la tecla Enter.

Ejemplo:

```
Program Ejemplo06;

Uses Forms;

Var nombre : string[20];

edad : integer;

Begin

Readln(nombre);

Readln(edad);

Readln;

End.
```

Salida de datos

Permite mostrar datos durante la ejecución de un programa.

El formato es el siguiente:

Write (item1, item2, itemN);

Writeln(item1, item2, ...itemN);

ítem representa al objeto a visualizar, puede ser una constante literal, una variable, una llamada a una función ó una combinación de ellos.

Al ejecutar la sentencia Writeln, el cursor salta a la siguiente línea, en cambio con Write el cursor se queda al final del último elemento en la misma línea.

Writeln sin ningún item proporciona saltos ó avances de línea.

Formatos de Salida

Permiten definir la longitud de los items de salida, es decir especifican el número de posiciones del campo de escritura y para los números reales es posible precisar además el número de decimales deseados.

Tiene el siguiente formato:

Writeln (ítem :longitud,);

Writeln (ítem : longitud: dígitos, ..);

Anchura, especifica la cantidad de caracteres del campo en que se escribe el ítem.

Dígitos, indica la cantidad de decimales de un número real. Si no vienen dígitos significa que el ítem no es real.

Ejemplo: X := 12.345 ;

Y := 67 ;

Writeln (X:7:4); da 12.3450

Writeln (X:7:1); da 12.3

Writeln (Y:4); da 67

Writeln(`USIL':5); da USIL

Funciones Matemáticas Standard

Función Descripción

Abs(X) Da el valor absoluto de X

Sqr(x) Da el cuadrado de X

Sqrt(X) Devuelve la raíz cuadrada de X

Int(X) Devuelve la parte entera de un

número Real. El resultado es Real.

Trunc(X) Devuelve la parte entera de X

Round(X) Devuelve el entero más próximo a X

Ln(X) Devuelve el logaritmo natural de x>0

Exp(X) Devuelve 2.71828 elevado al

exponente X

Sin(X) Da el valor del seno de X radianes

Cos(X) Da el valor del coseno de X radianes

ArcTan(X) Da el valor en radianes del arco

tangente de X.

En todos los casos se supone que X es un dato ó expresión que representa valores numéricos.

Las funciones TRUNC(x) y ROUND(x) sólo dan resultados de tipo entero.

Control de Flujo de un Programa

El Control de flujo de un programa, se refiere al orden de ejecución de las acciones individuales del programa. El control de flujo es lineal, pero es posible salir del flujo usando estructuras de control selectivas ó repetitivas.

Estructuras de Control Selectivas, usamos estas estructuras cuando encontramos acciones con dos ó mas alternativas. Las sentencias que tienen estas estructuras son el IF y el CASE.

Sentencia IF .− Permite elegir uno de dos posibles caminos, de acuerdo a la evaluación de una expresión lógica. Una expresión lógica es una expresión que puede ser verdadera ó falsa.

Formato:

IF expresión lógica

THEN

Sentencia A

ELSE

Sentencia B

Modo de ejecución :

1.− Se evalúa la expresión lógica

2.− Si la expresión lógica es verdadera ejecuta la sentencia A y luego el control pasa a la siguiente sentencia después del IF THEN ELSE.

3.− Si la expresión es falsa ejecuta la sentencia B y el control pasa a la sgte. Sentencia después del IF THEN ELSE.

4.− La sentencia Else es opcional.

Para evaluar la expresión lógica es necesario utilizar **operadores de relación** tales como :

operador significado

> mayor que

< menor que

= igual a

> = mayor ó igual que

< = menor ó igual que

< > diferente a

Además podemos usar **operadores lógicos** como:

operador significado

And Conjunción

Or Disyunción

Xor Disyunción exclusiva

Not Negación

Ejemplo :

Begin

Readln (número) ;

If número > 0

Then Writeln (`Número positivo')

Else Writeln (`Número negativo') ;

End.

El punto y coma separa dos sentencias. La sentencia If del ejemplo, termina con la sentencia del ELSE.

En vez de una sola sentencia, el Then y/o el Else pueden tener sentencias compuestas (son conjunto de sentencias separadas por puntos y comas que van entre un Begin y un end End).

Ejemplo:

```
Begin

Readln (número) ;

If número > 0

Then

Begin

Writeln (`Número positivo') ;

Writeln (`Número mayor que 0') ;

End

Else

Begin

Writeln (`Número negativo') ;

Writeln (`Número menor que 0') ;

End ;

Readln ;

End.
```

Consideración:

Si la expresión lógica contiene mas de una comparación, estas tienen que ir encerradas entre paréntesis, sino da error.

Ejemplo :

```
If (a > 0) and (b < 15)
```

Esta expresión es verdadera si a es mayor que 0 **y** b es menor que 15.

```
If (a > 0) or (b < 15)
```

Esta expresión es verdadera si a es mayor que 0 **o** b es menor que 15.

Sentencia CASE .− Se utiliza para elegir una de dos ó mas alternativas. Estas alternativas se generan de acuerdo al valor que puede tomar una variable ordinal. Una variable ordinal es Integer, Char, Boolean ó enumerado.

Formato:

CASE variable OF

Lista de constantes 1 : Sentencia 1 ;

Lista de constantes 2 : Sentencia 2 ;

. ..

Lista de constantes n : Sentencia n ;

[Else sentencia x]

END ;

Modo de ejecución :

1.− Se evalúa la variable y se compara con los valores de las listas de constantes.

2.− Si el valor de la variable se encuentra en una de las listas de constantes, se ejecuta la sentencia correspondiente y luego el control pasa a la siguiente sentencia después del End del Case.

3.− Si el valor de la variable no se encuentra en las listas de constantes y existe el Else, se ejecuta la sentencia del Else; si no existe el Else no se ejecuta ninguna sentencia del Case.

Consideraciones:

1.− La cláusula Else es opcional.

2.− Si la variable es tipo Real, produce un error ya que el tipo real no es ordinal.

3.− Todas las constantes del Case deben ser únicas y del mismo tipo de dato de la variable evaluada.

Ejemplo :

Program Case01;

Uses Forms;

Var operación : char;

Begin

Write(`Ingrese un valor `) ;

Readln(operación) ;

```
Case operación Of

`*' : Writeln(`Ud. va a multiplicar') ;

`/' : Writeln(`Ud. va a dividir') ;

`+' : Writeln(`Ud. va a sumar') ;

`−' : Writeln(`Ud. va a restar') ;

End;

Readln;

End.
```

```
Program Case02;

Uses Forms;

Var caracter : char;

Begin

Write(`Ingrese un caracter `) ;

Readln(caracter) ;

Case caracter Of

`0'. . `9' : Writeln(`es una cifra') ;

`a'. . `z' : Writeln(`es una letra minuscula') ;

`A'. .'Z': Writeln(`es una letra mayuscula') ;

`*','/','−`,'+' : Writeln(`es un simbolo') ;

End;

Readln;

End.
```

Los dos puntos seguidos que aparece en `0' y `9', indican que el valor de la variable se va a evaluar entre el rango de valores comprendidos entre esas dos constantes (0 y 9).

Estructuras de Control Repetitivas, son aquellas que permiten repetir varias veces una sentencia ó un grupo de sentencias. A este grupo de sentencias se le denomina ciclo, lazo ó bucle (loop).

Las sentencias que tienen esta estructura son :

While do, Repeat Until, For to do, For downto do.

Sentencia WHILE.− Permite repetir un grupo de sentencias **mientras** se cumple una condición

Formato :

a.− WHILE expresión lógica DO

Sentencia A;

b.− WHILE expresión lógica DO

Begin

Sentencia 1 ;

. . . . .

Sentencia N ;

End;

En el caso (a), el bucle es la sentencia A.

En el caso (b), el bucle son las sentencias que están entre el Begin y el End.

Modo de ejecución :

1.− Se evalúa la expresión lógica.

2.− Si es verdadera se ejecuta el bucle, si es falso, el control del programa pasa a la siguiente sentencia del bucle.

3.− Cada vez que ejecuta el bucle, regresa al While y vuelve a evaluar la expresión lógica.

Consideraciones:

1.− Antes de cada iteración se evalúa la condición.

2.− Si en la primera evaluación, la expresión lógica es falsa, el bucle no se ejecuta ninguna vez.

3.− Si la expresión lógica siempre es verdadera, la ejecución del bucle es infinita.

Ejemplo : Sumar los n primeros números enteros.

```
Program While01;

Uses Forms;

Var n,i : integer;

acum : integer;
```

```
Begin

Write(`Ingrese un numero');

Readln(n) ;

i := 0 ;

acum := 0 ;

While i < n do

Begin

i := i + 1;

acum := acum + i ;

end ;

Writeln(`La suma es `,acum:5);

Readln;

End.
```

En este ejemplo, se repiten dos sentencias (i:=i+1 y acum:=acum+1) mientras que el valor de i sea menor que el valor de n.

Sentencia REPEAT.− Permite repetir un grupo de sentencias **hasta** que la condición se cumpla (sea verdadera).

Formato :

```
REPEAT

Sentencia 1;

. . . . . .

Sentencia n;

UNTIL expresión lógica ;
```

Modo de ejecución :

1.− Se ejecuta el bucle.

2.− La expresión lógica se evalúa después de ejecutar el bucle.

3.− Si la expresión lógica es falsa, se ejecuta otra vez el bucle, si es verdadera termina la ejecución del Repeat.

Consideraciones :

1.− Después de cada iteración se evalúa la condición.

2.− El bucle se ejecuta por lo menos una vez.

3.− El bucle no necesita de un Begin y de un End.

Ejemplo : Sumar los n primeros números enteros.

```
Program Repeat01;

Uses Forms;

Var n,i : integer;

acum : integer;

Begin

Write(`Ingrese un numero');

Readln(n) ;

i := 0 ;

acum := 0 ;

Repeat

i := i + 1;

acum := acum + i ;

Until i > = n ;

Writeln(`La suma es `,acum:5);

Readln;

End.
```

En este ejemplo, se repiten dos sentencias (i:=i+1 y acum:=acum+1) hasta que el valor de i sea mayor ó igual que el valor de n.

Analice las diferencias entre el While y el Repeat.

Sentencia FOR .− Permite ejecutar una ó un grupo de sentencias un número determinado de veces. Esta sentencia se utiliza cuando se conoce el número de iteraciones.

Formato :

(a)

FOR variable:= valor inicial TO valor final DO

Sentencia A;

(b)

FOR variable:= valor inicial TO valor final DO

Begin

Sentencia 1;

. . . . . .

Sentencia n;

End;

En el caso (a), el bucle es la sentencia A.

En el caso (b), el bucle son las sentencias que están entre el Begin y el End.

Modo de ejecución :

1.− La primera vez la variable toma el valor inicial 2.− Se ejecuta el bucle.

3.− Se incrementa el valor de la variable en una unidad.

4.− Vuelve a la linea del For y evalúa si el valor de la variable es menor ó igual al valor final.

5.− Si es verdadero, ejecuta otra vez el bucle, si es falso termina con el For.

Consideraciones :

1.− La variable,el valor inicial y el valor final deben ser del mismo tipo.

2.− El tipo real no es permitido.

3.− El valor inicial y/o el valor final, pueden ser variables ó constantes.

4.− No es posible modificar el valor de la variable.

5.− El valor inicial debe ser menor que el valor final.

6.− En cada iteración se incrementa en una unidad el valor de la variable.

7.− Si desea **decrementar** el valor de la variable, use en vez del TO el DOWNTO y el valor inicial debe ser mayor que el valor final.

8.− La variable debe ser de tipo ordinal.

Ejemplo:

```
Program For01;
Uses Forms;
Var n,i : integer;
acum : integer;
Begin
Write(`Ingrese un numero');
Readln(n) ;
acum := 0 ;
For i:= 1 to n Do
acum := acum + i ;
Writeln(`La suma es `,acum:5);
Readln;
End.
```

Si desea mostrar las sumas parciales, el cuerpo del programa sería así:

```
Begin
Write(`Ingrese un numero');
Readln(n) ;
acum := 0 ;
For i:= 1 to n Do
Begin
acum := acum + i ;
writeln(acum) ;
end ;
Writeln(`La suma es `,acum:5);
Readln;
```

End

Tipos de Datos Ordinales

Un tipo de dato es Ordinal si sus elementos tienen las siguientes Características :

- Existe un primer y un último elemento.
- Cada elemento tiene un elemento que le sucede, salvo el último elemento.
- Cada elemento tiene un elemento que le precede, salvo el primero.

Los tipos de datos que tiene estas características son: Integer, Shortint, Longint, Byte, Word, Boolean, Char y los definidos por el usuario como los enumerados y los subrango.

Ejemplo de un Integer:

. . . ,−2,−1,0,1,2,3,4, ...

Los tipos reales no son ordinales porque no existe un primer y último valor real, además un valor real no tiene un sucesor ó un predecesor definido.

Ejemplo de real : 12.0757 qué número le sucede ó precede??.

Tipos de Datos Subrango

También llamado de intervalo. Es un tipo ordinal que permite definir un valor inicial y un valor final de un conjunto de datos del mismo tipo.

Ejemplo de un subrango:

1..5 está compuesto del 1,2,3,4 y 5.

`A'..'C' está compuesto de la `A', `B' y `C'.

Formato:

Type Nombre = Valor menor . . Valor mayor

Ejemplo :

Type numeros = 1..9;

Type letras = `A'..'C;

Si declaramos un tipo subrango en la sección **Type**, tenemos que declarar la variable en la sección **Var**.

Ejemplo de una declaración de subrangos :

Program Subrango01;

Uses Forms;

Type numero=1..9;

letras = `A'..'C;

Var rangonumeros: numero;

rangoletras: letras;

Es posible definir directamente en la Sección Var, pero no es recomendable.

Tipos de Datos Enumerados

El tipo enumerado es un tipo de dato definido por el usuario, que se compone de una serie de constantes. Este tipo de dato se define en la cláusula Type.

Formato :

Type nombre = (constante1, constante2, ..

constante n)

Ejemplo :

Type verano = (diciembre, enero, febrero);

Var mesverano : verano;

Consideraciones:

1.− El orden se indica por la posición de los valores definidos en Type.

2.− El número de orden de cada conjunto empieza en 0.

3.− No se pueden leer ó escribir estos tipos de datos.

4.− Una constante no puede figurar en 2 listas de tipos enumerados.

Conjuntos

Es un grupo de elementos que puede ser asociado con un solo nombre.

Un conjunto puede tener valores sencillos, así como subrangos.

Formato:

Type Nombre = Set Of tipo base

Ejemplo 1 :

Type

autos = (opel, toyota, datsun);

marca = set of autos;

Var carro: marca;

Todos los datos que podemos asignar a la variable carro deben contener algunos ó todos los valores que definen el conjunto autos. Si asignamos un valor diferente al conjunto autos, produce un error de compilación.

Ejemplo 2:

Type colores = set of (amarillo,rojo,verde);

letras = set of 'A'..'Z';

numeros = set of 0..9;

Var numero : números;

color : colores;

letra : letras;

Arreglos

Un arreglo es un conjunto de elementos dispuestos uno a continuación del otro, que nos permite conservar datos en la memoria de la computadora.

Consideraciones :

1.− Todos los arreglos tienen un tipo de dato asociado.

2.− El acceso se realiza por medio de un índice que identifica el lugar donde se encuentra el dato. El índice empieza en 1.

3.− El arreglo tiene un nombre.

4.− El tipo de dato debe ser igual para todos los elementos del arreglo pero cada elemento puede tener diferentes valores. Al arreglo que tiene un solo índice, se le conoce como vector ó listas.

Al arreglo que tiene 2 índices, se le conoce como matriz bidimensional ó tabla.

Para definir un arreglo usamos uno de los dos formatos siguientes:

Formato 1

```
Var

variable : Array [1. .X,1. .Y] OF tipo de dato
```

Formato2

Type

TArreglo= Array [1. .X,1. .Y] OF tipo de Dato

Var variable : TArreglo

Donde :

Variable = Es el nombre del Arreglo.

Array = Es una palabra reservada que indica que se está definiendo un arreglo.

[1..X] = Indica que el arreglo es de una dimensión y tiene X elementos.

[1..X,1..Y] = Indica que el arreglo es de dos dimensiones y tiene X por Y elementos.

Tipo de dato = Debemos indicar el tipo de dato del arreglo. Por ejemplo Integer, Real, Char, etc.

Tarreglo = Es el nombre del tipo de dato que se esta definiendo. En este caso se está definiendo como Array.

Para hacer referencia a un array usamos el nombre del arreglo .

Para hacer referencia a un elemento del array usamos un índice (si es de una dimensión) ó 2 índices (si es de 2 dimensiones), encerrados entre corchetes. Este índice puede ser una constante ó una variable. Este índice ó índices indican la posición del elemento dentro del arreglo.

Ejemplo : A[5] quinto elemento del arreglo A

B[2,10] hace referencia al elemento que se encuentra en la fila 2 columna 10 del arreglo B.

Con los elementos del arreglo podemos realizar operaciones de lectura, escritura, asignación y de calculo.

Ejemplo de definición de un arreglo de una dimensión con 5 elementos, además asignamos el valor de 26 al tercer elemento:

```
Type TARREGLO=array [1..5] of Integer;

Var A : TARREGLO;
```

ó

```
Var A : array [1..5] of Integer;

N := Byte;

Begin

N := 2;

Readln (A[1] ) ;

Readln (A[N] ) ;

A[3]:= 26;

A[4] := A[1] * A[N] * A[N+1] ;
```

```
Writeln ( A[4] ) ;

End.
```

Ejemplo de definición de un arreglo de dos dimensiones con 20 elementos, además asignamos el valor de 40 al elemento que se encuentra en la intersección de la fila 2 y la columna 3:

```
Type TARREGLO = array [1..4,1..5] of Integer;

Var A : TARREGLO;
```

ó

```
Var A : array [1..4,1..5] of Integer;

Begin

Readln (A[1,2] ) ;

A[2,3]:= 40;

A[3,4] := A[1,2] * A[2,3] ;

Writeln ( A[3,4] ) ;

End.
```

La lectura ó grabación de arreglos, se realiza elemento por elemento. Para realizar una de estas operaciones se pueden utilizar estructuras repetitivas como el For., el While ó el Repeat

Ejemplo.

```
Var A : array [1..5] of Integer;

N := Byte;

Begin

For N:= 1 to 5 Do

Readln (A[N] ) ;

End.
```

Otro ejemplo :

```
Var A : array [1..4,1..5] of Integer;

N,M : Integer;

Begin
```

```
For N:= 1 to 4 Do

Begin

For M:= 1 to 5 do

Begin

Write (`Ingrese el elemento `,N,' `,M);

Readln (A[N,M] ) ;

End;

End;

For N:= 1 to 4 Do

For M:= 1 to 5 Do

Writeln ( A[3,4] ) ;

Readln;

End.
```

En este ejemplo la lectura del arreglo se realiza fila por fila. Es decir ingresamos datos para todos los elementos de la primera fila, luego para todos los elementos de la segunda fila y así sucesivamente hasta la última fila.

¿Que modificaría en este ejemplo para que la lectura sea columna por columna?.

Tipo de dato Registro (Récord)

Un registro (Récord) es un tipo de dato estructurado que consta de un conjunto de elementos que pueden ser del mismo ó de diferentes tipos de datos.

Los componentes de un registro se denominan campos. Cada campo tienen un nombre y un tipo de dato. El tipo Récord se crea con el **Type**.

Formato:

```
Type tipo_registro = Record

campo_1 : tipo 1 ;

campo_2 : tipo 2 ;

. . . . . .

campo_n : tipo n ;
```

End ;

Tipo_registro es el nombre de la estructura de datos.

Campo_1 . . . campo_n , son los campos del registro. Una línea puede tener mas de un campo.

End indica fin de la estructura de datos.

Después de crear un tipo de dato registro, tenemos que crear una variable de ese tipo. Usamos lo siguiente :

Var

Nombre_registro : Nombre Tipo_registro

Nombre_registro, es el nombre que vamos a utilizar en el cuerpo del programa.

Tipo_registro, es el nombre del tipo de registro definido en el Type.

Ejemplo:

```
Program registro01;

uses Forms;

Type

alumnos = record

nombre : string[25];

edad : integer;

Domicilio: string[30];

end;

Var alumno : alumnos;
```

Podemos realizar operaciones de lectura, grabación y asignación con los campos del registro.

Para poder acceder directamente a un campo de un registro utilizamos un selector de campo, como figura en el formato siguiente:

nombre_registro.nombre_campo

Ejemplo (continuando con el ejemplo anterior)

```
begin

Write(`Ingrese el nombre');

readln(alumno.nombre);
```

```
alumno.direccion:=alumno.nombre;

writeln(`La direccion es ',alumno.direccion);

readln

end.
```

Como vemos en el ejemplo para hacer referencia a un campo tenemos que nombrar primero al registro luego un punto y finalmente al nombre del campo.

Hay una forma de abreviar la referencia a un campo, usando la sentencia WITH.

With

La sentencia With permite nombrar a un registro y posteriormente trabajar con los campos solo con indicar sus nombres.

Formato:

With nombre_registro **do**

Begin

{ sentencias }

End;

El cuerpo del ejemplo anterior sería de la siguiente manera;

```
begin

Write(`Ingrese el nombre');

with alumno do

begin

readln(nombre);

dirección := nombre;

writeln(`La dirección es ', dirección);

end;

readln

end.
```

Arreglo de Registros (Array of Record)

El Record definido anteriormente sirve para un solo registro, Podemos agrupar registros en un conjunto de ellos. Esta agrupación lo podemos realizar mediante arreglos ó arrays de registros.

Para definir un arreglo de registro utilizamos la siguiente notación :

Formato:

Type tipo_registro = **Record**

campo_1 : tipo 1 ;

campo_2 : tipo 2 ;

. . . . . .

campo_n : tipo n ;

End ;

Var

Nombre_registro : Array[ ] of Tipo_registro

Usemos el ejemplo anterior de Registros para generar un array de record.

Ejemplo:

```
Program registro02;

uses Forms;

Type

alumnos = record

nombre : string[25];

edad : integer;

Domicilio: string[30];

end;

Var alumno : array [1..10] of alumnos;
```

{ en este caso estamos creando un arreglo de 10 registros, cada fila del arreglo tiene la estructura del record Alumno}

Una forma de trabajar con esta estructura es la siguiente. Vamos a ingresar el nombre de 5 alumnos y después lo vamos a imprimir.

```
begin
```

```
for n:= 1 to 5 do

begin

Write(`Ingrese el nombre');

readln(alumno[n].nombre);

alumno[n].dirección :=alumno[n].nombre;

writeln(`La dirección es ',alumno[n].dirección);

end;

readln;

end.
```

El mismo ejemplo usando el With

```
begin

for n:= 1 to 5 do

begin

With alumno[n] do

Begin

Write(`Ingrese el nombre');

readln(nombre);

dirección := nombre;

writeln(`La dirección es ',dirección);

end;

end;

readln;

end.
```

Podemos definir un arreglo dentro de un registro (tienen el nombre de **registro de arreglo**) . Usemos un ejemplo para ilustrar este concepto.

Un alumno lleva un curso en la universidad, ese curso puede tener 6 notas. La estructura puede tener una de dos formas.

Sin usar registro de arreglo

Type cursos = record

codigo :curso : integer;

codigo_alumno : integer;

nota1 : byte;

nota2 : byte;

nota3: byte;

nota4 : byte;

nota5 : byte;

nota6 : byte;

end;

Var curso : array [1..100] of cursos;

Usando registro de arreglo

Type cursos = record

codigo:curso : integer;

codigo_alumno : integer;

nota : array [1..6] of byte;

end;

Var curso : array [1..100] of cursos;

Note la diferencia en ambas estructuras.

PROCEDIMIENTOS

Es un subprograma que realiza una tarea específica.

Está compuesto por un grupo de sentencias que constituyen una unidad de programación.

Todo procedimiento debe tener un nombre y debe ser declarado antes de ser referenciado en el cuerpo del programa.

El procedimiento debe estar en cualquier parte, pero antes del Begin del programa principal.

Formato 1

Procedure Nompre_procedimiento;

Var { declaración de variables locales}

Begin

{ Cuerpo del procedimiento}

End;

Formato 2

Procedure Nombre_proced (parámetros);

Var { declaración de variables locales}

Begin

{ Cuerpo del procedimiento}

End;

El primer formato no recibe datos del programa que lo llama.

El segundo formato es para procedimientos que van a recibir datos del programa que lo llama, estos datos los recibe a través de los parámetros.

A estos parámetros se le conoce como parámetros formales ó ficticios.

Como vemos en los formatos, los procedimientos tienen 3 partes :

- La cabecera donde aparece la palabra Procedure y el nombre del procedimiento.
- La sección de declaración de variables (es opcional).
- El cuerpo ejecutable del procedimiento que está limitado por un Begin y un End.

Un procedimiento es ejecutado cuando es llamado directamente por su nombre, por un programa ó por otro procedimiento, de acuerdo a uno de los formatos.

Formato 1

Nombre_procedimiento

Formato 2

Nombre_procedimiento (parámetros)

Los parámetros del formato 2 son llamados parámetros actuales, y sirven para enviar valores al procedimiento.

Un procedimiento puede recibir valores ó no, también puede devolver valores ó no

Ejemplo :

Calcular el área de un cilindro

```
Program area2;
Uses Forms;
Const Pi = 3.141592;
Var Radio, altura : Real;
Area : Real;
Procedure Leer;
Begin
Write (ingrese el valor del radio');
readln(radio);
Write (ingrese el valor de la altura');
readln(altura);
End;
Procedure Calcula_Area;
Begin
Area := Pi * Radio * Altura;
End;
Procedure Salida;
Begin
Writeln(`La Altura es `, altura:8:2);
Writeln(`El Radio es `, radio:8:2);
Writeln(`La Area es `, area:8:2);
End;
Begin
Leer;
Calcula_Area;
```

Salida;

readln

End.

En este ejemplo tenemos tres procedimiento, que son ejecutados en el programa principal, tan solo con mencionar su nombre.

PARAMETROS

Permiten la comunicación entre procedimientos ó entre un programa y un procedimiento.

Un parámetro pasa información del programa principal al procedimiento y viceversa.

Existen 2 tipos de parámetros :

- De entrada
- De salida

Los parámetros de entrada tienen valores que deben ser proporcionados por el programa principal.

Los parámetros de salida tienen valores que son calculados en los procedimientos y son devueltos al programa principal.

Los parámetros actuales deben coincidir en número, orden y tipo con los parámetros formales.

Todos los parámetros formales tienen indicado un tipo de dato en la definición del procedure.

Los parámetros formales pueden ser de dos tipos: Parámetros Valor y Variable.

- Parámetro Valor, son parámetros de una sola dirección y se utilizan para proporcionar información a un procedimiento, pero no devuelven valores.
- Parámetro Variable, son parámetros que se utilizan para recibir y/o transmitir valores entre el programa principal y el procedimiento. Aparecen después de la palabra reservada Var.

Ejemplo:

Procedure Opera(P1, P2: Real; VAR P3,P4:Byte);

Los parámetros P1, P2, P3 y P4 son parámetros formales

P1 y P2 son parámetros Valor.

P3 y P4 son parámetros variables.

Ejemplo :

Program area2;

Uses Forms;

```
Const Pi = 3.141592;

Var Radio, altura : Real;

Area : Real;

Procedure Calcula_Area(Xradio,Xaltura:Real);

Begin

Area := Pi * XRadio * XAltura;

End;

Procedure Salida;

Begin

Writeln(`La Altura es `, altura:8:2);

Writeln(`El Radio es `, radio:8:2);

Writeln(`La Area es `, area:8:2);

End;

Begin

Write (`ingrese el valor del radio');

readln(radio);

Write (`ingrese el valor de la altura');

readln(altura);

Calcula_Area(radio, altura);;

Salida;

readln

End.
```

En este ejemplo estamos ejecutando el procedimiento Calcula_Area y estamos enviando valores a través de las variables radio y altura, estos valores son recibidos por los parámetros xradio y xaltura.

FUNCIONES

Una función es un subprograma que recibe valores a través de parámetros y devuelve un resultado.

Existen dos tipos de funciones :

- Funciones predefinidas
- Funciones definidas por el usuario.

Las funciones predefinidas son proporcionadas por el Pascal. A continuación enumeramos algunas de ellas.

Frac.− Devuelve la parte decimal de un número Real. Ej. Frac(4.29) devuelve 0.29

Int.− Devuelve la parte entera de un número real. Ej. Int(4.29) devuelve el real 4.

Round.− Redondea un número real al entero mas próximo. Ej. Round(12.9) devuelve 13.

Inc.− Incrementa un valor de tipo ordinal.

Inc(X) es igual a x := x + 1

Inc(X,n) es igual a x := x + n

Dec.− Decrementa un valor de tipo ordinal.

Dec(X) es igual a x := x − 1

Dec(X,n) es igual a x := x − n

Random .− Genera números seudoaleatorios. Los números se generan a partir de formulas matemáticas complejas.

Ej. Random devuelve un número entre 0 y 1.

Random(5) devuelve un valor entre 0 y 4 .

Las funciones definidas por el usuario, son creadas por el usuario y se declaran en forma similar a los procedimientos.

Formato

FUNCTION Nombre (p1,p2,) : tipo de dato

{ declaración de variables locales}

Begin

{ cuerpo de la función }

nombre := valor de la función {*}

End;

A diferencia del procedimiento, el nombre de la función actúa como una variable que recibe un valor.

Consideraciones :

- Toda función debe tener una sentencia como la que está marcada con {*}.

- La función solo devuelve un valor (el procedimiento 0, 1 ó varios valores.
- El nombre de la función debe tener un tipo de dato que es indicado en la cabecera de la función.
- Para ejecutar una función se nombra el nombre de la función en una sentencia de asignación ó en una sentencia de salida.

El ejemplo utilizado en el procedimiento lo utilizamos en la función.

```
program funcion01;

uses Forms;

const Pi = 3.141592;

Var Radio, altura : Real;

Function Area(Xradio,Xaltura:Real):real;

Begin

Area := Pi * XRadio * XAltura;

End;

Procedure Salida;

Begin

Writeln('La Altura es ', altura:8:2);

Writeln('El Radio es ', radio:8:2);

End;

Begin

Write ('ingrese el valor del radio');

readln(radio);

Write ('ingrese el valor de la altura');

readln(altura);

salida;

Write ('El area es ',Area(radio, altura):8:3);

readln ;

end.
```

En este ejemplo ejecutamos la función dentro de la sentencia Write y además la tratamos como una variable

ya que le estamos dando un formato al momento de imprimir.

Programación Orientada a Objetos −OOP−

(OBJECT PASCAL)

La programación orientada a Objetos permite combinar datos y procesos con el fin de utilizarlos de manera adecuada y eficiente en las tareas relacionadas con los objetos.

Un objeto está compuesto de atributos y de métodos.

Los atributos representan las características del objeto.

Los métodos representan a las operaciones que podemos realizar con los objetos. Son los únicos que pueden utilizar directamente los atributos de los objetos.

Todos los objetos que tienen las mismas características se agrupan en Clases.

En Object Pascal el término Clase es considerado como objeto. Además para definir los métodos se usan los procedimientos y las funciones.

Para trabajar con objetos primero tenemos que definir un tipo de datos en la sección Type de la siguiente manera:

Formato:

Type Nombre−clase = CLASS

Atributo 1 : tipo de dato;

Atributo 2 : tipo de dato;

Atributo N : tipo de dato;

Método A;

Método B;

.

Método Z;

End;

Nombre−clase es un tipo de dato que representa a un conjunto de objetos (clase).

Atributo1, Atributo2, .., AtributoN, son los atributos de la clase. Una línea puede tener mas de un atributo.

Método A, Método B,.. Método Z, son los métodos asociados a la clase definida.

End, representa el fin de la declaración de la clase.

Podemos crear varias clases utilizando un solo Type ó podemos crear cada clase con un Type.

Ejemplo :

```
Program Objeto1;

Uses Forms;

Type Alumno = Class

Codigo : Integer;

Nombre : String;

Edad : Byte;

Procedure Ingresar_Datos;

Procedure Mostrar_Datos;

End;
```

En este ejemplo estamos definiendo a la clase Alumno y a tres de sus atributos. Además se definen 2 métodos. El End indica el fin de la definición de la clase.

El encapsulamiento es una característica de los objetos/clases que consiste en agrupar atributos y métodos dentro de un tipo de objeto.

En el ejemplo anterior los tres atributos y los dos métodos están encapsulados dentro de la clase ALUMNO.

Los métodos se definen en dos partes del programa.

La primera va dentro de la definición de la clase, (en el ejemplo anterior serían Ingresar_datos y Mostrar_datos).

La otra parte aparece en la implementación de los métodos, también llamada definición externa de los métodos. Esto se realiza cuando especificamos el contenido del método. Esta definición se realiza con el nombre de la clase un punto y el nombre del método.

La implementación de Mostrar_Datos sería de la siguiente manera:

```
Procedure Alumno.Mostrar_Datos;

begin

Writeln (`Código del alumno : ',Código) ;

Writeln (`Nombre del alumno : `, Nombre);

Writeln (`Edad del alumno : `, Edad);

End;
```

Si omitimos la palabra **Alumno** en la cabecera del procedimiento, el compilador nos envía un mensaje indicando que las variables código, nombre y edad no han sido definidas.

La definición externa de los métodos se realiza antes del Begin del programa principal y después de haberlos definido dentro de la clase.

Para usar un objeto de la clase en el programa principal, es necesario definirlo en la cláusula Var de la siguiente manera :

Var nombre del objeto : nombre de la clase;

A continuación un ejemplo usando clases:

Ejemplo: definir una clase Circulo e implementar los métodos necesarios para calcular el área y la circunferencia.

```
program ClaseCirculo;

uses Forms;

{−−−−−−−−−−−−−−−−−−}

{ Definimos la clase circulo}

Type circulo = class

{ atributos}

{ definimos 3 atributos}

radio : real;

area : real;

circun: real;

{ metodos}

{ definimos 6 metodos}

constructor crear;

destructor destruir;

procedure calcula_area;

procedure asigna_datos(r:real);

procedure calcula_circun;

procedure muestra_resultado;
```

```
end; { Indica fin de la definición de la clase }

{−−−−−−−−−−−−−−−−−}

{Implementación de los métodos}

constructor circulo.crear;

begin

writeln('objeto creado');

end;

Destructor circulo.destruir;

begin

writeln('objeto destruido');

end;

Procedure circulo.asigna_datos(r:real);

begin

radio := r;

end;

Procedure circulo.calcula_area;

begin

area:= Pi * radio*radio;

end;

Procedure circulo.calcula_circun;

begin

circun:= 2*Pi * radio;

end;

Procedure circulo.muestra_resultado;

begin

Writeln('El area es = ',area:8:2);
```

```
Writeln('La Circunferencia es = ',circun:8:2);

readln;

end;

var

ocirculo : circulo;

wradio : real;

{−−−−−−−−−−− Cuerpo del programa −−−−−−−−−−−−−−−−}

begin

ocirculo:=circulo.crear;

Write('Ingrese el valor del radio '); readln(wradio);

ocirculo.asigna_datos(wradio);

ocirculo.calcula_area;

ocirculo.calcula_circun;

ocirculo.muestra_resultado;

ocirculo.destruir;

end.
```

En este ejemplo estamos definiendo 3 atributos para la clase circulo, además 6 métodos. Existen 2 métodos especiales como son el **Constructor** y el **Destructor** .

El constructor es un método que se asocia a un objeto particular. Un constructor puede inicializar los campos de una variable objeto.

El método constructor debe ser el primero en ejecutarse, pues es el que le da vida al objeto.

El Destructor permite limpiar y liberar ó disponer de los objetos asignados dinámicamente

En el ejemplo de la clase circulo el nombre del constructor es CREAR y en la implementación usamos CIRCULO.CREAR ya que ese método pertenece a la clase circulo. Cuando este método se ejecute va a crear a un objeto de la clase circulo y va a mostrar el mensaje objeto creado .

En el cuerpo del programa la sentencia que crea un objeto es **ocirculo:=circulo.crear**; y a partir de esa sentencia podemos trabajar con los objeto de la clase circulo.

1