

ESTRUCTURA

TEMARIO:

REPASO DE PASCAL.

ESTRUCTURAS DE DATOS LISTA.

Variables de tipo puntero.

- **Definición de estructura.**
- **Implementación en Pascal.**
- **Operaciones.**

PILAS, COLAS Y RECURSIVIDAD.

ESTRUCTURA DE DATOS ARBOLES.

ESTRUCTURA DE DATOS GRAFOS.

LIBROS:

- Programación avanzada y resolución de problemas en Pascal estructura de datos, metodología de la programación e ingeniería del Software (Autores G. M. Steven C. Bruell) Editorial Anaya.
- Gusta a ella es: Pascal y estructura de datos. Date/Lilly Mc Graw Hill.
- Otro clásico es: Estructura de datos. Seymoul Lipschntz Mc Craw Hill. Es todo en Pseudocódigo.

Apuntes realizados por Ignacio Domínguez (Nacho) y Jose Luis Blanco (Chevere)

TEMA I – REPASO PASCAL

1.– ORGANIZACION DE UN PROGRAMA EN PASCAL.

2.– TIPOS DE DATOS.

3.– ENTRADA SALIDA.

4.– ESTRUCTURAS DE CONTROL.

5.– SUBALGORITMOS DE PROCEDIMIENTO Y

SUBALGORITMO DE FUNCION.

6.– PASO DE PARAMETROS.

7.– ESTRUCTURA DE DATOS VECTOR.

8.- ESTRUCTURA DE DATOS REGISTRO.

9.- ESTRUCTURA DE DATOS FICHERO.

1.- ORGANIZACION DE UN PROGRAMA PASCAL:

A) Area de encabezamiento. C) Area de subalgoritmos.

PROGRAM nombre; PROCEDURE

USES librerias; FUNCTION

B) Area de declaraciones. D) Area de programa principal.

CONST identif=valor; BEGIN

TYPE identif=tipo; ...

VAR identif:tipo; END.

2.- TIPOS DE DATOS EN PASCAL:

Los datos se dividen en datos simples y estructurados: son varios datos de tipo simple.

Integer String

SIMPLES Char ESTRUCTURADOS Array

Boolean El resto....

Real

3. ENTRADA. SALIDA.

READ(); WRITE();

READLN(); WRITELN();

4.- ESTRUCTURAS DE CONTROL:

Se dividen en estructuras de:

– **lógica secuencial:** asignaciones y llamadas a procedimientos.

– **lógica condicional:** simple, doble, multiple (**IF ; CASE**)

– **lógica iterativa o repetitiva:** **FOR ; WHILE ; REPEAT**

SIMPLE DOBLE

IF Condición THEN IF Condición THEN

BEGIN BEGIN

END; END

ELSE

BEGIN

END;

IF Múltiple CASE:

CASE Variable OF

Valor1:-----

Valor2:-----

End;

FOR:

FOR Variable:=Valor1 TO Valorn DO

BEGIN

END;

WHILE: REPEAT:

WHILE Condición DO REPEAT

BEGIN -----

----- UNTIL Condición;

END;

EJER 1: *Hacer un programa en Pascal que acepte 20 números de teclado comprendidos entre 1 y 50, y visualice en pantalla cuantos son mayores o iguales que 25 y cuantos son menores.*

Program ejer1;

```

Uses CRT;

Var i,num,may,men: integer;

BEGIN

may := 0; men := 0;

for i := 1 to 20 do

begin

repeat

readln(num);

until (num >= 1) and ( num < 50);

if num >= 25 then may := may+1;

else men := men+1;

end;

writeln(Hay, may, n° mayores o igual a 25 años);

writeln(Hay, men, n° menores a 25 años);

END.

```

EJER 2: *Aceptar el termino cuadrático, lineal e independiente de una ecuación de 2º grado y muestre por pantalla todas sus posibles soluciones.*

$ax^2 + bx + c = 0$ ---cuadrado

1º) si $a=0$ $x = -c/b$

2º) si $a <> 0$ y $b^2 - 4ac = 0$ $x = -b/2a$

3º) si $a <> 0$ y $b^2 - 4ac < 0$ no hay soluciones reales

4º) si $a <> 0$ y $b^2 - 4ac > 0$

PROGRAM RAIZ;

VAR

X,A,B,C,X1,X2:INTEGER;

BEGIN

WRITE('TECLEE TÉRMINO CUADRÁTICO');

```

READLN(A);

WRITE('TECLEE TÉRMINO REAL');

READLN(B);

WRITE('TECLEE TÉRMINO INDEPENDIENTE');

READLN(C);

IF A=0 THEN

BEGIN

X := -C/B;

WRITE('EL RESULTADO ES: ',X);

END;

IF (A<>0) AND (SQR(B)-4AC=0) THEN

BEGIN

X:= -B/2A;

WRITE('EL RESULTADO ES: ',X);

END;

IF (A=0) AND (SQR(B)-4AC<0) THEN

WRITE('NO EXISTEN SOLUCIONES REALES. ');

IF (A<>0) AND (SQR(B)-4AC>0) THEN

BEGIN

X1:= (-B-SQRT(SQR(B)-4AC))/2A;

X2:= (-B+SQRT(SQR(B)-4AC))/2A;

WRITE('LOS RESULTADOS SON: ',X1,',',X2);

END;

END.

```

5.- SUBALGORITMOS DE PROCEDIMIENTOS Y DE FUNCION:

PROCEDURE nombre(); FUNCTION nombre():tipo;

CONST

TYPE

VAR

EJER 3: *Hacer una función que devuelva la suma de los elementos de un vector de n posiciones. Esta cargado con números enteros:*

```
FUNCTION SUMA:integer;
```

```
Var i,aux:integer;
```

```
BEGIN
```

```
aux := 0;
```

```
for i := 1 to n do
```

```
aux := aux+vector[i];
```

```
suma:=aux;
```

```
END;
```

EJER 4: *Hacer una función de tipo boolean que devuelve verdadero si el número n es > 10 y falso en el caso contrario.*

```
FUNCTION ERROR:boolean; **programa principal**
```

```
BEGIN if ERROR then writeln(n es > que 10)
```

```
if n > 10 then error := true else writeln(n no es > que 10);
```

```
else error := false;
```

```
END;
```

EJER 5: *Codificar la potencia.*

```
FUNCTION POTENCIA:integer;
```

```
VAR aux:integer;
```

```
BEGIN
```

```
aux:=1;
```

```
for i:=1 to exp do
```

```
aux:=aux*base;
```

potencia:=aux;

END;

EJER 6: Programa que acepte un número entero y mediante una función de tipo boolean visualice en pantalla si el número es > 10 ó ≤ 10

PROGRAM ent; FUNCTION DECIM : boolean;

VAR num:integer; BEGIN

BEGIN if num>10 then decim:=true

writeln('Teclee un número: '); else decim:=false;

read(num); END;

if decim=T then writeln('El n° es > que 10')

else writeln('El n° es > que 10');

END.

6.- PASO DE PARAMETROS:

PARAMETROS: sirven para enviar informacion desde el punto de llamada del procedimiento o función al subalgoritmo y viceversa.

Parametros formales en las cabeceras: PROCEDURE UNO (parametros formales);

Parametros actuales en la llamada: UNO (parametros actuales);

Según el sentido del flujo de información hay dos tipos de parámetros:

–**PARAMETROS POR VALOR:** son de ida. Desde el punto de llamada hasta el procedimiento. No se modifican a la vuelta porque sólo son de ida.

Procedure Uno (x:integer ; a:char);

–**PARAMETROS POR VARIABLE:** de ida y vuelta. Llevan y devuelven información.

Procedure Dos (VAR x:real ; VAR z:char);

Nº parametros formales = Nº parametros actuales

Las variables **Por Valor** deben de ser:

– Mismo nº de parámetros actuales que en la cabecera de la función (formales).

– Del mismo tipo.

– En el mismo orden.

– En la llamada se pondría una variable, un valor o una expresión.

Las variables **Por Referencia** deben de ser:

– Sólo pueden ponerse variables que contengan un valor.

Ejemplo paso de parámetros.

```
PROGRAM EJERCICIO;
```

```
VAR
```

```
a,b,c:Integer;
```

```
PROCEDURE Parametros (Var x,y:Integer;Z:Integer);
```

```
VAR h:Integer
```

```
BEGIN
```

```
h:=x+2; y:=y+h;
```

```
z:=2*y; x:=z;
```

```
END;
```

```
BEGIN
```

```
a:=1; b:=2; c:=3;
```

```
Parametros (a,b,c);
```

```
WRITE (A); WRITE (B); WRITE (C);
```

```
END.
```

EJER 7: *Dada la siguiente declaración de tipos y de variables que representan la cantidad de vehículos vendidos en los últimos 20 años.*

Se pide:

1. – hacer una funcion que calcule el promedio de ventas de cualquiera de ellos.

2. – hacer un procedimiento que visualica aquellas posiciones del vector que tenga una cantidad superior al promedio de ventas.

```
TYPE VECTOR : ARRAY [1..20] OF INTEGER;
```

```
VAR
```

```
CANTT : INTEGER;
```



```
COCHES,MOTOS,CAMIONES,CICLOMOTORES,TRACTORES:VECTOR;
```

```
FUNCTION PROMEDIO (V1 : VECTOR):REAL;
```

```
VAR
```

```
ACUM,I : INTEGER;
```

```
BEGIN
```

```
ACUM := 0;
```

```
FOR I:=1 TO 20 DO ACUM := ACUM+V1[I];
```

```
PROMEDIO := ACUM/20;
```

```
END;
```

```
PROCEDURE SUPERIOR(X:VECTOR);
```

```
VAR
```

```
PROM:REAL;
```

```
BEGIN
```

```
PROM:=PROMEDIO(X);
```

```
FOR I:=1 TO 20 DO
```

```
IF X[I] > PROM THEN WRITELN(I);
```

```
END;
```

```
BEGIN
```

```
WRITELN('EL PROMEDIO DE COCHES ES: ' PROMEDIO(COCHES));
```

```
WRITELN('EL PROMEDIO DE MOTOS ES: ' PROMEDIO(MOTOS));
```

```
....
```

```
....
```

```
END.
```

7.- ESTRUCTURA DE DATOS VECTOR:

VECTOR: es una estructura de datos lineal, finita de elementos homogéneos que se almacenan secuencialmente en memoria y son referenciados mediante un índice.

IMPLEMENTACION:

TYPE

VECTOR: = Array [lim.inf. ... lim.sup.[OF tipo;

VAR

v1,v2,v3: VECTOR;

Búsqueda de un elemento en un Vector de 2 formas:

Secuencial: Del primero al último.

Dicotómica: En un vector ordenado se va partiendo por la mitad y se va buscando, así sucesivamente.

– **BUSQUEDA DICOTOMICA:**

1.– Calcular el elemento mitad.

2.– Comparar el elemento a buscar con el elemento mitad.

3.– Hacer el vector mas pequeño.

Inf := LI; Sup := LS;

Mitad := [inf+sup[DIV 2;

Parametros: – vector donde busco.

– elemento a buscar.

– posición donde lo encuentra,si no lo encuentra un 0.

TYPE

VECTOR: = Array [lim.inf. ... lim.sup.[OF tipo;

VAR

v1,v2: VECTOR;

PROCEDURE **DICOTOMICA** (v:vector;elem: ;VAR pos:integer);

VAR

Inf, Sup, Mitad: integer;

Enc: boolean;

BEGIN

Inf := LINF;

```

Sup := LSUP;

Enc:= false;

REPEAT

mitad:= (inf+sup) DIV 2;

IF elem = v[mitad] THEN

BEGIN

enc := true;

pos := mitad;

END;

ELSE

IF elem < v[mitad] THEN sup := mitad+1

else inf := mitad+1;

UNTIL (enc = true) OR (inf = sup);

IF NOT enc THEN pos := 0;

END;

```

8.- ESTRUCTURA DE DATOS REGISTRO:

REGISTRO: es una estructura de datos homogeneos o no a los cuales se les llama campos tales que se accede a ellos mediante su propio nombre. Tienen un número ilimitado de elementos y no tienen porque guardar un orden natural en memoria.

IMPLEMENTACION:

```

TYPE registro = RECORD

campo1: tipo;

campo2: tipo;

...

END;

VAR reg1, reg2: registro;

Para acceder : reg1.campo1

```

EJER 8: Se tiene la siguiente declaración de tipos y de variables:

```
TYPE datos=RECORD
```

```
nombre:string;
```

```
provincia:string;
```

```
ventas:ARRAY[1..12] of real;
```

```
END;
```

```
VAR tiendas:ARRAY[1..50] of datos;
```

El vector tiendas almacena el nombre, provincia y las ventas realizadas durante los últimos 12 meses de 50 tiendas de España.

Suponiendo cargada la estructura, se pide un algoritmo que visualice el nombre de las tiendas de cuenca cuyo promedio mensual de ventas sea inferior a 100.000 pts.

```
FOR I:=1 TO 50 DO
```

```
  BEGIN
```

```
    IF TIENDAS[I].PROVINCIA='CUENCA' THEN
```

```
      BEGIN
```

```
        ACUM:=0;
```

```
        FOR J:=1 TO 12 DO
```

```
          SUM:=SUM+TIENDAS[I].VENTAS[J] ;
```

```
        IF SUM/12 < 100000 THEN
```

```
          WRITELN(TIENDAS[I].NOMBRE);
```

```
        END;
```

```
      END;
```

EJER 9: Dada la siguiente declaración de tipos y de variables:

```
TYPE Fecha =RECORD
```

```
Día:1..31;
```

```
Mes: 1..12;
```

```
Año:1960..1995;
```

END;

Alumno=RECORD

Nombre: String;

F–Nac: Fecha;

Notas: ARRAY [1..8] OF REAL;

END;

VAR CLASE2TA: ARRAY [1..35] OF ALUMNO;

Supuesta la estructura que está cargada hacer un algoritmo que visualice el nombre de todos aquellos alumnos con nota media superior a 7.

Visualizar el nombre de los alumnos que han nacido en el mes de Febrero.

Funcion que cuente el número de alumnos.

PROCEDURE VISUALIZAR:

VAR X,Y: INTEGER; PROM: REAL;

BEGIN

FOR X:=1 TO 35 DO

BEGIN

PROM:=0;

FOR Y:=1 TO 8 DO

PROM:=PROM+CLASE2TA[X].NOTAS[Y];

PROM:=PROM/8;

IF PROM >7 THEN WRITELN (CLASE2TA[X].NOMBRE):

END;

END;

FUNCTION NUMERO:INTEGER;

VAR I, CON: INTEGER;

BEGIN

```

CONT:=0;

FOR I:=1 TO 35 DO

BEGIN

IF CLASE2TA[I].F_NAC.MED=2 THEN CONT:=CONT+1;

NUMERO:=CONT;

END;

END;

```

EJER 10: Con la misma declaración de tipos que el ejercicio anterior y con la variable CLASES hacer un algoritmo que visualice el nombre de los alumnos del curso 2TA cuyo año de nacimiento sea 1970 y cuya nota en la asignatura 3 sea inferior a 5.

```

VAR CLASES: ARRAY [1..35] OF ALUMNO;

FOR X:=1 TO 35 DO

BEGIN

IF (CLASES[X].FECHA_NAC.AÑO=1970) AND (CLASES[X].CURSO='2MD') AND (CLASES[X].NOTAS[3] < 5)

THEN WRITELN(CLASES[X].NOMBRE);

END;

```

9.- FICHEROS:

FICHERO: estructura de almacenamiento de información, todos del mismo tipo, el almacenamiento se realiza en disco.

IMPLEMENTACION:

TYPE registro = ...

fichero = FILE OF registro;

VAR f: fichero;

OPERACIONES:

ASSIGN (VAR Fichero, 'C:\Fichero físico'); Asignar.

RESET (Var Fichero);

Para leer un fichero que ya existe; situa el puntero en el primer registro del fichero. Si no existe el fichero daría error.

– Detectar errores:

{ \$I- }

RESET(f)

{ \$I+ }

IF IORESULT = 0 THEN WRITELN("El fichero ya existe")

ELSE WRITELN("El fichero no existe");

REWRITE (Var Fichero);

Sirve para crear un fichero. Situa el puntero en el primer fichero.

READ (Var Fichero, Var Registro);

Sirve para leer 1 registro, leemos un registro.

WRITE (Var Fichero, Var Registro); Escritura.

CLOSE (Var Fichero); Para cerrar el fichero.

EOF (): Devuelve TRUE si es fin de fichero.

FILEPOS (Var Fichero): Requiere un parámetro y me devuelve la posición actual del puntero.

FILESIZE (Var Fichero): Devuelve el nº de registros del fichero.

SEEK (Var Fichero, nº Registro): Para acceder a un determinado registro.

acceder al 4º registro: SEEK(f,3); – Pascal empieza a numerar desde el 0.

EJER 11: *Dada la siguiente declaración de tipos y de variables:*

TYPE Fecha =RECORD

Día:1..31;

Mes: 1..12;

Año:1900..1995;

END;

Alumno=RECORD

Nombre: String;

F–Nac: Fecha;

Notas: ARRAY [1..6] OF REAL;

END;

fichero=FILE OF alumno;

VAR

F:fichero;

Hacer un procedimiento que cargue de teclado el fichero hasta que el usuario diga que no quiere meter mas datos.

Una vez creado hacer un algoritmo que visualice el nombre de los alumnos que tengan más de un 6 de nota media.

PROCEDURE CARGAR;

VAR

REG:ALUMNO;

BEGIN

ASSIGN(F,'A:\PASCAL\ALUMNOS.DAT);

{ \$I- }

RESET(F);

{ \$I+ }

REWRITE(F);

SEGUIR:=`S';

REPEAT

WRITELN(^ Teclee nombre: `);READLN(Reg.nombre);

WRITELN(^ Teclee dia de nacimiento: `);READLN(Reg.fecha_nac.dia);

WRITELN(^ Teclee mes: `);READLN(Reg.fecha_nac.mes);

WRITELN(^ Teclee año: `);READLN(Reg.fecha_nac.año);

WRITELN(^ Teclee el curso: `);READLN(Reg.curso);

FOR I:=1 TO 6 DO

BEGIN


```

WRITE(`Teclee la nota: `);

READ(Reg.notas[I] ;

END;

WRITELN(`¿DESEA SEGUIR?`);

SEGUIR:=READKEY;

UNTIL SEGUIR:=`N`;

CLOSE(F);

END;

```

TEMA II – PUNTEROS

1.– ASIGNACION ESTATICA Y DINAMICA DE MEMORIA.

2.– VARIABLES DE TIPO PUNTERO.

1.– ASIGNACION ESTATICA Y DINAMICA DE MEMORIA:

Una variable tiene **asignación estática** de memoria cuando su tamaño se define en el momento de la compilación. (Ej: A:INTEGER;)

Una variable tiene **asignación dinámica** de memoria cuando se define en la compilación, pero no ocupa memoria (no existe realmente) hasta la ejecución. (Ej: X:INTEGER;).

2.– VARIABLES DE TIPO PUNTERO:

PUNTERO: Es una variable que almacena una dirección de memoria. Las **variables dinámicas** se definen y se accede a ellas a través de las variables de tipo puntero.

IMPLEMENTACION

```
TYPE
```

```
Puntero=^Integer;
```

```
VAR
```

```
P,Q:PUNTERO;
```

```
BEGIN
```

```
NEW (P);
```

NEW: Este procedimiento asigna al puntero P, a través del parámetro una dirección de memoria libre. En esta dirección es donde se almacena la **variable dinámica**.

P[^]: Guarda el contenido de la dirección de memoria.

Ejemplo:

```
TYPE Puntero=^Integer;
```

```
VAR P,Q: puntero;
```

```
BEGIN
```

```
NEW (P);
```

```
NEW (Q);
```

```
Q^:=7; Dirección de memoria de q guarda un 7.
```

```
P^:=5; Dirección de memoria de p guarda un 5.
```

```
P:=Q; Asignación de P, lo que tenga Q.
```

```
WRITE (P^); Visualizo un 7.
```

```
WRITE (Q^); Visualizo un 7.
```

```
END;
```

NIL: Indica que la dirección de memoria es nula.

TEMA III– LISTAS ENLAZADAS

1.– DEFINICION DE LA ESTRUCTURA. LISTA ENLAZADA.

2.– IMPLEMENTACION EN PASCAL.

3.– OPERACIONES CON LISTAS ENLAZADAS:

3.1.– CREACION.

3.2.– RECORRIDO.

3.3.– BUSQUEDA DE UN ELEMENTO.

3.4.– INSERCION DE UN ELEMENTO.

3.5.– BORRADO DE UN ELEMENTO.

4.– LISTAS DOBLES O LISTAS DOBLEMENTE ENLAZADAS.

1.– DEFINICION DE LISTA ENLAZADA:

LISTA ENLAZADA: Es una estructura de datos lineal con un número no limitado de elementos homogéneos llamados NODOS, a los cuales se accede mediante punteros.

¿Que es un NODO?:

Es un elemento formado por 2 partes, la parte de la Izquierda es el **INFO**, es donde guarda el dato y el de la Derecha, es el **SIG**, me da la dirección de memoria del siguiente **NODO**.

info sig NIL

Lista

2.- IMPLEMENTACION EN PASCAL:

TYPE

Puntero=^nodo;

Nodo=RECORD

INFO:...;

SIG:Puntero;

END;

VAR

Lista,Aux:Puntero;

Lista: Puntero comienzo, **me garantiza la lista completa.**

Aux: Puntero auxiliar.

3.- OPERACIONES CON LISTAS ENLAZADAS:

3.1.- CREACION: Estructura dinámica.

NEW (LISTA); Lista es el primer NODO.

READ (LISTA^.Info); Aceptar el campo Info del NODO.

LISTA^.Sig:=NIL; Meto NIL en Sig, para saber que es el último NODO.

Todos los NODOS son iguales, menos el 1º (**jamás debe tocarse el 1º**).

Hay 2 pasos, creación del primer NODO y el Resto.

CREACION CON N NODOS:

BEGIN

NEW (Lista);

READ (Lista^.Info);

```

Aux:=Lista;

FOR I:=1 TO N DO

BEGIN

NEW (Aux^.Sig);

Aux:=Aux^.Sig;

READ (Aux^.Info);

END;

Aux^.Sig:=NIL;

END.

```

3.2.– RECORRIDO EN UNA LISTA ENLAZADA:

Me sitúo en el primer NODO, con el puntero auxiliar y hasta el último no termina.

```

Aux:=Lista;

WHILE Aux <> NIL DO

BEGIN

WRITE (Aux^.Info);

Aux:=Aux^.Sig;

END;

```

3.3.– BUSQUEDA DE UN ELEMENTO EN UNA LISTA ENLAZADA:

Realizar un Procedure que reciba como parámetro el puntero comienzo de una lista enlazada, un elemento del mismo tipo y un puntero sobre el que se devuelve la dirección de memoria del NODO que contiene dicho elemento si es que esta y NIL si no está.

```

PROCEDURE BUSQUEDA(Comienzo:Puntero,Elem:Integer,VAR pos:Puntero)

```

```

VAR

Aux:Puntero;

Enc:Boolean;

BEGIN

Aux:=COMIENZO;

```

```

Enc:=FALSE;

WHILE (NOT Enc) AND (Aux<>NIL) DO

BEGIN

IF AUX^.Info=Elem THEN ENC:=TRUE

ELSE AUX:=AUX^.Sig;

END;

POS:=AUX; Me da la dirección del NODO que busco.

END.

```

3.4.– INSERCIÓN DE UN NODO EN UNA LISTA ENLAZADA:

Por el PRINCIPIO de la Lista:

Inserta el elemento Elem, en el principio de la lista.

NEW (Aux); Pedir memoria.

Aux^.Info:=Elem; Guardar el elemento.

Aux^.Sig:=Lista; Engancharlo.

Lista:=Aux; Cambiar lista al primero.

Por el FINAL de la Lista:

Inserta el Elemento Elem en el final de la lista.(otra forma en pag. sig.)

NEW (Aux);

Aux^.Info:=Elem;

Aux^.Sig:=NIL;

P:=Lista;

WHILE P^.Sig <> NIL DO

BEGIN

P:=P^.Sig;

END;

P^.Sig:=Aux;

EJER 12: Insertar el elemento Elem despues del NODO que apunta P.

NEW (Aux);

Aux^.Info:=Elem;

Aux^.Sig:=P^.Sig;

P^.Sig:=Aux;

Por el FINAL de la Lista: Insertar.Otra forma:

Aux:=Lista;

WHILE Aux^.Sig <> NIL DO

BEGIN

Aux:=Aux^.Sig;

END;

NEW (Aux^.Sig);

Aux:=Aux^.Sig;

Aux^.Info:=Elem;

Aux^.Sig:=NIL;

3.5.– BORRADO DE UN ELEMENTO EN UNA LISTA ENLAZADA:

DISPOSE: Libera memoria, al revés del NEW. La dirección que tiene el puntero que se le pasa como parámetro pasa a ser una dirección libre.

Si la lista no tiene elementos repetidos:

Eliminar de la lista el NODO que contiene el elemento 3.

IF Lista^.Info=Elem THEN

BEGIN

Aux:=Lista;

Lista:=Lista^.Sig;

DISPOSE (Aux);

END

ELSE

BEGIN

Ant:=Lista;

Aux:=Lista^.Sig;

Enc:=FALSE;

END;

WHILE (NOT Enc) AND (Aux<>NIL) DO

BEGIN

IF Aux^.Info=Elem THEN

Enc:=TRUE

ELSE

BEGIN

Ant:=Aux;

Aux:=Aux^.Sig;

END;

END;

IF Enc THEN

BEGIN

Ant^.Sig:=Aux^.Sig;

DISPOSE (Aux);

END;

Si la lista tiene elementos repetidos:

Eliminar de la lista el NODO que contiene el elemento 3.

WHILE (Lista^.Info=Elem) DO

BEGIN

Aux:=Lista;

Lista:=Lista^.Sig;

```

DISPOSE ( Aux );

END;

Aux:=Lista^.Sig;

Ant:=Lista;

WHILE ( Aux< >NIL ) DO

BEGIN

IF Aux^.Info=Elem THEN

BEGIN

Ant^.Sig:=Aux^.Sig;

DISPOSE (Aux);

Aux:=Ant^.Sig;

END

ELSE

BEGIN

Ant:=Aux;

Aux:=Aux^.Sig;

END;

END;

```

EJER 13: *Se desea almacenar en un vector el nombre de 20 especialidades médicas de un hospital. Mediante una estructura de `lista enlazada`, de cada especialidad debe depender la lista de médicos que la practican (solo el nombre); además mediante otra estructura enlazada , de cada médico debe depender la lista de pacientes que tiene a su cargo (solo el nombre) .*

Se pide : a) Hacer el dibujo de la estructura .

b) Declaración de tipos y de variables para su implementación .

c) Suponiendo cargada la estructura, hacer un algoritmo que visualiza el nombre del medico/s de la especialidad de ALERGIA y que tengan más de 10 pacientes.

a)

b) TYPE


```

DIRECCION = ^ PACIENTE ;

PACIENTE = RECORD

NOM_ PAC : STRING ;

PROX : DIRECCION ;

END ;

PUNTERO = ^MEDICO ;

MEDICO = RECORD

NOM_MED : STRING ;

SIG : PUNTERO ;

PROX : DIRECCION ;

END ;

ESPECIALIDAD = RECORD

NOMBRE : STRING ;

SIG : PUNTERO ;

END ;

VAR

HOSPITAL : ARRAY [ 1.. 20 ] OF ESPECIALIDAD ;

AUX_MED : PUNTERO ;

AUX_PAC : DIRECCION ;

c) := 1;

WHILE ( HOSPITAL [ ] . NOMBRE <> `ALERGIA' ) DO

:= +1 ;

AUX_MED := HOSPITAL [ ] . SIG ;

WHILE ( AUX_MED <> NIL ) DO

BEGIN

:= 0 ;

```

```

AUX_PAC := AUX_MED ^ . PROX ;

WHILE ( AUX_PAC <> NIL ) DO

BEGIN

C := C + 1 ;

AUX_PAC := AUX_PAC ^ . PROX ;

END ;

IF ( C > 10 ) THEN WRITELN ( AUX_MED ^ . NOM_MED ) ;

AUX_MED := AUX_MED ^ . SIG ;

END ;

```

EJER 14: *Dada esta declaración de tipos y suponiendo que está cargada la estructura hacer un algoritmo que acepte por teclado una palabra y la busque en la estructura, en caso de no encontrarla la inserta.*

```

TYPE

PUNTERO = ^PALABRA;

PALABRA = RECORD

INFO : STRING;

SIG : PUNTERO;

END;

VAR

DICCIONARIO = ARRAY[ 'A'..'Z' ] OF PUNTERO;

PAL : STRING;

AUX : PUNTERO;

ENC : BOOLEAN;

BEGIN

READLN(PAL);

AUX := DICCIONARIO[PAL[1]];

ENC := FALSO;

WHILE (AUX<>NIL) AND (ENC=FALSO) DO

```

```

BEGIN
IF AUX^.INFO = PAL THEN
ENC := VERDAD
ELSE
AUX := AUX^.SIG;
END;
IF (ENC = FALSO) THEN
BEGIN
NEW(AUX);
AUX^.INFO := PAL;
AUX^.SIG := DICCIONARIO[PAL[1]];
DICCIONARIO[PAL[1]] := AUX;
END;
END;
'A'
'B'
'Z'

```

EJER 15: *Se tiene un fichero PALABRAS.DAT cuya estructura es la siguiente:*

palabra1 Se desea almacenar al contenido del fich en una tabla de esta forma:

palabra2 – cada palabra se almacenará en un nodo de una lista enlazada simple : cuyo puntero a comienzo se encuentra en la posición de la tabla

palabraN correspondiente a la primera letra de dicha palabra (ver dibujo).

Se pide: declarar la estructura y hacer la carga del fichero.

árbol ala

'A'

bueno

'B'

```

READLN(F,PAL); leo una palabra,
X := PAL[1]; una linea del fichero.
'Z'
TABLA : ARRAY [^A..'Z'] OF ... ;
TYPE
FICHERO = TEXT;
PUNERO = ^PALABRA;
PALABRA = RECORD
CONTENIDO : STRING;
SIG : PUNTERO;
END;
LETRA = RECORD
CARAC : CHAR;
SIG : PUNTERO;
END;
VECTOR = ARRAY [^A..'Z'] OF LETRA;
VAR
DICCIONARIO : VECTOR; F : FICHERO;
REG : STRING; AUX : PUNTERO;
BEGIN
FOR C := `A' TO `Z' DO
BEGIN
DICCIONARIO[C].CARAC := C;
DICCIONARIO[C].SIG := NIL;
END;
ASSIGN (F, `A:\PALABRAS.DAT');

```

```

RESET (F);

WHILE NOT EOF(F) DO

BEGIN

READLN (F,REG);

NEW (AUX);

AUX^.CONTENIDO := REG;

AUX^.SIG := DICCIONARIO[REG[1]].SIG;

DICCIONARIO[REG[1]].SIG := AUX;

END;

CLOSE (F);

END;

```

EJER 16: *Se desea almacenar en un vector el nombre de 15 universidades españolas. En forma dinámica de cada universidad dependen las facultades que la forman (solamente el nombre de la facultad). Además de cada facultad y también en forma dinámica dependen los cursos de los cuales se conoce su nº de alumnos.*

Se pide: – implementación de la estructura

– suponiendo cargada la estructura hacer un algoritmo que visualice el nombre de las facultades de la universidad complutense de Madrid que

tengan más de 1000 alumnos en primer curso.

```

TYPE

DIRECCION = ^CURSO;

CURSO = RECORD

NOMBRE : STRING;

NUMERO : INTEGER;

PROX : DIRECCION;

END;

PUNTERO = ^FACULTAD;

FACULTAD = RECORD

NOMBRE : STRIG;

```

```

PROX : DIRECCION;

SIG : PUNTERO;

END;

UNIVER = RECORD

NOMBRE : STRING;

SIG : PUNTERO;

END;

VECTOR = ARRAY[1..15] OF UNIVER;

VAR

UNIVERSIDAD : VECTOR;

AUX_FAC : PUNTERO;

AUX_CUR : DIRECCION;

BEGIN

I:=1;

WHILE (UNIVERSIDAD[I].NOMBRE <> `UCM') DO

I:=I+1;

AUX_FAC := UNIVERSIDAD[I].SIG;

WHILE (AUX_FAC <> NIL) DO

BEGIN

AUX_CUR := AUX_FAC^.PROX;

WHILE (AUX_CUR<>NIL) AND (AUX_CUR^.NOMBRE<>`PRIMERO')

DO

AUX_CUR := AUX_CUR^.PROX;

IF AUX_CUR^.NUMERO >1000 THEN

WRITELN(AUX_FAC^.NOMBRE);

AUX_FAC := AUX_FAC^.SIG;

```

END;

END;

EJER 17: *Hacer una función que reciba como parámetro el puntero a comienzo de una lista enlazada simple cuyos elementos son enteros y devuelva el número de nodos con contenido impar.*

La llamada a la función sería:

```
WRITE(IMPARG(LISTA));
```

```
FUNCTION IMPARG (P : PUNTERO):INTEGER;
```

```
VAR
```

```
TOTAL : INTEGER;
```

```
AUX : PUNTERO;
```

```
BEGIN
```

```
TOTAL := 0;
```

```
AUX := P;
```

```
WHILE (AUX <> NIL) DO
```

```
BEGIN
```

```
IF (AUX^.INFO MOD 2 <> 0) THEN
```

```
TOTAL := TOTAL+1;
```

```
AUX := AUX^.SIG;
```

```
END;
```

```
IMPARG := TOTAL;
```

```
END;
```

4.- LISTAS DOBLES O LISTAS DOBLEMENTE ENLAZADAS:

Es una lista que está enlazada en dos sentidos en la que cada nodo contiene la dirección del anterior y del siguiente.

ant info sig ant info sig ant info sig ant info sig

Comienzo Fin

IMPLEMENTACION:

TYPE

puntero=^nodo;

nodo=RECORD

info: ...;

ant,sig: puntero;

END;

VAR

comienzo, fin: puntero;

CREACIÓN DE UNA LISTA DOBLE CON N NODOS:

NEW (comienzo);

READ (comienzo^.info);

comienzo^.ant:=NIL;

fin:=comienzo;

FOR I:=1 TO N DO

BEGIN

NEW(fin^.sig);

fin^. sig^.ant:=fin;

fin:=fin^.sig

READ (fin^.info);

END;

fin^.sig:=NIL;

INSERTAR UN NODO DESPUES DEL NODO QUE APUNTA P:

NEW (aux);

aux^.info:=elem;

aux^.sig:=p^.sig;

aux^.ant:=p;

p^.sig:=aux;

aux^. sig^ .ant:=aux;

EJER 18: *Hacer un algoritmo que intercambie los nodos que ocupan las posiciones k y k+1, modificando unicamente los campos sig.*

PROCEDURE INTERCAMBIAR(VAR LISTA:PUNTERO; K:INTEGER);

VAR

AUX, ANT : PUNTERO;

CONT : INTEGER;

BEGIN

IF K>=1 THEN

BEGIN

ANT := LISTA;

AUX := LISTA^.SIG;

IF (K=1) AND (AUX<>NIL) THEN

BEGIN

ANT^.SIG := AUX^.SIG;

AUX^.SIG := ANT;

LISTA := AUX;

END

ELSE

BEGIN

CONT := 2; {contador de nodos}

WHILE (CONT<>K) AND (AUX<>NIL) DO

BEGIN

AUX := AUX^.SIG;

ANT := ANT^.SIG;

CON := CONT+1;

```

END;

IF (CONT = K) AND (AUX^.SIG<>NIL) THEN

BEGIN

ANT^.SIG := AUX^.SIG;

AUX^.SIG := ANT^.SIG^.SIG;

AUT^.SIG^.SIG := ANX;

END;

END;

END;

END;

END;

```

TEMA IV– PILAS

1.– Definición de la estructura de datos pila .

2.– Implementación en Pascal .

2.1. Implementación estática o secuencial .

2.2. Implementación dinámica o enlazada .

3.– Operaciones con pilas .

3.1. Limpiar pila .

3.1.1. Función pila vacía .

3.1.2. Función pila llena .

3.2. Insertar un elemento en una pila .

3.3. Extraer un elemento de una pila .

1.– DEFINICIÓN DE LA ESTRUCTURA DE DATOS PILA:

PILA:Estructura de datos lineal de elementos homogéneos , en la cual los elementos entran y salen por un mismo extremo , llamado **tope**, **cabeza** o **cima de la pila**.

Las pilas también son conocidas como listas **LIFO** (`Last In ,First Out ` , el último en entrar es el primero en salir) .

El tipo PILA no existe, hay que diseñarlo:

2.- IMPLEMENTACIÓN EN PASCAL:

2.1.- ESTÁTICA O SECUENCIAL.

TYPE

TIPOPILA = RECORD

DATOS : ARRAY [1.. Max] OF ;

CAB : 0 .. Max ;

END ;

VAR

PILA1,PILA2 : TIPOPILA ;

2.2.- DINÁMICA O ENLAZADA.

TYPE

TIPOPILA = ^ NODO

NODO = RECORD

INFO : ;

SIG : TIPOPILA ;

END ;

VAR

PILA1,PILA2 : TIPOPILA ;

3.- OPERACIONES CON PILAS .

A continuación vamos a ver los algoritmos de las operaciones más comunes que se realizan con pilas :

3.1. LIMPIAR PILA .

En ambos casos (tanto en la estática como en la dinámica) será un procedimiento en el que se pasa como parámetro una pila .

a) **Estática**

PROCEDURE LIMPIA_PILA (VAR PILA1: TIPOPILA) ;

BEGIN

```
PILA1. CAB := 0 ;
```

```
END ;
```

b) **Dinámica**

```
PROCEDURE LIMPIA_PILA ( VAR PILA1: TIPOPILA ) ;
```

```
BEGIN
```

```
PILA1:= NIL ;
```

```
END ;
```

3.2. FUNCION PILA VACIA .

Vamos a hacer una función BOOLEAN que recibe una pila como parámetro y devuelve .T. si está vacía o .F. si no lo está.

a) **Estática**

```
FUNTION PILA_VACIA ( VAR PILA1: TIPOPILA ) : BOOLEAN ;
```

```
BEGIN
```

```
PILA_VACIA := PILA1 . CAB = 0 ;
```

```
END;
```

b) **Dinámica**

```
FUNTION PILA_VACIA ( VAR PILA1: TIPOPILA ) : BOOLEAN ;
```

```
BEGIN
```

```
PILA_VACIA := PILA1 = NIL ;
```

```
END;
```

3.3. FUNCION PILA LLENA .

Vamos a hacer una función BOOLEAN que recibe una pila como parámetro y devuelve .T. si está vacía o .F. si no lo está.

a) **Estática**

```
FUNTION PILA_LLENA ( VAR PILA1: TIPOPILA ) : BOOLEAN ;
```

```
BEGIN
```

```
PILA_LLENA := PILA1 . CAB = MAX ;
```

END;

b) **Dinámica**

* **En la estructura dinámica nunca estará llena .**

3.4. INSERTAR UN ELEMENTO EN UNA PILA .

a) **Estática**

```
PROCEDURE INSERTAR ( VAR PILA1: TIPOPILA ; ELEM : ....) ; BEGIN
```

```
IF PILA_LLENA ( PILA1 ) = FALSE THEN
```

```
BEGIN
```

```
PILA1. CAB := PILA1. CAB+1 ;
```

```
PILA1 . DATOS [ PILA1. CAB ] := ELEM ;
```

```
END;
```

```
END;
```

b) **Dinámica**

```
PROCEDURE INSERTAR ( VAR PILA1: TIPOPILA ; ELEM : ....) ; VAR AUX : TIPOPILA ;
```

```
BEGIN
```

```
NEW ( AUX ) ;
```

```
AUX ^ . INFO := ELEM ;
```

```
AUX ^ . SIG := PILA1 ;
```

```
PILA1:= AUX ;
```

```
END;
```

3.5. EXTRAER UN ELEMENTO DE UNA PILA .

a) **Estática .**

```
PROCEDURE SACAR ( VAR PILA1: TIPOPILA ; ELEM : ....) ;
```

```
BEGIN
```

```
IF NOT ( PILA_VACIA ( PILA1 )) THEN
```

```
BEGIN
```

```
ELEM := PILA1. DATOS [PILA1. CAB];
```

```
PILA1 . CAB:= PILA1. CAB-1;
```

```
END;
```

```
END;
```

b) **Dinámica** .

```
PROCEDURE SACAR ( VAR PILA1: TIPOPILA ; ELEM : ....) ;
```

```
VAR AUX : TIPOPILA ;
```

```
BEGIN
```

```
IF NOT ( PILA_VACIA ( PILA1 )) THEN
```

```
BEGIN
```

```
ELEM := PILA1. INFO ;
```

```
AUX := PILA 1 ;
```

```
PILA1 := PILA1^. SIG ;
```

```
DISPOSE ( AUX ) ;
```

```
END;
```

```
END;
```

EJER 19: *Hacer un algoritmo que visualice el contenido de una lista enlazada simple en orden inverso . El puntero a comienzo es ` lista' . Es decir, dada una lista enlazada simple visualizarla al revés.*

```
LIMPIA_PILA ( PILA1 ) ;
```

```
AUX := LISTA ;
```

```
WHILE ( AUX <>NIL ) DO
```

```
BEGIN
```

```
INSERTAR ( PILA1 , AUX ^. INFO ) ;
```

```
AUX := AUX ^. SIG ;
```

```
END ;
```

```
WHILE NOT PILA_VACIA ( PILA1 ) DO
```

```
BEGIN
```

SACAR (PILA1 , ELEM) ;

WRITELN (ELEM) ;

END ;

EJER 20: *Se dispone de una pila de números enteros y de 2 variables enteras que se llaman VIEJA y NUEVA. Hacer un algoritmo que reemplace de la pila el elemento que contiene VIEJA por el valor de NUEVA, dejando el resto de la pila como estaba.*

limpiar_pila(pilaaux);

enc := falso;

mientras (no pilavacia(pila)) y (no enc)

sacar(pila,h)

si h = vieja

enc := verdad

meter(pila,nueva)

sino

meter(pilaaux,h)

fin

fin

mientras no pilavacia(pilaaux)

sacar(pilaaux,h)

meter(pila,h)

fin

EJER 21: *Averiguar si una frase es políndroma o no lo es.*

nota : cuidado con los blancos. EJ: dabale arroz a la zorra el abad

FUNCTION POLINDROMA(FRASE : STRING):BOOLEAN;

VAR PILA : TIPOPILA;

BEGIN

LIMPIARPILA(PILA);

```

FOR I:=1 TO LENGTH(FRASE) DO
IF FRASE[I]<>` ` THEN EMPILAR(PILA,FRASE[I]);
I:=I+1; SI:=TRUE;
WHILE (NOT PILAVACIA(PILA)) AND (SI) DO
BEGIN
WHILE FRASE[I]=` ` DO I:=I+1;
DESEMPILAR(PILA,VAR);
IF CAR <> FRASE[I] THEN SI:=FALSE
ELSE I:=I+1;
END;
POLINDROMA:=SI;
END;

```

TEMA V- COLAS

1.- Definición de la estructura datos cola.

2.- Implementaciones en Pascal.

2.1.- Estática.

2.2.- Dinámica.

3.- Operaciones.

3.1.- Inicializar cola.

3.2.- Función Colavacia.

3.3.- Función Colallena.

3.4.- Insertar un elemento en la pila.

3.5.- Sacar un elemento de la pila.

1.- DEFINICIÓN.

Estructura lineal de elementos homogeneos, los cuales entran (por final) y salen (por frente) por extremos opuestos, es decir el 1º que entra es el 1º que sale.

Listas **FIFO**.

FRENTE FINAL

2.- IMPLEMENTACIÓN.

Declaración de tipos y variables.

2.1.- ESTÁTICA.

Se representa con un vector y dos números. El **nº frente** me da la posición del primero en salir y el **nº final** el último en entrar.

Vamos a hacer un vector circular.

COLALLEN A COLAVACIA

frente siguiente a final frente siguiente a final

IMPLEMENTACIÓN ERRONEA

Para obtener una implementación correcta dejaremos una posición libre en el vector, tenemos dos opciones para dejar una posición vacía:

1ª- final va una posición por delante del último elemento que entró.

2ª- frente va una posición por detrás del primer elemento en salir.

Vamos a optar por la 2ª opción.

COLALLEN A COLAVACIA

frente siguiente a final frente = final

CONST max= ?

TYPE tipocola = RECORD

datos : ARRAY [1..max] OF ... ;

frente,final : 1..max ;

END;

VAR cola : tipocola;

2.2.- DINÁMICA.

Frente Final

TYPE puntero = ^nodo;

nodo = RECORD

dato : ... ;

sig : puntero; frente final

END;

tipocola = RECORD

frente,final : puntero; COLA

END;

VAR cola : tipocola;

3. OPERACIONES.

3.1.- INICIALIZAR COLAS.

ESTÁTICA (Secuencial):

PROCEDURE INICOLA (VAR cola : tipocola);

BEGIN

cola.frente := max;

cola.final := max;

END;

DINÁMICA:

PROCEDURE INICOLA (VAR cola : tipocola);

BEGIN

cola.frente := NIL;

cola.final := NIL;

END;

3.2.- FUNCIÓN COLAVACIA.

SECUENCIAL:

FUNCTION COLAVACIA (cola : tipocola): BOOLEAN;

BEGIN

colavacia := cola.frente :=cola.final;

END;

DINÁMICA: (igual anterior)

FUNCTION COLAVACIA (cola : tipocola): BOOLEAN;

BEGIN

colavacia := cola.frente :=cola.final;

END;

3.3.– FUNCIÓN COLALLENA.

SECUENCIAL:

FUNCTION COLALLENA (cola : tipocola): BOOLEAN;

VAR siguiente : 1..max;

BEGIN

IF cola.final=max THEN siguiente :=1

ELSE siguiente :=cola.final+1;

END;

DINÁMICA:

La representación de colallena en una lista enlazada no existe ya que una lista nunca se llena.

3.4.– INSERTAR UN ELEMENTO EN LA PILA.

SECUENCIAL:

PROCEDURE INSERTAR (VAR cola : tipocola; elem : ...);

BEGIN

IF colallena(cola) THEN cola.final :=cola.final+1

ELSE IF cola.final=Max THEN cola.final:=1;

cola.datos[cola.final] :=elem;

END;

DINÁMICA:

PROCEDURE INSERTAR (VAR cola : tipocola; elem : ...);

```

BEGIN
IF NOT colavacia cola THEN
BEGIN
NEW(aux);
aux^.dato:=elem;
cola.final^.sig:=aux;
cola.final:=aux;
END
ELSE
BEGIN
NEW(aux);
aux^.dato:=elem;
cola.frente:=aux;
cola.final:=aux;
END;
END;

```

3.5.– SACAR UN ELEMENTO DE LA PILA.

SECUENCIAL: (se extrae por el frente)

```

PROCEDURE EXTRAER (VAR cola : tipocola;VAR elem: ... );
BEGIN
IF NOT colavacia cola THEN
BEGIN
IF cola.frente=Max THEN cola.frente:=1
ELSE cola.frente:=cola.frente+1;
elem:=cola.datos[cola.frente] ;
END;

```

```

END;

DINÁMICA:

PROCEDURE EXTRAER (VAR cola : tipocola;VAR elem: ... );

BEGIN

IF NOT colavacia(cola) THEN

BEGIN

IF cola.frente=cola.final THEN

BEGIN

aux:=cola.frente;

cola.frente:=NIL;

cola.final:=NIL;

elem:=cola.frente^.dato;

DISPOSE(aux);

END

ELSE

BEGIN

elem:=cola.frente^.dato;

aux:=cola.frente;

cola.frente:=aux^.sig;

DISPOSE(aux);

END;

END;

END;

```

EJER 22: *Se tiene un array de 20 colas cuyo nº de elementos puede variar entre 0 y 1000 elementos.*

*Cada cola tiene una prioridad distinta de 1 a 20, siendo la 1 la mas alta y la 20 la mas baja. 1–Hacer la **declaración**. 2–**Insertar** un elemento en la cola de prioridad n. 3–**Extraer** un elemento de la cola de mayor prioridad no vacia.*

```

TYPE
PUNTERO = ^NODO;
NODO = RECORD
DATO : TIPODATO;
SIG : PUNTERO;
END;
TIPOCOLA = RECORD
FRENTE, FINAL : PUNTERO;
END;
VAR COLAS : ARRAY [1..20] OF TIPOCOLA;
PROCEDURE INSERTARELEM(ELEM:TIPODATO; N:INTEGER);
BEGIN inserta un elem en la posición N
INSERTAR(COLAS[N], ELEM);
END;
PROCEDURE DEVOLVERELEM(ELEM:TIPODATO; N:INTEGER);
VAR I:INTEGER; devuelve un elem de la posición
BEGIN con más prioridad no vacia
I:=1;
WHILE (I<=20) AND (COLAVACIA(COLAS[I])) DO I:=I+1;
IF I>20 THEN WRITE('TODAS VACIAS');
END;

```

TEMA VI- RECURSIVIDAD

factorial (n) 1 n=0

n*factorial (n-1) en otro caso

```

FUNCTION FACTORIAL (N:INTEGER):INTEGER;
BEGIN

```

```

IF N = 0 THEN
FACTORIAL := 1
ELSE
FACTORIAL := N*FACTORIAL(N-1)
END;

```

si N=3

factorial(3) = 3*? = 3*2 = 6

factorial(2) = 2*? = 2*1 = 2

factorial(1) = 1*? = 1*1 = 1

factorial(0)= 1

REGLAS DE ALGORITMOS RECURSIVOS

Que tenga salida (N=0), caso base.

Que cada llamada recursiva haga referencia a un problema más cercano a la salida.

EJER 23: *suma de 2 números, X e Y. SUMA(X;Y). Para enteros positivos.*

suma(x,y) x si y=0

1+suma(x,y-1)

3+2 = (3+1)+1 = (3+0)+1+1

```

FUNCTION SUMA (X,Y:INTEGER):INTEGER;

```

```

BEGIN

```

```

IF Y=0 THEN SUMA := X

```

```

ELSE SUMA := 1+SUMA(X,Y-1)

```

```

END;

```

EJER 24: *Hacer un seguimiento:*

```

PROCEDURE UNO (X,Y,Z:CHAR; N:INTEGER);

```

```

BEGIN UNO ('a','b','c',2) `a' ? `c'

```

```

IF N>0 THEN

```

```

BEGIN UNO (^'b','c','a',1) `b' ? `a'
WRITE(X);
UNO(Y,Z,X,N-1); UNO (^'c','a','b',0)
WRITE(Z); no devuelve nada ya que N=0
END; Al final devuelve : a b a c
END;

```

EJER 25:

```

FUNCTION DOLORCABEZA(LISTA:PUNTERO):INTEGER;
BEGIN
IF LISTA <>NIL THEN DOLORCABEZA :=1+DOLORCABEZA(LISTA^.SIG)
ELSE DOLORCABEZA :=DOLORCABEZA(LISTA)
END; no funciona por que no tiene salida

```

EJER 26: *Hacer una función que cuente en forma recursiva el número de nodos de una lista enlazada simple.*

SALIDA LISTA VACIA

```

FUNCTION CONTAR (LISTA:PUNTERO):INTEGER;
BEGIN
IF LISTA=NIL THEN CONTAR:=0
ELSE CONTAR :=1+CONTAR(LISTA^.SIG);
END;

```

EJER 27: *Procedimiento que visualice los contenidos de los nodos de una lista enlazada simple en forma capicua.*

```

PROCEDURE VISUAL (LISTA:PUNTERO);
BEGIN
IF LISTA <>NIL THEN
BEGIN
WRITE (LISTA^.INFO);

```


VISUAL (LISTA^.SIG);

WRITE (LISTA^.INFO);

END;

END;

EJER 28: *Función recursiva que devuelva la suma de los elementos de un vector sabiendo que el primer elemento está en la posición LI y el último en la posición LS.*

FUNCTION SUMA (V:VECTOR ; LI,LS : INTEGER) : INTEGER ;

BEGIN

IF LI = LS THEN SUMA := V[LI] ;

ELSE SUMA := V[LI] + SUMA (V, LI+1, LS);

END;

EJER 29: *Función recursiva de tipo BOOLEAN que devuelva TRUE si el elemento ELEM es miembro de la lista enlazada simple LISTA.*

FUNCTION MIEMBRO (LISTA:PUNTERO ; ELEM: ...):BOOLEAN;

BEGIN

IF LISTA = NIL THEN MIEMBRO := FALSE

ELSE MIEMBRO := MIEMBRO(LISTA^.SIG, ELEM);

END;

EJER 30: *Procedimiento recursivo que reciba una lista enlazada simple como parámetro y devuelva una copia exacta de la lista.*

PROCEDURE COPIAR (LISTA:PUNTERO; VAR COPIA:PUNTERO);

BEGIN

IF LISTA = NIL THEN COPIA := NIL

ELSE

BEGIN

NEW(COPIA);

COPIA^.INFO := LISTA^.INFO;

COPIAR (LISTA^.SIG; COPIA^.SIG);

END;

END;

EJER 31: *Multiplicación recursiva.*

FUNCTION MULTIPLICACION (X, Y: INTEGER):INTEGER;

BEGIN

IF X = 0 THEN MULTIPLICACION := 0

ELSE IF Y = 0 THEN MULTIPLICACION := 0

ELSE MULTIPLICACION := MULTIPLICACION

END;

EJER 32: *División recursiva.*

FUNCTION DIVIDIR (X,Y: INTEGER):INTEGER;

BEGIN

IF Y = 0 OR X = 0 THEN DIVIDIR := 0

ELSE IF X<Y THEN DIVIDIR := 0

ELSE DIVIDIR := 1+DIVIDIR (X-Y, Y);

END;

EJER 33: *Función recursiva que cuente el número de nodos de una lista enlazada simple.*

FUNCTION CONTAR (LISTA:PUNTERO):INTEGER;

BEGIN

IF LISTA = NIL THEN CONTAR := 0

ELSE CONTAR := 1+CONTAR(LISTA^.SIG);

END;

EJER 34: *Hacer una función recursiva que devuelva el número de elementos pares que hay en un vector cuyas dimensiones son LI...LS*

FUNCTION PAR(LI,LS:INTEGER):INTEGER;

BEGIN

IF LI=LS THEN

```

IF V[LI] MOD 2=0 THEN PAR:=1
ELSE PAR:=0
ELSE
IF V[LI] MOD 2=0 THEN PAR:=1+PAR(LI+1,LS)
ELSE PAR:=PAR(LI+1,LS);
END;

```

EJER 35: *Hacer un procedimiento recursivo que visualice el contenido de una lista enlazada simple tal que así: si la lista tiene a b c, visualizaría a b c c b a*

```

PROCEDURE VISUAL(LISTA:PUNTERO);
BEGIN
IF LISTA<>NIL THEN
BEGIN
WRITE(LISTA^.INFO);
VISUAL(LISTA^.SIG);
WRITE(LISTA^.INFO);
END;
END;

```

TEMA VII-ÁRBOLES

- 1.- Definición de la estructura de datos árbol.**
- 2.- Árboles Binarios. Definición y terminología.**
- 3.- Implementación en pascal de los arboles binarios.**
- 4.- Recorrido de los arboles:**
 - 4.1.- Recorrido en INORDEN.**
 - 4.2.- Recorrido en PREORDEN.**
 - 4.3.- Recorrido en POSTORDEN.**
- 5.- Árboles binarios de búsqueda (ABB)**
- 6.- Operaciones con arboles binarios de búsqueda:**

6.1.– Búsqueda de un elemento en un ABB.

6.2.– Inserción de un elemento en un ABB.

6.3.– Borrado de un elemento en un ABB.

7.– Árboles binarios de expresión aritmética.

1.– DEFINICIÓN DE LA ESTRUCTURA DE DATOS ÁRBOL:

Es una estructura no lineal de datos homogéneos tal que establece una jerarquía entre sus elementos.

2.– ÁRBOLES BINARIOS.

Es un árbol que o bien está vacío, o bien está formado por un nodo o elemento Raíz y dos árboles binarios, llamados subárbol izquierdo y subárbol derecho.

Es un árbol que tiene 0, 1, 2 hijos su nodo. Como máximo dos hijos por elem.

TERMINOLOGÍA:

NODO: cada uno de los elementos de un árbol.

SUCESORES DE UN NODO: son los elementos de su subárbol izquierdo y de su subárbol derecho.

NODO HIJO: son los sucesores directos de un Nodo.

NODO TERMINAL O NODO HOJA: es aquel que no tiene hijos.

NIVEL DE UN NODO: es un número entero que se define como 0 para la raíz y

uno más que el nivel de su padre para cualquier otro nodo.

n=0 Para la Raíz.

para cualquier otro nodo 1+ el nivel de su padre.

RAMA: es cualquier camino que se establece entre la raíz y un nodo terminal.

ALTURA O PROFUNDIDAD DE UN ÁRBOL: es el máximo nivel de los nodos de un árbol que coincide con el número de nodos de la rama más larga menos 1(-1)

3.–IMPLEMENTACIÓN EN PASCAL DE LOS ÁRBOLES BINARIOS.

TYPE ARBOL = ^NODO;

NODO=RECORD

INFO:.....;

IZQ,DER:ARBOL;

END;

VAR RAIZ:ARBOL;

RAIZ

izq info der

EJER 36: *Hacer una función recursiva que cuente el n° de nodos que tiene el árbol.*

FUNCTION CONTAR (RAIZ:ARBOL): INTEGER;

BEGIN

IF RAIZ=NIL THEN CONTAR:=0

ELSE

CONTAR:=1 + CONTAR(RAIZ^.IZQ) + CONTAR(RAIZ^.DER)

END;

EJER 37: *Hacer una función que devuelva el n° de nodos terminales que hay en un árbol.*

FUNCTION TERMINAL (RAIZ:ARBOL): INTEGER;

BEGIN

IF RAIZ=NIL THEN TERMINAL:=0

ELSE

IF (RAIZ^.IZQ=NIL) AND (RAIZ^.DER=NIL) THEN

TERMINAL:=1

ELSE

TERMINAL:= TERMINAL(RAIZ^.IZQ)+ TERMINAL(RAIZ^.DER)

END;

4.- RECORRIDOS DE UN ARBOL:

4.1.- RECORRIDO EN INORDEN:

El árbol sería recorrido así: 5,15,17,9,3,1,2,19,25,23

1

15 2

5 9 19

17 3 23

25

Si el árbol no está vacío se realizan los siguientes pasos:

1º.– Recorrer en INORDEN el subárbol izquierdo.

2º.– Procesar la Raíz.

3º.– Recorrer en INORDEN el subárbol derecho.

Recorrido de un árbol en INORDEN:

PROCEDURE INORDEN (RAIZ:ARBOL);

BEGIN

IF RAIZ <> NIL THEN

BEGIN

INORDEN (RAIZ^.IZQ);

PROCESAR (RAIZ^.INFO);

INORDEN (RAIZ^.DER);

END;

END;

4.2.– RECORRIDO EN PREORDEN:

El árbol sería recorrido así: 1,15,5,9,17,3,2,19,23,25

Si el árbol no está vacío se realizan los siguientes pasos:

1º.– Procesar Raíz.

2º.– Recorrer en PREORDEN el izquierdo.

3º.– Recorrer en PREORDEN el derecho.

PROCEDURE PREORDEN (RAIZ:ARBOL);

BEGIN

IF RAIZ <> NIL THEN

```

BEGIN
PROCESAR (RAIZ^.INFO);
PREORDEN (RAIZ^.IZQ);
PREORDEN (RAIZ^.DER);
END;
END;

```

4.2.– RECORRIDO EN POSTORDEN:

El árbol sería recorrido así: 5,17,3,9,15,25,23,19,2,1

Si el árbol no esta vacio se realizan los siguientes pasos:

- 1º.– Recorrer en POSTORDEN el izquierdo.
- 2º.– Recorrer en POSTORDEN el derecho.
- 3º.– Procesar raíz.

```

PROCEDURE POSTORDEN (RAIZ:ARBOL);
BEGIN
IF RAIZ <>NIL THEN
BEGIN
POSORDEN (RAIZ^.IAZQ);
POSORDEN (RAIZ^.DER);
PROCESAR (RAIZ^.INFO);
END;
END;

```

EJER 38: *Hacer un seguimiento del siguiente algoritmo con el árbol que se da.*

```

PROCEDURE EJERCICIO(RAIZ:PUNTERO);
BEGIN
IF RAIZ<>NIL THEN
BEGIN

```

```
IF RAIZ^.INFO MOD 2=0 THEN WRITE(RAIZ^.INFO);
```

```
EJERCICIO(RAIZ^.IZQ);
```

```
EJERCICIO(RAIZ^.DER);
```

```
IF RAIZ^.INFO>0 THEN WRITE(RAIZ^.INFO);
```

```
END;
```

```
END;
```

```
1
```

```
2 9
```

```
3 8 10 15
```

```
11 16
```

```
4 5
```

```
12
```

```
6 13
```

```
7 14
```

Obtendríamos: 2, 4, 4, 6, 7, 6, 5, 3, 8, 8, 2, 10, 12, 14, 14, 13, 12, 11, 10, 16, 16, 15, 9,1

5.- ARBOLES BINARIOS DE BUSQUEDA (A.B.B)

Arbol binario en el cual todos sus nodos cumplen que los nodos de su subarbol izquierdo tienen un valor inferior a él, y los nodos de su subarbol derecho tienen un valor superior a él. No existen valores repetidos.

EJER 39: *Dada la siguiente secuencia de números:*

25, 12, 1, 9, 37, 30, 63, 48, 29, 11, 40

construir el ABB que genera dicha secuencia si el orden en el que entran en el árbol es el que se establece.

```
25
```

```
12 37
```

```
1 30 63
```

```
9 29 48
```

```
11 40
```

EJER 40: *Hacer una función que cuente el número de nodos que tienen contenido impar en un árbol binario.*

(¿cuantos números impares hay?)

primer caso – arbol vacio=0

segundo caso – si la raiz es impar sería 1+los impares del subarbol izquierdo+los impares del subarbol derecho

```
FUNCTION IMPARES(RAIZ:ARBOL):INTEGER;  
  
BEGIN  
  
IF RAIZ=NIL THEN IMPARES:=0  
  
ELSE  
  
IF RAIZ^.INFO MOD 2<>0 THEN  
  
IMPARES:=1+IMPARES(RAIZ^.IZQ)+IMARES(RAIZ^.DER)  
  
ELSE  
  
IMPARES:=IMPARES(RAIZ^.IZQ)+IMARES(RAIZ^.DER);  
  
END;
```

6.– OPERACIONES CON ARBOLES BINARIOS DE BUSQUEDA.

6.1.– BUSQUEDA DE UN ELEMENTO.

Hacer un procedimiento que se llama BUSCAR que reciba como parámetro el puntero raiz de un ABB, un elemento a buscar en la variable ELEM del mismo tipo que los que forman el arbol y una variable POS en la que devolverá la dirección de memoria del nodo que contiene el elemento ELEM, en caso de no encontrarse ese elemento devolverá nil.

```
PROCEDURE BUSCAR (RAIZ:ARBOL; ELEM:INTEGER;VAR POS:ARBOL); BEGIN  
  
IF RAIZ=NIL THEN POS:=NIL  
  
ELSE  
  
IF RAIZ^.INFO=ELEM THEN POS :=RAIZ  
  
ELSE  
  
IF ELEM < RAIZ^.INFO THEN  
  
BUSCAR ( RAIZ^.IZQ, ELEM, POS)  
  
ELSE  
  
BUSCAR (RAIZ^.DER, ELEM, POS)
```

END;

6.2.– INSERCION DE UN ELEMENTO.

Hacer un procedimiento que reciba como parámetro el puntero a la raíz de un ABB y el elemento ELEM y lo inserte en su sitio correspondiente.

PROCEDURE INSERTAR (VAR RAIZ:ARBOL; ELEM:INTEGER);

BEGIN

IF RAIZ=NIL THEN

BEGIN

NEW (RAIZ);

RAIZ^.INFO:=ELEM;

RAIZ^.IZQ:= NIL;

RAIZ^.DER:= NIL;

END

ELSE

IF RAIZ^.INFO=ELEM THEN WRITE (REPETIDO)

ELSE

IF ELEM < RAIZ^.INFO THEN INSERTAR (RAIZ^.IZQ, ELEM)

ELSE INSERTAR (RAIZ^.DER, ELEM);

END;

6.3.– BUSCAR Y ELIMINAR UN ELEMENTO.

elem 1º– encontrar 1–nodo sin hijos

2º– suprimir 2–nodo con 1 hijo

3–nodo con 2 hijos

Con 2 hijos: se sustituye el elemento a borrar por su anterior en un recorrido en INORDEN, y una vez hecho hay que deshacerse de este último. Dentro del subarbol izquierdo del elemento a suprimir, el que esté más a la derecha es el que ocupa su posición, ya que es su anterior en un recorrido en INORDEN.

PROCEDURE ENCONTAR(RAIZ:ARBOL;ELEM:...;VAR ANT:ARBOL);

VAR

```

P:ARBOL;

BEGIN

ANT := NIL ;

P := RAIZ ;

WHILE ( P^. INFO <> ELEM ) AND ( P<> NIL ) DO

BEGIN

ANT := P ;

IF ( ELEM < P^. INFO ) THEN P:= P^. IZQ ;

ELSE P := P^. DER ;

END ;

IF ( P <> NIL ) THEN

BEGIN

IF ( P = RAIZ ) THEN SUPRIMIR ( RAIZ ) ;

ELSE

IF P=ANT^.IZQ THEN SUPRIMIR(ANT^.IZQ);

ELSE SUPRIMIR ( ANT ^. DER ) ;

END ;

END ;

PROCEDURE SUPRIMIR ( VAR P : ARBOL ) ;

VAR

TEMP , ANT : ARBOL ;

BEGIN

IF ( P ^. IZQ = NIL ) THEN P:= P^. DER

ELSE

IF ( P ^. DER = NIL ) THEN P:= P^. IZQ

ELSE

```

```

BEGIN
ANT := P ;
TEMP := P^. IZQ ;
WHILE ( TEMP ^. DER <> NIL ) DO
BEGIN
ANT := TEMP ;
TEMP := TEMP ^. DER ;
END ;
P ^. INFO := TEMP ^. INFO ;
IF ( ANT=P ) THEN ANT ^. IZQ := TEMP ^. IZQ
ELSE ANT ^. DER := TEMP ^. IZQ ;
END ;
END;

```

EJER 41: *Dado un árbol binario de búsqueda , que contiene el nombre, el apellido y la edad de un grupo de personas , ordenados por edades .*

Se pide :

Hacer un algoritmo que visualice los datos de las personas de mayor a menor edad .

```

TYPE
ARBOL = ^ NODO ;
NODO = RECORD
NOMBRE : STRING[20] ;
APELLIDO : STRING[30] ;
EDAD : BYTE ;
DER , IZQ : ARBOL ;
END ;
VAR
RAIZ : ARBOL ;

```

```
PROCEDURE VISUALIZAR(RAIZ : ARBOL) ;
```

```
BEGIN
```

```
IF ( RAIZ <> NIL ) THEN
```

```
BEGIN
```

```
VISUALIZAR(RAIZ ^. DER) ;
```

```
WRITELN(RAIZ ^. NOMBRE) ;
```

```
WRITELN(RAIZ ^. APELLIDO) ;
```

```
WRITELN(RAIZ ^. EDAD ) ;
```

```
VISUALIZAR(RAIZ ^. IZQ) ;
```

```
END ;
```

```
END;
```

EJER 42: *Dado un ABB que contiene números enteros .*

Se pide :

– Hacer un procedimiento que reciba como parámetros: el puntero a raíz del árbol , y un número entero . El procedimiento debe buscar el elemento NUM y una vez encontrado visualizar todos sus antecesores (los anteriores).

```
TYPE
```

```
ARBOL = ^ NODO ;
```

```
NODO = RECORD
```

```
INFO : INTEGER ;
```

```
DER , IZQ : ARBOL ;
```

```
END ;
```

```
VAR
```

```
RAIZ : ARBOL ;
```

```
ENC : BOOLEAN ;
```

```
NUM : INTEGER;
```

```
PROCEDURE BUSCAR(RAIZ : ARBOL ; NUM : INTEGER) ;
```

```

BEGIN
IF (RAIZ<>NIL ) THEN ENC := FALSE
ELSE
IF (NUM = RAIZ ^ . INFO) THEN
BEGIN
ENC := TRUE ;
WRITELN (RAIZ ^ . INFO) ;
END
ELSE
IF ( NUM < RAIZ ^ . INFO ) THEN
BEGIN
BUSCAR( RAIZ ^ . IZQ , NUM ) ;
IF (ENC) THEN WRITELN ( RAIZ ^ . INFO )
END
ELSE
BEGIN
BUSCAR( RAIZ ^ . DER, NUM ) ;
IF (ENC) THEN WRITELN ( RAIZ ^ . INFO )
END ;
END ;

```

TEMA VIII – GRAFOS

1.– Definición de la estructura de datos ` grafo ` y su terminología .

2.– Implementación en Pascal .

2.1.– Matrices de adyacencia (estática) .

2.2.– Lista de adyacencia (dinámica) .

3.– Operaciones .

3.1.– Procedimiento grafovacio o iniciar grafo .

3.2.– Añadir vértice .

3.3.– Añadir arco .

3.4.– Borrar vértice .

3.5.– Borrar arco .

4.– Recorrido de un grafo .

4.1.– En profundidad .

4.2.– En anchura .

1.–DEFINICIÓN DE LA ESTRUCTURA TIPO GRAFO Y TERMINOLOGIA.

Es una estructura de datos no lineal formada por un conjunto finito de **Vertices** unidos por un conjunto finito de **Arcos** .

1 2

3

4 5

6

G : grafo

$V(G) = \{1,2,3,4,5,6\}$ conjunto de vertices

$A(G) = \{(1,2),(1,4),(2,1),(2,3),(2,5),(3,2),(3,5),$

$(4,1),(4,5),(4,6),(5,2),(5,3),(5,4),(6,4)\}$

TERMINOLOGÍA

Grafo no dirigido : Cuando los arcos tienen los dos sentidos.

Grafo dirigido : Cuando los arcos tienen un único sentido.

Vertices adyacentes : Dos vertices son adyacentes cuando están conectados

mediante un arco . Se dice que el arco es incidente a dichos vertices.

Grado de un vertice : El nº de arcos que inciden en dicho vertice . Si el grafo es

dirigido se habla de grado de entrada y grado de salida de dicho vertice.

Camino de un vertice : Del V_i al V_j . Es la secuencia de vertices que hay desde

uno a otro.

Longitud de un camino : Es el nº de arcos que forman el camino .

Grafo valorado : Es un grafo en el que cada arco tienen asociado un valor o peso.

2.- IMPLEMENTACIÓN EN PASCAL .

2.1.- MATRICES DE ADYACENCIA.

El conjunto de arcos se representa mediante una matriz cuadrada de grado igual al número máximo de vértices que puede tener el grafo. La matriz se representa mediante elementos booleanos (un valor verdadero (.T.) , significa que hay camino.

	1	2	3	4	5	6
1	T	T	T	F	F	F
2	T	F	F	F	T	T
3	T	F	F	T	T	F
4	F	F	T	F	T	F
5	F	T	T	T	F	F
6	F	T	F	F	F	F

* **Conjuntovértices**

a) Implementación Estática .

TYPE

TIPOVERTICE = 1.. N ;

TIPOARCO = RECORD

ORIGEN,DESTINO : TIPOVERTICES ;

END ;

CONJUNTOVERTICES = ARRAY [1.. N] OF BOOLEAN ;

CONJUNTOARCOS = ARRAY [1.. N , 1 .. N] OF BOOLEAN ;

TIPOGRAFO = RECORD

VERTICES : CONJUNTOVERTICES ;

ARCOS : CONJUNTOARCOS ;

END ;

VAR

GRAFO : TIPOGRAFO ;

VECTOR : ARRAY[1..N] OF STRING ;

2.2.– LISTA DE ADYACENCIA .

El conjunto de vertices se representa como un vector, el cual tiene un número máximo de posiciones que es igual al número máximo de vertices del grafo . Tambien se llena con valores booleanos .

1	2	3	4	5	6
T	T	T	T	T	T

* Conjuntoarcos

b) Implementación Dinámica .

TYPE

TIPOVERTICE = ;

TIPOARCO = RECORD

ORIGEN,DESTINO : TIPOVERTICES ;

END ;

PUNTEROARCO = ^ NODOARCO ;

NODOARCO = RECORD

INFO : TIPOVERTICE ;

ADYA : PUNTEROARCO ;

END ;

TIPOGRAFO = ^ NODOVERTICE ;

NODOVERTICE = RECORD

INFO : TIPOVERTICES ;

SIG : TIPOGRAFO ;

ADYA : PUNTEROARCOS ;

END ;

VAR

GRAFO : TIPOGRAFO ;

3.– OPERACIONES CON BÁSICAS .

A continuación vamos a ver los algoritmos de las operaciones más comunes que se realizan con grafos:

3.1.– PROCEDIMIENTO GRAFOVACIO O INICIAR GRAFO .

a) Estática .

```
PROCEDURE GRAFO_VACIO ( VAR GRAFO : TIPOGRAFO );  
  
VAR  
  
X , Y : INTEGER ;  
  
BEGIN  
  
FOR X := 1 TO N DO  
  
BEGIN  
  
GRAFO . VERTICES [ X ] := FALSE ;  
  
FOR I:= 1 TO N DO  
  
BEGIN  
  
GRAFO . ARCOS [ X , Y ]:= FALSE ;  
  
END ;  
  
END ;  
  
END ;
```

a) Dinámica .

```
PROCEDURE GRAFO_VACIO ( VAR GRAFO : TIPOGRAFO );  
  
BEGIN  
  
GRAFO ^ . SIG:= NIL ;  
  
GRAFO ^ . ADYA := NIL ;  
  
END ;
```

3.2.– AÑADIR VERTICE .

a) Estática .

```
PROCEDURE AÑADE_VER ( VAR GRAFO : TIPOGRAFO ; VERT :  
TIPOVERTICE ) ;
```

```
BEGIN  
GRAFO . VERTICE [ VERT ] := TRUE ;  
END ;
```

3.3.– AÑADIR ARCO .

a) Estática .

```
PROCEDURE AÑADE_ARC ( VAR GRAFO : TIPOGRAFO ; ARC :  
TIPOARCO ) ;  
BEGIN  
IF ( GRAFO . VERTICES [ ARC . ORIGEN ] = TRUE ) AND  
( GRAFO . VERTICES [ ARC . DESTINO ] = TRUE ) THEN  
GRAFO . ARCOS [ ARC . ORIGEN , ARC . DESTINO ] := TRUE ;  
END ;
```

3.4.– BORRAR VERTICE .

a) Estática .

```
PROCEDURE BORRA_VER ( VAR GARFO : TIPOGRAFO ; VER :  
TIPOVERTICE ) ;  
BEGIN  
GRAFO . VERTICE [ VER ] := FALSE ;  
END ;
```

3.5.– BORRAR ARCO .

a) Estática .

```
PROCEDURE BORRA_ARC ( VAR GRAFO : TIPOGRAFO ; ARC :  
TIPOARCO ) ;  
BEGIN  
GARFO . ARCOS [ ARC . ORIGEN , ARC . DESTINO ] := FALSE ;  
END ;
```

b) Dinámica .

```
PROCEDURE BORRA_ARC (VAR GRAFO : TIPOGRAFO ; ARC :
TIPOARCO ) ;
VAR
POS1 , POS2 : TIPOGRAFO ;
AUX , ANT : PUNTEROARCO ;
ENC : BOOLEAN ;
BEGIN
BUSCAR ( GRAFO , ARC . ORGIGEN , POS1 ) ; --- Buscamos el origen
IF ( POS1 <> NIL ) THEN
BEGIN
BUSCAR ( GRAFO , ARC . DESTINO , POS2 ) ; --- Buscamos destino
IF ( POS2 <> NIL ) THEN
BEGIN
IF ( POS1 ^ . ADYA ^ . INFO = ARC . DESTINO ) THEN
BEGIN
AUX := POS1 ^ . ADYA ;
Eliminamos sí es el 1º ----- POS1 ^ . ADYA := AUX ^ . ADYA ;
DISPOSE ( AUX ) ;
END ;
ELSE
BEGIN
ANT := POS1 ^ . ADYA ;
AUX := ANT ^ . ADYA ;
ENC := FALSE ;
WHILE ( AUX <> NIL ) AND ( NOT ENC ) DO
```

```

BEGIN
IF ( AUX ^ . INFO = ARC . DESTINO ) THEN BEGIN
ENC := TRUE ;
ANT ^ . ADYA := AUX ^ . ADYA ;
DISPOSE ( AUX ) ;
END;
ELSE
BEGIN
ANT := AUX ;
AUX := AUX ^ . ADYA ;
END;
END;
END;
END;
END;
END;

```

4.- RECORRIDO DE UN GRAFO .

Recorrer un grafo significa pasar por todos sus vertices y procesar la información que de esto se desprende , dependiendo del problema que se nos planteó . Este recorrido se puede hacer de dos maneras distintas :

- En profundidad .
- En anchura .

4.1.- EN PROFUNDIDAD .

El recorrido en profundidad (a pesar de la idea anterior) no implica que debamos pasar por todos los nodos (o vertices) y los tengamos que procesar.

Los pasos a realizar son los siguientes :

- Se parte de un vertice V . Se `marca' como vertice procesado o visitado y se procesa .
- Se meten en una pila todos sus adyacentes y se marcan como visitados , y se repite el `proceso 2` hasta que la pila quede vacia .

PROCESO 2 : Se saca un elemento de la pila ; se procesa y se meten en la pila todos sus adyacentes no

visitados .

4.2.– EN ANCHURA .

El recorrido en anchura consta de los siguientes pasos :

– Se parte de un vertice V . Se `marca' como vertice procesado o visitado y se mete en una cola . A continuación se repetirán los siguientes pasos hasta que la cola esté vacía :

1º .– Se saca un elemento de la cola y se procesa .

2º .– Se meten en la cola sus vertices adyacentes no visitados y se marcan como visitados .

	A	B	D	C	E	F
--	---	---	---	---	---	---

Visitados : A , B , D , C , E , F

***Ejercicio .** Se tiene un grafo no dirigido valorado cuyos vertices contienen nombres de ciudades y los arcos representan los vuelos entre las ciudades , siendo el peso , la distancia en Km .*

Se pide :

– *Declaración de tipos y algoritmo que visualice la ciudad más cercana a Madrid con vuelo directo .*

a) TYPE

TIPOARCO = RECORD

ORIGEN,DESTINO : STRING ;

DISTANCIA : INTEGER ;

END ;

PUNTEROARCO = ^ NODOARCO ;

NODOARCO = RECORD

DISTANCIA : INTEGER ;

CIUDAD : STRING ;

ADYA : PUNTEROARCO ;

END ;

TIPOGRAFO = ^ NODOVERTICE ;

NODOVERTICE = RECORD

INFO : TIPOVERTICES ;

```

SIG : TIPOGRAFO ;

ADYA : PUNTEROARCOS ;

END ;

VAR

GRAFO : TIPOGRAFO ;

b) BUSCAR ( GRAFO , ` Madrid ` , POS ) ;

IF ( POS <> NIL ) THEN

BEGIN

AUX := POS ^ . ADYA ;

MENOR := AUX ^ . DISTANCIA ;

CIUDAD := AUX ^ . INFO ;

WHILE ( AUX <> NIL ) DO

BEGIN

AUX := AUX ^ . ADYA ;

IF ( AUX ^ . DISTANCIA < MENOR ) THEN

BEGIN

MENOR := AUX ^ . DISTANCIA ;

CIUDAD := AUX ^ . INFO ;

END ;

WRITELN ( CIUDAD ) ;

END ;

```

Ejercicio . Hacer el ejercicio anterior pero con la implementación estática .

```

a) TYPE

TIPOVERTICE = 1.. N ;

TIPOARCO = RECORD

ORIGEN,DESTINO : TIPOVERTICES ;

```

```

DISTANCIA : INTEGER ;

END ;

CONJUNTOVERTICES = ARRAY [1.. N ] OF BOOLEAN ;

CONJUNTOARCOS = ARRAY [1.. N , 1 .. N ] OF BOOLEAN ;

CIUDADES = ARRAY [ 1 .. N ] OF STRING ;

TIPOGRAFO = RECORD

VERTICES : CONJUNTOVERTICES ;

ARCOS : CONJUNTOARCOS ;

END ;

VAR

GRAFO : TIPOGRAFO ;

, J : INTEGER ;

ENC : BOOLEAN ;

b) I:=1 ;

ENC := FALSE ;

WHILE ( I <= N ) AND ( NOT ENC ) DO

BEGIN

IF ( CIUDADES [ ] = ` Madrid ` ) THEN ENC := TRUE ;

ELSE I:=I+1 ;

MENOR := GRAFO . ARCOS [ ,1] ;

POS := 1 ;

FOR := 2 TO N DO

BEGIN

IF ( GRAFO . ARCOS [ , J] < MENOR THEN

BEGIN

MENOR := GRAFO . ARCOS [ , J] ;

```


POS := J ;

END;

END;

WRITELN (CIUDADES) ;

END;

ESTRUCTURA Pag: 36

Medicos

Pacientes

Especialidades

Vertice

Arco

D

A

B

G

F

E

C