**Resumen**

**0. CONSTANTES Y TIPOS:**

program resumen;

*{########## CONSTANTES ##########}*

const

*{Se definen constantes de salida, de objetivos, de tamaño de vectores, etc.}*

*{########## TIPOS ##########}*

type

  t\_str20=string[20];

  t\_rango\_num=1..tam;

  t\_rango\_str1='a'..'z';

  t\_rango\_str2=(juan,ignacio);

  t\_vector=array[t\_rango\_num] of integer;

  t\_registro=record

    ele1: integer;

    ele2: string;

    ..

  end;

  t\_lista=^t\_nodo;

  t\_nodo=record

    ele: *{integer, string, record, array, etc.}*;

    sig: t\_lista;

  end;

  t\_abb=^t\_nodo;

  t\_nodo=record

    ele: *{integer, string, record, array, etc.}*;

    hi: t\_abb;

    hd: t\_abb;

  end;

**1. MÓDULOS DEL MÓDULO IMPERATIVO:**

*{########## 1. MÓDULO IMPERATIVO ##########}*

*{ORDENACIÓN - VECTORES}*

procedure ordenacion\_seleccion(var vector: t\_vector; dimL: integer);

var

  i, j, k, item: integer;

begin

  for i:= 1 to (dimL-1) do

  begin

    k:=i;

    for j:= (i+1) to dimL do

      if (vector[j]<vector[k]) then

        k:=j;

    item:=vector[k];

    vector[k]:=vector[i];

    vector[i]:=item;

  end;

end;

procedure ordenacion\_insercion(var vector: t\_vector; dimL: integer);

var

  i, j, actual: integer;

begin

  for i:= 2 to dimL do

  begin

    actual:=v[i];

    j:=j-1;

    while ((j>0) and (vector[j]>actual)) do

    begin

      vector[j+1]:=vector[j];

      j:=j-1;

    end;

    v[j+1]:=actual;

  end;

end;

*{RECURSIÓN}*

procedure recursion\_imprimir(lista: t\_lista);

begin

  if (lista<>nil) then

  begin

    writeln(lista^.ele);

    lista:=lista^.sig;

    recursion\_imprimir(lista);

  end;

end;

procedure recursion\_potencia(var pot: integer; x, n: integer);

var

  i: integer;

begin

  if (n=0) then

    pot:=1

  else

    if (n=1) then

      pot:=x

    else

    begin

      recursion\_potencia(x,n-1,pot);

      pot:=pot\*n;

    end;

end;

function recursion\_potencia(x, n: integer): integer;

begin

  if (n=0) then

    recursion\_potencia:=1

  else

    if (n=1) then

      recursion\_potencia:=x

    else

      recursion\_potencia:=x\*recursion\_potencia(x,n-1);

end;

*{ÁRBOLES BINARIOS DE BÚSQUEDA (ABB) - CREACIÓN}*

procedure abb\_crear(var abb: t\_abb);

var

  num: integer;

begin

  readln(num);

  while (num<>num\_salida) do

  begin

    abb\_agregar(abb,num);

    readln(num);

  end;

end;

procedure abb\_agregar(var abb: t\_abb; num: integer);

begin

  if (abb=nil) then

  begin

    new(abb);

    abb^.ele:=num;

    abb^.hi:=nil;

    abb^.hd:=nil;

  end

  else

    if (num<=abb^.ele) then

      abb\_agregar(abb^.hi,num)

    else

      abb\_agregar(abb^.hd,num);

end;

*{ÁRBOLES BINARIOS DE BÚSQUEDA (ABB) - RECORRER}*

procedure abb\_recorrer1(abb: t\_abb);

begin

  if (abb<>nil) then

  begin

    abb\_recorrer1(abb^.hi);

    writeln(abb^.ele);

    abb\_recorrer1(abb^.hd);

  end;

end;

procedure abb\_recorrer2(abb: t\_abb);

begin

  if (abb<>nil) then

  begin

    writeln(abb^.ele);

    abb\_recorrer2(abb^.hi);

    abb\_recorrer2(abb^.hd);

  end;

end;

procedure abb\_recorrer3(abb: t\_abb);

begin

  if (abb<>nil) then

  begin

    abb\_recorrer3(abb^.hi);

    abb\_recorrer3(abb^.hd);

    writeln(abb^.ele);

  end;

end;

*{ÁRBOLES BINARIOS DE BÚSQUEDA (ABB) - MENOR ELEMENTO}*

abb\_cargar(abb);

if (abb<>nil) then

begin

  min:=abb\_minimo1(abb);

  writeln(min);

end.

function abb\_minimo1(abb: t\_abb): integer;

begin

  if (abb^.hi=nil) then

    abb\_minimo1:=abb^.ele

  else

    abb\_minimo1:=abb\_minimo1(abb^.hi);

end;

abb\_cargar(abb);

min:=abb\_minimo2(abb);

if (min<>nil) then

  writeln(min^.ele);

end.

function abb\_minimo2(abb: t\_abb): t\_abb;

begin

  if (abb=nil) then

    abb\_minimo2:=nil

  else

    if (abb^.hi=nil) then

      abb\_minimo2:=abb

    else

      abb\_minimo2:=abb\_minimo2(abb^.hi);

end;

*{ÁRBOLES BINARIOS DE BÚSQUEDA (ABB) - MAYOR ELEMENTO}*

abb\_cargar(abb);

if (abb<>nil) then

begin

  max:=abb\_maximo1(abb);

  writeln(max);

end.

function abb\_maximo1(abb: t\_abb): integer;

begin

  if (abb^.hd=nil) then

    abb\_maximo1:=abb^.ele

  else

    abb\_maximo1:=abb\_maximo1(abb^.hd);

end;

abb\_cargar(abb);

max:=abb\_maximo2(abb);

if (max<>nil) then

  writeln(max^.ele);

end.

function abb\_maximo2(abb: t\_abb): t\_abb;

begin

  if (abb=nil) then

    abb\_maximo2:=nil

  else

    if (abb^.hd=nil) then

      abb\_maximo2:=abb

    else

      abb\_maximo2:=abb\_maximo2(abb^.hd);

end;

*{ÁRBOLES BINARIOS DE BÚSQUEDA (ABB) - BUSCAR}*

abb\_cargar(abb);

readln(num);

if (abb<>nil) then

begin

  ok:=abb\_buscar1(abb,num);

  writeln(ok);

end.

function abb\_buscar1(abb: t\_abb; num: integer): boolean;

begin

  if (abb=nil) then

    abb\_buscar1:=false

  else

    if (abb^.ele=num) then

      abb\_buscar1:=true

    else

      if (num<=abb^.ele) then

        abb\_buscar1:=abb\_buscar1(abb^.hi,num)

      else

        abb\_buscar1:=abb\_buscar1(abb^.hd,num);

end;

abb\_cargar(abb);

readln(num);

buscar:=abb\_buscar2(abb,num);

if (buscar<>nil) then

  writeln('OK');

end.

function abb\_buscar2(abb: t\_abb; num: integer): t\_abb;

begin

  if (abb=nil) then

    abb\_buscar2:=nil

  else

    if (abb^.ele=num) then

      abb\_buscar1:=abb

    else

      if (num<=abb^.ele) then

        abb\_buscar1:=abb\_buscar1(abb^.hi,num)

      else

        abb\_buscar1:=abb\_buscar1(abb^.hd,num);

end;

**2. MÓDULOS DEL MÓDULO OBJETOS:**

**3. MÓDULOS DEL MÓDULO CONCURRENTE:**