

## Resumen

### 0. CONSTANTES Y TIPOS:

```
{##### RESUMEN #####}

program resumen;

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

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

const

    const1=1;
    const2=1.0;
    const3='a';

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

type

    t_str10=string[10];
    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: real;
        ele3: 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:=vector[i];
      j:=i-1;
      while ((j>0) and (vector[j]>actual)) do
        begin
          vector[j+1]:=vector[j];
          j:=j-1;
        end;
      vector[j+1]:=actual;
    end;
  end;

{RECURSIÓN}

procedure recursion_imprimir(lista: t_lista);
begin
  if (lista<>nil) then
    begin
      writeln(lista^.ele);
      recursion_imprimir(lista^.sig);
    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;

```

```
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;  
    end;  
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;  
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;  
    end;  
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;  
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;  
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 (abb^.ele>num) 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_buscar2:=abb
        else
            if (abb^.ele>num) then
                abb_buscar2:=abb_buscar2(abb^.hi,num)
            else
                abb_buscar2:=abb_buscar2(abb^.hd,num);
    end;
```

## **2. MÓDULOS DEL MÓDULO OBJETOS:**

### **3. MÓDULOS DEL MÓDULO CONCURRENTE:**